1

PRODUCT OVERVIEW

SAM8 PRODUCT FAMILY

Samsung's SAM8 family of 8-bit single-chip CMOS microcontrollers offers a fast and efficient CPU with a wide range of integrated peripherals, in various mask-programmable ROM sizes. Analog its major CPU features are:

— Efficient register-oriented architecture
— Selectable CPU clock sources
— Idle and Stop power-down mode release by interrupt
— Built-in basic timer with watchdog function

The sophisticated interrupt structure recognizes up to eight interrupt levels. Each level can have one or more interrupt sources and vectors. Fast interrupt processing (within a minimum of four CPU clocks) can be assigned to specific interrupt levels.

S3C8639/C863A/P863A MICROCONTROLLERS

S3C8639/C863A/P863A single-chip 8-bit microcontrollers are based on the powerful SAM8 CPU architecture. The internal register file is logically expanded to increase the on-chip register space. S3C8639/C863A/P863A contain 32/48 Kbytes of on-chip program ROM.

In line with Samsung's modular design approach, the following peripherals are integrated with the SAM8 core:

— Four programmable I/O ports (total 27 pins)
— One 8-bit basic timer for oscillation stabilization and watchdog functions
— One 8-bit general-purpose timer/counter with selectable clock sources
— One interval timer
— One 12-bit counter with selectable clock sources, including Hsync or Csnc input
— PWM block with seven 8-bit PWM circuits
— Sync processor block (for Vsync and Hsync I/O, Csync input, and Clamp signal output)
— DDC Multi-master and slave-only IIC-Bus
— 4-channel A/D converter (8-bit resolution)

S3C8639/C863A/P863A are a versatile microcontrollers which are ideal for use in multi-sync monitors or in general-purpose applications that require sophisticated timer/counter, PWM, sync signal processing, A/D converter, and multi-master IIC-bus support with DDC. They are available in a 42-pin SDIP or a 44-pin QFP package.

OTP

S3C8639/C863A microcontrollers are also available in OTP (One Time Programmable) version named, S3P863A. S3P863A microcontroller has an on-chip 48-Kbyte one-time-programmable EPROM instead of masked ROM. S3P863A is comparable to S3C8639/C863A, both in function and pin configuration except its ROM size.
S3C8647/F8647 MICROCONTROLLERS

S3C8647/F8647 single-chip 8-bit microcontrollers are based on the powerful SAM8 CPU architecture. The internal register file is logically expanded to increase the on-chip register space. S3C8647/F8647 contain 24 Kbytes of on-chip program ROM.

In line with Samsung’s modular design approach, the following peripherals are integrated with the SAM8 core:

- Three programmable I/O ports (total 19 pins)
- One 8-bit basic timer for oscillation stabilization and watchdog functions
- One 8-bit general-purpose timer/counter with selectable clock sources
- One interval timer
- One 12-bit counter with selectable clock sources, including Hsync or Csync input
- PWM block with six 8-bit PWM circuits
- Sync processor block (for Vsync and Hsync I/O, Csync input, and Clamp signal output)
- DDC Multi-master IIC-Bus
- 4-channel A/D converter (4-bit resolution)

S3C8647/F8647 are a versatile microcontrollers which are ideal for use in multi-sync monitors or in general-purpose applications that require sophisticated timer/counter, PWM, sync signal processing, A/D converter, and multi-master IIC-bus support with DDC. They are available in a 32-pin SDIP/SOP package.

FLASH

S3C8647 microcontroller is also available in Flash version named, S3F8647. S3F8647 microcontroller has an on-chip 24-Kbyte flash cells instead of masked ROM. S3F8647 is comparable to S3C8647, both in function and pin configuration.
## FEATURES

### CPU
- SAM88RC CPU core

### Memory
- S3C8639: 32-Kbyte program memory (ROM)
- S3C863A: 48-Kbyte program memory (ROM)
- S3C8647: 24-Kbyte program memory (ROM)
- S3C8639: 784-byte general-purpose register area
- S3C863A: 1040-byte general-purpose register area
- S3C8647: 400-byte general-purpose register area

### Instruction Set
- 78 instructions
- IDLE and STOP instructions added for power-down modes

### Instruction Execution Time
- Minimum 333 ns (with 12 MHz CPU clock)

### Interrupts
- Ten (nine)* interrupt sources/vectors (S3C8647)*
- Eight (seven)* interrupt level (S3C8647)*
- Fast interrupt feature

### General I/O
- S3C863X: four I/O ports (total 27pins)
- S3C8647: three I/O ports (total 19pins)

### 8-Bit Basic Timer
- Programmable timer for oscillation stabilization interval control or watchdog timer function
- Three selective internal clock frequencies

### Timer/Counters
- One 8-bit Timer/Counter with several clock sources (Capture mode)
- One 12-bit Counter with H-/C-sync and several clock sources
- One Interval Timer

### Low Voltage Detector (LVD & POR)

### Pulse Width Modulator (PWM)
- 8-bit PWM: 7(6)*-Ch (S3C8647)*
  (6-bit basic frame with 2-bit extension)

### Sync-Processor Block
- Vsync-I, Hsync-I, Csync-I input and Vsync-O, Hsync-O, Clamp-O output pins
- Programmable Pseudo sync signal generation
- Auto SOG detection
- Auto H-/V-sync polarity detection
- Composite sync detection

### DDC Multi-Master IIC-Bus 1-Ch
- Serial Peripheral Interface
- Support for Display Data Channel (DDC1/DDC2B/DDC2Bi/DDC2B+)

### Slave Only IIC-Bus 1-Ch (Only S3C863X)
- Serial Peripheral Interface

### A/D Converter
- 4-channel; 8(4)*-bit resolution (S3C8647)*

### Oscillator Frequency
- 8 MHz to 12 MHz crystal operation
- Internal Max. 12 MHz CPU clock

### Operating Temperature Range
- –40 °C to +85 °C

### Operating Voltage Range
- 3.0(4.0)* V to 5.5 V (S3C8647)*

### Package Types
- S3C863X: 42-pin SDIP, 44-pin QFP
- S3C8647: 32-pin SDIP, 32-pin SOP
BLOCK DIAGRAM

Figure 1-1. Block Diagram (S3C863X)
Figure 1-2. Block Diagram (S3C8647)
PIN ASSIGNMENTS

NOTE: The TEST pin must connect to Vss (GND) in the normal operation mode.

Figure 1-3. S3C8639/C863A Pin Assignment (42-SDIP)
NOTE: The TEST pin must connect to Vss (GND) in the normal operation mode.

Figure 1-4. S3C8639/C863A Pin Assignment (44-QFP)
Figure 1-5. S3C8647 Pin Assignment (32-SDIP)

Figure 1-6. S3C8647 Pin Assignment (32-SOP)
## PIN DESCRIPTIONS

Table 1-1. S3C8639/C863A Pin Descriptions

<table>
<thead>
<tr>
<th>Pin Names</th>
<th>Pin Type</th>
<th>Pin Description</th>
<th>Circuit Type</th>
<th>SDIP Pin Numbers</th>
<th>Shared Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0.0</td>
<td>I/O</td>
<td>General-purpose, 8-bit I/O port. Shared functions include three external interrupt inputs and I/O for timer M0. Selective configuration of port 0 pins to input or output mode is supported.</td>
<td>D-1</td>
<td>1</td>
<td>INT0</td>
</tr>
<tr>
<td>P0.1</td>
<td></td>
<td></td>
<td>D-1</td>
<td>2</td>
<td>INT1</td>
</tr>
<tr>
<td>P0.2</td>
<td></td>
<td></td>
<td>D-1</td>
<td>3</td>
<td>INT2</td>
</tr>
<tr>
<td>P0.3 (note)</td>
<td></td>
<td></td>
<td>D-1</td>
<td>4</td>
<td>TM0CAP</td>
</tr>
<tr>
<td>P0.4</td>
<td></td>
<td></td>
<td>D-1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>P0.5 (note)</td>
<td></td>
<td></td>
<td>D-1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>P0.6 (note)</td>
<td></td>
<td></td>
<td>D-1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>P0.7 (note)</td>
<td></td>
<td></td>
<td>D-1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>P1.0 (note)</td>
<td></td>
<td>General-purpose, 8-bit I/O port. Selective configuration is available for port 1 pins to input, push-pull output, n-channel open-drain mode, or IIC-bus clock and data I/O.</td>
<td>E-1</td>
<td>9</td>
<td>SDA1</td>
</tr>
<tr>
<td>P1.1 (note)</td>
<td></td>
<td></td>
<td>E-1</td>
<td>10</td>
<td>SCL1</td>
</tr>
<tr>
<td>P1.2 (note)</td>
<td></td>
<td></td>
<td>E-1</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>P2.0</td>
<td>I/O</td>
<td>General-purpose, 8-bit I/O port Selective configuration of port 2 pins to input or output mode is supported. The port 2 pin circuits are designed to push-pull PWM output and Csyc1 (SOG) signal input.</td>
<td>D-1</td>
<td>20</td>
<td>PWM0</td>
</tr>
<tr>
<td>P2.1</td>
<td></td>
<td></td>
<td>D-1</td>
<td>21</td>
<td>PWM1</td>
</tr>
<tr>
<td>P2.2</td>
<td></td>
<td></td>
<td>D-1</td>
<td>22</td>
<td>PWM2</td>
</tr>
<tr>
<td>P2.3</td>
<td></td>
<td></td>
<td>D-1</td>
<td>23</td>
<td>PWM3</td>
</tr>
<tr>
<td>P2.4</td>
<td></td>
<td></td>
<td>E-1</td>
<td>24</td>
<td>PWM4</td>
</tr>
<tr>
<td>P2.5</td>
<td></td>
<td></td>
<td>E-1</td>
<td>25</td>
<td>PWM5</td>
</tr>
<tr>
<td>P2.6 (note)</td>
<td></td>
<td></td>
<td>E-1</td>
<td>26</td>
<td>PWM6</td>
</tr>
<tr>
<td>P2.7</td>
<td></td>
<td></td>
<td>D-1</td>
<td>32</td>
<td>Csyc1</td>
</tr>
<tr>
<td>P3.0–P3.3</td>
<td>I/O</td>
<td>General-purpose, 8-bit I/O port Selective configuration port 3 pins to input or output mode is supported. Multiplexed for alternative use as A/D converter inputs AD0–AD3.</td>
<td>E-1</td>
<td>35–38</td>
<td>AD0–AD3</td>
</tr>
<tr>
<td>P3.4–P3.7</td>
<td></td>
<td></td>
<td>E</td>
<td>39–42</td>
<td></td>
</tr>
<tr>
<td>Hsync-I</td>
<td>I</td>
<td></td>
<td>A-3</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Vsync-I</td>
<td>I</td>
<td>The pins are sync processor signal I/O and IIC-bus clock and data I/O.</td>
<td>A-3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Clamp-O</td>
<td>O</td>
<td></td>
<td>A</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Hsync-O</td>
<td>O</td>
<td></td>
<td>A</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Vsync-O</td>
<td>O</td>
<td></td>
<td>A</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>SDA0</td>
<td>I/O</td>
<td></td>
<td>G-3</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>SCL0</td>
<td>I/O</td>
<td></td>
<td>G-3</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>V&lt;sub&gt;DD1&lt;/sub&gt;, V&lt;sub&gt;SS1&lt;/sub&gt; (note)</td>
<td>–</td>
<td>Power pins</td>
<td>–</td>
<td>11, 12</td>
<td>–</td>
</tr>
<tr>
<td>V&lt;sub&gt;DD2&lt;/sub&gt;, V&lt;sub&gt;SS2&lt;/sub&gt; (note)</td>
<td>–</td>
<td></td>
<td>–</td>
<td>34, 33</td>
<td>–</td>
</tr>
<tr>
<td>X&lt;sub&gt;IN&lt;/sub&gt;, X&lt;sub&gt;OUT&lt;/sub&gt;</td>
<td>–</td>
<td>System clock I/O pins</td>
<td>–</td>
<td>14, 13</td>
<td>–</td>
</tr>
<tr>
<td>RESET</td>
<td>I</td>
<td>System RESET pin</td>
<td>B</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>I</td>
<td>Factory test pin input</td>
<td>–</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** Not used in S3C8647.
PIN CIRCUITS DIAGRAM

Figure 1-7. Pin Circuit Type A

Figure 1-8. Pin Circuit Type A-3

Figure 1-9. Pin Circuit Type B (RESET)

Figure 1-10. Pin Circuit Type D-1

NOTE: The noise filter must be built in the external interrupts.
Figure 1-11. Pin Circuit Type E

Figure 1-12. Pin Circuit Type E-1

Figure 1-13. Pin Circuit Type G-3
OVERVIEW

S3C8639/C863A/C8647 microcontrollers have two types of address space:

- Internal program memory (ROM)
- Internal register file

The 16-bit address and data bus support program memory operations. The separate 8-bit register bus carries addresses and data between the CPU and the internal register file. S3C8639/C863A/C8647 employ an internal 32/48/24-Kbyte mask-programmable ROM. External memory interface is not implemented.

There are 852/1108/462 8-bit registers in the internal register file. In this space, there are 784/1040/400 registers for general use, 19 for CPU and system control, and 49(43) for peripheral control and data. An area of 16-byte common working register (scratch) is part of the general-purpose register space. Most of these registers serve as either a source or destination address, or as accumulators for data memory operations.
ADDRESS SPACES
S3C8639/C863A/P863A/C8647/F8647

PROGRAM MEMORY (ROM)

Program memory (ROM) stores program code or table data. S3C8639/C863A employ 32/48-Kbytes of mask-programmable program memory. The memory address range is 0H–7FFFH/BFFFH (see Figure 2-1).

S3C8647 employs 24-Kbytes of mask-programmable program memory. The memory address large is 0H-5FFFH.

The first 256 bytes of the ROM (0H–FFH) are reserved for interrupt vector addresses. Unoccupied locations in the address range can be used as normal program memory. When you use the vector address area to store program code, be careful not to overwrite vector addresses stored in these locations.

The ROM address at which program execution starts after a reset is 0100H.

Figure 2-1. Program Memory Address Space
REGISTER ARCHITECTURE

The upper 64-byte area of the S3C8639/C863A/C8647 files is logically expanded to two 64-byte areas, called set 1 and set 2. The upper 32-byte area of set 1 is divided into two register banks, bank 0 and bank 1. The total physical register space is thereby expanded internal register to 864/1120 bytes. Within this physical space, there are 864/1120/462-byte registers, of which 852/1108/450 are addressable.

Given the microcontroller’s 8-bit register bus architecture, up to 256 bytes of physical register space can be addressed as a single page. The S3C8639 register files have three pages, page 0, page 1 and page 2. And the S3C863A register files have four pages, page 0, page 1, page 2 and page 3. The S3C8647 register files have two pages, page 0, and page 1. All page contain 256 bytes respectively.

The extension of physical register space into separately addressable areas (sets, banks, and pages) is enabled by addressing mode restrictions, the select bank instructions SB0 and SB1, and the register page pointer, PP.

Specific register types and areas (in bytes) they occupy in the S3C8639/C863A/C8647 internal register files are summarized in Table 2-1.

<table>
<thead>
<tr>
<th>Register Type</th>
<th>Number of Bytes (S3C8639/C863A)</th>
<th>Number of Bytes (S3C8647)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General-purpose registers (including the 16-byte common working register area)</td>
<td>784/1040</td>
<td>400</td>
</tr>
<tr>
<td>CPU and system control registers</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>Clock, peripheral, I/O control, and data registers</td>
<td>49</td>
<td>43</td>
</tr>
<tr>
<td><strong>Total Addressable Bytes</strong></td>
<td><strong>852/1108</strong></td>
<td><strong>462</strong></td>
</tr>
</tbody>
</table>
NOTE: To address registers in bank 0, bank 1, and the system register area, you must use the register addressing mode. To address working registers, you must use working register addressing mode.

Figure 2-2. Internal Register File Organization (S3C863X)
Figure 2-3. Register File Layout (S3C8647)
REGISTER PAGE POINTER (PP)

The SAM8 architecture supports the logical expansion of the physical 256-byte internal register file (which use an 8-bit data bus) to as many as 16 separately addressable register pages. Page addressing is controlled by the register page pointer (PP, DFH). Two logical pages are implemented in S3C8639/C863A/C8647. These pages are used as general purpose register space.

### Register Page Pointer (PP)

DFH, Set 1, R/W

<table>
<thead>
<tr>
<th>MSB</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
<th>LSB</th>
</tr>
</thead>
</table>

#### Destination register page selection bits:

- 0 0 0 0 B: Destination: page 0
- 0 0 0 1 B: Destination: page 1
- 0 0 1 0 B: Destination: page 2 (Not used for the S3C8647)
- 0 0 1 1 B: Destination: page 3 (Not used for the S3C8639)
- 0 1 0 0 B: Not used for the S3C8639/C863A/C8647
- 1 1 1 1 B: Not used for the S3C8639/C863A/C8647

#### Source register page selection bits:

- 0 0 0 0 B: Source: page 0
- 0 0 0 1 B: Source: page 1
- 0 0 1 0 B: Source: page 2 (Not used for the S3C8647)
- 0 0 1 1 B: Source: page 3 (Not used for the S3C8639)
- 0 1 0 0 B: Not used for the S3C8639/C863A/C8647
- 1 1 1 1 B: Not used for the S3C8639/C863A/C8647

### REGISTER SET 1

The term *set 1* refers to the upper 64 bytes of the register file, locations C0H–FFH. The upper 32-byte area of this 64-byte space (E0H–FFH) is divided into two 32-byte register banks, *bank 0* and *bank 1*. You execute the set register bank instructions SB0 or SB1 to address one bank or the other. Bank 0 is automatically selected by a reset operation.

In S3C8639/C863A, register locations of only E0H–F4H are addressable in the bank 1 area; the remaining locations (F5H–FFH) are not mapped. The lower 32-byte area of set 1 is not banked and can be addressed at any time. It contains 16 mapped system registers (D0H–DFH) and a 16-byte “scratch” area (C0H–CFH) for working register addressing.

Registers in set 1 are directly accessible at all times using Register addressing mode. The 16-byte working register area can only be accessed using working register addressing. (For more information about working register addressing, please refer to Chapter 3, “Addressing Modes.”)

### REGISTER SET 2

The same 64-byte physical space that is used for set 1 register locations C0H–FFH is logically duplicated to add another 64 bytes of space. This expanded area of the register file is called *set 2*. All set 2 locations (C0H–FFH) can be addressed in all page of the S3C8639/C863A register space.

The logical division of set 1 and set 2 is maintained by means of addressing mode. In order to access set 1, you should use register addressing mode. When you want to access register locations in set 2, you have to select Register Indirect addressing mode or Indexed addressing mode access register locations in set 2.
PRIME REGISTER SPACE

The lower 192 bytes of the 256-byte physical internal register file (00H–BFH) is called the prime register space, or more simply, the prime area. You can access registers in this address range at all page using any of the seven explicit addressing modes (see chapter 3, "Addressing Modes"). All registers in the prime area can be addressed immediately after a reset.

Figure 2-5. Set 1, Set 2, and Prime Area Register Map (S3C863X)
Register pointers RP0 and RP1 point to the common working register area, locations C0H-CFH, after a reset.

RP0 = \[\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\]

RP1 = \[\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}\]

Figure 2-6. Set 1, Set 2, and Prime Area Register Map (S3C8647)
WORKING REGISTERS

Instructions can access specific 8-bit registers or 16-bit register pairs using either 4-bit or 8-bit address fields. When 4-bit working register addressing is used, the 256-byte register file can be seen by the programmer as one that consists of 32 8-byte register groups or "slices." Each slice comprises of eight 8-bit registers.

With the two 8-bit register pointers, RP1 and RP0 employed, two working register slices can be selected at any time to form a 16-byte working register block. The register pointers help you move this 16-byte register block to anywhere in the addressable register file, except for the set 2 area.

The terms slice and block are used in this manual to help you visualize the size and relative locations of selected working register spaces:

- One working register slice is 8 bytes (eight 8-bit working registers; R0–R7 or R8–R15)
- One working register block is 16 bytes (sixteen 8-bit working registers; R0–R15)

All the registers in an 8-byte working register slice have the same binary value for their five most significant address bits. This makes it possible for each register pointer to point to one of the 24 slices in the register file. The base addresses for the two 8-byte register slices selected are contained in register pointers RP0 and RP1.

After a reset, RP0 and RP1 always point to the 16-byte common area in set 1 (C0H–CFH).

---

**Figure 2-7. 8-Byte Working Register Areas (Slices)**

- **RP1 (Registers R8-R15)**
  - 1 1 1 1 1 X X X
  - Each register pointer points to one 8-byte slice of the register space, selecting a total of 16-byte working register block.

- **RP0 (Registers R0-R7)**
  - 0 0 0 0 0 X X X

---
USING THE REGISTER POINTERS

Register pointers of RP0 and RP1 which are mapped to the addresses D6H and D7H in set 1, are used to select two movable 8-byte working register slices in the register file. After a reset, they point to the working register common area: RP0 points to the addresses C0H–C7H, and RP1 points to the addresses C8H–CFH.

You can change a register pointer value, by loading a new value to RP0 and/or RP1 using an SRP or LD instruction (see Figures 2-6 and 2-7).

In working register addressing, you can only access those two 8-bit slices of the register file that are currently pointed to by RP0 and RP1. You cannot use the register pointers to select a working register area in set 2, C0H–FFH, because these locations can be accessed only with Indirect Register or Indexed addressing modes.

The 16-byte working register block selected usually consists of two contiguous 8-byte slices. As a general programming guideline, we recommend that RP0 point to the "lower" slice and RP1 point to the "upper" slice (see Figure 2-6). In some cases, it may be necessary to define working register areas in different (non-contiguous) areas of the register file. In Figure 2-7, RP0 points to the "upper" slice and RP1 to the "lower" slice.

As a register pointer can point to either of the two 8-byte slices in the working register block, you can flexibly define the working register area to support a variety of program requirements.

PROGRAMMING TIP — Setting the Register Pointers

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Register Pointer</th>
<th>RP0 Value</th>
<th>RP1 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRP #70H</td>
<td>RP0</td>
<td>70H</td>
<td>78H</td>
</tr>
<tr>
<td>SRP1 #48H</td>
<td>RP0</td>
<td>no change</td>
<td>48H</td>
</tr>
<tr>
<td>SRP0 #0A0H</td>
<td>RP0</td>
<td>00H</td>
<td>no change</td>
</tr>
<tr>
<td>CLR RP0</td>
<td>RP0</td>
<td>A0H</td>
<td>no change</td>
</tr>
<tr>
<td>LD RP1,#0F8H</td>
<td>RP0</td>
<td>no change</td>
<td>0F8H</td>
</tr>
</tbody>
</table>

![Figure 2-8. Contiguous 16-Byte Working Register Block](image-url)
PROGRAMMING TIP — Calculate the Sum of a Series of Registers Using the RPs

Calculate the sum of registers 80H–85H using the register pointer and working register addressing. The register addresses from 80H through 85H contain the values 10H, 11H, 12H, 13H, 14H, and 15H, respectively:

\[
\begin{align*}
\text{SRP0} & \#80H & \text{; R0} & \leftarrow 80H \\
\text{ADD} & \text{R0,R1} & \text{; R0} & \leftarrow \text{R0} + \text{R1} \\
\text{ADC} & \text{R0,R2} & \text{; R0} & \leftarrow \text{R0} + \text{R2} + \text{C} \\
\text{ADC} & \text{R0,R3} & \text{; R0} & \leftarrow \text{R0} + \text{R3} + \text{C} \\
\text{ADC} & \text{R0,R4} & \text{; R0} & \leftarrow \text{R0} + \text{R4} + \text{C} \\
\text{ADC} & \text{R0,R5} & \text{; R0} & \leftarrow \text{R0} + \text{R5} + \text{C}
\end{align*}
\]

The sum of these six registers, 6FH, is located in the register R0 (80H). The instruction string used in this example takes 12 bytes of instruction code and its execution time is 24 cycles. If the register pointer is not used to calculate the sum of these registers, the following instruction sequence would have to be used:

\[
\begin{align*}
\text{ADD} & \text{80H,81H} & \text{; 80H} & \leftarrow (\text{80H}) + (\text{81H}) \\
\text{ADC} & \text{80H,82H} & \text{; 80H} & \leftarrow (\text{80H}) + (\text{82H}) + \text{C} \\
\text{ADC} & \text{80H,83H} & \text{; 80H} & \leftarrow (\text{80H}) + (\text{83H}) + \text{C} \\
\text{ADC} & \text{80H,84H} & \text{; 80H} & \leftarrow (\text{80H}) + (\text{84H}) + \text{C} \\
\text{ADC} & \text{80H,85H} & \text{; 80H} & \leftarrow (\text{80H}) + (\text{85H}) + \text{C}
\end{align*}
\]

The sum of the six registers, here, is also located in the register 80H. This instruction string, however, takes 15 bytes of instruction code instead of 12 bytes, and its execution time is 30 cycles instead of 24 cycles.
REGISTER ADDRESSING

The SAM8 register architecture provides an efficient method of working register addressing that takes full advantage of shorter instruction formats to reduce execution time.

With Register (R) addressing mode, in which the operand value is the content of a specific register or register pair, you can access all locations in the register file except for set 2. With working register addressing, you use a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space.

Registers are addressed either as a single 8-bit register or as a paired 16-bit register space. In a 16-bit register pair, the address of the first 8-bit register is always an even number and the address of the next register is always an odd number. The most significant byte of the 16-bit data is always stored in the even-numbered register; the least significant byte is always stored in the next (+1) odd-numbered register.

Working register addressing differs from Register addressing in a way that it uses a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space (see Figure 3-2).

\[
\begin{array}{|c|c|}
\hline
\text{MSB} & \text{LSB} \\
R_n & R_{n+1} \\
\hline
\end{array}
\]

\[ n = \text{Even address} \]

**Figure 2-10. 16-Bit Register Pair**
Each register pointer (RP) can independently point to one of the 24 8-byte "slices" of the register file (other than set 2). After a reset, RP0 points to locations C0H-C7H and RP1 to locations C8H-CFH (the common working register area).
COMMON WORKING REGISTER AREA (C0H–CFH)

After a reset, register pointers RP0 and RP1 automatically select two 8-byte register slices in set 1, locations C0H–CFH, as the active 16-byte working register block:

\[
\begin{align*}
\text{RP0} & \rightarrow \text{C0H–C7H} \\
\text{RP1} & \rightarrow \text{C8H–CFH}
\end{align*}
\]

This 16-byte address range is called common working register area. That is, locations in this area can be used as working registers by operations that address any location on any page in the register file. Typically, these working registers serve as temporary buffers for data operations between different pages.

![Diagram of common working register area](image-url)

Figure 2-12. Common Working Register Area (S3C863X)
Register pointers RP0 and RP1 point to the common working register area, locations C0H-CFH, after a reset.

RP0 = 1 1 0 0 0 0 0 0
RP1 = 1 1 0 0 1 0 0 0

Figure 2-13. Common Working Register Area (S3C8647)
PROGRAMMING TIP — Addressing the Common Working Register Area

As the following examples show, you should access working registers in the common area, locations C0H–CFH, using working register addressing mode only.

**Examples:**

1. **LD 0C2H,40H** ; Invalid addressing mode!
   
   Use working register addressing instead:
   
   ```
   SRP #0C0H
   LD R2,40H ; R2 (C2H) ← the value in location 40H
   ```

2. **ADD 0C3H,#45H** ; Invalid addressing mode!
   
   Use working register addressing instead:
   
   ```
   SRP #0C0H
   ADD R3,#45H ; R3 (C3H) ← R3 + 45H
   ```
4-BIT WORKING REGISTER ADDRESSING

Each register pointer defines a movable 8-byte slice of working register space. The address information stored in a register pointer serves as an addressing "window" that makes it possible for instructions to access working registers very efficiently using short 4-bit addresses. When an instruction addresses a location in the selected working register area, the address bits are concatenated in the following way to form a complete 8-bit address:

— The high-order bit of the 4-bit address selects one of the register pointers ("0" selects RP0; "1" selects RP1);
— The five high-order bits in the register pointer select an 8-byte slice of the register space;
— The three low-order bits of the 4-bit address select one of the eight registers in the slice.

As shown in Figure 2-11, the result of this operation is that the five high-order bits from the register pointer are concatenated with the three low-order bits from the instruction address to form the complete address. As long as the address stored in the register pointer remains unchanged, the three bits from the address will always point to an address in the same 8-byte register slice.

Figure 2-12 shows a typical example of 4-bit working register addressing: the high-order bit of the instruction "INC R6" is "0", which selects RP0. The five high-order bits stored in RP0 (01110B) are concatenated with the three low-order bits of the instruction's 4-bit address (110B) to produce the register address 76H (01110110B).

Figure 2-14. 4-Bit Working Register Addressing
Figure 2-15. 4-Bit Working Register Addressing Example
You can also use 8-bit working register addressing to access registers in a selected working register area. To initiate 8-bit working register addressing, the upper four bits of the instruction address must contain the value of 1100B. This 4-bit value (1100B) indicates that the remaining four bits have the same effect as 4-bit working register addressing.

As shown in Figure 2-13, the lower nibble of the 8-bit address is concatenated in much the same way as for 4-bit addressing: Bit 3 selects either RP0 or RP1, which then supplies the five high-order bits of the final address; the three low-order bits of the complete address are provided by the original instruction.

Figure 2-14 shows an example of 8-bit working register addressing: the four high-order bits of the instruction address (1100B) specify 8-bit working register addressing. Bit 4 ("1") selects RP1 and the five high-order bits in RP1 (10101B) become the five high-order bits of the register address. The three low-order bits of the register address (011) are provided by the three low-order bits of the 8-bit instruction address. The five address bits from RP1 and the three address bits from the instruction are concatenated to form the complete register address, 0ABH (10101011B).

Figure 2-16. 8-Bit Working Register Addressing
Figure 2-17. 8-Bit Working Register Addressing Example
SYSTEM AND USER STACKS

S3-series microcontrollers can be programmed to use the system stack for subroutine calls, returns and interrupts and to store data. The PUSH and POP instructions are used to control system stack operations. The S3C8639/C863A architecture supports stack operations in the internal register file.

Stack Operations

Return addresses for procedure calls and interrupts and data are stored on the stack. The contents of the PC are saved to stack by a CALL instruction and restored by the RET instruction. When an interrupt occurs, the contents of the PC and the FLAGS register are pushed to the stack. The IRET instruction then pops these values back to their original locations. The stack address is always decremented before a push operation and incremented after a pop operation. The stack pointer (SP) always points to the stack frame stored on the top of the stack, as shown in Figure 2-15.

User-Defined Stacks

You can freely define stacks in the internal register file as data storage locations. The instructions PUSHUI, PUSHUD, POPUI, and POPUD support user-defined stack operations.

Stack Pointers (SPL, SPH)

Register locations D8H and D9H contain the 16-bit stack pointer (SP) that is used for system stack operations. The most significant byte of the SP address, SP15–SP8, is stored in the SPH register (D8H) and the least significant byte, SP7–SP0, is stored in the SPL register (D9H). After a reset, the SP value is undetermined.

Because only internal memory space is implemented in S3C8639/C863A, the SPL must be initialized to an 8-bit value in the range 00H–FFH. The SPH register is not needed here and can be used as a general-purpose register, if necessary.

When the SPL register contains the only stack pointer value (that is, when it points to a system stack in the register file), you can use the SPH register as a general-purpose data register. However, if an overflow or underflow condition occurs as the result of incrementing or decrementing the stack address in the SPL register during normal stack operations, the value in the SPL register will overflow (or underflow) to the SPH register, overwriting any data that is currently stored there. To avoid overwriting data in the SPH register, you can initialize the SPL value to "FFH" rather than "00H".
PROGRAMMING TIP — Standard Stack Operations Using PUSH and POP

The following example shows you how to perform stack operations in the internal register file using PUSH and POP instructions:

LD SPL,#0FFH ; SPL ← FFH
 ; (Normally, the SPL is set to 0FFH by the initialization routine)

PUSH PP ; Stack address 0FEH ← PP
PUSH RP0 ; Stack address 0FDH ← RP0
PUSH RP1 ; Stack address 0FCH ← RP1
PUSH R3 ; Stack address 0FBH ← R3

POP R3 ; R3 ← Stack address 0FBH
POP RP1 ; RP1 ← Stack address 0FCH
POP RP0 ; RP0 ← Stack address 0FDH
POP PP ; PP ← Stack address 0FEH
OVERVIEW

The program counter is used to fetch instructions that are stored in program memory for execution. Instructions indicate the operation to be performed and the data to be operated on. Addressing mode is used to determine the location of the data operand. The operands specified in SAM8 instructions may be condition codes, immediate data, or a location in the register file, program memory, or data memory.

The SAM8 instruction set supports seven explicit addressing modes. Not all of these addressing modes are available for each instruction:

- Register (R)
- Indirect Register (IR)
- Indexed (X)
- Direct Address (DA)
- Indirect Address (IA)
- Relative Address (RA)
- Immediate (IM)
REGISTER ADDRESSING MODE (R)

In Register addressing mode, the operand is the content of a specified register or register pair (see Figure 3-1). Working register addressing differs from Register addressing as it uses a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space (see Figure 3-2).

Sample Instruction:

DEC CNTR ; Where CNTR is the label of an 8-bit register address

Sample Instruction:

ADD R1, R2 ; Where R1 and R2 are registers in the working register area currently selected
INDIRECT REGISTER ADDRESSING MODE (IR)

In Indirect Register (IR) addressing mode, the content of the specified register or register pair is the address of the operand. Depending on the instruction used, the actual address may point to a register in the register file, to program memory (ROM), or to an external memory space, if implemented (see Figures 3-3 through 3-6).

You can use any 8-bit register to indirectly address another register. Any 16-bit register pair can be used to indirectly address another memory location. Remember, however, that locations C0H–FFH in set 1 cannot be accessed using Indirect Register addressing mode.

Sample Instruction:

```plaintext
RL @SHIFT ; Where SHIFT is the label of an 8-bit register address
```

Figure 3-3. Indirect Register Addressing to Register File
INDIRECT REGISTER ADDRESSING MODE (Continued)

Sample Instructions:

- CALL @RR2
- JP @RR2

Figure 3-4. Indirect Register Addressing to Program Memory
INDIRECT REGISTER ADDRESSING MODE (Continued)

Sample Instruction:
OR R3,@R6

Figure 3-5. Indirect Working Register Addressing to Register File
INDIRECT REGISTER ADDRESSING MODE (Concluded)

Sample Instructions:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD R5, @RR6</td>
<td>Program memory access</td>
</tr>
<tr>
<td>LDE R3, @RR14</td>
<td>External data memory access</td>
</tr>
<tr>
<td>LDE @RR4, R8</td>
<td>External data memory access</td>
</tr>
</tbody>
</table>

Figure 3-6. Indirect Working Register Addressing to Program or Data Memory
INDEXED ADDRESSING MODE (X)

Indexed (X) addressing mode adds an offset value to a base address during instruction execution in order to calculate the effective operand address (see Figure 3-7). You can use Indexed addressing mode to access locations in the internal register file or in external memory (if implemented). You cannot, however, access locations C0H–FFH in set 1 using Indexed addressing.

In short offset Indexed addressing mode, the 8-bit displacement is treated as a signed integer in the range from –128 to +127. This applies to external memory accesses only (see Figure 3-8).

For register file addressing, an 8-bit base address provided by the instruction is added to an 8-bit offset contained in a working register. For external memory access, the base address is stored in the working register pair designated in the instruction. The 8-bit or 16-bit offset given in the instruction is then added to the base address (see Figure 3-9).

The only instruction that supports Indexed addressing mode for the internal register file is the Load instruction (LD). The LDC and LDE instructions support Indexed addressing mode for internal program memory and for external data memory (if implemented).

---

Sample Instruction:

```
LD R0,#BASE[R1] ; Where BASE is an 8-bit immediate value
```

Figure 3-7. Indexed Addressing to Register File
INDEXED ADDRESSING MODE (Continued)

Sample Instructions:

LDC R4, #04H[RR2] : The values in the program address (RR2 + 04H) are loaded into register R4.
LDE R4,#04H[RR2] : Identical operation to LDC example, except that external program memory is accessed.

Figure 3-8. Indexed Addressing to Program or Data Memory with Short Offset
INDEXED ADDRESSING MODE (Concluded)

Sample Instructions:

LDC R4,#1000H[RR2] : The values in the program address (RR2 + 1000H) are loaded into register R4.
LDE R4,#1000H[RR2] : Identical operation to LDC example, except that external program memory is accessed.

Figure 3-9. Indexed Addressing to Program or Data Memory
DIRECT ADDRESS MODE (DA)

In Direct Address (DA) mode, the instruction provides the operand's 16-bit memory address. Jump (JP) and Call (CALL) instructions use this addressing mode to specify the 16-bit destination address that is loaded into the PC whenever a JP or CALL instruction is executed.

The LDC and LDE instructions can use Direct Address mode to specify the source or destination address for Load operations to program memory (LDC) or to external data memory (LDE), if implemented.

**Figure 3-10. Direct Addressing for Load Instructions**
DIRECT ADDRESS MODE (Continued)

Sample Instructions:

- **JP C, JOB1** ; Where JOB1 is a 16-bit immediate address
- **CALL DISPLAY** ; Where DISPLAY is a 16-bit immediate address

Figure 3-11. Direct Addressing for Call and Jump Instructions
INDIRECT ADDRESS MODE (IA)

In Indirect Address (IA) mode, the instruction specifies an address located in the lowest 256 bytes of the program memory. The selected pair of memory locations contains the actual address of the next instruction to be executed. Only the CALL instruction can use Indirect Address mode.

Because Indirect Address mode assumes that the operand is located in the lowest 256 bytes of program memory, only an 8-bit address is supplied in the instruction. The upper bytes of the destination address are assumed to be all zeros.

Sample Instruction:
CALL #40H ; The 16-bit value in program memory addresses 40H and 41H is the subroutine start address.

Figure 3-12. Indirect Addressing
RELATIVE ADDRESS MODE (RA)

In Relative Address (RA) mode, a two’s-complement signed displacement between –128 and +127 is specified in the instruction. The displacement value is then added to the current PC value. The result is the address of the next instruction to be executed. Before this addition occurs, the PC contains the address of the next instruction immediately following the current instruction.

Several program control instructions use the Relative Address mode to perform conditional jumps. The instructions that support RA addressing are BTJRF, BTJRT, DJNZ, CPIJE, CPIJNE, and JR.

![Figure 3-13. Relative Addressing](image-url)

Sample Instruction:

```
JR ULT,$+OFFSET ; Where OFFSET is a value in the range +127 to -128
```
IMMEDIATE MODE (IM)

In Immediate (IM) mode, the operand value used in the instruction is the value supplied in the operand field itself. The operand may be one byte or one word in length, depending on the instruction used. Immediate addressing mode is useful for loading constant values into registers.

<table>
<thead>
<tr>
<th>Program Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERAND</td>
</tr>
<tr>
<td>OPCODE</td>
</tr>
</tbody>
</table>

(TheOperand value is in the instruction)

Sample Instruction:

LD R0,#0AAH

Figure 3-14. Immediate Addressing
OVERVIEW

In this chapter, detailed descriptions of the S3C8639/C863A/C8647 control registers are presented in an easy-to-read format. You can use this chapter as a quick-reference source when writing application programs.

The locations and read/write characteristics of all mapped registers in the S3C8639/C863A/C8647 register files are presented in Tables 4-1, 4-2, and 4-3. The hardware reset values for these registers are described in Chapter 8, "RESET and Power-Down."

Figure 4-1 illustrates the important features of the standard register description format.

Control register descriptions are arranged in alphabetical order according to register mnemonic. More detailed information about control registers is presented in the context of the specific peripheral hardware descriptions in Part II of this manual.
### Table 4-1. Set 1 Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Mnemonic</th>
<th>Decimal</th>
<th>Hex</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer M0 counter register</td>
<td>TM0CNT</td>
<td>208</td>
<td>D0H</td>
<td>R (note)</td>
</tr>
<tr>
<td>Timer M0 data register</td>
<td>TM0DATA</td>
<td>209</td>
<td>D1H</td>
<td>R (note)</td>
</tr>
<tr>
<td>Timer M0 control register</td>
<td>TM0CON</td>
<td>210</td>
<td>D2H</td>
<td>R/W</td>
</tr>
<tr>
<td>Basic timer control register</td>
<td>BTCON</td>
<td>211</td>
<td>D3H</td>
<td>R/W</td>
</tr>
<tr>
<td>Clock control register</td>
<td>CLKCON</td>
<td>212</td>
<td>D4H</td>
<td>R/W</td>
</tr>
<tr>
<td>System flags register</td>
<td>FLAGS</td>
<td>213</td>
<td>D5H</td>
<td>R/W</td>
</tr>
<tr>
<td>Register pointer 0</td>
<td>RP0</td>
<td>214</td>
<td>D6H</td>
<td>R/W</td>
</tr>
<tr>
<td>Register pointer 1</td>
<td>RP1</td>
<td>215</td>
<td>D7H</td>
<td>R/W</td>
</tr>
<tr>
<td>Stack pointer (high byte)</td>
<td>SPH</td>
<td>216</td>
<td>D8H</td>
<td>R/W</td>
</tr>
<tr>
<td>Stack pointer (low byte)</td>
<td>SPL</td>
<td>217</td>
<td>D9H</td>
<td>R/W</td>
</tr>
<tr>
<td>Instruction pointer (high byte)</td>
<td>IPH</td>
<td>218</td>
<td>DAH</td>
<td>R/W</td>
</tr>
<tr>
<td>Instruction pointer (low byte)</td>
<td>IPL</td>
<td>219</td>
<td>DBH</td>
<td>R/W</td>
</tr>
<tr>
<td>Interrupt request register</td>
<td>IRQ</td>
<td>220</td>
<td>DCH</td>
<td>R (note)</td>
</tr>
<tr>
<td>Interrupt mask register</td>
<td>IMR</td>
<td>221</td>
<td>DDH</td>
<td>R/W</td>
</tr>
<tr>
<td>System mode register</td>
<td>SYM</td>
<td>222</td>
<td>DEH</td>
<td>R/W</td>
</tr>
<tr>
<td>Page pointer register</td>
<td>PP</td>
<td>223</td>
<td>DFH</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**NOTE:** You cannot use a read-only register (TM0CNT, TM0DATA, IRQ) as a destination field for the instructions OR, AND, LD, or LDB.
### Table 4-2. Set 1, Bank 0 Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Mnemonic</th>
<th>Decimal</th>
<th>Hex</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 0 data register</td>
<td>P0</td>
<td>224</td>
<td>E0H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 1 data register</td>
<td>P1</td>
<td>225</td>
<td>E1H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 2 data register</td>
<td>P2</td>
<td>226</td>
<td>E2H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 3 data register</td>
<td>P3</td>
<td>227</td>
<td>E3H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 0 control register (high byte)</td>
<td>P0CONH</td>
<td>228</td>
<td>E4H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 0 control register (low byte)</td>
<td>P0CONL</td>
<td>229</td>
<td>E5H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 1 control register (high byte)</td>
<td>P1CON</td>
<td>230</td>
<td>E6H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 1 control register (low byte)</td>
<td>P1CON</td>
<td>230</td>
<td>E6H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 2 control register (high byte)</td>
<td>P2CONH</td>
<td>231</td>
<td>E7H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 2 control register (low byte)</td>
<td>P2CONL</td>
<td>232</td>
<td>E8H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 3 control register (high byte)</td>
<td>P3CONH</td>
<td>233</td>
<td>E9H</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 3 control register (low byte)</td>
<td>P3CONL</td>
<td>234</td>
<td>EAH</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 0 external interrupt control register</td>
<td>P0INT</td>
<td>235</td>
<td>EBH</td>
<td>R/W</td>
</tr>
<tr>
<td>Watchdog time control register</td>
<td>WDTCON</td>
<td>236</td>
<td>ECH</td>
<td>R/W</td>
</tr>
<tr>
<td>Sync control register 0</td>
<td>SYNCON0</td>
<td>237</td>
<td>EDH</td>
<td>R/W</td>
</tr>
<tr>
<td>Sync control register 1</td>
<td>SYNCON1</td>
<td>238</td>
<td>EEH</td>
<td>R/W</td>
</tr>
<tr>
<td>Sync control register 2</td>
<td>SYNCON2</td>
<td>239</td>
<td>EFH</td>
<td>R/W</td>
</tr>
<tr>
<td>Sync port read data register</td>
<td>SYNCRD</td>
<td>240</td>
<td>F0H</td>
<td></td>
</tr>
<tr>
<td>Timer M1 counter register (high byte)</td>
<td>TM1CNTH</td>
<td>241</td>
<td>F1H</td>
<td></td>
</tr>
<tr>
<td>Timer M1 counter register (low byte)</td>
<td>TM1CNTL</td>
<td>242</td>
<td>F2H</td>
<td></td>
</tr>
<tr>
<td>Timer M1 data register (high byte)</td>
<td>TM1DATAH</td>
<td>243</td>
<td>F3H</td>
<td></td>
</tr>
<tr>
<td>Timer M1 data register (low byte)</td>
<td>TM1DATAL</td>
<td>244</td>
<td>F4H</td>
<td></td>
</tr>
<tr>
<td>Timer M1 control register</td>
<td>TM1CON</td>
<td>245</td>
<td>F5H</td>
<td>R/W</td>
</tr>
<tr>
<td>Timer M2 control register</td>
<td>TM2CON</td>
<td>246</td>
<td>F6H</td>
<td>R/W</td>
</tr>
<tr>
<td>A/D converter control register</td>
<td>ADCON</td>
<td>247</td>
<td>F7H</td>
<td>R/W</td>
</tr>
<tr>
<td>A/D converter data register</td>
<td>ADDATA</td>
<td>248</td>
<td>F8H</td>
<td></td>
</tr>
<tr>
<td>Pseudo Hsync generation register</td>
<td>PHGEN</td>
<td>249</td>
<td>F9H</td>
<td>R/W</td>
</tr>
<tr>
<td>Pseudo Vsync generation register</td>
<td>PVGEN</td>
<td>250</td>
<td>FAH</td>
<td>R/W</td>
</tr>
<tr>
<td>Stop control register</td>
<td>STOPCON</td>
<td>251</td>
<td>FBH</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**NOTES:**

1. You cannot use a read-only register (SYNCRD, TM1CNTH, TM1TNCL, TM1DATAH, TM1DATAL, ADDATA, BTCNT) as a destination field for the instructions OR, AND, LD, or LDB.
2. Not used for the S3C8647.
<table>
<thead>
<tr>
<th>Register Name</th>
<th>Mnemonic</th>
<th>Decimal</th>
<th>Hex</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM 0 data register</td>
<td>PWM0</td>
<td>224</td>
<td>E0H</td>
<td>R/W</td>
</tr>
<tr>
<td>PWM 1 data register</td>
<td>PWM1</td>
<td>225</td>
<td>E1H</td>
<td>R/W</td>
</tr>
<tr>
<td>PWM 2 data register</td>
<td>PWM2</td>
<td>226</td>
<td>E2H</td>
<td>R/W</td>
</tr>
<tr>
<td>PWM 3 data register</td>
<td>PWM3</td>
<td>227</td>
<td>E3H</td>
<td>R/W</td>
</tr>
<tr>
<td>PWM 4 data register</td>
<td>PWM4</td>
<td>228</td>
<td>E4H</td>
<td>R/W</td>
</tr>
<tr>
<td>PWM 5 data register</td>
<td>PWM5</td>
<td>229</td>
<td>E5H</td>
<td>R/W</td>
</tr>
<tr>
<td>PWM 6 data register (2)</td>
<td>PWM6</td>
<td>230</td>
<td>E6H</td>
<td>R/W</td>
</tr>
<tr>
<td>PWM control register</td>
<td>PWMCON</td>
<td>231</td>
<td>E7H</td>
<td>R/W</td>
</tr>
<tr>
<td>PWM counter register</td>
<td>PWMCNT</td>
<td>232</td>
<td>E8H</td>
<td>R (1)</td>
</tr>
<tr>
<td>DDC control register</td>
<td>DCON</td>
<td>233</td>
<td>E9H</td>
<td>R/W</td>
</tr>
<tr>
<td>DDC address register 0</td>
<td>DAR0</td>
<td>234</td>
<td>EAH</td>
<td>R/W</td>
</tr>
<tr>
<td>DDC clock control register</td>
<td>DCCR</td>
<td>235</td>
<td>EBH</td>
<td>R/W</td>
</tr>
<tr>
<td>DDC control/status register 0</td>
<td>DCSR0</td>
<td>236</td>
<td>ECH</td>
<td>R/W</td>
</tr>
<tr>
<td>DDC control/status register 1</td>
<td>DCSR1</td>
<td>237</td>
<td>EDH</td>
<td>R/W</td>
</tr>
<tr>
<td>DDC address register 1</td>
<td>DAR1</td>
<td>238</td>
<td>EEH</td>
<td>R/W</td>
</tr>
<tr>
<td>Transmit prebuffer data register</td>
<td>TBDR</td>
<td>239</td>
<td>EFH</td>
<td>R/W</td>
</tr>
<tr>
<td>Receive prebuffer data register</td>
<td>RBDR</td>
<td>240</td>
<td>F0H</td>
<td>R (1)</td>
</tr>
<tr>
<td>DDC data shift register</td>
<td>DDSR</td>
<td>241</td>
<td>F1H</td>
<td>R/W</td>
</tr>
<tr>
<td>Slave IIC-Bus control/status register (2)</td>
<td>SICSR</td>
<td>243</td>
<td>F2H</td>
<td>R/W</td>
</tr>
<tr>
<td>Slave IIC-Bus address register (2)</td>
<td>SIAR</td>
<td>242</td>
<td>F3H</td>
<td>R/W</td>
</tr>
<tr>
<td>Slave IIC-Bus data shift register (2)</td>
<td>SIDS R</td>
<td>244</td>
<td>F4H</td>
<td>R/W</td>
</tr>
</tbody>
</table>

Locations F5H–FFH are not mapped

NOTES:
1. You cannot use a read-only register (PWMCNT, RBDR) as a destination field for the instructions OR, AND, LD, or LDB.
2. Not used for the S3C8647.
Figure 4-1. Register Description Format
### ADCON — A/D Converter Control Register

**F7H**  Set 1, Bank 0

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>–</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **.7**
  - Not used for the S3C8639/C863A/C8647

- **.6 and .4**
  - Analog Input Pin Selection Bits
  - 0 0 0 ADC0 (Port 3.0)
  - 0 0 1 ADC1 (Port 3.1)
  - 0 1 0 ADC2 (Port 3.2)
  - 0 1 1 ADC3 (Port 3.3)
  - Others Not used

- **.3**
  - End-of Conversion (EOC) Flag (read-only)
  - 0 Conversion not complete
  - 1 Conversion is complete

- **.2 and .1**
  - Clock Source Selection Bits
  - 0 0 \( f_{\text{OSC}}/16 \)
  - 0 1 \( f_{\text{OSC}}/8 \)
  - 1 0 \( f_{\text{OSC}}/4 \)
  - 1 1 \( f_{\text{OSC}} \)

- **.0**
  - Start or Enable Bit
  - 0 Disable operation
  - 1 Start operation
**BTCON — Basic Timer Control Register**

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7—.4**

**Watchdog Timer Function Disable Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0</td>
<td>Disable watchdog timer function</td>
</tr>
<tr>
<td>Others</td>
<td>Enable watchdog timer function</td>
</tr>
</tbody>
</table>

**.3 and .2**

**Basic Timer Input Clock Selection Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>(f_{\text{OSC}}/4096)</td>
</tr>
<tr>
<td>0 1</td>
<td>(f_{\text{OSC}}/1024)</td>
</tr>
<tr>
<td>1 0</td>
<td>(f_{\text{OSC}}/128)</td>
</tr>
<tr>
<td>1 1</td>
<td>Invalid setting; not used for the S3C8639/C863A/C8647</td>
</tr>
</tbody>
</table>

**.1**

**Basic Timer Counter Clear Bit (1)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No effect</td>
</tr>
<tr>
<td>1</td>
<td>Clear the basic timer counter value</td>
</tr>
</tbody>
</table>

**.0**

**Clock Frequency Divider Clear Bit for Basic Timer and Timer M0 (2)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No effect</td>
</tr>
<tr>
<td>1</td>
<td>Clear basic timer and timer M0 frequency dividers</td>
</tr>
</tbody>
</table>

**NOTES:**

1. When you write a “1” to BTCON.1, the basic timer counter value is cleared to "00H". Immediately after the write operation, the BTCON.1 value is automatically cleared to “0”.
2. When you write a “1” to BTCON.0, the corresponding frequency divider is cleared to "00H". Immediately after the write operation, the BTCON.0 value is automatically cleared to "0".
### CLKCON — System Clock Control Register

**D4H**  
Set 1

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>( .7 )</th>
<th>( .6 )</th>
<th>( .5 )</th>
<th>( .4 )</th>
<th>( .3 )</th>
<th>( .2 )</th>
<th>( .1 )</th>
<th>( .0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### \( .7 \)  
**Oscillator IRQ Wake-up Function Enable Bit**

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

#### \( .6 \) and \( .5 \)  
**Main Oscillator Stop Control Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>No effect</td>
</tr>
<tr>
<td>0 1</td>
<td>No effect</td>
</tr>
<tr>
<td>1 0</td>
<td>Stop main oscillator</td>
</tr>
<tr>
<td>1 1</td>
<td>No effect</td>
</tr>
</tbody>
</table>

#### \( .4 \) and \( .3 \)  
**CPU Clock (System Clock) Selection Bits**

(1)  

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Divide by 16 ( f_{OSC}/16 )</td>
</tr>
<tr>
<td>0 1</td>
<td>Divide by 8 ( f_{OSC}/8 )</td>
</tr>
<tr>
<td>1 0</td>
<td>Divide by 2 ( f_{OSC}/2 )</td>
</tr>
<tr>
<td>1 1</td>
<td>Non-divided clock ( f_{OSC} )</td>
</tr>
</tbody>
</table>

#### \( .2\)–\( .0 \)  
**Subsystem Clock Selection Bits**

(3)  

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0</td>
<td>Invalid setting for S3C8639/C863A/C8647</td>
</tr>
<tr>
<td>Others</td>
<td>Select main system clock (MCLK)</td>
</tr>
</tbody>
</table>

**NOTES:**

1. After a reset, the slowest clock (divided by 16) is selected as the system clock. To select faster clock speeds, load the appropriate values to CLKCON.3 and CLKCON.4.
2. If the oscillator frequency is higher than 12 MHz, this selection is invalid.
3. These selection bits are required only for systems that have a main clock and a subsystem clock. S3C8639/C863A/C8647 use only the main oscillator clock circuit. For this reason, the setting "101B" is invalid.
### DAR0 — DDC Address Register 0

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .7–.4

**4-Slave Address Bits**

These bits are operate only when receive the slave address. Read enable anytime. Write enable when DCSR0.4 is "0".

#### .3–.0

Not used for the S3C8639/C863A/C8647

### DAR1 — DDC Address Register 1

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>–</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>–</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .7–.1

**7-Slave Address Bits**

These bits are operate only when receive the slave address. Read enable anytime. Write enable when DCSR0.4 is "0".

#### .0

Not used for the S3C8639/C863A/C8647
### DCCR — DDC Clock Control Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bit .7**
Transmit acknowledgement enable mode when this bit is "1".

**Bit .6**
*Tx Clock Selection Bit*

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$f_{\text{osc}}/10$</td>
</tr>
<tr>
<td>1</td>
<td>$f_{\text{osc}}/256$</td>
</tr>
</tbody>
</table>

**Bit .5**
*DDC Module Interrupt Enable Bit*

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable interrupt</td>
</tr>
</tbody>
</table>

**Bit .4**
*DDC Module Interrupt Pending Bit*

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When write &quot;0&quot; to this bit (write &quot;1&quot; has no effect)</td>
</tr>
<tr>
<td>0</td>
<td>When DCSR0.4 is &quot;0&quot;</td>
</tr>
<tr>
<td>1</td>
<td>When slave address match occurred</td>
</tr>
<tr>
<td>1</td>
<td>When arbitration lost (master mode)</td>
</tr>
<tr>
<td>1</td>
<td>When an 1-byte transmit or receive operation is terminated</td>
</tr>
<tr>
<td>1</td>
<td>As soon as the DDC1 mode is enabled after the prebuffer is used</td>
</tr>
</tbody>
</table>

**Bit .3–.0**
*Transmit Clock 4-Bit Prescaler Bits (CCR3–CCR0)*

$$\text{SCL clock} = \frac{IICLK}{(CCR < 3: 0 > +1)}$$

where, $IICLK$ is $f_{\text{osc}}/10$ when DCCR.6 is "0"

$IICLK$ is $f_{\text{osc}}/256$ when DCCR.6 is "1"
**DCON — DDC Control Register**

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>
| Addressing Mode| Register addressing mode only

**.7—.4**
Not used for the S3C8639/C863A/C8647.

**.3**
Tx/Rx Pre-Buffer Data Registers Enable Bit

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 Normal IIC-bus mode (Pre-buffer data registers are not used.)

1 Pre-buffer data registers enable mode. This bit is set by writing "1" or by a reset.

**.2**
DDC Address Match Bit

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 When start or stop or reset

1 When DDC received address matches to DAR0 register

**.1**
DDC1 Tx Mode Enable Bit

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 IIC-bus interface mode (SCL pin is also selected)

1 DDC1 Tx mode (VCLK pin is also selected)

**.0**
SCL Pin Falling Edge Detection Flag (note)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 SCL pin level remains high after a reset (when read)

0 This bit can be cleared by S/W written "0" (when write)

1 Falling edge can be detected at the SCL pin after a reset or after this flag is cleared by software (when read)
After start condition, the clock source of DDC module automatically charges from VCLK (Vsync-I) to SCL0 (DCON.1 is "1" to "0") and slave address match possible.

1 No effect (when write)

**NOTE:** When DDC interrupt is occurred, the SCL line is not pull-down in the DDC1 mode and Tx/Rx pre-buffer data registers enable bit, DCON.3 is "1" (only slave mode).
### DCSR0 — DDC Control/Status Register 0

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>–</td>
<td>R</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register addressing mode only

#### .7–.6 Master/Slave, Tx/Rx Mode Selection Bits

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>Slave receiver mode (Default mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Slave transmitter mode</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Master receiver mode</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Master transmitter mode</td>
</tr>
</tbody>
</table>

#### .5 Bus Busy Bit

<table>
<thead>
<tr>
<th>0</th>
<th>IIC-bus is not busy (when read), stop condition generation (when write)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IIC-bus is busy (when read), start condition generation (when write)</td>
</tr>
</tbody>
</table>

#### .4 DDC Module Enable Bit

<table>
<thead>
<tr>
<th>0</th>
<th>Disable DDC module</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enable DDC module</td>
</tr>
</tbody>
</table>

#### .3 Arbitration Lost Bit

<table>
<thead>
<tr>
<th>0</th>
<th>Bus arbitration status okay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bus arbitration failed during serial I/O</td>
</tr>
</tbody>
</table>

#### .2 DDC Address/Data classification Bit

<table>
<thead>
<tr>
<th>0</th>
<th>When reset or start/stop condition is generated, or when the received data is in the data field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>When the received slave address matches to DAR0, DAR1 register</td>
</tr>
</tbody>
</table>

#### .1 Not used for the S3C8639/C863A/C8647

#### .0 Received Acknowledgement (ACK) Bit

<table>
<thead>
<tr>
<th>0</th>
<th>ACK is received</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACK is not received</td>
</tr>
</tbody>
</table>
## DCSR1 — DDC Control/Status Register 1

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### .7–.3

Not used for the S3C8639/C863A/C8647

### .2

**Stop Condition Detection Bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When it writes &quot;0&quot; to this bit, it is reset or master mode.</td>
</tr>
<tr>
<td>1</td>
<td>When a STOP condition is detected after START and slave address reception</td>
</tr>
</tbody>
</table>

### .1

**Data Buffer Empty Status Bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When the CPU writes the transmitted data into the TBDR register</td>
</tr>
<tr>
<td>1</td>
<td>When the data of the TBDR register is loads to the DDSR register or when a STOP condition is detected in DCSR0.7-.6 (slave transmitter mode) = &quot;01&quot;</td>
</tr>
</tbody>
</table>

### .0

**Data Buffer Full Status Bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When the CPU reads the received data from the RBDR register or STOP condition</td>
</tr>
<tr>
<td>1</td>
<td>When the data or matched address is transferred from the DDSR register to the RBDR register</td>
</tr>
</tbody>
</table>
### DDSR — DDC Data Shift Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **.7–.0**
  - Write enable when DCSR0.4 is "1" and DCON.3 is "0". Read enable anytime.
## EMT — External Memory Timing Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>–</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7**

**External WAIT Input Function Enable Bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable WAIT input function for external device</td>
</tr>
<tr>
<td>1</td>
<td>Enable WAIT input function for external device</td>
</tr>
</tbody>
</table>

**.6**

**Slow Memory Timing Enable Bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable slow memory timing</td>
</tr>
<tr>
<td>1</td>
<td>Enable slow memory timing</td>
</tr>
</tbody>
</table>

**.5 and .4**

**Program Memory Automatic Wait Control Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>No wait (Normal Operation)</td>
</tr>
<tr>
<td>0 1</td>
<td>Wait one cycle</td>
</tr>
<tr>
<td>1 0</td>
<td>Wait two cycles</td>
</tr>
<tr>
<td>1 1</td>
<td>Wait three cycles</td>
</tr>
</tbody>
</table>

**.3 and .2**

**Data Memory Automatic Wait Control Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>No wait (Normal Operation)</td>
</tr>
<tr>
<td>0 1</td>
<td>Wait one cycle</td>
</tr>
<tr>
<td>1 0</td>
<td>Wait two cycles</td>
</tr>
<tr>
<td>1 1</td>
<td>Wait three cycles</td>
</tr>
</tbody>
</table>

**.1**

**Stack Area Selection Bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Select internal register file area</td>
</tr>
<tr>
<td>1</td>
<td>Select external data memory area</td>
</tr>
</tbody>
</table>

**.0**

Not used for the S3C8639/C863A/C8647

**NOTE:** As an external peripheral interface is not implemented in S3C8639/C863A/C8647, EMT register is not used. The program initialization routine should clear the EMT register to "00H" after a reset. Modification of EMT values during the normal operation may cause a system malfunction.
## FLAGS — System Flags Register

### Bit Identifier

<table>
<thead>
<tr>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>.7</td>
<td>.6</td>
<td>.5</td>
<td>.4</td>
<td>.3</td>
<td>.2</td>
<td>.1</td>
<td>.0</td>
</tr>
</tbody>
</table>

### RESET Value

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Read/Write

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addressing Mode</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

### Addressing Mode

Register addressing mode only

#### .7 Carry Flag (C)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Operation does not generate a carry or borrow condition</td>
</tr>
<tr>
<td>1</td>
<td>Operation generates a carry-out or borrow into high-order bit 7</td>
</tr>
</tbody>
</table>

#### .6 Zero Flag (Z)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Operation result is a non-zero value</td>
</tr>
<tr>
<td>1</td>
<td>Operation result is zero</td>
</tr>
</tbody>
</table>

#### .5 Sign Flag (S)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Operation generates a positive number (MSB = &quot;0&quot;)</td>
</tr>
<tr>
<td>1</td>
<td>Operation generates a negative number (MSB = &quot;1&quot;)</td>
</tr>
</tbody>
</table>

#### .4 Overflow Flag (V)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Operation result is $\leq +127$ or $\geq -128$</td>
</tr>
<tr>
<td>1</td>
<td>Operation result is $&gt; +127$ or $&lt; -128$</td>
</tr>
</tbody>
</table>

#### .3 Decimal Adjust Flag (D)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Add operation completed</td>
</tr>
<tr>
<td>1</td>
<td>Subtraction operation completed</td>
</tr>
</tbody>
</table>

#### .2 Half-Carry Flag (H)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No carry-out of bit 3 or no borrow into bit 3 by addition or subtraction</td>
</tr>
<tr>
<td>1</td>
<td>Addition generated carry-out of bit 3 or subtraction generated borrow into bit 3</td>
</tr>
</tbody>
</table>

#### .1 Fast Interrupt Status Flag (FIS)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cleared automatically during an interrupt return (IRET)</td>
</tr>
<tr>
<td>1</td>
<td>Automatically set to logic one during a fast interrupt service routine</td>
</tr>
</tbody>
</table>

#### .0 Bank Address Selection Flag (BA)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bank 0 is selected (by executing the instruction SB0)</td>
</tr>
<tr>
<td>1</td>
<td>Bank 1 is selected (by executing the instruction SB1)</td>
</tr>
</tbody>
</table>
### IMR — Interrupt Mask Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7**

**Interrupt Level 7 (IRQ7) Enable Bit; Slave Only IIC-Bus Interrupt (Only S3C863X)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable IRQ7 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable IRQ7 interrupt</td>
</tr>
</tbody>
</table>

**.6**

**Interrupt Level 6 (IRQ6) Enable Bit; P0.2 External Interrupt (INT2)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable IRQ6 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable IRQ6 interrupt</td>
</tr>
</tbody>
</table>

**.5**

**Interrupt Level 5 (IRQ5) Enable Bit; P0.1 External Interrupt (INT1)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable IRQ5 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable IRQ5 interrupt</td>
</tr>
</tbody>
</table>

**.4**

**Interrupt Level 4 (IRQ4) Enable Bit; P0.0 External Interrupt (INT0)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable IRQ4 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable IRQ4 interrupt</td>
</tr>
</tbody>
</table>

**.3**

**Interrupt Level 3 (IRQ3) Enable Bit; DDC (Multi-Master IIC-Bus) Interrupt**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable IRQ3 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable IRQ3 interrupt</td>
</tr>
</tbody>
</table>

**.2**

**Interrupt Level 2 (IRQ2) Enable Bit; Timer M1 Capture/Overflow Interrupt**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable IRQ2 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable IRQ2 interrupt</td>
</tr>
</tbody>
</table>

**.1**

**Interrupt Level 1 (IRQ1) Enable Bit; Timer M2 Interval Interrupt**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable IRQ1 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable IRQ1 interrupt</td>
</tr>
</tbody>
</table>

**.0**

**Interrupt Level 0 (IRQ0) Enable Bit; Timer M0 Overflow/Capture Interrupt**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable IRQ0 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable IRQ0 interrupt</td>
</tr>
</tbody>
</table>
### IPH — Instruction Pointer (High Byte)

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Instruction Pointer Address (High Byte)**

  The high-byte instruction pointer value is the upper eight bits of the 16-bit instruction pointer address (IP15–IP8). The lower byte of the IP address is located in the IPL register (DBH).

### IPL — Instruction Pointer (Low Byte)

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Instruction Pointer Address (Low Byte)**

  The low-byte instruction pointer value is the lower eight bits of the 16-bit instruction pointer address (IP7–IP0). The upper byte of the IP address is located in the IPH register (DAH).
# IPR — Interrupt Priority Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## .7, .4 and .1

### Priority Control Bits for Interrupt Groups A, B and C

<table>
<thead>
<tr>
<th></th>
<th>.7</th>
<th>.4</th>
<th>.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>B &gt; C &gt; A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>A &gt; B &gt; C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>B &gt; A &gt; C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>C &gt; A &gt; B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>C &gt; B &gt; A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>A &gt; C &gt; B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>Not used</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## .6

### Interrupt Sub-group C Priority Control Bit

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IRQ6 &gt; IRQ7</td>
</tr>
<tr>
<td>1</td>
<td>IRQ7 &gt; IRQ6</td>
</tr>
</tbody>
</table>

## .5

### Interrupt Group C Priority Control Bit

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IRQ5 &gt; (IRQ6, IRQ7)</td>
</tr>
<tr>
<td>1</td>
<td>(IRQ6, IRQ7) &gt; IRQ5</td>
</tr>
</tbody>
</table>

## .3

### Interrupt Sub-group B Priority Control Bit

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IRQ3 &gt; IRQ4</td>
</tr>
<tr>
<td>1</td>
<td>IRQ4 &gt; IRQ3</td>
</tr>
</tbody>
</table>

## .2

### Interrupt Group B Priority Control Bit

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IRQ2 &gt; (IRQ3, IRQ4)</td>
</tr>
<tr>
<td>1</td>
<td>(IRQ3, IRQ4) &gt; IRQ2</td>
</tr>
</tbody>
</table>

## .0

### Interrupt Group A Priority Control Bit

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IRQ0 &gt; IRQ1</td>
</tr>
<tr>
<td>1</td>
<td>IRQ1 &gt; IRQ0</td>
</tr>
</tbody>
</table>

---

**NOTE:** Interrupt group A is IRQ0 and IRQ1. Interrupt group B is IRQ2, IRQ3, and IRQ4. Interrupt group C is IRQ5, IRQ6 and IRQ7.
## IRQ — Interrupt Request Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7 Level 7 (IRQ7) Request Pending Bit; Slave Only IIC-Bus Interrupt (Only S3C863X)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No IRQ7 interrupt pending</td>
</tr>
<tr>
<td>1</td>
<td>IRQ7 interrupt is pending</td>
</tr>
</tbody>
</table>

**.6 Level 6 (IRQ6) Request Pending Bit; P0.2 External Interrupt (INT2)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No IRQ6 interrupt pending</td>
</tr>
<tr>
<td>1</td>
<td>IRQ6 interrupt is pending</td>
</tr>
</tbody>
</table>

**.5 Level 5 (IRQ5) Request Pending Bit; P0.1 External Interrupt (INT1)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No IRQ5 interrupt pending</td>
</tr>
<tr>
<td>1</td>
<td>IRQ5 interrupt is pending</td>
</tr>
</tbody>
</table>

**.4 Level 4 (IRQ4) Request Pending Bit; P0.0 External Interrupt (INT0)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No IRQ4 interrupt pending</td>
</tr>
<tr>
<td>1</td>
<td>IRQ4 interrupt is pending</td>
</tr>
</tbody>
</table>

**.3 Level 3 (IRQ3) Request Pending Bit; DDC (Multi-Master IIC-Bus) Interrupt**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No IRQ3 interrupt pending</td>
</tr>
<tr>
<td>1</td>
<td>IRQ3 interrupt is pending</td>
</tr>
</tbody>
</table>

**.2 Level 2 (IRQ2) Request Pending Bit; Timer M1 Capture/Overflow Interrupt**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No IRQ2 interrupt pending</td>
</tr>
<tr>
<td>1</td>
<td>IRQ2 interrupt is pending</td>
</tr>
</tbody>
</table>

**.1 Level 1 (IRQ1) Request Pending Bit; Timer M2 Interval Interrupt**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No IRQ1 interrupt pending</td>
</tr>
<tr>
<td>1</td>
<td>IRQ1 interrupt is pending</td>
</tr>
</tbody>
</table>

**.0 Level 0 (IRQ0) Request Pending Bit; Timer M0 Overflow/Capture Interrupt**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No IRQ0 interrupt pending</td>
</tr>
<tr>
<td>1</td>
<td>IRQ0 interrupt is pending</td>
</tr>
</tbody>
</table>
**P0CONH — Port 0 Control Register (High Byte)**

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **.7 and .6**
  - **P0.7 Mode Selection Bits (Not Used for S3C8647)**
    - 0: x Input mode
    - 1: x Push-pull output mode

- **.5 and .4**
  - **P0.6 Mode Selection Bits (Not Used for S3C8647)**
    - 0: x Input mode
    - 1: x Push-pull output mode

- **.3 and .2**
  - **P0.5 Mode Selection Bits (Not Used for S3C8647)**
    - 0: x Input mode
    - 1: x Push-pull output mode

- **.1 and .0**
  - **P0.4/TM0CAP Mode Selection Bits**
    - 0: 0 Input mode
    - 0: 1 TM0CAP input mode
    - 1: x Push-pull output mode
### P0CONL — Port 0 Control Register (Low Byte)

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register addressing mode only

#### P0.3 Mode Selection Bits (Not Used for S3C8647)

<table>
<thead>
<tr>
<th>.7 and .6</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Input mode</td>
</tr>
<tr>
<td>0 1</td>
<td>Input mode</td>
</tr>
<tr>
<td>1 0</td>
<td>Input mode</td>
</tr>
<tr>
<td>1 1</td>
<td>Push-pull output mode</td>
</tr>
</tbody>
</table>

#### P0.2/INT2 Mode Selection Bits

<table>
<thead>
<tr>
<th>.5 and .4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Input mode (P0.2)</td>
</tr>
<tr>
<td>0 1</td>
<td>Input mode, rising edge interrupt detection (INT2)</td>
</tr>
<tr>
<td>1 0</td>
<td>Input mode, falling edge interrupt detection (INT2)</td>
</tr>
<tr>
<td>1 1</td>
<td>Push-pull output mode</td>
</tr>
</tbody>
</table>

#### P0.1/INT1 Mode Selection Bits

<table>
<thead>
<tr>
<th>.3 and .2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Input mode (P0.1)</td>
</tr>
<tr>
<td>0 1</td>
<td>Input mode, rising edge interrupt detection (INT1)</td>
</tr>
<tr>
<td>1 0</td>
<td>Input mode, falling edge interrupt detection (INT1)</td>
</tr>
<tr>
<td>1 1</td>
<td>Push-pull output mode</td>
</tr>
</tbody>
</table>

#### P0.0/INT0 Mode Selection Bits

<table>
<thead>
<tr>
<th>.1 and .0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Input mode (P0.0)</td>
</tr>
<tr>
<td>0 1</td>
<td>Input mode, rising edge interrupt detection (INT0)</td>
</tr>
<tr>
<td>1 0</td>
<td>Input mode, falling edge interrupt detection (INT0)</td>
</tr>
<tr>
<td>1 1</td>
<td>Push-pull output mode</td>
</tr>
<tr>
<td>Bit Identifier</td>
<td>.7</td>
</tr>
<tr>
<td>----------------</td>
<td>----</td>
</tr>
<tr>
<td>RESET Value</td>
<td>–</td>
</tr>
<tr>
<td>Read/Write</td>
<td>–</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>.7 and .3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>POINT — Port 0 External Interrupt Control Register</td>
<td>EBH</td>
</tr>
<tr>
<td>P0.2 External Interrupt (INT2, IRQ6) Pending Flag</td>
<td>(note)</td>
</tr>
<tr>
<td>0</td>
<td>No P0.2 external interrupt pending (when read)</td>
</tr>
<tr>
<td>0</td>
<td>Clear P0.2 interrupt pending condition (when write)</td>
</tr>
<tr>
<td>1</td>
<td>P0.2 external interrupt is pending (when read)</td>
</tr>
<tr>
<td>P0.1 External Interrupt (INT1, IRQ5) Pending Flag</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>No P0.1 external interrupt pending (when read)</td>
</tr>
<tr>
<td>0</td>
<td>Clear P0.1 interrupt pending condition (when write)</td>
</tr>
<tr>
<td>1</td>
<td>P0.1 external interrupt is pending (when read)</td>
</tr>
<tr>
<td>P0.0 External Interrupt (INT0, IRQ4) Pending Flag</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>No P0.0 external interrupt pending (when read)</td>
</tr>
<tr>
<td>0</td>
<td>Clear P0.0 interrupt pending condition (when write)</td>
</tr>
<tr>
<td>1</td>
<td>P0.0 external interrupt is pending (when read)</td>
</tr>
<tr>
<td>P0.2 External Interrupt (INT2, IRQ6) Enable Bit</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable P0.2 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable P0.2 interrupt</td>
</tr>
<tr>
<td>P0.1 External Interrupt (INT1, IRQ5) Enable Bit</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable P0.1 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable P0.1 interrupt</td>
</tr>
<tr>
<td>P0.0 External Interrupt (INT0, IRQ4) Enable Bit</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable P0.0 interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Enable P0.0 interrupt</td>
</tr>
</tbody>
</table>

**NOTE:** Writing a “1” to an interrupt pending flag (P0.2, P0.1, P0.0) has no effect.
## P1CON — Port 1 Control Register (Only S3C863X)

### E6H  Set 1, Bank 0

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>–</td>
<td>–</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Addressing Mode**

- Register addressing mode only

---

### .7 and .6

Not used for the S3C8639/C863A/C8647

---

### .5 and .4

**P1.2 Mode Selection Bits**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Input mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Push-pull output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>N-channel open-drain output mode (5 V load capability)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Not used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### .3 and .2

**P1.1/SCL1 Mode Selection Bits**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Input mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Push-pull output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>N-channel open-drain output mode (5 V load capability)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Multiplexed mode (SCL1 (P1.1))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### .1 and .0

**P1.0/SDA1 Mode Selection Bits**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Input mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Push-pull output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>N-channel open-drain output mode (5 V load capability)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Multiplexed mode (SDA1 (P1.0))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### P2CONH — Port 2 Control Register (High Byte)

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .7 and .6

**P2.7/Csync-I**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x TTL input mode (Csync-I)</td>
</tr>
<tr>
<td>1</td>
<td>x Push-pull output mode</td>
</tr>
</tbody>
</table>

#### .5 and .4

**P2.6/PWM6 Mode Selection Bits (Not Used for S3C8647)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Input mode</td>
</tr>
<tr>
<td>0 1</td>
<td>Push-pull output mode</td>
</tr>
<tr>
<td>1 0</td>
<td>Push-pull PWM output mode</td>
</tr>
<tr>
<td>1 1</td>
<td>N-channel open-drain PWM output mode (5 V load capability)</td>
</tr>
</tbody>
</table>

#### .3 and .2

**P2.5/PWM5 Mode Selection Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Input mode</td>
</tr>
<tr>
<td>0 1</td>
<td>Push-pull output mode</td>
</tr>
<tr>
<td>1 0</td>
<td>Push-pull PWM output mode</td>
</tr>
<tr>
<td>1 1</td>
<td>N-channel open-drain PWM output mode (5 V load capability)</td>
</tr>
</tbody>
</table>

#### .1 and .0

**P2.4/PWM4 Mode Selection Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Input mode</td>
</tr>
<tr>
<td>0 1</td>
<td>Push-pull output mode</td>
</tr>
<tr>
<td>1 0</td>
<td>Push-pull PWM output mode</td>
</tr>
<tr>
<td>1 1</td>
<td>N-channel open-drain PWM output mode (5 V load capability)</td>
</tr>
</tbody>
</table>
## P2CONL — Port 2 Control Register (Low Byte)

**E8H**  
Set 1, Bank 0

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Register addressing mode only</td>
</tr>
</tbody>
</table>

### .7 and .6

**P2.3/PWM3 Mode Selection Bits**

- 0 x Input mode
- 1 0 Push-pull output mode
- 1 1 Push-pull PWM output mode (5 V load capability)

### .5 and .4

**P2.2/PWM2 Mode Selection Bits**

- 0 x Input mode
- 1 0 Push-pull output mode
- 1 1 Push-pull PWM output mode (5 V load capability)

### .3 and .2

**P2.1/PWM1 Mode Selection Bits**

- 0 x Input mode
- 1 0 Push-pull output mode
- 1 1 Push-pull PWM output mode (5 V load capability)

### .1 and .0

**P2.0/PWM0 Mode Selection Bits**

- 0 x Input mode
- 1 0 Push-pull output mode
- 1 1 Push-pull PWM output mode (5 V load capability)
### P3CONH — Port 3 Control Register (High Byte)

**Bit Identifier**

<table>
<thead>
<tr>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
</table>

#### RESET Value

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

#### Read/Write

| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

#### Addressing Mode

Register addressing mode only

#### .7 and .6

**P3.7 Mode Selection Bits**

| 0 | 0 | Input mode |
| 0 | 1 | Input mode with pull-up resistor |
| 1 | 0 | Push-pull output mode |
| 1 | 1 | N-channel open-drain output mode |

#### .5 and .4

**P3.6 Mode Selection Bits**

| 0 | 0 | Input mode |
| 0 | 1 | Input mode with pull-up resistor |
| 1 | 0 | Push-pull output mode |
| 1 | 1 | N-channel open-drain output mode |

#### .3 and .2

**P3.5 Mode Selection Bits**

| 0 | 0 | Input mode |
| 0 | 1 | Input mode with pull-up resistor |
| 1 | 0 | Push-pull output mode |
| 1 | 1 | N-channel open-drain output mode |

#### .1 and .0

**P3.4 Mode Selection Bits**

| 0 | 0 | Input mode |
| 0 | 1 | Input mode with pull-up resistor |
| 1 | 0 | Push-pull output mode |
| 1 | 1 | N-channel open-drain output mode |
### P3CONL — Port 3 Control Register (Low Byte)

**EAH**  
Set 1, Bank 0

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .7 and .6  
P3.3/AD3 Mode Selection Bits

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Input mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Analog Input mode (AD3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Push-pull output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>N-channel open-drain output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .5 and .4  
P3.2/AD2 Mode Selection Bits

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Input mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Analog Input mode (AD2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Push-pull output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>N-channel open-drain output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .3 and .2  
P3.1/AD1 Mode Selection Bits

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Input mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Analog Input mode (AD1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Push-pull output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>N-channel open-drain output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .1 and .0  
P3.0/AD0 Mode Selection Bits

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Input mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Analog Input mode (AD0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Push-pull output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>N-channel open-drain output mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# PHGEN — Pseudo Hsync Generation Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W R/W</td>
<td></td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Write enable when SYNCON2.4 is "0". (General Pseudo H/Vsync generation mode)
Read enable any time
## PP — Page Pointer Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### .7–.4 Destination Register Page Selection Bits

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Destination: page 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Destination: page 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Destination: page 2 (Not used for the S3C8647)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Destination: page 3 (Not used for the S3C8639)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Not used for the S3C8639/C863A/C8647</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### .3–.0 Source Register Page Selection Bits

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Source: page 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Source: page 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Source: page 2 (Not used for the S3C8647)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Source: page 3 (Not used for the S3C8639)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Not used for the S3C8639/C863A/C8647</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Not used for the S3C8639/C863A/C8647

- Not used for the S3C8639/C863A/C8647
PVGEN — Pseudo Vsync Generation Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Write enable SYNCON2.4 is "0". (General Pseudo H/Vsync generation mode) Read enable any time
**PWMCON — PWM Control Register**

**E7H Set 1, Bank 1**

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7 and .6**

2-Bit Prescaler Value for PWM Counter Input Clock

<table>
<thead>
<tr>
<th>.7</th>
<th>.6</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Non-divided input clock</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Input clock divided into two</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Input clock divided into three</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Input clock divided into four</td>
</tr>
</tbody>
</table>

**.5**

PWM Counter Enable Bit

<table>
<thead>
<tr>
<th>.5</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Stop PWM counter operation (No current leakage)</td>
</tr>
<tr>
<td>1</td>
<td>Start (or resume) PWM counter operation</td>
</tr>
</tbody>
</table>

**.4–.0**

Not used for the S3C8639/C863A/C8647
## RBDR — Receive Pre-Buffer Data Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is a read-only register. Read enable anytime. This register will be updated after a data byte is received when the DCSR0.2 is "1" and the DCSR1.0 will be "1". The read operation of this register will clear the DCSR1.0. After the DCSR1.0 is cleared, the register can load the received data again and set the DCSR1.0.
### CONTROL REGISTERS

#### S3C8639/C863A/P863A/C8647/F8647

**RP0 — Register Pointer 0**

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7–.3**

Register Pointer 0 can independently point to one of the 18 8-byte working register areas in the register file. Using the register pointers, RP0 and RP1, you can select two 8-byte register slices at one time as active working register space. After a reset, RP0 points to address C0H in register set 1, selecting the 8-byte working register slice C0H–C7H.

**.2–.0**

Not used for the S3C8639/C863A/C8647

---

**RP1 — Register Pointer 1**

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7–.3**

Register pointer 1 can independently point to one of the 18 8-byte working register areas in the register file. Using the register pointers, RP0 and RP1, you can select two 8-byte register slices at one time as active working register space. After a reset, RP1 points to address C8H in register set 1, selecting the 8-byte working register slice C8H–CFH.

**.2–.0**

Not used for the S3C8639/C863A/C8647
### SIAR — Slave IIC-Bus Address Register (Only S3C863X)

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESET Value</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>Read/Write</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>–</td>
</tr>
<tr>
<td><strong>Addressing Mode</strong></td>
<td>Register addressing only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .7–.1

**7-Bit Slave Address Bits**

These bits are operated only when receive the slave address and general call. Write enable when SICSR.6 is "0", but read enable anytime.

#### .0

Not used for the S3C8639/C863A/C8647
### SICSR — Slave IIC-Bus Control/Status Register (Only S3C863X)

#### Bit Identifier

<table>
<thead>
<tr>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>.7 Acknowledgement Enable Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable ACK generation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enable ACK generation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.6 Slave IIC-Bus Module Enable Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable IIC-Bus module</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enable IIC-Bus module (Enable serial data Tx/Rx)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.5 Slave IIC-Bus Tx/Rx Interrupt Enable Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable interrupt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enable interrupt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4 Slave IIC-Bus Tx/Rx Interrupt Pending Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>No interrupt pending (when read) clear pending condition (when write)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>When SICSR.6 is &quot;0&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>When 1-Byte Tx/Rx is terminated</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>When slave address match occurred</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.3 Slave IIC-Bus Tx/Rx Mode Status Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Slave receive mode (Default mode)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Slave transmitter mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.2 IIC-Bus Busy Status Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>IIC-Bus is not busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>IIC-Bus is busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.1 Slave Address Match Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>When start or stop or reset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>When received slave address matches to SIAR register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.0 Received Acknowledge (ACK) Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>ACK is received</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ACK is not received</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** Bit 2-0 are read only.
SIDSR — Slave IIC-Bus Tx/Rx Data Shift Register (Only S3C863X)  
F4H Set 1, Bank 1

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Slave IIC-Bus Transmit/Receive Data Shift Bus

Write enable when SICS.R.6 is "1", but read enable anytime.
### SPH — Stack Pointer (High Byte)

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Stack Pointer Address (High Byte)**

The high-byte stack pointer value is the upper eight bits of the 16-bit stack pointer address (SP15–SP8). The lower byte of the stack pointer value is located in the register SPL (D9H). The SP value is undefined after a reset.

### SPL — Stack Pointer (Low Byte)

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Stack Pointer Address (Low Byte)**

The low-byte stack pointer value is the lower eight bits of the 16-bit stack pointer address (SP7–SP0). The upper byte of the stack pointer value is located in the register SPH (D8H). The SP value is undefined after a reset.
**STOPCON** — Stop Control Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7–.0 Stop Operation Enable Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 1 0 1</td>
<td>Enable the stop (power saving) function</td>
</tr>
<tr>
<td>Others</td>
<td>Disable the stop function</td>
</tr>
</tbody>
</table>

**NOTES:**

1. If you intend to stop function for power saving, before Stop OP-code, you must set this register value to A5H (10100101B).
2. When STOP mode is released, stop control register (STOPCON) value is cleared automatically.
### SYM — System Mode Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>–</td>
<td>–</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>.7</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tri-State External Interface Control Bit (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Normal operation (disable tri-state operation)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Set external interface lines to high impedance (enable tri-state operation)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>.6 and .5</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not used for the S3C8639/C863A/C8647</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>.4—.2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fast Interrupt Level Selection Bits (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>IRQ0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>IRQ1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>IRQ2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>IRQ3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>IRQ4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>IRQ5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>IRQ6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>IRQ7 (Not used for the S3C8647)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>.1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fast Interrupt Enable Bit (3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable fast interrupt processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enable fast interrupt processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>.0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global Interrupt Enable Bit (4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable global interrupt processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enable global interrupt processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**
1. As external interface is not implemented in S3C8639/C863A/C8647, SYM.7 must always be "0".
2. You can select only one interrupt level at a time for fast interrupt processing.
3. Setting SYM.1 to "1" enables fast interrupt processing for the interrupt level currently selected by SYM.2–SYM.4.
4. After a reset, you must enable global interrupt processing by executing an EI instruction (not by writing a "1" to SYM.0).
SYNCON0 — Sync Processor Control Register 0

Bit Identifier

<table>
<thead>
<tr>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

.7  
Sync Input Selection (SIS) Bit

<table>
<thead>
<tr>
<th>0</th>
<th>Hsync-I input is selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Csync-I input is selected</td>
</tr>
</tbody>
</table>

.6  
Hsync Blanking Enable Bit

| 0  | Disable (Hsync signal by-pass) (When SYNCON0.5 = "0") |
| 1  | Enable Hsync blanking automatically (During the Vsync signal extraction period) (When SYNCON0.5 = "1") |

.5  
Vsync-O Output Selection (VOS) Bit

<table>
<thead>
<tr>
<th>0</th>
<th>Select Vsync-I port input (when separate sync input mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select 5-bit compare output (when composite sync input mode)</td>
</tr>
</tbody>
</table>

.4–.0  
5-Bit Counter Value Bits

5-bit counter increases when a high level is detected, while an overflow does not occur (Stop at "11111"). It decreases when a low level is detected, while an underflow does not occur (Stop at "00000").

When SYNCON0.5 is "1": Sync separation and output (When counter value increases to "11111", output the high through the MUX and when counter value decreases to "00000", output becomes low. Resume the previous status when "11111" > counter value > "00000")
### SYNCON1 — Sync Processor Control Register 1

**Bit Identifier**

<table>
<thead>
<tr>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**RESET Value**

- Read/Write: R/W
- Addressing Mode: Register addressing mode only

#### .7 and .6 Clamp Signal Generator Selection Bits

- 0 0: Inhibit Clamp signal output (Clamp-O)
- 0 1: \((f_{OSC} \times 2)\) clock pulse output (250 ns at 8 MHz \(f_{OSC}\))
- 1 0: \((f_{OSC} \times 4)\) clock pulse output (500 ns at 8 MHz \(f_{OSC}\), 333 ns at 12 MHz \(f_{OSC}\))
- 1 1: \((f_{OSC} \times 8)\) clock pulse output (1 \(\mu\)s at 8 MHz \(f_{OSC}\), 666 ns at 12 MHz \(f_{OSC}\))

#### .5 "Front Porch"/"Back Porch" Mode Selection Bit

- 0: Generate Clamp-O after the rising edge of Hsync ("front porch" mode)
- 1: Generate Clamp-O after the falling edge of Hsync ("back porch" mode)

#### .4 Clamp Signal Output Status Control Bit (COSC)

- 0: Negative polarity
- 1: Positive polarity

#### .3 Vsync-O Status Control Bit

- 0: Do not invert (by-pass)
- 1: Invert output signal

#### .2 Hsync-O Status Control Bit (HOSC)

- 0: Do not invert (by-pass)
- 1: Invert output signal

#### .1 Vsync Polarity Detection Bit (1)

- 0: Negative polarity
- 1: Positive polarity

#### .0 Hsync Polarity Detection Bit (2)

- 0: Negative polarity
- 1: Positive polarity

**NOTES:**

1. To check Hsync/Vsync polarity, it uses 16 clocks of timer M2 \((f_{OSC}/1000)\). If the Vsync polarity is changing, this bit will be updated after a typical delay of 2 ms, at 8 MHz \(f_{OSC}\) (1.33 ms at 12 MHz \(f_{OSC}\)).
2. The SYNCON1.0 may not be accurate when the Hsync-I is composite-sync signal output.
### SYNCON2 — Sync Processor Control Register 2

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .7 Unmixed Hsync Detection Bit (When SYNCON0.5 is “1”)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Mixed Hsync period with Vsync of a composite sync input (This bit still cleared before being read this bit or it is been in mixed Hsync period)</td>
</tr>
<tr>
<td>1</td>
<td>Unmixed Hsync periods</td>
</tr>
</tbody>
</table>

#### .6 Not used for the S3C8639/C863A/C8647 (Only “0”)

#### .5 5-Bit Counter Source Clock (fsync) Input Selection Bit (1)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>f_{OSC}/3 (when f_{CPU} is 12 MHz)</td>
</tr>
<tr>
<td>1</td>
<td>f_{OSC}/2 (when f_{CPU} is 8 MHz)</td>
</tr>
</tbody>
</table>

#### .4 Pseudo Sync Generation Disable Bit (Positive Polarity Only)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Enable Pseudo Hsync/Vsync generation</td>
</tr>
<tr>
<td>1</td>
<td>Normal Sync-processor operation (by-pass)</td>
</tr>
</tbody>
</table>

#### .3 Sync Signal Output Disable Bit

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Enable Sync signal output</td>
</tr>
<tr>
<td>1</td>
<td>Inhibit Sync signal output (Output level is low)</td>
</tr>
</tbody>
</table>

#### .2 SOG (Sync On Green) Detection Bit

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No SOG signal (when read)</td>
</tr>
<tr>
<td>0</td>
<td>Clear SOG detection 6-bit counter (when write)</td>
</tr>
<tr>
<td>1</td>
<td>Csync-I is SOG signal (2)</td>
</tr>
</tbody>
</table>

#### .1 5-Bit up/down Counter Latch Status Changing Detection Bit

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When the latch status is not changed or it writes “0” to this bit</td>
</tr>
<tr>
<td>1</td>
<td>When the latch status changing is detected.</td>
</tr>
</tbody>
</table>

#### .0 V_{DD} Level Selection Bit for TTL Sync-Input Port (Not used for the S3C8647)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When V_{DD} is +5 V</td>
</tr>
<tr>
<td>1</td>
<td>When V_{DD} is +3 V</td>
</tr>
</tbody>
</table>

### NOTES:

1. Countable maximum Hsync pulse width = 7.85 us (when fsync is 4 MHz)
2. To check SOG presence, it uses 64 Csync-I input edge signal.
3. The SYNCON2.1 can be used to check the presence of composite-sync signal input.
### SYNCRD — Sync Processor Port Read Data Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**.7—.4**

Not used for the S3C8639/C863A/C8647

**.3**

Vertical Sync Signal Output Data Bit (Vsync-O)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Low data</td>
</tr>
<tr>
<td>1</td>
<td>High data</td>
</tr>
</tbody>
</table>

**.2**

Horizontal Sync Signal Output Data Bit (Hsync-O)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Low data</td>
</tr>
<tr>
<td>1</td>
<td>High data</td>
</tr>
</tbody>
</table>

**.1**

Vertical Sync Signal Input Data Bit (Vsync-I)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Low data</td>
</tr>
<tr>
<td>1</td>
<td>High data</td>
</tr>
</tbody>
</table>

**.0**

Horizontal Sync Signal Input Data Bit (Hsync-I)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Low data</td>
</tr>
<tr>
<td>1</td>
<td>High data</td>
</tr>
</tbody>
</table>
### TBDR — Transmit Pre-Buffer Data Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Bit Identifier Description
- **.7**–**.0**
  - Write enable when DCSR0.4 is "1", Read enable anytime
  - When DCON.3 (TBDR Enable bit) = "1" and DCSR1.1 = "0", the data written into his register will be automatically downloaded to the DDC Data Shift Register (DDSR) and generate the interrupt request when the module detects the calling address is matched and the bit 0 of the received data is "1" (DCSR0.7-6 = "01") and when the data in the DDSR register has been transmitted with received acknowledge bit, DCSR0.0 = "0". At this interrupt service routine, the CPU must write the next data to the TBDR register to clear DCSR1.1 and for the auto downloading of data to the DDSR register after the data in the DDSR register is transmitted over again with DCSR0.0 = "0".
  - When DCON.3 = "1" and DDSR1.1 = "1", the data stored in this register will not be downloaded to the DDSR register and generated the interrupt request when the module detects the calling address is matched and the bit 0 of the received data is "1". At this interrupt service routine, the CPU must write the current data and rewrite the next data to the TBDR register to clear DCSR1.1.
  - If the master receiver doesn’t acknowledge the transmitted data, DCSR0.0 = "1", the module will release the SDI line for master to generate STOP or Repeated START conditions.
  - If DCON.3 (TBDR Enable bit) is "0", the module will pull-down the SCL line in the IIC-Bus interrupt service routine when the DCSR0.2 is "1". And the module will release the SCL line if the CPU writes a data to the DDSR registers and the interrupt pending bit is cleared.
## TM0CON — Timer m0 Control Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Bit .7
**Timer M0 Input Clock Selection Bit**
- 0: $f_{\text{OSC}}/128$
- 1: $f_{\text{OSC}}/8$

### Bits .6 and .5
**2 Bit Prescaler Bits**
- 0 0: No division
- 0 1: Divided by 2
- 1 0: Divided by 3
- 1 1: Divided by 4

### Bit .4
**Timer M0 Capture Mode Selection Bit**
- 0: Capture on rising mode
- 1: Capture on falling mode

### Bit .3
**Timer M0 Counter Clear Bit (TM0CLR)**
- 0: No effect
- 1: Clear timer M0 counter, TM0CNT (when write)

### Bit .2
**Timer M0 Overflow Interrupt Enable Bit (TM0OVINT)**
- 0: Disable timer M0 overflow interrupt
- 1: Enable timer M0 overflow interrupt

### Bit .1
**Timer M0 Capture Interrupt Enable Bit (TM0INT)**
- 0: Disable timer M0 interrupt
- 1: Enable timer M0 interrupt

### Bit .0
**Timer M0 Capture Input Selection Bit (TM0CAPSEL)**
- 0: TM0CAP input pin selection
- 1: Vsync output path selection from sync-processor

### NOTES:
1. When the captured value is #0FFH, the overflow interrupt does not occurred. If the captured value is changed from #0FFH to #00H, the overflow interrupt occurs. When the captured value is #00H, the overflow interrupt occurs first.
2. When the timer M0 interrupt is disabled, the timer M0 overflow interrupt by $f_{\text{OSC}}$ can happen.
### TM1CON — Timer M1 Control Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capture Signal Source Selection Bit</td>
<td>0</td>
<td>Signal from timer M2 interval time</td>
<td>1</td>
<td>Vsync-O from sync-processor</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vsync-O Capture Edge Selection Bit (When TM1CON.7 = &quot;1&quot;)</td>
<td>0</td>
<td>Capture Vsync-O (from sync-processor) on rising edge</td>
<td>1</td>
<td>Capture Vsync-O (from sync-processor) on falling edge</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M1 Capture Interrupt Enable Bit (TM1INT)</td>
<td>0</td>
<td>Disable timer M1 capture</td>
<td>1</td>
<td>Enable timer M1 capture</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M1 Capture Pending Bit (TM1PND)</td>
<td>0</td>
<td>Interrupt is not pending (when read)</td>
<td>0</td>
<td>Clear this pending bit (when write)</td>
<td>1</td>
<td>Interrupt is pending (when read)</td>
<td>1</td>
<td>No effect (when write)</td>
</tr>
<tr>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M1 Counter Clear Bit (TM1CLR; when write)</td>
<td>0</td>
<td>No effect</td>
<td>1</td>
<td>Clear timer M1 counter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M1 Overflow Interrupt Enable Bit (TM1OVF)</td>
<td>0</td>
<td>Disable timer M1 overflow interrupt</td>
<td>1</td>
<td>Enable timer M1 overflow interrupt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.1--.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M1 Clock Input Selection Bit</td>
<td>0</td>
<td>0</td>
<td>Hsync-I or Csync-I from sync processor</td>
<td>0</td>
<td>1</td>
<td>( f_{OSC}/2 )</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### TM2CON — Timer M2 Control Register

**F6H  Set 1, Bank 0**

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### .7–.3

<table>
<thead>
<tr>
<th>5-bit Prescale Bits (TM2PS4–TM2PS0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0  No division</td>
</tr>
<tr>
<td>0 0 0 0 0 1  1  Divide by 2</td>
</tr>
<tr>
<td>0 0 0 1 0 0  1  Divide by 3</td>
</tr>
<tr>
<td>0 0 1 0 0 0  1  Divide by 32</td>
</tr>
</tbody>
</table>

#### .2

<table>
<thead>
<tr>
<th>Timer M2 Interrupt Enable Bit (TM2INT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  Disable timer M2 interrupt</td>
</tr>
<tr>
<td>1  Enable timer M2 interrupt</td>
</tr>
</tbody>
</table>

#### .1 and .0

| Timer M2 Capture Interval Time Selection Bits (When TM2CON.5 is "1") |
|---------------|----------------|
| 0 0  Timer M2 interval (by pass)                        |
| 0 1  Timer M2 interval × 10                             |
| 1 0  Timer M2 interval × 20                             |
| 1 1  Timer M2 interval × 30                             |

**NOTES:**

1. When the timer M1 capture mode is enabled (TM1CON.5 = "1"), the value of 5/2-bit prescaler is changed only in the timer M1 capture interrupt routine.
2. When the timer M1 capture mode is disabled (TM1CON.5 = "0"), the value of 5-bit prescaler is changed only in the timer M2 interval interrupt routine.
## WDTCN — Watchdog Time Control Register

<table>
<thead>
<tr>
<th>Bit Identifier</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Value</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Addressing Mode</td>
<td>Register addressing mode only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### .7–.4

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.7–.4</td>
<td>Not used for the S3C8639/C863A/C8647</td>
</tr>
</tbody>
</table>

### .3

**Hsync-O Divide Enable Bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Hsync-O = Hsync-I (Non-divide)</td>
</tr>
<tr>
<td>1</td>
<td>Hsync-O = Hsync-I/2</td>
</tr>
</tbody>
</table>

### .2–.0

**Watchdog Time Generation Control Bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>tBTOVF (note)</td>
</tr>
<tr>
<td>0 0 1</td>
<td>tBTOVF/2</td>
</tr>
<tr>
<td>0 1 0</td>
<td>tBTOVF/3</td>
</tr>
<tr>
<td>0 1 1</td>
<td>tBTOVF/4</td>
</tr>
<tr>
<td>1 0 0</td>
<td>tBTOVF/5</td>
</tr>
<tr>
<td>1 0 1</td>
<td>tBTOVF/6</td>
</tr>
<tr>
<td>1 1 0</td>
<td>tBTOVF/7</td>
</tr>
<tr>
<td>1 1 1</td>
<td>tBTOVF/8</td>
</tr>
</tbody>
</table>

**NOTE:** tBTOVF = \((1/f_{OSC}) \times \text{(Divider count of basic timer input clock)}) \times 256
5

INTERRUPT STRUCTURE

OVERVIEW

The SAM8 interrupt structure has three basic components: levels, vectors, and sources. The CPU recognizes eight interrupt levels and supports up to 128 interrupt vectors. When a specific interrupt level has more than one vector address, the vector priorities are established in hardware. Each vector can have one or more sources.

Levels

Interrupt levels are the main unit for interrupt priority assignment and recognition. All peripherals and I/O blocks can issue interrupt requests. In other words, peripheral and I/O operations are interrupt-driven. There are eight interrupt levels: IRQ0–IRQ7, also called level 0–level 7. Each interrupt level directly corresponds to an interrupt request number (IRQn). The total number of interrupt levels used in the interrupt structure varies from device to device.

The interrupt level numbers 0 through 7 do not necessarily indicate the relative priority of the levels. They are just identifiers for the interrupt levels that are recognized by the CPU. The relative priority of different interrupt levels is determined by settings in the interrupt priority register, IPR. Interrupt group and subgroup logic controlled by IPR settings lets you define more complex priority relationships between different levels.

Vectors

Each interrupt level can have one or more interrupt vectors, or it may have no vector address assigned at all. The maximum number of vectors that can be supported for a given level is 128. (The actual number of vectors used for S3C8-series devices will always be much smaller.) If an interrupt level has more than one vector address, the vector priorities are set in hardware. S3C8639/C863A/C8647* have ten (nine)* vectors — one corresponding to each of the ten (nine)* possible interrupt sources.

Sources

A source is any peripheral that generates an interrupt. A source can be an external pin or a counter overflow. Each vector can have several interrupt sources. In the S3C8639/C863A/C8647* interrupt structure, each source has its own vector address.

When a service routine starts, the respective pending bit should be either cleared automatically by hardware or cleared "manually" by program software. The characteristics of the source's pending mechanism determine which method would be used to clear its respective pending bit.
INTERRUPT TYPES

The three components of the SAM8 interrupt structure described before — levels, vectors, and sources — are combined to determine the interrupt structure of an individual device and to make full use of its available interrupt logic. There are three possible combinations of interrupt structure components, called interrupt types 1, 2, and 3. The types differ in the number of vectors and interrupt sources assigned to each level (see Figure 5-1):

Type 1: One level (IRQn) + one vector (V1) + one source (S1)

Type 2: One level (IRQn) + one vector (V1) + multiple sources (S1–Sn)

Type 3: One level (IRQn) + multiple vectors (V1–Vn) + multiple sources (S1–Sn, Sn+1–Sn+m)

In the S3C8639/C863A/C8647 microcontrollers, only interrupt types 1 and 3 are implemented.

![Figure 5-1. S3C8-Series Interrupt Types](image-url)
The S3C8639/C863A/C8647 microcontrollers support ten interrupt sources. Each interrupt source has a corresponding interrupt vector address. All eight interrupt levels are used in the device-specific interrupt structure, which is shown in Figure 5-2.

When multiple interrupt levels are active, the interrupt priority register (IPR) determines the order in which contending interrupts are to be serviced. If multiple interrupts occur within the same interrupt level, the interrupt with the lowest vector address is usually processed first. (The relative priorities of multiple interrupts within a single level are fixed in hardware.)

When the CPU grants an interrupt request, interrupt processing starts: All other interrupts are disabled and the program counter value and status flags are pushed to stack. The starting address of the service routine is fetched from the appropriate vector address (plus the next 8-bit value to concatenate the full 16-bit address) and the service routine is executed.

<table>
<thead>
<tr>
<th>Levels</th>
<th>Vectors</th>
<th>Sources</th>
<th>Reset/Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRQ0</td>
<td>E0H</td>
<td>Timer M0 overflow interrupt</td>
<td>H/W</td>
</tr>
<tr>
<td>IRQ1</td>
<td>E2H</td>
<td>Timer M0 capture interrupt</td>
<td>H/W</td>
</tr>
<tr>
<td>IRQ2</td>
<td>E4H</td>
<td>Timer M2 interval interrupt</td>
<td>H/W</td>
</tr>
<tr>
<td>IRQ3</td>
<td>E6H</td>
<td>Timer M1 overflow interrupt</td>
<td>H/W</td>
</tr>
<tr>
<td>IRQ4</td>
<td>E8H</td>
<td>Timer M1 capture interrupt</td>
<td>S/W</td>
</tr>
<tr>
<td>IRQ5</td>
<td>EAH</td>
<td>DDC (Multi-master IIC-bus) interrupt</td>
<td>S/W</td>
</tr>
<tr>
<td>IRQ6</td>
<td>ECH</td>
<td>P0.0 external interrupt (INT0)</td>
<td>S/W</td>
</tr>
<tr>
<td>IRQ7</td>
<td>EEH</td>
<td>P0.1 external interrupt (INT1)</td>
<td>S/W</td>
</tr>
<tr>
<td></td>
<td>F0H</td>
<td>P0.2 external interrupt (INT2)</td>
<td>S/W</td>
</tr>
<tr>
<td></td>
<td>F2H</td>
<td>Slave only IIC-bus interrupt (note)</td>
<td>S/W</td>
</tr>
</tbody>
</table>

**NOTE:** Not used for the S3C8647.

Figure 5-2. S3C8639/C863A/C8647 Interrupt Structure
INTERRUPT VECTOR ADDRESSES

All interrupt vector addresses for the S3C8639/C863A/C8647 interrupt structure are stored in the vector address area of the ROM, 00H–FFH (see Figure 5-3). You can allocate unused locations in the vector address area as normal program memory. If you do so, please be careful not to overwrite any of the stored vector addresses. (Table 5-1 lists all vector addresses.)

The program reset address in the ROM is 0100H.

![Figure 5-3. ROM Vector Address Area](image-url)
<table>
<thead>
<tr>
<th>Vector Address</th>
<th>Interrupt Source</th>
<th>Request</th>
<th>Reset/Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Interrupt Level</td>
<td>Priority in Level</td>
</tr>
<tr>
<td>Decimal Value</td>
<td>Hex Value</td>
<td>Timer M0 overflow interrupt</td>
<td>IRQ0</td>
</tr>
<tr>
<td>224</td>
<td>E0H</td>
<td>Timer M0 capture interrupt</td>
<td>IRQ0</td>
</tr>
<tr>
<td>226</td>
<td>E2H</td>
<td>Timer M0 overflow interrupt</td>
<td>IRQ0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Timer M0 capture interrupt</td>
<td>IRQ0</td>
</tr>
<tr>
<td>228</td>
<td>E4H</td>
<td>Timer M2 interval interrupt</td>
<td>IRQ1</td>
</tr>
<tr>
<td>230</td>
<td>E6H</td>
<td>Timer M1 overflow interrupt</td>
<td>IRQ2</td>
</tr>
<tr>
<td>232</td>
<td>E8H</td>
<td>Timer M1 capture interrupt</td>
<td>IRQ2</td>
</tr>
<tr>
<td>234</td>
<td>EAH</td>
<td>DDC (Multi-master IIC-bus) interrupt</td>
<td>IRQ3</td>
</tr>
<tr>
<td>236</td>
<td>ECH</td>
<td>P0.0 external interrupt (INT0)</td>
<td>IRQ4</td>
</tr>
<tr>
<td>238</td>
<td>EEH</td>
<td>P0.1 external interrupt (INT1)</td>
<td>IRQ5</td>
</tr>
<tr>
<td>240</td>
<td>F0H</td>
<td>P0.2 external interrupt (INT2)</td>
<td>IRQ6</td>
</tr>
<tr>
<td>242</td>
<td>F2H</td>
<td>Slave only IIC-bus interrupt (^{(3)})</td>
<td>IRQ7</td>
</tr>
</tbody>
</table>

**NOTES:**
1. Interrupt priorities are identified in inverse order: “0” is the highest priority, “1” is the next highest, and so on.
2. If two or more interrupts within the same level contend, the interrupt with the lowest vector address usually has priority over one with a higher vector address. The priorities within a given level are fixed in hardware.
3. Not used for the S3C8647.
ENABLE/DISABLE INTERRUPT INSTRUCTIONS (EI, DI)

Executing the Enable Interrupts (EI) instruction enables the interrupt structure. All interrupts are then serviced as they occur according to the established priorities.

NOTE

The system initialization routine executed after a reset must always contain an EI instruction (assuming one or more interrupts are used in the application).

During the normal operation, you can execute the DI (Disable Interrupt) instruction at any time to globally disable interrupt processing. The EI and DI instructions change the value of bit 0 in the SYM register. Although you can directly manipulate SYM.0 to enable or disable interrupts, it is recommended that you use the EI and DI instructions instead.

SYSTEM-LEVEL INTERRUPT CONTROL REGISTERS

In addition to the control registers for specific interrupt sources, four system-level registers control interrupt processing:

— The interrupt mask register, IMR, enables (un-masks) or disables (masks) interrupt levels.
— The interrupt priority register, IPR, controls the relative priorities of interrupt levels.
— The interrupt request register, IRQ, contains interrupt pending flags for each interrupt level (as opposed to each interrupt source).
— The system mode register, SYM, enables or disables global interrupt processing. (SYM settings also enable fast interrupts and control the activity of external interface, if implemented.)

<table>
<thead>
<tr>
<th>Control Register</th>
<th>ID</th>
<th>R/W</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt mask register</td>
<td>IMR</td>
<td>R/W</td>
<td>Bit settings in the IMR register enable or disable interrupt processing for each of the eight interrupt levels, IRQ0–IRQ7.</td>
</tr>
<tr>
<td>Interrupt priority register</td>
<td>IPR</td>
<td>R/W</td>
<td>Controls the relative processing priorities of the interrupt levels. The eight levels are organized into three groups: A, B, and C. Group A is IRQ0 and IRQ1, group B is IRQ2–IRQ4, and group C is IRQ5–IRQ7.</td>
</tr>
<tr>
<td>Interrupt request register</td>
<td>IRQ</td>
<td>R</td>
<td>This register contains a request pending bit for each of the seven interrupt levels, IRQ0–IRQ7.</td>
</tr>
<tr>
<td>System mode register</td>
<td>SYM</td>
<td>R/W</td>
<td>This register enables and disables dynamic global interrupt processing and fast interrupt processing.</td>
</tr>
</tbody>
</table>
INTERRUPT PROCESSING CONTROL POINTS

Interrupt processing can therefore be controlled in two ways: globally or by specific interrupt level and source. Among the system-level control points in the interrupt structure are:

— Global interrupt enabled and disabled (by EI and DI instructions or by direct manipulation of SYM.0)
— Interrupt level enable/disable settings (IMR register)
— Interrupt level priority settings (IPR register)
— Interrupt source enable/disable settings in the corresponding peripheral control registers

NOTE

When writing an application program that handles interrupt processing, be sure to include the necessary register file address (register pointer) information.

Figure 5-4. Interrupt Function Diagram
PERIPHERAL INTERRUPT CONTROL REGISTERS

For each interrupt source there is a corresponding peripheral control register (or registers) controlling the interrupts generated by the related peripheral. These registers and their locations are listed in Table 5-3.

<table>
<thead>
<tr>
<th>Interrupt Source</th>
<th>Interrupt Level</th>
<th>Control Register(s)</th>
<th>Register Location(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer M0 overflow interrupt</td>
<td>IRQ0</td>
<td>TM0CON</td>
<td>Set 1, D2H</td>
</tr>
<tr>
<td>Timer M0 capture interrupt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M2 interval interrupt</td>
<td>IRQ1</td>
<td>TM2CON</td>
<td>Set 1, bank 0, F6H</td>
</tr>
<tr>
<td>Timer M1 overflow interrupt</td>
<td>IRQ2</td>
<td>TM1CON</td>
<td>Set 1, bank 0, F5H</td>
</tr>
<tr>
<td>Timer M1 capture interrupt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDC (Multi-master IIC-bus) interrupt</td>
<td>IRQ3</td>
<td>DCCR, DCSR0</td>
<td>Set 1, bank 1, EBH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set 1, bank 1, ECH</td>
</tr>
<tr>
<td>P0.0 external interrupt</td>
<td>IRQ4</td>
<td>P0CONL, P0INT</td>
<td>Set 1, bank 0, E5H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set 1, bank 0, EBH</td>
</tr>
<tr>
<td>P0.1 external interrupt</td>
<td>IRQ5</td>
<td>P0CONL, P0INT</td>
<td>Set 1, bank 0, E5H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set 1, bank 0, EBH</td>
</tr>
<tr>
<td>P0.2 external interrupt</td>
<td>IRQ6</td>
<td>P0CONL, P0INT</td>
<td>Set 1, bank 0, E5H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set 1, bank 0, EBH</td>
</tr>
<tr>
<td>Slave only IIC-bus interrupt (note)</td>
<td>IRQ7</td>
<td>SICSR</td>
<td>Set 1, bank 1, F2H</td>
</tr>
</tbody>
</table>

NOTE: Not used for the S3C8647.
SYSTEM MODE REGISTER (SYM)

The system mode register, SYM (set 1, DEH), is used to globally enable and disable interrupt processing and to control fast interrupt processing. Figure 5-5 shows the effect of the various control settings.

A reset clears SYM.7, SYM.1, and SYM.0 to "0". Other SYM bit values (for fast interrupt level selection) are undetermined.

The instructions EI and DI enable and disable global interrupt processing, respectively, by modifying the bit 0 value of the SYM register. In order to enable interrupt processing an Enable Interrupt (EI) instruction must be included in the initialization routine, which follows a reset operation. Although you can manipulate SYM.0 directly to enable and disable interrupts during the normal operation, it is recommended to use the EI and DI instructions for this purpose.

NOTE: Not used for S3C8639/C863A/C8647

Figure 5-5. System Mode Register (SYM)
**INTERRUPT MASK REGISTER (IMR)**

The interrupt mask register, IMR (set 1, DDH) is used to enable or disable interrupt processing for individual interrupt levels. After a reset, all IMR bit values are undetermined and must therefore be written to their required settings by the initialization routine.

Each IMR bit corresponds to a specific interrupt level: bit 1 to IRQ1, bit 2 to IRQ2, and so on. When the IMR bit of an interrupt level is cleared to "0", interrupt processing for that level is disabled (masked). When you set a level's IMR bit to "1", interrupt processing for the level is enabled (not masked).

The IMR register is mapped to register location DDH in set 1. Bit values can be read and written by instructions using the Register addressing mode.

---

**Figure 5-6. Interrupt Mask Register (IMR)**

---

**NOTE:** Not used for the S3C8647.
INTERRUPT PRIORITY REGISTER (IPR)

The interrupt priority register, IPR (set 1, bank 0, FFH), is used to set the relative priorities of the interrupt levels in the microcontroller’s interrupt structure. After a reset, all IPR bit values are undetermined and must therefore be written to their required settings by the initialization routine.

When more than one interrupt sources are active, the source with the highest priority is serviced first. If two sources belong to the same interrupt level, the source with the lower vector address usually has priority. (This priority is fixed in hardware.)

To support programming of the relative interrupt level priorities, they are organized into groups and subgroups by the interrupt logic. Please note that these groups (and subgroups) are used only by IPR logic for the IPR register priority definitions (see Figure 5-7):

- Group A  IRQ0, IRQ1
- Group B  IRQ2, IRQ3, IRQ4
- Group C  IRQ5, IRQ6, IRQ7

As you can see in Figure 5-8, IPR.7, IPR.4, and IPR.1 control the relative priority of interrupt groups A, B, and C. For example, the setting “001B” for these bits would select the group relationship B > C > A. The setting “101B” would select the relationship C > B > A.

The functions of the other IPR bit settings are as follows:

- Interrupt group C includes a sub group that has an additional priority relationship among interrupt levels 5, 6, and 7. IPR.6 defines the subgroup C relationship.
- IPR.5 controls the relative priorities of group C interrupts.
- Interrupt group B includes a subgroup that has an additional priority relationship among interrupt levels 2, 3, and 4. IPR.3 defines the subgroup B relationship.
- IPR.2 controls interrupt group B.
- IPR.0 controls the relative priority setting of IRQ0 and IRQ1 interrupts.
Interrupt Priority Register (IPR)
FFH, Set 1, Bank 0, R/W

D7 D4 D1
0 0 0 = Not used
0 0 1 = B > C > A
0 1 0 = A > B > C
0 1 1 = B > A > C
1 0 0 = C > A > B
1 0 1 = C > B > A
1 1 0 = A > C > B
1 1 1 = Not used

Group priority:
Group A
0 = IRQ0 > IRQ1
1 = IRQ1 > IRQ0

Group B
0 = IRQ3 > (IRQ, IRQ4)
1 = (IRQ3, IRQ4) > IRQ2

Group C
0 = IRQ5 > (IRQ6, IRQ7)
1 = (IRQ6, IRQ7) > IRQ5

Subgroup C
0 = IRQ6 > IRQ7
1 = IRQ7 > IRQ6

B > C > A
A > B > C
B > A > C
C > A > B
C > B > A
A > C > B
Not used

Figure 5-8. Interrupt Priority Register (IPR)
**INTERRUPT REQUEST REGISTER (IRQ)**

You can poll bit values in the interrupt request register, IRQ (set 1, DCH), to monitor interrupt request status for all levels in the microcontroller’s interrupt structure. Each bit corresponds to the interrupt level of the same number: bit 0 to IRQ0, bit 1 to IRQ1, and so on. A "0" indicates that no interrupt request is currently being issued for that level. A "1" indicates that an interrupt request has been generated for that level.

IRQ bit values are addressable in read-only using Register addressing mode. You can read (test) the contents of the IRQ register at any time using bit or byte addressing to determine the current interrupt request status of specific interrupt levels. After a reset, all IRQ status bits are cleared to “0”.

You can poll IRQ register values even if a DI instruction has been executed (that is, if global interrupt processing is disabled). If an interrupt occurs while the interrupt structure is disabled, the CPU will not service it. You can, however, still detect the interrupt request by polling the IRQ register. In this way, you can determine which events occurred while the interrupt structure was globally disabled.

![Interrupt Request Register (IRQ)](image)

**NOTE:** Not used for the S3C8647.
INTERRUPT PENDING FUNCTION TYPES

Overview
There are two types of interrupt pending bits: One type that is automatically cleared by hardware after the interrupt service routine is acknowledged and executed; the other that must be cleared by the application program's interrupt service routine.

Pending Bits Cleared Automatically by Hardware
For interrupt pending bits that are cleared automatically by hardware, interrupt logic sets the corresponding pending bit to "1" when a request occurs. It then issues an IRQ pulse to inform the CPU that an interrupt is waiting to be serviced. The CPU acknowledges the interrupt source, executes the service routine, and clears the pending bit to "0". This type of pending bit is not mapped and cannot, therefore, be read or written by application software.

In the S3C8639/C863A/C8647 interrupt structure, the timer M0 overflow interrupt (IRQ0, vector E0H), the timer M0 capture interrupt (IRQ0, vector E2H), the timer M2 interval interrupt (IRQ1, vector E4H), and the timer M1 overflow interrupt (IRQ2, vector E6H) belong to this category of interrupts in which pending conditions are cleared automatically by hardware.

Pending Bits Cleared by the Service Routine
The second type of pending bit is the one that should be cleared by program software. The service routine must clear the appropriate pending bit before a return-from-interrupt subroutine (IRET) occurs. To do this, a "0" must be written to the corresponding pending bit location in the source's mode or control register.

In the S3C8639/C863A/C8647 interrupt structure, pending conditions for all interrupt sources, except the timer M0 overflow/capture, the timer M2 interval interrupt and the timer M1 overflow interrupt, must be cleared by the program software's interrupt service routine.
INTERRUPT SOURCE POLLING SEQUENCE

The interrupt request polling and servicing sequence is as follows:

1. A source generates an interrupt request by setting the interrupt request bit to "1".
2. The CPU polling procedure identifies a pending condition for that source.
3. The CPU checks the source's interrupt level.
4. The CPU generates an interrupt acknowledge signal.
5. Interrupt logic determines the interrupt's vector address.
6. The service routine starts and the source's pending bit is cleared to "0" (by hardware or by software).
7. The CPU continues polling for interrupt requests.

INTERRUPT SERVICE ROUTINES

Before an interrupt request is serviced, the following conditions must be met:

— Interrupt processing must be globally enabled (EI, SYM.0 = "1")
— The interrupt level must be enabled (IMR register)
— The interrupt level must have the highest priority if more than one levels are currently requesting service
— The interrupt must be enabled at the interrupt's source (peripheral control register)

When all of the above conditions are met, the interrupt request is acknowledged at the end of the instruction cycle. The CPU then initiates an interrupt machine cycle that completes the following processing sequence:

1. Reset (clear to "0") the interrupt enable bit in the SYM register (SYM.0) to disable all subsequent interrupts.
2. Save the program counter (PC) and status flags to the system stack.
3. Branch to the interrupt vector to fetch the address of the service routine.
4. Pass control to the interrupt service routine.

When the interrupt service routine is completed, the CPU issues an Interrupt Return (IRET). The IRET restores the PC and status flags, setting SYM.0 to "1". It allows the CPU to process the next interrupt request.
GENERATING INTERRUPT VECTOR ADDRESSES

The interrupt vector area in the ROM (00H–FFH) contains the addresses of interrupt service routines that correspond to each level in the interrupt structure. Vectored interrupt processing follows this sequence:

1. Push the program counter's low-byte value to the stack.
2. Push the program counter's high-byte value to the stack.
3. Push the FLAG register values to the stack.
4. Fetch the service routine's high-byte address from the vector location.
5. Fetch the service routine's low-byte address from the vector location.
6. Branch to the service routine specified by the concatenated 16-bit vector address.

NOTE

A 16-bit vector address always begins at an even-numbered ROM address within the range of 00H–FFH.

NESTING OF VECTORED INTERRUPTS

It is possible to nest a higher-priority interrupt request while a lower-priority request is being serviced. To do this, you must follow these steps:

1. Push the current 8-bit interrupt mask register (IMR) value to the stack (PUSH IMR).
2. Load the IMR register with a new mask value that enables only the higher priority interrupt.
3. Execute an EI instruction to enable interrupt processing (a higher priority interrupt will be processed if it occurs).
4. When the lower-priority interrupt service routine ends, restore the IMR to its original value by returning the previous mask value from the stack (POP IMR).
5. Execute an IRET.

Depending on the application, you may be able to simplify the above procedure to some extent.

INSTRUCTION POINTER (IP)

The instruction pointer (IP) is adopted by all the S3C8-series microcontrollers to control the optional high-speed interrupt processing feature called fast interrupts. The IP consists of register pair DAH and DBH. The names IP of registers are IPH (high byte, IP15–IP8) and IPL (low byte, IP7–IP0).
FAST INTERRUPT PROCESSING

The feature called fast interrupt processing allows an interrupt within a given level to be completed in approximately six clock cycles rather than the usual 10 clock cycles. SYM.4–SYM.2 are used to select a specific interrupt level for fast processing and SYM.1 enables or disables fast interrupt processing.

Two other system registers support fast interrupt processing:

— The instruction pointer (IP) contains the starting address of the service routine (and is later used to swap the program counter values), and
— When a fast interrupt occurs, the contents of the FLAGS register is stored in an unmapped, dedicated register called FLAGS' ("FLAGS prime").

NOTES

1. For the S3C8639/C863A/C8647 microcontrollers, the service routine for any of the seven interrupt levels (IRQ0–IRQ7) can be selected for fast interrupt processing. The S3C8647 microcontroller has six interrupt levels (IRQ0-IRQ6) for fast interrupt processing.

2. When you use a fast interrupt in a multi-source interrupt vector, the fast interrupt may not be processed if you use two sources as interrupt vector in normal mode. But it is possible when you use only one source as interrupt vector.

Procedure for Initiating Fast Interrupts

To initiate fast interrupt processing, follow these steps:

1. Load the start address of the service routine into the instruction pointer (IP).
2. Load the interrupt level number (IRQn) into the fast interrupt selection field (SYM.4–SYM.2)
3. Write a “1” to the fast interrupt enable bit in the SYM register.

Fast Interrupt Service Routine

When an interrupt occurs in the level selected for fast interrupt processing, the following events occur:

1. The contents of the instruction pointer and the PC are swapped.
2. The FLAG register values are written to the FLAGS' ("FLAGS prime") register.
3. The fast interrupt status bit in the FLAGS register is set.
4. The interrupt is serviced.
5. Assuming that the fast interrupt status bit is set, when the fast interrupt service routine ends, the instruction pointer and PC values are swapped back.
6. The content of FLAGS' ("FLAGS prime") is copied automatically back to the FLAGS register.
7. The fast interrupt status bit in FLAGS is cleared automatically.

Relationship to Interrupt Pending Bit Types

As described previously, there are two types of interrupt pending bits: One is the type that is automatically cleared by hardware after the interrupt service routine is acknowledged and executed, and the other is the one that must be cleared by the application program's interrupt service routine. You can select fast interrupt processing for interrupts with either type of pending condition clear function — by hardware or by software.
Programming Guidelines

Remember that the only way to enable/disable a fast interrupt is to set/clear the fast interrupt enable bit in the SYM register, SYM.1. Executing an EI or DI instruction globally enables or disables all interrupt processing, including fast interrupts.

NOTE

If you use fast interrupts, remember to load the IP with a new start address when the fast interrupt service routine ends.

PROGRAMMING TIP — Setting Up the Interrupt Control Structure

This example shows you how to enable interrupts for select interrupt sources, disable interrupts for other sources, and set interrupt priorities for the S3C8639/C863A/C8647 interrupt structure. The following is a sample program:

— Disables the watchdog function.
— Enables the following interrupts: P0.0 external interrupt, timer M0 capture/overflow, timer M1 capture/overflow, timer M2 interval interrupt, and DDC interrupt.
— Disables the following interrupts: P0.1 and P0.2 external interrupts, and slave only IIC-bus interrupt.
— Sets interrupt priorities as P0.0 > timer M2 > timer M0 > timer M1 > DDC.

```
DI ; Disable interrupts globally
LD BTCON,#0A0H ; Disable watchdog function
LD P0CONL,#01H ; P0.0 ← enable rising edge interrupts
LD P0INT,#01H ; Enable P0.0 external interrupt
 ; Disable P0.1 and P0.2 external interrupts
LD TM0CON,#8FH ; Enable timer M0 capture interrupt
 ; (capture on rising edges)
LD TM1CON,#3CH ; Enable timer M1 capture/overflow interrupt
LD TM2CON,#3DH ; Enable timer M2 interval interrupt
LD TM2DATA,#249 ; Setting 1ms interval
LD DCCR,#0A3H ; Enable DDC interrupt, SCL clock = 100 kHz
LD IMR,#1FH ; Enable interrupt levels IRQ0, IRQ1, IRQ2, IRQ3 and IRQ4
LD IPR,#1EH ; IRQ4 > IRQ0 > IRQ1 > IRQ2 > IRQ3
 ; (P0.0 > timer M0 > timer M2 > timer M1 > DDC)
EI ; Enable interrupts globally
```
PROGRAMMING TIP — Programming Level IRQ0 as a Fast Interrupt

The following example shows you how to program fast interrupt processing for a selected interrupt level — in this case, for the timer M0 capture interrupt:

```
LD TM0CON,#8FH ; Enable TM0OVF interrupt
; Enable TM0CAP interrupt
; Capture mode (on rising signal edges)
; Select fOSC/8 as the T0 clock source
LD P0CONH,#01H ; Set P0.4 to capture input mode
LDW IPH,#T0_INT ; IPH ← high byte of interrupt service routine
; IPL ← low byte of interrupt service routine
LD SYM,#02H ; Enable fast interrupt processing
; Select IRQ0 for fast interrupt service
EI ; Enable interrupts

FAST_RET: IRET ; IP ← Address of T0_INT (again)
T0_INT:
; (Fast service routine executes)
LD TM0CON,#8FH ; Clear TM0INT interrupt pending bit
JP T,FAST_RET
```
INSTRUCTION SET

OVERVIEW

The SAM8RC instruction set is specifically designed to support the large register files that are typical of most SAM8RC microcontrollers. There are 78 instructions. The powerful data manipulation capabilities and features of the instruction set include:

- A full complement of 8-bit arithmetic and logic operations, including multiply and divide
- No special I/O instructions (I/O control/data registers are mapped directly into the register file)
- Decimal adjustment included in binary-coded decimal (BCD) operations
- 16-bit (word) data can be incremented and decremented
- Flexible instructions for bit addressing, rotate, and shift operations

DATA TYPES

The SAM8RC CPU performs operations on bits, bytes, BCD digits, and two-byte words. Bits in the register file can be set, cleared, complemented, and tested. Bits within a byte are numbered from 7 to 0, where bit 0 is the least significant (right-most) bit.

REGISTER ADDRESSING

To access an individual register, an 8-bit address in the range 0-255 or the 4-bit address of a working register is specified. Paired registers can be used to construct 16-bit data or 16-bit program memory or data memory addresses. For detailed information about register addressing, please refer to Section 2, "Address Spaces."

ADDRESSING MODES

There are seven explicit addressing modes: Register (R), Indirect Register (IR), Indexed (X), Direct (DA), Relative (RA), Immediate (IM), and Indirect (IA). For detailed descriptions of these addressing modes, please refer to Section 3, "Addressing Modes."
Table 6-1. Instruction Group Summary

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>dst</td>
<td>Clear</td>
</tr>
<tr>
<td>LD</td>
<td>dst,src</td>
<td>Load</td>
</tr>
<tr>
<td>LDB</td>
<td>dst,src</td>
<td>Load bit</td>
</tr>
<tr>
<td>LDE</td>
<td>dst,src</td>
<td>Load external data memory</td>
</tr>
<tr>
<td>LDC</td>
<td>dst,src</td>
<td>Load program memory</td>
</tr>
<tr>
<td>LDED</td>
<td>dst,src</td>
<td>Load external data memory and decrement</td>
</tr>
<tr>
<td>LDCD</td>
<td>dst,src</td>
<td>Load program memory and decrement</td>
</tr>
<tr>
<td>LDEI</td>
<td>dst,src</td>
<td>Load external data memory and increment</td>
</tr>
<tr>
<td>LDCI</td>
<td>dst,src</td>
<td>Load program memory and increment</td>
</tr>
<tr>
<td>LDEPD</td>
<td>dst,src</td>
<td>Load external data memory with pre-decrement</td>
</tr>
<tr>
<td>LDCPD</td>
<td>dst,src</td>
<td>Load program memory with pre-decrement</td>
</tr>
<tr>
<td>LDEPI</td>
<td>dst,src</td>
<td>Load external data memory with pre-increment</td>
</tr>
<tr>
<td>LDCPI</td>
<td>dst,src</td>
<td>Load program memory with pre-increment</td>
</tr>
<tr>
<td>LDW</td>
<td>dst,src</td>
<td>Load word</td>
</tr>
<tr>
<td>POP</td>
<td>dst</td>
<td>Pop from stack</td>
</tr>
<tr>
<td>POPUD</td>
<td>dst,src</td>
<td>Pop user stack (decrementing)</td>
</tr>
<tr>
<td>POPUI</td>
<td>dst,src</td>
<td>Pop user stack (incrementing)</td>
</tr>
<tr>
<td>PUSH</td>
<td>src</td>
<td>Push to stack</td>
</tr>
<tr>
<td>PUSHUD</td>
<td>dst,src</td>
<td>Push user stack (decrementing)</td>
</tr>
<tr>
<td>PUSHUI</td>
<td>dst,src</td>
<td>Push user stack (incrementing)</td>
</tr>
</tbody>
</table>
### Table 6-1. Instruction Group Summary (Continued)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic Instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td>dst,src</td>
<td>Add with carry</td>
</tr>
<tr>
<td>ADD</td>
<td>dst,src</td>
<td>Add</td>
</tr>
<tr>
<td>CP</td>
<td>dst,src</td>
<td>Compare</td>
</tr>
<tr>
<td>DA</td>
<td>dst</td>
<td>Decimal adjust</td>
</tr>
<tr>
<td>DEC</td>
<td>dst</td>
<td>Decrement</td>
</tr>
<tr>
<td>DECW</td>
<td>dst</td>
<td>Decrement word</td>
</tr>
<tr>
<td>DIV</td>
<td>dst,src</td>
<td>Divide</td>
</tr>
<tr>
<td>INC</td>
<td>dst</td>
<td>Increment</td>
</tr>
<tr>
<td>INCW</td>
<td>dst</td>
<td>Increment word</td>
</tr>
<tr>
<td>MULT</td>
<td>dst,src</td>
<td>Multiply</td>
</tr>
<tr>
<td>SBC</td>
<td>dst,src</td>
<td>Subtract with carry</td>
</tr>
<tr>
<td>SUB</td>
<td>dst,src</td>
<td>Subtract</td>
</tr>
<tr>
<td><strong>Logic Instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>dst,src</td>
<td>Logical AND</td>
</tr>
<tr>
<td>COM</td>
<td>dst</td>
<td>Complement</td>
</tr>
<tr>
<td>OR</td>
<td>dst,src</td>
<td>Logical OR</td>
</tr>
<tr>
<td>XOR</td>
<td>dst,src</td>
<td>Logical exclusive OR</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Operands</td>
<td>Instruction</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>BTJRF</td>
<td>dst,src</td>
<td>Bit test and jump relative on false</td>
</tr>
<tr>
<td>BTJRT</td>
<td>dst,src</td>
<td>Bit test and jump relative on true</td>
</tr>
<tr>
<td>CALL</td>
<td>dst</td>
<td>Call procedure</td>
</tr>
<tr>
<td>CPIJE</td>
<td>dst,src</td>
<td>Compare, increment and jump on equal</td>
</tr>
<tr>
<td>CPIJNE</td>
<td>dst,src</td>
<td>Compare, increment and jump on non-equal</td>
</tr>
<tr>
<td>DJNZ</td>
<td>r,dst</td>
<td>Decrement register and jump on non-zero</td>
</tr>
<tr>
<td>ENTER</td>
<td></td>
<td>Enter</td>
</tr>
<tr>
<td>EXIT</td>
<td></td>
<td>Exit</td>
</tr>
<tr>
<td>IRET</td>
<td></td>
<td>Interrupt return</td>
</tr>
<tr>
<td>JP</td>
<td>cc,dst</td>
<td>Jump on condition code</td>
</tr>
<tr>
<td>JP</td>
<td>dst</td>
<td>Jump unconditional</td>
</tr>
<tr>
<td>JR</td>
<td>cc,dst</td>
<td>Jump relative on condition code</td>
</tr>
<tr>
<td>NEXT</td>
<td></td>
<td>Next</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td>Return</td>
</tr>
<tr>
<td>WFI</td>
<td></td>
<td>Wait for interrupt</td>
</tr>
</tbody>
</table>

| Program Control Instructions |

| Bit Manipulation Instructions |

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAND</td>
<td>dst,src</td>
<td>Bit AND</td>
</tr>
<tr>
<td>BCP</td>
<td>dst,src</td>
<td>Bit compare</td>
</tr>
<tr>
<td>BITC</td>
<td>dst</td>
<td>Bit complement</td>
</tr>
<tr>
<td>BITR</td>
<td>dst</td>
<td>Bit reset</td>
</tr>
<tr>
<td>BITS</td>
<td>dst</td>
<td>Bit set</td>
</tr>
<tr>
<td>BOR</td>
<td>dst,src</td>
<td>Bit OR</td>
</tr>
<tr>
<td>BXOR</td>
<td>dst,src</td>
<td>Bit XOR</td>
</tr>
<tr>
<td>TCM</td>
<td>dst,src</td>
<td>Test complement under mask</td>
</tr>
<tr>
<td>TM</td>
<td>dst,src</td>
<td>Test under mask</td>
</tr>
</tbody>
</table>
### Table 6-1. Instruction Group Summary (Concluded)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rotate and Shift Instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RL</td>
<td>dst</td>
<td>Rotate left</td>
</tr>
<tr>
<td>RLC</td>
<td>dst</td>
<td>Rotate left through carry</td>
</tr>
<tr>
<td>RR</td>
<td>dst</td>
<td>Rotate right</td>
</tr>
<tr>
<td>RRC</td>
<td>dst</td>
<td>Rotate right through carry</td>
</tr>
<tr>
<td>SRA</td>
<td>dst</td>
<td>Shift right arithmetic</td>
</tr>
<tr>
<td>SWAP</td>
<td>dst</td>
<td>Swap nibbles</td>
</tr>
<tr>
<td><strong>CPU Control Instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCF</td>
<td></td>
<td>Complement carry flag</td>
</tr>
<tr>
<td>DI</td>
<td></td>
<td>Disable interrupts</td>
</tr>
<tr>
<td>EI</td>
<td></td>
<td>Enable interrupts</td>
</tr>
<tr>
<td>IDLE</td>
<td></td>
<td>Enter Idle mode</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td>No operation</td>
</tr>
<tr>
<td>RCF</td>
<td></td>
<td>Reset carry flag</td>
</tr>
<tr>
<td>SB0</td>
<td></td>
<td>Set bank 0</td>
</tr>
<tr>
<td>SB1</td>
<td></td>
<td>Set bank 1</td>
</tr>
<tr>
<td>SCF</td>
<td></td>
<td>Set carry flag</td>
</tr>
<tr>
<td>SRP</td>
<td>src</td>
<td>Set register pointers</td>
</tr>
<tr>
<td>SRP0</td>
<td>src</td>
<td>Set register pointer 0</td>
</tr>
<tr>
<td>SRP1</td>
<td>src</td>
<td>Set register pointer 1</td>
</tr>
<tr>
<td>STOP</td>
<td></td>
<td>Enter Stop mode</td>
</tr>
</tbody>
</table>
FLAGS REGISTER (FLAGS)

The flags register FLAGS contains eight bits that describe the current status of CPU operations. Four of these bits, FLAGS.7–FLAGS.4, can be tested and used with conditional jump instructions; two others FLAGS.3 and FLAGS.2 are used for BCD arithmetic.

The FLAGS register also contains a bit to indicate the status of fast interrupt processing (FLAGS.1) and a bank address status bit (FLAGS.0) to indicate whether bank 0 or bank 1 is currently being addressed. FLAGS register can be set or reset by instructions as long as its outcome does not affect the flags, such as, Load instruction.

Logical and Arithmetic instructions such as, AND, OR, XOR, ADD, and SUB can affect the Flags register. For example, the AND instruction updates the Zero, Sign and Overflow flags based on the outcome of the AND instruction. If the AND instruction uses the Flags register as the destination, then simultaneously, two write will occur to the Flags register producing an unpredictable result.

![Figure 6-1. System Flags Register (FLAGS)](image-url)
FLAG DESCRIPTIONS

C  Carry Flag (FLAGS.7)
   The C flag is set to "1" if the result from an arithmetic operation generates a carry-out from or a borrow to
   the bit 7 position (MSB). After rotate and shift operations, it contains the last value shifted out of the
   specified register. Program instructions can set, clear, or complement the carry flag.

Z  Zero Flag (FLAGS.6)
   For arithmetic and logic operations, the Z flag is set to "1" if the result of the operation is zero. For
   operations that test register bits, and for shift and rotate operations, the Z flag is set to "1" if the result is
   logic zero.

S  Sign Flag (FLAGS.5)
   Following arithmetic, logic, rotate, or shift operations, the sign bit identifies the state of the MSB of the
   result. A logic zero indicates a positive number and a logic one indicates a negative number.

V  Overflow Flag (FLAGS.4)
   The V flag is set to "1" when the result of a two's-complement operation is greater than + 127 or less than
   – 128. It is also cleared to "0" following logic operations.

D  Decimal Adjust Flag (FLAGS.3)
   The DA bit is used to specify what type of instruction was executed last during BCD operations, so that a
   subsequent decimal adjust operation can execute correctly. The DA bit is not usually accessed by
   programmers, and cannot be used as a test condition.

H  Half-Carry Flag (FLAGS.2)
   The H bit is set to "1" whenever an addition generates a carry-out of bit 3, or when a subtraction borrows
   out of bit 4. It is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous
   addition or subtraction into the correct decimal (BCD) result. The H flag is seldom accessed directly by a
   program.

FIS  Fast Interrupt Status Flag (FLAGS.1)
   The FIS bit is set during a fast interrupt cycle and reset during the IRET following interrupt servicing.
   When set, it inhibits all interrupts and causes the fast interrupt return to be executed when the IRET
   instruction is executed.

BA  Bank Address Flag (FLAGS.0)
   The BA flag indicates which register bank in the set 1 area of the internal register file is currently
   selected, bank 0 or bank 1. The BA flag is cleared to "0" (select bank 0) when you execute the SB0
   instruction and is set to "1" (select bank 1) when you execute the SB1 instruction.
INSTRUCTION SET NOTATION

Table 6-2. Flag Notation Conventions

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Carry flag</td>
</tr>
<tr>
<td>Z</td>
<td>Zero flag</td>
</tr>
<tr>
<td>S</td>
<td>Sign flag</td>
</tr>
<tr>
<td>V</td>
<td>Overflow flag</td>
</tr>
<tr>
<td>D</td>
<td>Decimal-adjust flag</td>
</tr>
<tr>
<td>H</td>
<td>Half-carry flag</td>
</tr>
<tr>
<td>0</td>
<td>Cleared to logic zero</td>
</tr>
<tr>
<td>1</td>
<td>Set to logic one</td>
</tr>
<tr>
<td>*</td>
<td>Set or cleared according to operation</td>
</tr>
<tr>
<td>–</td>
<td>Value is unaffected</td>
</tr>
<tr>
<td>x</td>
<td>Value is undefined</td>
</tr>
</tbody>
</table>

Table 6-3. Instruction Set Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>Destination operand</td>
</tr>
<tr>
<td>src</td>
<td>Source operand</td>
</tr>
<tr>
<td>@</td>
<td>Indirect register address prefix</td>
</tr>
<tr>
<td>PC</td>
<td>Program counter</td>
</tr>
<tr>
<td>IP</td>
<td>Instruction pointer</td>
</tr>
<tr>
<td>FLAGS</td>
<td>Flags register (D5H)</td>
</tr>
<tr>
<td>RP</td>
<td>Register pointer</td>
</tr>
<tr>
<td>#</td>
<td>Immediate operand or register address prefix</td>
</tr>
<tr>
<td>H</td>
<td>Hexadecimal number suffix</td>
</tr>
<tr>
<td>D</td>
<td>Decimal number suffix</td>
</tr>
<tr>
<td>B</td>
<td>Binary number suffix</td>
</tr>
<tr>
<td>opc</td>
<td>Opcode</td>
</tr>
</tbody>
</table>
### Table 6-4. Instruction Notation Conventions

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Actual Operand Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc</td>
<td>Condition code</td>
<td>See list of condition codes in Table 6-6.</td>
</tr>
<tr>
<td>r</td>
<td>Working register only</td>
<td>Rn (n = 0–15)</td>
</tr>
<tr>
<td>rb</td>
<td>Bit (b) of working register</td>
<td>Rn.b (n = 0–15, b = 0–7)</td>
</tr>
<tr>
<td>r0</td>
<td>Bit 0 (LSB) of working register</td>
<td>Rn (n = 0–15)</td>
</tr>
<tr>
<td>rr</td>
<td>Working register pair</td>
<td>RRp (p = 0, 2, 4, ..., 14)</td>
</tr>
<tr>
<td>R</td>
<td>Register or working register</td>
<td>reg or Rn (reg = 0–255, n = 0–15)</td>
</tr>
<tr>
<td>Rb</td>
<td>Bit 'b' of register or working register</td>
<td>reg.b (reg = 0–255, b = 0–7)</td>
</tr>
<tr>
<td>RR</td>
<td>Register pair or working register pair</td>
<td>reg or RRp (reg = 0–254, even number only, where p = 0, 2, ..., 14)</td>
</tr>
<tr>
<td>IA</td>
<td>Indirect addressing mode</td>
<td>addr (addr = 0–254, even number only)</td>
</tr>
<tr>
<td>Ir</td>
<td>Indirect working register only</td>
<td>@Rn (n = 0–15)</td>
</tr>
<tr>
<td>IR</td>
<td>Indirect register or indirect working register</td>
<td>@Rn or @reg (reg = 0–255, n = 0–15)</td>
</tr>
<tr>
<td>Irr</td>
<td>Indirect working register pair only</td>
<td>@RRp (p = 0, 2, ..., 14)</td>
</tr>
<tr>
<td>IRR</td>
<td>Indirect register pair or indirect working register pair</td>
<td>@RRp or @reg (reg = 0–254, even only, where p = 0, 2, ..., 14)</td>
</tr>
<tr>
<td>X</td>
<td>Indexed addressing mode</td>
<td>#reg [Rn] (reg = 0–255, n = 0–15)</td>
</tr>
<tr>
<td>XS</td>
<td>Indexed (short offset) addressing mode</td>
<td>#addr [RRp] (addr = range –128 to +127, where p = 0, 2, ..., 14)</td>
</tr>
<tr>
<td>XL</td>
<td>Indexed (long offset) addressing mode</td>
<td>#addr [RRp] (addr = range 0–65535, where p = 0, 2, ..., 14)</td>
</tr>
<tr>
<td>DA</td>
<td>Direct addressing mode</td>
<td>addr (addr = range 0–65535)</td>
</tr>
<tr>
<td>RA</td>
<td>Relative addressing mode</td>
<td>addr (addr = number in the range +127 to –128 that is an offset relative to the address of the next instruction)</td>
</tr>
<tr>
<td>IM</td>
<td>Immediate addressing mode</td>
<td>#data (data = 0–255)</td>
</tr>
<tr>
<td>IML</td>
<td>Immediate (long) addressing mode</td>
<td>#data (data = range 0–65535)</td>
</tr>
</tbody>
</table>
### Table 6-5. Opcode Quick Reference

#### OPCODE MAP

**LOWER NIBBLE (HEX)**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>DEC R1</td>
<td>DEC IR1</td>
<td>ADD r1, r2</td>
<td>ADD r1,lr2</td>
<td>ADD ADD R2,R1</td>
<td>ADD ADD R1,IM</td>
<td>ADD R1,IM</td>
<td>BOR r0–Rb</td>
</tr>
<tr>
<td>P</td>
<td>RLC R1</td>
<td>RLC IR1</td>
<td>ADC r1, r2</td>
<td>ADC r1,lr2</td>
<td>ADC ADC R2,R1</td>
<td>ADC ADC R2,R1</td>
<td>ADC R1,IM</td>
<td>BCP r1,b, R2</td>
</tr>
<tr>
<td>P</td>
<td>INC R1</td>
<td>INC IR1</td>
<td>SUB r1, r2</td>
<td>SUB r1,lr2</td>
<td>SUB SUB R2,R1</td>
<td>SUB SUB R2,R1</td>
<td>SUB R1,IM</td>
<td>BXOR r0–Rb</td>
</tr>
<tr>
<td>E</td>
<td>JP IRR1</td>
<td>SRP/0/1 IM</td>
<td>SBC r1, r2</td>
<td>SBC r1,lr2</td>
<td>SBC SBC R2,R1</td>
<td>SBC SBC R2,R1</td>
<td>SBC R1,IM</td>
<td>BTJR r2,b, RA</td>
</tr>
<tr>
<td>R</td>
<td>DA R1</td>
<td>DA IR1</td>
<td>OR r1, r2</td>
<td>OR r1,lr2</td>
<td>OR OR R2,R1</td>
<td>OR OR R2,R1</td>
<td>OR R1,IM</td>
<td>LDB r0–Rb</td>
</tr>
<tr>
<td>N</td>
<td>POP R1</td>
<td>POP IR1</td>
<td>AND r1, r2</td>
<td>AND r1,lr2</td>
<td>AND AND R2,R1</td>
<td>AND AND R2,R1</td>
<td>AND R1,IM</td>
<td>BITC r1,b</td>
</tr>
<tr>
<td>I</td>
<td>COM R1</td>
<td>COM IR1</td>
<td>TCM r1, r2</td>
<td>TCM r1,lr2</td>
<td>TCM TCM R2,R1</td>
<td>TCM TCM R2,R1</td>
<td>TCM R1,IM</td>
<td>BAND r0–Rb</td>
</tr>
<tr>
<td>B</td>
<td>PUSH R2</td>
<td>PUSH IR2</td>
<td>TM r1, r2</td>
<td>TM r1,lr2</td>
<td>TM TM R2,R1</td>
<td>TM TM R2,R1</td>
<td>TM R1,IM</td>
<td>BIT r1,b</td>
</tr>
<tr>
<td>B</td>
<td>DECW RR1</td>
<td>DECW IR1</td>
<td>PUSHUD IRR1, R2</td>
<td>PUSHUI IRR1, R2</td>
<td>MULT MULT R2,RR1</td>
<td>MULT MULT R2,RR1</td>
<td>MULT R1,IM</td>
<td>LD r1, x, r2</td>
</tr>
<tr>
<td>B</td>
<td>RL R1</td>
<td>RL IR1</td>
<td>POPUD IRR2, R1</td>
<td>POPUI IRR2, R1</td>
<td>DIV DIV R2,RR1</td>
<td>DIV DIV R2,RR1</td>
<td>DIV R1,IM</td>
<td>LD r2, x, r1</td>
</tr>
<tr>
<td>L</td>
<td>INCW RR1</td>
<td>INCW IR1</td>
<td>CP r1, r2</td>
<td>CP r1,lr2</td>
<td>CP CP R2,R1</td>
<td>CP CP R2,R1</td>
<td>CP R1,IM</td>
<td>LDC r1, Irr2, xL</td>
</tr>
<tr>
<td>E</td>
<td>CLR R1</td>
<td>CLR IR1</td>
<td>XOR r1, r2</td>
<td>XOR r1,lr2</td>
<td>XOR XOR R2,R1</td>
<td>XOR XOR R2,R1</td>
<td>XOR R1,IM</td>
<td>LDC r2, Irr2, xL</td>
</tr>
<tr>
<td>C</td>
<td>RRC R1</td>
<td>RRC IR1</td>
<td>CPJE Irr, r2, RA</td>
<td>LDC LDC R2,RR1</td>
<td>LDW LDW R2,RR1</td>
<td>LDW LDW R2,RR1</td>
<td>LDW R1,IML</td>
<td>LD r1, Irr2</td>
</tr>
<tr>
<td>H</td>
<td>SRA R1</td>
<td>SRA IR1</td>
<td>CPIJNE Irr, r2, RA</td>
<td>LDC LDC r2, Irr2</td>
<td>CALL CALL IA1</td>
<td>CALL CALL IA1</td>
<td>CALL IA1, IM</td>
<td>LD r1, Irr2</td>
</tr>
<tr>
<td>E</td>
<td>RR R1</td>
<td>RR IR1</td>
<td>LDDC r1, r2, RA</td>
<td>LDCI LDCI r2, Irr2</td>
<td>LD LD R2, RR1</td>
<td>LD LD R2, RR1</td>
<td>LD R1,IM</td>
<td>LDC r1, Irr2, xs</td>
</tr>
<tr>
<td>X</td>
<td>SWAP R1</td>
<td>SWAP IR1</td>
<td>LDCPD r2, Irr1</td>
<td>LDCPI LDCPI r2, Irr1</td>
<td>CALL CALL IRR1</td>
<td>CALL CALL IRR1</td>
<td>CALL IRR1</td>
<td>LDC r2, Irr1, xs</td>
</tr>
</tbody>
</table>
### Table 6-5. Opcode Quick Reference (Continued)

<table>
<thead>
<tr>
<th>OPCODE MAP</th>
<th>LOWER NIBBLE (HEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>–</td>
</tr>
<tr>
<td><strong>U</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>P</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>P</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>I</strong></td>
<td>6</td>
</tr>
<tr>
<td><strong>I</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>8</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>9</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td>A</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>B</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>C</td>
</tr>
<tr>
<td><strong>H</strong></td>
<td>D</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>E</td>
</tr>
<tr>
<td><strong>X</strong></td>
<td>F</td>
</tr>
</tbody>
</table>
CONDITION CODES

The opcode of a conditional jump always contains a 4-bit field called the condition code (cc). This specifies under which conditions it is to execute the jump. For example, a conditional jump with the condition code for "equal" after a compare operation only jumps if the two operands are equal. Condition codes are listed in Table 6-6.

The carry (C), zero (Z), sign (S), and overflow (V) flags are used to control the operation of conditional jump instructions.

Table 6-6. Condition Codes

<table>
<thead>
<tr>
<th>Binary</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Flags Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>F</td>
<td>Always false</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>T</td>
<td>Always true</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>C</td>
<td>Carry</td>
<td>C = 1</td>
</tr>
<tr>
<td>1111</td>
<td>NC</td>
<td>No carry</td>
<td>C = 0</td>
</tr>
<tr>
<td>0110</td>
<td>Z</td>
<td>Zero</td>
<td>Z = 1</td>
</tr>
<tr>
<td>1110</td>
<td>NZ</td>
<td>Not zero</td>
<td>Z = 0</td>
</tr>
<tr>
<td>1101</td>
<td>PL</td>
<td>Plus</td>
<td>S = 0</td>
</tr>
<tr>
<td>0101</td>
<td>MI</td>
<td>Minus</td>
<td>S = 1</td>
</tr>
<tr>
<td>0100</td>
<td>OV</td>
<td>Overflow</td>
<td>V = 1</td>
</tr>
<tr>
<td>1100</td>
<td>NOV</td>
<td>No overflow</td>
<td>V = 0</td>
</tr>
<tr>
<td>0110</td>
<td>EQ</td>
<td>Equal</td>
<td>Z = 1</td>
</tr>
<tr>
<td>1110</td>
<td>NE</td>
<td>Not equal</td>
<td>Z = 0</td>
</tr>
<tr>
<td>1001</td>
<td>GE</td>
<td>Greater than or equal</td>
<td>(S XOR V) = 0</td>
</tr>
<tr>
<td>0001</td>
<td>LT</td>
<td>Less than</td>
<td>(S XOR V) = 1</td>
</tr>
<tr>
<td>1010</td>
<td>GT</td>
<td>Greater than</td>
<td>(Z OR (S XOR V)) = 0</td>
</tr>
<tr>
<td>0010</td>
<td>LE</td>
<td>Less than or equal</td>
<td>(Z OR (S XOR V)) = 1</td>
</tr>
<tr>
<td>1111</td>
<td>UGE</td>
<td>Unsigned greater than or equal</td>
<td>C = 0</td>
</tr>
<tr>
<td>0111</td>
<td>ULT</td>
<td>Unsigned less than</td>
<td>C = 1</td>
</tr>
<tr>
<td>1011</td>
<td>UGT</td>
<td>Unsigned greater than</td>
<td>(C = 0 AND Z = 0) = 1</td>
</tr>
<tr>
<td>0011</td>
<td>ULE</td>
<td>Unsigned less than or equal</td>
<td>(C OR Z) = 1</td>
</tr>
</tbody>
</table>

NOTES:
1. It indicates condition codes that are related to two different mnemonics but which test the same flag. For example, Z and EQ are both true if the zero flag (Z) is set, but after an ADD instruction, Z would probably be used; after a CP instruction, however, EQ would probably be used.
2. For operations involving unsigned numbers, the special condition codes UGE, ULT, UGT, and ULE must be used.
INSTRUCTION DESCRIPTIONS

This section contains detailed information and programming examples for each instruction in the SAM8RC instruction set. Information is arranged in a consistent format for improved readability and for fast referencing. The following information is included in each instruction description:

— Instruction name (mnemonic)
— Full instruction name
— Source/destination format of the instruction operand
— Shorthand notation of the instruction's operation
— Textual description of the instruction's effect
— Specific flag settings affected by the instruction
— Detailed description of the instruction's format, execution time, and addressing mode(s)
— Programming example(s) explaining how to use the instruction
ADC — Add with carry

**ADC**

dst, src

**Operation:**
dst ← dst + src + c

The source operand, along with the setting of the carry flag, is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed. In multiple precision arithmetic, this instruction permits the carry from the addition of low-order operands to be carried into the addition of high-order operands.

**Flags:**

- **C:** Set if there is a carry from the most significant bit of the result; cleared otherwise.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result is negative; cleared otherwise.
- **V:** Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.
- **D:** Always cleared to "0".
- **H:** Set if there is a carry from the most significant bit of the low-order four bits of the result; cleared otherwise.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>R</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>16</td>
</tr>
</tbody>
</table>

**Example:**

Given: R1 = 10H, R2 = 03H, C flag = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

- ADC R1,R2 → R1 = 14H, R2 = 03H
- ADC R1,@R2 → R1 = 1BH, R2 = 03H
- ADC 01H,02H → Register 01H = 24H, register 02H = 03H
- ADC 01H,@02H → Register 01H = 2BH, register 02H = 03H
- ADC 01H,#11H → Register 01H = 32H

In the first example, destination register R1 contains the value 10H, the carry flag is set to "1", and the source working register R2 contains the value 03H. The statement "ADC R1,R2" adds 03H and the carry flag value ("1") to the destination value 10H, leaving 14H in register R1.
**ADD — Add**

ADD \( \text{dst,src} \)

**Operation:** \( \text{dst} \leftarrow \text{dst} + \text{src} \)

The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed.

**Flags:**
- **C:** Set if there is a carry from the most significant bit of the result; cleared otherwise.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result is negative; cleared otherwise.
- **V:** Set if arithmetic overflow occurred, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.
- **D:** Always cleared to "0".
- **H:** Set if a carry from the low-order nibble occurred.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc dst</td>
<td>src</td>
<td>2</td>
<td>4</td>
<td>02</td>
</tr>
<tr>
<td>opc dst</td>
<td>src</td>
<td>6</td>
<td>03</td>
<td>r</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc dst</td>
<td>src</td>
<td>3</td>
<td>6</td>
<td>04</td>
</tr>
<tr>
<td>opc dst</td>
<td>src</td>
<td>6</td>
<td>05</td>
<td>R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc dst</td>
<td>src</td>
<td>3</td>
<td>6</td>
<td>06</td>
</tr>
</tbody>
</table>

**Example:**

Given: \( R1 = 12H, R2 = 03H, \) register \( 01H = 21H, \) register \( 02H = 03H, \) register \( 03H = 0AH: \)

- ADD \( R1,R2 \) \( \rightarrow \) \( R1 = 15H, R2 = 03H \)
- ADD \( R1,@R2 \) \( \rightarrow \) \( R1 = 1CH, R2 = 03H \)
- ADD \( 01H,02H \) \( \rightarrow \) \( \) register \( 01H = 24H, \) register \( 02H = 03H \)
- ADD \( 01H,@02H \) \( \rightarrow \) \( \) \( \) \( \) \( 01H = 2BH, \) \( \) \( 02H = 03H \)
- ADD \( 01H,#25H \) \( \rightarrow \) \( \) \( \) \( \) \( \) \( 01H = 46H \)

In the first example, destination working register \( R1 \) contains \( 12H \) and the source working register \( R2 \) contains \( 03H. \) The statement "ADD \( R1,R2 \)" adds \( 03H \) to \( 12H, \) leaving the value \( 15H \) in register \( R1. \)
AND — Logical AND

AND  dst,src

Operation:  dst ← dst AND src

The source operand is logically ANDed with the destination operand. The result is stored in the destination. The AND operation results in a "1" bit being stored whenever the corresponding bits in the two operands are both logic ones; otherwise a "0" bit value is stored. The contents of the source are unaffected.

Flags:  
C:  Unaffected.
Z:  Set if the result is "0"; cleared otherwise.
S:  Set if the result bit 7 is set; cleared otherwise.
V:  Always cleared to "0".
D:  Unaffected.
H:  Unaffected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc dst</td>
<td>src 2</td>
<td>52</td>
<td>r</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>opc</td>
<td>src dst 3</td>
<td>53</td>
<td>r</td>
<td>Ir</td>
<td></td>
</tr>
<tr>
<td>opc src dst 3</td>
<td>54</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opc dst src 3</td>
<td>55</td>
<td>R</td>
<td>IR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opc dst src 3</td>
<td>56</td>
<td>R</td>
<td>IM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:  Given:  R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

AND  R1,R2  →  R1 = 02H, R2 = 03H
AND  R1,@R2  →  R1 = 02H, R2 = 03H
AND  01H,02H  →  Register 01H = 01H, register 02H = 03H
AND  01H,@02H  →  Register 01H = 00H, register 02H = 03H
AND  01H,#25H  →  Register 01H = 21H

In the first example, destination working register R1 contains the value 12H and the source working register R2 contains 03H. The statement "AND R1,R2" logically ANDs the source operand 03H with the destination operand value 12H, leaving the value 02H in register R1.
BAND — Bit AND

**Operation:**
- \( \text{dst}(0) \leftarrow \text{dst}(0) \ AND \ \text{src}(b) \)
- or
- \( \text{dst}(b) \leftarrow \text{dst}(b) \ AND \ \text{src}(0) \)

The specified bit of the source (or the destination) is logically ANDed with the zero bit (LSB) of the destination (or source). The resultant bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

**Flags:**
- **C:** Unaffected.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Cleared to "0".
- **V:** Undefined.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>b</th>
<th>0</th>
<th>src</th>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>67</td>
<td>dst</td>
<td>src</td>
<td>r0</td>
<td>Rb</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**
Given: \( R1 = 07H \) and register \( 01H = 05H \):

BAND \( R1,01H.1 \) → \( R1 = 06H \), register \( 01H = 05H \)

BAND \( 01H.1,R1 \) → Register \( 01H = 05H \), \( R1 = 07H \)

In the first example, source register \( 01H \) contains the value \( 05H \) (00000101B) and destination working register \( R1 \) contains \( 07H \) (00000111B). The statement "BAND R1,01H.1" ANDs the bit 1 value of the source register ("0") with the bit 0 value of register \( R1 \) (destination), leaving the value \( 06H \) (00000110B) in register \( R1 \).
BCP — Bit Compare

**Operation:**

\[
dst(0) - src(b)
\]

The specified bit of the source is compared to (subtracted from) bit zero (LSB) of the destination. The zero flag is set if the bits are the same; otherwise it is cleared. The contents of both operands are unaffected by the comparison.

**Flags:**
- **C:** Unaffected.
- **Z:** Set if the two bits are the same; cleared otherwise.
- **S:** Cleared to "0".
- **V:** Undefined.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>b</th>
<th>0</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>17</td>
<td>r0</td>
<td>Rb</td>
</tr>
</tbody>
</table>

**NOTE:** In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

**Example:**

Given: \( R1 = 07H \) and register \( 01H = 01H \):

\[
BCP \ R1,01H.1 \rightarrow R1 = 07H, \text{ register } 01H = 01H
\]

If destination working register \( R1 \) contains the value \( 07H \) (00000111B) and the source register \( 01H \) contains the value \( 01H \) (00000001B), the statement "BCP \( R1,01H.1 \)" compares bit one of the source register (01H) and bit zero of the destination register (R1). Because the bit values are not identical, the zero flag bit (Z) is cleared in the FLAGS register (0D5H).
BITC — Bit Complement

BITC dst.b

Operation: dst(b) ← NOT dst(b)

This instruction complements the specified bit within the destination without affecting any other bits in the destination.

Flags:
- C: Unaffected.
- Z: Set if the result is "0"; cleared otherwise.
- S: Cleared to "0".
- V: Undefined.
- D: Unaffected.
- H: Unaffected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example:

Given: R1 = 07H
BITC R1.1 → R1 = 05H

If working register R1 contains the value 07H (00000111B), the statement "BITC R1.1" complements bit one of the destination and leaves the value 05H (00000101B) in register R1. Because the result of the complement is not "0", the zero flag (Z) in the FLAGS register (0D5H) is cleared.
**BITR — Bit Reset**

**BITR**

dst.b

**Operation:**

dst(b) ← 0

The BITR instruction clears the specified bit within the destination without affecting any other bits in the destination.

**Flags:**

No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

2 4 77 rb

**NOTE:** In the second byte of the instruction format, the destination address is four bits, the bit address ‘b’ is three bits, and the LSB address value is one bit in length.

**Example:**

Given: R1 = 07H:

BITR R1.1 → R1 = 05H

If the value of working register R1 is 07H (00000111B), the statement “BITR R1.1” clears bit one of the destination register R1, leaving the value 05H (00000101B).
**BITS — Bit Set**

**BITS**

\[ \text{dst.b} \]

**Operation:**

\[ \text{dst(b)} \leftarrow 1 \]

The BITS instruction sets the specified bit within the destination without affecting any other bits in the destination.

**Flags:**

No flags are affected.

**Format:**

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Bytes} & \text{Cycles} & \text{Opcode (Hex)} & \text{Addr Mode} \\
\hline
\text{opc} & \text{dst | b | 1} & 2 & 4 & 77 & rb \\
\hline
\end{array}
\]

**NOTE:** In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

**Example:**

Given: \( R1 = 07H \):

BITS \( R1.3 \) \( \rightarrow \) \( R1 = 0FH \)

If working register \( R1 \) contains the value 07H (00000111B), the statement "BITS \( R1.3 \)" sets bit three of the destination register \( R1 \) to "1", leaving the value 0FH (00001111B).
BOR — Bit OR

**BOR**

<table>
<thead>
<tr>
<th>Operation</th>
<th>dst(0) ← dst(0) OR src(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>dst(b) ← dst(b) OR src(0)</td>
</tr>
</tbody>
</table>

The specified bit of the source (or the destination) is logically ORed with bit zero (LSB) of the destination (or the source). The resulting bit value is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

**Flags:**

- **C:** Unaffected.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Cleared to "0".
- **V:** Undefined.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>b</th>
<th>0</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>r0</td>
<td>Rb</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>src</th>
<th>b</th>
<th>1</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>Rb</td>
<td>r0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** In the second byte of the 3-byte instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit.

**Example:**

Given: R1 = 07H and register 01H = 03H:

- BOR R1, 01H.1 → R1 = 07H, register 01H = 03H
- BOR 01H.2, R1 → Register 01H = 07H, R1 = 07H

In the first example, destination working register R1 contains the value 07H (00000111B) and source register 01H the value 03H (00000011B). The statement "BOR R1,01H.1" logically ORs bit one of register 01H (source) with bit zero of R1 (destination). This leaves the same value (07H) in working register R1.

In the second example, destination register 01H contains the value 03H (00000011B) and the source working register R1 the value 07H (00000111B). The statement "BOR 01H.2,R1" logically ORs bit two of register 01H (destination) with bit zero of R1 (source). This leaves the value 07H in register 01H.
BTJRF — Bit Test, Jump Relative on False

BTJRF dst, src.b

Operation: If src(b) is a "0", then PC ← PC + dst

The specified bit within the source operand is tested. If it is a "0", the relative address is added to the program counter and control passes to the statement whose address is now in the PC; otherwise, the instruction following the BTJRF instruction is executed.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>src</td>
<td>b</td>
<td>0 dst</td>
</tr>
</tbody>
</table>

NOTE: In the second byte of the instruction format, the source address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BTJRF SKIP, R1.3 → PC jumps to SKIP location

If working register R1 contains the value 07H (00000111B), the statement "BTJRF SKIP, R1.3" tests bit 3. Because it is "0", the relative address is added to the PC and the PC jumps to the memory location pointed to by the SKIP. (Remember that the memory location must be within the allowed range of +127 to −128.)
**BTJRT — Bit Test, Jump Relative on True**

**BTJRT**  
dst, src.b

**Operation:**  
If src(b) is a "1", then PC ← PC + dst  
The specified bit within the source operand is tested. If it is a "1", the relative address is added to the program counter and control passes to the statement whose address is now in the PC; otherwise, the instruction following the BTJRT instruction is executed.

**Flags:**  
No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>3</td>
<td>37</td>
<td>RA</td>
</tr>
<tr>
<td>src</td>
<td>10</td>
<td></td>
<td>rb</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dst</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** In the second byte of the instruction format, the source address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

**Example:**

Given: R1 = 07H:

```
BTJRT SKIP,R1.1
```

If working register R1 contains the value 07H (00000111B), the statement "BTJRT SKIP,R1.1" tests bit one in the source register (R1). Because it is a "1", the relative address is added to the PC and the PC jumps to the memory location pointed to by the SKIP. (Remember that the memory location must be within the allowed range of +127 to −128.)
BXOR — Bit XOR

**BXOR**

dst, src.b

**Operation:**

dst(0) ← dst(0) XOR src(b)

or

dst(b) ← dst(b) XOR src(0)

The specified bit of the source (or the destination) is logically exclusive-ORed with bit zero (LSB) of the destination (or source). The result bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

**Flags:**

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Cleared to "0".
V: Undefined.
D: Unaffected.
H: Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>b</th>
<th>0</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dst</td>
<td>b</td>
<td>0</td>
<td>src</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>r0</td>
<td>Rb</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>src</th>
<th>b</th>
<th>1</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dst</td>
<td>b</td>
<td>1</td>
<td>src</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Rb</td>
<td>r0</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** In the second byte of the 3-byte instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

**Example:**

Given: R1 = 07H (00000111B) and register 01H = 03H (00000011B):

BXOR R1,01H.1  →  R1 = 06H, register 01H = 03H
BXOR 01H.2,R1  →  Register 01H = 07H, R1 = 07H

In the first example, destination working register R1 has the value 07H (00000111B) and source register 01H has the value 03H (00000011B). The statement "BXOR R1,01H.1" exclusive-ORs bit one of register 01H (source) with bit zero of R1 (destination). The result bit value is stored in bit zero of R1, changing its value from 07H to 06H. The value of source register 01H is unaffected.
CALL — Call Procedure

CALL dst

Operation:  
SP ← SP – 1  
@SP ← PCL  
SP ← SP – 1  
@SP ← PCH  
PC ← dst

The current contents of the program counter are pushed onto the top of the stack. The program counter value used is the address of the first instruction following the CALL instruction. The specified destination address is then loaded into the program counter and points to the first instruction of a procedure. At the end of the procedure the return instruction (RET) can be used to return to the original program flow. RET pops the top of the stack back into the program counter.

Flags:  
No flags are affected.

Format:

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
</tr>
</tbody>
</table>

Example:  
Given: R0 = 35H, R1 = 21H, PC = 1A47H, and SP = 0002H:

CALL 3521H  →  SP = 0000H  
(Memory locations 0000H = 1AH, 0001H = 4AH, where 4AH is the address that follows the instruction.)

CALL @RR0  →  SP = 0000H (0000H = 1AH, 0001H = 49H)

CALL #40H  →  SP = 0000H (0000H = 1AH, 0001H = 49H)

In the first example, if the program counter value is 1A47H and the stack pointer contains the value 0002H, the statement "CALL 3521H" pushes the current PC value onto the top of the stack. The stack pointer now points to memory location 0000H. The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed.

If the contents of the program counter and stack pointer are the same as in the first example, the statement "CALL @RR0" produces the same result except that the 49H is stored in stack location 0001H (because the two-byte instruction format was used). The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed. Assuming that the contents of the program counter and stack pointer are the same as in the first example, if program address 0040H contains 35H and program address 0041H contains 21H, the statement "CALL #40H" produces the same result as in the second example.
CCF — Complement Carry Flag

**Operation:**
\[ C \leftarrow \text{NOT } C \]

The carry flag (C) is complemented. If \( C = \text{"1"} \), the value of the carry flag is changed to logic zero; if \( C = \text{"0"} \), the value of the carry flag is changed to logic one.

**Flags:**
- **C:** Complemented.
  - No other flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

**Example:**

Given: The carry flag = "0":

CCF

If the carry flag = "0", the CCF instruction complements it in the FLAGS register (0D5H), changing its value from logic zero to logic one.
CLR — Clear

CLR  dst

Operation:  dst ← "0"
The destination location is cleared to "0".

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>voc</td>
<td>dst</td>
<td>2</td>
<td>B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>B1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IR</td>
</tr>
</tbody>
</table>

Example: Given: Register 00H = 4FH, register 01H = 02H, and register 02H = 5EH:

CLR  00H  →  Register 00H = 00H
CLR  @01H →  Register 01H = 02H, register 02H = 00H

In Register (R) addressing mode, the statement "CLR  00H" clears the destination register 00H value to 00H. In the second example, the statement "CLR  @01H" uses Indirect Register (IR) addressing mode to clear the 02H register value to 00H.
COM — Complement

COM  dst

Operation:  dst ← NOT dst

The contents of the destination location are complemented (one’s complement); all "1s" are changed to "0s", and vice-versa.

Flags:

C:  Unaffected.
Z:  Set if the result is "0"; cleared otherwise.
S:  Set if the result bit 7 is set; cleared otherwise.
V:  Always reset to "0".
D:  Unaffected.
H:  Unaffected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IR</td>
</tr>
</tbody>
</table>

Example:  Given:  R1 = 07H and register 07H = 0F1H:

COM  R1  →  R1 = 0F8H
COM  @R1  →  R1 = 07H, register 07H = 0EH

In the first example, destination working register R1 contains the value 07H (00000111B). The statement "COM R1" complements all the bits in R1: all logic ones are changed to logic zeros, and vice-versa, leaving the value 0F8H (11111000B).

In the second example, Indirect Register (IR) addressing mode is used to complement the value of destination register 07H (11110001B), leaving the new value 0EH (00001110B).
**CP — Compare**

**Operation:**

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags are set accordingly. The contents of both operands are unaffected by the comparison.

**Flags:**

- **C:** Set if a "borrow" occurred (src > dst); cleared otherwise.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result is negative; cleared otherwise.
- **V:** Set if arithmetic overflow occurred; cleared otherwise.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>r</td>
<td>r</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>r</td>
<td>lr</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>R</td>
<td>R</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>R</td>
<td>IR</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>R</td>
<td>IM</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Examples:**

1. Given: R1 = 02H and R2 = 03H:

   CP R1,R2 → Set the C and S flags

   Destination working register R1 contains the value 02H and source register R2 contains the value 03H. The statement "CP R1,R2" subtracts the R2 value (source/subtrahend) from the R1 value (destination/minuend). Because a "borrow" occurs and the difference is negative, C and S are "1".

2. Given: R1 = 05H and R2 = 0AH:

   CP R1,R2
   JP UGE,SKIP
   INC R1
   SKIP LD R3,R1

   In this example, destination working register R1 contains the value 05H which is less than the contents of the source working register R2 (0AH). The statement "CP R1,R2" generates C = "1" and the JP instruction does not jump to the SKIP location. After the statement "LD R3,R1" executes, the value 06H remains in working register R3.
**CPIJE — Compare, Increment, and Jump on Equal**

**CPIJE**  
dst,src,RA

**Operation:**  
If dst – src = "0", PC ← PC + RA  
Ir ← Ir + 1

The source operand is compared to (subtracted from) the destination operand. If the result is "0", the relative address is added to the program counter and control passes to the statement whose address is now in the program counter. Otherwise, the instruction immediately following the CPIJE instruction is executed. In either case, the source pointer is incremented by one before the next instruction is executed.

**Flags:**  
No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>src</td>
<td>dst</td>
<td>RA</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>C2</td>
<td>r Ir</td>
</tr>
</tbody>
</table>

**NOTE:** Execution time is 18 cycles if the jump is taken or 16 cycles if it is not taken.

**Example:**  
Given: R1 = 02H, R2 = 03H, and register 03H = 02H:

CPIJE R1,@R2,SKIP → R2 = 04H, PC jumps to SKIP location

In this example, working register R1 contains the value 02H, working register R2 the value 03H, and register 03 contains 02H. The statement "CPIJE R1,@R2,SKIP" compares the @R2 value (00000010B) to 02H (00000010B). Because the result of the comparison is equal, the relative address is added to the PC and the PC then jumps to the memory location pointed to by SKIP. The source register (R2) is incremented by one, leaving a value of 04H. (Remember that the memory location must be within the allowed range of +127 to −128.)
CPIJNE — Compare, Increment, and Jump on Non-Equal

CPIJNE     dst, src, RA

Operation:  If dst – src != “0”, PC ← PC + RA
            Ir ← Ir + 1
            The source operand is compared to (subtracted from) the destination operand. If the result is not
            “0”, the relative address is added to the program counter and control passes to the statement
            whose address is now in the program counter; otherwise the instruction following the CPIJNE
            instruction is executed. In either case the source pointer is incremented by one before the next
            instruction.

Flags:      No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>src</td>
<td>dst</td>
<td>RA</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>D2</td>
<td>r</td>
</tr>
</tbody>
</table>

NOTE: Execution time is 18 cycles if the jump is taken or 16 cycles if it is not taken.

Example:   Given: R1 = 02H, R2 = 03H, and register 03H = 04H:
            CPIJNE R1, @R2, SKIP → R2 = 04H, PC jumps to SKIP location

Working register R1 contains the value 02H, working register R2 (the source pointer) the value
03H, and general register 03 the value 04H. The statement "CPIJNE R1, @R2, SKIP" subtracts
04H (00000100B) from 02H (00000010B). Because the result of the comparison is non-equal,
the relative address is added to the PC and the PC then jumps to the memory location pointed to
by SKIP. The source pointer register (R2) is also incremented by one, leaving a value of 04H.
(Remember that the memory location must be within the allowed range of +127 to –128.)
DA — Decimal Adjust

**DA**  
**dst**

**Operation:**  
dst ← DA dst

The destination operand is adjusted to form two 4-bit BCD digits following an addition or subtraction operation. For addition (ADD, ADC) or subtraction (SUB, SBC), the following table indicates the operation performed. (The operation is undefined if the destination operand was not the result of a valid addition or subtraction of BCD digits):

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Carry Before DA</th>
<th>Bits 4–7 Value (Hex)</th>
<th>H Flag Before DA</th>
<th>Bits 0–3 Value (Hex)</th>
<th>Number Added to Byte</th>
<th>Carry After DA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0–9</td>
<td>0</td>
<td>0–9</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0–8</td>
<td>0</td>
<td>A–F</td>
<td>06</td>
<td>0</td>
</tr>
<tr>
<td>ADD</td>
<td>0</td>
<td>A–F</td>
<td>0</td>
<td>0–9</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>ADC</td>
<td>0</td>
<td>9–F</td>
<td>0</td>
<td>A–F</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>A–F</td>
<td>1</td>
<td>0–3</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0–2</td>
<td>0</td>
<td>0–9</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0–2</td>
<td>0</td>
<td>A–F</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0–3</td>
<td>1</td>
<td>0–3</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td>SUB</td>
<td>0</td>
<td>0–9</td>
<td>0</td>
<td>0–9</td>
<td>00</td>
<td>–00</td>
</tr>
<tr>
<td>SBC</td>
<td>1</td>
<td>7–F</td>
<td>0</td>
<td>0–9</td>
<td>A0</td>
<td>–60</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>6–F</td>
<td>1</td>
<td>6–F</td>
<td>9A</td>
<td>–66</td>
</tr>
</tbody>
</table>

**Flags:**
- **C:** Set if there was a carry from the most significant bit; cleared otherwise (see table).
- **Z:** Set if result is "0"; cleared otherwise.
- **S:** Set if result bit 7 is set; cleared otherwise.
- **V:** Undefined.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>40</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41</td>
<td>IR</td>
</tr>
</tbody>
</table>
DA — Decimal Adjust

Example: Given: Working register R0 contains the value 15 (BCD), working register R1 contains 27 (BCD), and address 27H contains 46 (BCD):

\[
\text{ADD R1,R0 ; } C \leftarrow \text{"0"}, H \leftarrow \text{"0"}, \text{ Bits 4–7 } = 3, \text{ bits 0–3 } = C, \text{ R1 } \leftarrow 3CH \\
\text{DA R1 ; } R1 \leftarrow 3CH + 06
\]

If addition is performed using the BCD values 15 and 27, the result should be 42. The sum is incorrect, however, when the binary representations are added in the destination location using standard binary arithmetic:

\[
\begin{array}{cccccc}
0 & 0 & 0 & 1 & 0 & 1 \\
+ & 0 & 0 & 1 & 0 & 1 1 \\
\hline
0 & 0 & 1 & 1 & 1 & 0 0
\end{array} = 3CH
\]

The DA instruction adjusts this result so that the correct BCD representation is obtained:

\[
\begin{array}{cccccc}
0 & 0 & 1 & 1 & 1 & 0 0 \\
+ & 0 & 0 & 0 & 0 & 1 1 0 \\
\hline
0 & 1 & 0 & 0 & 0 & 0 1 0 = 42
\end{array}
\]

Assuming the same values given above, the statements

\[
\text{SUB 27H,R0 ; } C \leftarrow \text{"0"}, H \leftarrow \text{"0"}, \text{ Bits 4–7 } = 3, \text{ bits 0–3 } = 1 \\
\text{DA @R1 ; } @R1 \leftarrow 31–0
\]

leave the value 31 (BCD) in address 27H (@R1).
**DEC — Decrement**

**DEC**  
\[ \text{dst} \]

**Operation:**  
\[ \text{dst} \leftarrow \text{dst} - 1 \]

The contents of the destination operand are decremented by one.

**Flags:**
- **C:** Unaffected.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if result is negative; cleared otherwise.
- **V:** Set if arithmetic overflow occurred; cleared otherwise.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>00</td>
<td>R</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td></td>
<td>IR</td>
</tr>
</tbody>
</table>

**Example:**

Given: \( R1 = 03H \) and register \( 03H = 10H \):

- **DEC R1**  
  \( R1 = 02H \)
- **DEC @R1**  
  Register \( 03H = 0FH \)

In the first example, if working register \( R1 \) contains the value \( 03H \), the statement "DEC R1" decrements the hexadecimal value by one, leaving the value \( 02H \). In the second example, the statement "DEC @R1" decrements the value \( 10H \) contained in the destination register \( 03H \) by one, leaving the value \( 0FH \).
DECW — Decrement Word

**Operation:**

\[ \text{dst} \leftarrow \text{dst} - 1 \]

The contents of the destination location (which must be an even address) and the operand following that location are treated as a single 16-bit value that is decremented by one.

**Flags:**

- **C:** Unaffected.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result is negative; cleared otherwise.
- **V:** Set if arithmetic overflow occurred; cleared otherwise.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>80 RR</td>
<td>81 IR</td>
</tr>
</tbody>
</table>

**Example:**

Given: \( R0 = 12H, R1 = 34H, R2 = 30H, \) register \( 30H = 0FH, \) and register \( 31H = 21H: \)

```
DECW RR0 \rightarrow R0 = 12H, R1 = 33H
DECW @R2 \rightarrow Register 30H = 0FH, register 31H = 20H
```

In the first example, destination register \( R0 \) contains the value \( 12H \) and register \( R1 \) the value \( 34H. \) The statement "DECW RR0" addresses \( R0 \) and the following operand \( R1 \) as a 16-bit word and decrements the value of \( R1 \) by one, leaving the value \( 33H. \)

**NOTE:**

A system malfunction may occur if you use a Zero flag (FLAGS.6) result together with a DECW instruction. To avoid this problem, we recommend that you use DECW as shown in the following example:

```
LOOP:  DECW RR0
       LD R2,R1
       OR R2,R0
       JR NZ,LOOP
```
**DI — Disable Interrupts**

**Operation:** SYM (0) ← 0

Bit zero of the system mode control register, SYM.0, is cleared to "0", globally disabling all interrupt processing. Interrupt requests will continue to set their respective interrupt pending bits, but the CPU will not service them while interrupt processing is disabled.

**Flags:** No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>1</td>
<td>8F</td>
</tr>
</tbody>
</table>

**Example:** Given: SYM = 01H:

DI

If the value of the SYM register is 01H, the statement "DI" leaves the new value 00H in the register and clears SYM.0 to "0", disabling interrupt processing.

Before changing IMR, interrupt pending and interrupt source control register, be sure DI state.
**DIV — Divide (Unsigned)**

**DIV**

dst, src

**Operation:**

dst ÷ src

dst (UPPER) ← REMAINDER

dst (LOWER) ← QUOTIENT

The destination operand (16 bits) is divided by the source operand (8 bits). The quotient (8 bits) is stored in the lower half of the destination. The remainder (8 bits) is stored in the upper half of the destination. When the quotient is ≥ 2^8, the numbers stored in the upper and lower halves of the destination for quotient and remainder are incorrect. Both operands are treated as unsigned integers.

**Flags:**

C: Set if the V flag is set and quotient is between 2^8 and 2^9 – 1; cleared otherwise.

Z: Set if divisor or quotient = "0"; cleared otherwise.

S: Set if MSB of quotient = "1"; cleared otherwise.

V: Set if quotient is ≥ 2^8 or if divisor = "0"; cleared otherwise.

D: Unaffected.

H: Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>RR</td>
<td>R</td>
</tr>
<tr>
<td>95</td>
<td>RR</td>
<td>IR</td>
</tr>
<tr>
<td>96</td>
<td>RR</td>
<td>IM</td>
</tr>
</tbody>
</table>

**Note:** Execution takes 10 cycles if the divide-by-zero is attempted; otherwise it takes 26 cycles.

**Examples:**

Given: R0 = 10H, R1 = 03H, R2 = 40H, register 40H = 80H:

DIV RR0,R2 → R0 = 03H, R1 = 40H

DIV RR0,@R2 → R0 = 03H, R1 = 20H

DIV RR0,#20H → R0 = 03H, R1 = 80H

In the first example, destination working register pair RR0 contains the values 10H (R0) and 03H (R1), and register R2 contains the value 40H. The statement "DIV RR0,R2" divides the 16-bit RR0 value by the 8-bit value of the R2 (source) register. After the DIV instruction, R0 contains the value 03H and R1 contains 40H. The 8-bit remainder is stored in the upper half of the destination register RR0 (R0) and the quotient in the lower half (R1).
DJNZ — Decrement and Jump if Non-Zero

**Operation:**
\[ r \leftarrow r - 1 \]
If \( r \neq 0 \), \( PC \leftarrow PC + dst \)

The working register being used as a counter is decremented. If the contents of the register are not logic zero after decrementing, the relative address is added to the program counter and control passes to the statement whose address is now in the PC. The range of the relative address is +127 to −128, and the original value of the PC is taken to be the address of the instruction byte following the DJNZ statement.

**Flags:**
No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>opc</td>
<td>dst</td>
<td>2</td>
</tr>
<tr>
<td>dst</td>
<td>8 (no jump)</td>
<td>r = 0 to F</td>
<td></td>
</tr>
</tbody>
</table>

**Example:**
Given: \( R1 = 02H \) and LOOP is the label of a relative address:

```
SRP  #0C0H
DJNZ  R1,LOOP
```

DJNZ is typically used to control a "loop" of instructions. In many cases, a label is used as the destination operand instead of a numeric relative address value. In the example, working register \( R1 \) contains the value 02H, and LOOP is the label for a relative address.

The statement "DJNZ R1, LOOP" decrements register R1 by one, leaving the value 01H. Because the contents of R1 after the decrement are non-zero, the jump is taken to the relative address specified by the LOOP label.
EI — Enable Interrupts

EI

Operation: SYM (0) ← 1

An EI instruction sets bit zero of the system mode register, SYM.0 to "1". This allows interrupts to be serviced as they occur (assuming they have highest priority). If an interrupt's pending bit was set while interrupt processing was disabled (by executing a DI instruction), it will be serviced when you execute the EI instruction.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>9F</td>
</tr>
</tbody>
</table>

Example: Given: SYM = 00H:

EI

If the SYM register contains the value 00H, that is, if interrupts are currently disabled, the statement "EI" sets the SYM register to 01H, enabling all interrupts. (SYM.0 is the enable bit for global interrupt processing.)
ENTER — Enter

Operation:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>← SP – 2</td>
</tr>
<tr>
<td>@SP</td>
<td>← IP</td>
</tr>
<tr>
<td>IP</td>
<td>← PC</td>
</tr>
<tr>
<td>PC</td>
<td>← @IP</td>
</tr>
<tr>
<td>IP</td>
<td>← IP + 2</td>
</tr>
</tbody>
</table>

This instruction is useful when implementing threaded-code languages. The contents of the instruction pointer are pushed to the stack. The program counter (PC) value is then written to the instruction pointer. The program memory word that is pointed to by the instruction pointer is loaded into the PC, and the instruction pointer is incremented by two.

Flags:
No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>1F</td>
</tr>
</tbody>
</table>

Example:
The diagram below shows one example of how to use an ENTER statement.
EXIT — Exit

EXIT

Operation:

\[
\begin{align*}
\text{IP} & \leftarrow @\text{SP} \\
\text{SP} & \leftarrow \text{SP} + 2 \\
\text{PC} & \leftarrow @\text{IP} \\
\text{IP} & \leftarrow \text{IP} + 2
\end{align*}
\]

This instruction is useful when implementing threaded-code languages. The stack value is popped and loaded into the instruction pointer. The program memory word that is pointed to by the instruction pointer is then loaded into the program counter, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

\[
\begin{array}{ccc}
\text{Bytes} & \text{Cycles} & \text{Opcode (Hex)} \\
1 & 14 & 2F \\
16 & 16 (internal stack) & 2F \\
16 (internal stack) & & \\
\end{array}
\]

Example: The diagram below shows one example of how to use an EXIT statement.
IDLE — Idle Operation

IDLE

Operation:
The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue. Idle mode can be released by an interrupt request (IRQ) or an external reset operation.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>6F</td>
<td>–</td>
</tr>
</tbody>
</table>

Example: The instruction

IDLE
NOP
NOP
NOP
NOP

stops the CPU clock but not the system clock.
INSTRUCTION SET

INC — Increment

INC    dst

Operation: dst ← dst + 1

The contents of the destination operand are incremented by one.

Flags:  
- **C:** Unaffected.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result is negative; cleared otherwise.
- **V:** Set if arithmetic overflow occurred; cleared otherwise.
- **D:** Unaffected.
- **H:** Unaffected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>rE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r = 0 to F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>20</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>21</td>
<td>IR</td>
</tr>
</tbody>
</table>

Example:  
Given: R0 = 1BH, register 00H = 0CH, and register 1BH = 0FH:

INC R0 → R0 = 1CH
INC 00H → Register 00H = 0DH
INC @R0 → R0 = 1BH, register 01H = 10H

In the first example, if destination working register R0 contains the value 1BH, the statement "INC R0" leaves the value 1CH in that same register.

The next example shows the effect an INC instruction has on register 00H, assuming that it contains the value 0CH.

In the third example, INC is used in Indirect Register (IR) addressing mode to increment the value of register 1BH from 0FH to 10H.
INCW — Increment Word

**INCW**

**dst**

**Operation:**

\[ dst \leftarrow dst + 1 \]

The contents of the destination (which must be an even address) and the byte following that location are treated as a single 16-bit value that is incremented by one.

**Flags:**

- **C:** Unaffected.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result is negative; cleared otherwise.
- **V:** Set if arithmetic overflow occurred; cleared otherwise.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>A0</td>
<td>RR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A1</td>
<td>IR</td>
</tr>
</tbody>
</table>

**Example:**

Given: \( R0 = 1AH, R1 = 02H, \) register \( 02H = 0FH, \) and register \( 03H = 0FFH: \)

- **INCW RR0** \( \rightarrow R0 = 1AH, R1 = 03H \)
- **INCW @R1** \( \rightarrow \) Register \( 02H = 10H, \) register \( 03H = 00H \)

In the first example, the working register pair RR0 contains the value 1AH in register R0 and 02H in register R1. The statement "INCW RR0" increments the 16-bit destination by one, leaving the value 03H in register R1. In the second example, the statement "INCW @R1" uses Indirect Register (IR) addressing mode to increment the contents of general register 03H from 0FFH to 00H and register 02H from 0FH to 10H.

**NOTE:**

A system malfunction may occur if you use a Zero (Z) flag (FLAGS.6) result together with an INCW instruction. To avoid this problem, we recommend that you use INCW as shown in the following example:

```
LOOP: INCW RR0
      LD R2,R1
      OR R2,R0
      JR NZ,LOOP
```
# IRET — Interrupt Return

**IRET** is used at the end of an interrupt service routine. It restores the flag register and the program counter. It also re-enables global interrupts.

** Flags:** All flags are restored to their original settings (that is, the settings before the interrupt occurred).

<table>
<thead>
<tr>
<th>Format:</th>
<th>IRET (Normal)</th>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>opc</td>
<td>1</td>
<td>10 (internal stack)</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12 (external stack)</td>
<td></td>
</tr>
</tbody>
</table>

** Example:** In the figure below, the instruction pointer is initially loaded with 100H in the main program before interrupts are enabled. When an interrupt occurs, the program counter and instruction pointer are swapped. This causes the PC to jump to address 100H and the IP to keep the return address. The last instruction in the service routine normally is a jump to IRET at address FFH. This causes the instruction pointer to be loaded with 100H "again" and the program counter to jump back to the main program. Now, the next interrupt can occur and the IP is still correct at 100H.

<table>
<thead>
<tr>
<th>0H</th>
<th>FFH</th>
<th>IRET</th>
</tr>
</thead>
<tbody>
<tr>
<td>100H</td>
<td></td>
<td>Interrupt Service Routine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP to FFH</td>
</tr>
<tr>
<td></td>
<td>FFFFH</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** In the fast interrupt example above, if the last instruction is not a jump to IRET, you must pay attention to the order of the last two instructions. The IRET cannot be immediately proceeded by a clearing of the interrupt status (as with a reset of the IPR register).
**JP — Jump**

**JP**  
cc,dst  (Conditional)

**JP**  
dst  (Unconditional)

**Operation:** If cc is true, PC ← dst

The conditional JUMP instruction transfers program control to the destination address if the condition specified by the condition code (cc) is true; otherwise, the instruction following the JP instruction is executed. The unconditional JP simply replaces the contents of the PC with the contents of the specified register pair. Control then passes to the statement addressed by the PC.

**Flags:** No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>cc</th>
<th>opc</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>ccD</td>
</tr>
</tbody>
</table>

cc = 0 to F

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

**NOTES:**
1. The 3-byte format is used for a conditional jump and the 2-byte format for an unconditional jump.
2. In the first byte of the three-byte instruction format (conditional jump), the condition code and the opcode are both four bits.

**Example:** Given: The carry flag (C) = "1", register 00 = 01H, and register 01 = 20H:

**JP** C,LABEL_W → LABEL_W = 1000H, PC = 1000H

**JP** @00H → PC = 0120H

The first example shows a conditional JP. Assuming that the carry flag is set to "1", the statement "JP C,LABEL_W" replaces the contents of the PC with the value 1000H and transfers control to that location. Had the carry flag not been set, control would then have passed to the statement immediately following the JP instruction.

The second example shows an unconditional JP. The statement "JP @00" replaces the contents of the PC with the contents of the register pair 00H and 01H, leaving the value 0120H.
**JR — Jump Relative**

**JR**  
cc,dst

**Operation:**  
If cc is true, PC ← PC + dst  
If the condition specified by the condition code (cc) is true, the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise, the instruction following the JR instruction is executed. (See list of condition codes).

The range of the relative address is +127, –128, and the original value of the program counter is taken to be the address of the first instruction byte following the JR statement.

**Flags:**  
No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>cc</th>
<th>opc</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>ccB</td>
</tr>
</tbody>
</table>

cc = 0 to F  

**NOTE:** In the first byte of the two-byte instruction format, the condition code and the opcode are each four bits.

**Example:**  
Given: The carry flag = "1" and LABEL_X = 1FF7H:

JR C,LABEL_X → PC = 1FF7H  

If the carry flag is set (that is, if the condition code is true), the statement "JR C,LABEL_X" will pass control to the statement whose address is now in the PC. Otherwise, the program instruction following the JR would be executed.
**LD — Load**

**LD**  \(\text{dst,src}\)

**Operation:**  \(\text{dst} \leftarrow \text{src}\)

The contents of the source are loaded into the destination. The source's contents are unaffected.

**Flags:**  No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>rC</td>
<td>r IM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>r8</td>
<td>r R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>r9</td>
<td>R r</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(r = 0) to (F)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>C7</td>
<td>r Ir</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>D7</td>
<td>Ir r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>E4</td>
<td>R R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>E5</td>
<td>R IR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>E6</td>
<td>R IM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>D6</td>
<td>IR IM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>F5</td>
<td>IR R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>87</td>
<td>r x [r]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>97</td>
<td>x [r] r</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INSTRUCTION SET

LD — Load

LD (Continued)

Examples: Given: R0 = 01H, R1 = 0AH, register 00H = 01H, register 01H = 20H, register 02H = 02H, LOOP = 30H, and register 3AH = 0FFH:

- LD R0,#10H → R0 = 10H
- LD R0,01H → R0 = 20H, register 01H = 20H
- LD 01H,R0 → Register 01H = 01H, R0 = 01H
- LD R1,@R0 → R1 = 20H, R0 = 01H
- LD @R0,R1 → R0 = 01H, R1 = 0AH, register 01H = 0AH
- LD 00H,01H → Register 00H = 20H, register 01H = 20H
- LD 02H,@00H → Register 02H = 20H, register 00H = 01H
- LD 00H,#0AH → Register 00H = 0AH
- LD @00H,#10H → Register 00H = 01H, register 01H = 10H
- LD @00H,02H → Register 00H = 01H, register 01H = 02, register 02H = 02H
- LD R0,#LOOP[R1] → R0 = 0FFH, R1 = 0AH
- LD #LOOP[R0],R1 → Register 31H = 0AH, R0 = 01H, R1 = 0AH
LDB — Load Bit

LDB dst, src.b
LDB dst.b, src

Operation:
dst(0) ← src(b)
or
dst(b) ← src(0)
The specified bit of the source is loaded into bit zero (LSB) of the destination, or bit zero of the source is loaded into the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:
No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>47</td>
</tr>
<tr>
<td>opc</td>
<td>src</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>47</td>
</tr>
</tbody>
</table>

NOTE: In the second byte of the instruction formats, the destination (or source) address is four bits, the bit address ‘b’ is three bits, and the LSB address value is one bit in length.

Examples:
Given: R0 = 06H and general register 00H = 05H:
LDB R0,00H.2 → R0 = 07H, register 00H = 05H
LDB 00H.0,R0 → R0 = 06H, register 00H = 04H

In the first example, destination working register R0 contains the value 06H and the source general register 00H the value 05H. The statement "LD R0,00H.2" loads the bit two value of the 00H register into bit zero of the R0 register, leaving the value 07H in register R0.

In the second example, 00H is the destination register. The statement "LD 00H.0,R0" loads bit zero of register R0 to the specified bit (bit zero) of the destination register, leaving 04H in general register 00H.
**LDC/LDE — Load Memory**

**LDC/LDE**

dst, src

**Operation:**

dst ← src

This instruction loads a byte from program or data memory into a working register or vice-versa. The source values are unaffected. LDC refers to program memory and LDE to data memory. The assembler makes 'Irr' or 'rr' values an even number for program memory and odd an odd number for data memory.

**Flags:**

No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>2</td>
<td>C3</td>
<td>r</td>
<td>Irr</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>D3</td>
<td>Irr</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>3</td>
<td>E7</td>
<td>r</td>
<td>XS</td>
<td>[rr]</td>
</tr>
<tr>
<td>4.</td>
<td>3</td>
<td>F7</td>
<td>XS</td>
<td>[rr]</td>
<td>r</td>
</tr>
<tr>
<td>5.</td>
<td>4</td>
<td>A7</td>
<td>r</td>
<td>XL</td>
<td>[rr]</td>
</tr>
<tr>
<td>6.</td>
<td>4</td>
<td>B7</td>
<td>XL</td>
<td>[rr]</td>
<td>r</td>
</tr>
<tr>
<td>7.</td>
<td>4</td>
<td>A7</td>
<td>r</td>
<td>DA</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>4</td>
<td>B7</td>
<td>DA</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>4</td>
<td>A7</td>
<td>r</td>
<td>DA</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>4</td>
<td>B7</td>
<td>DA</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**

1. The source (src) or working register pair [rr] for formats 5 and 6 cannot use register pair 0–1.
2. For formats 3 and 4, the destination address 'XS [rr]' and the source address 'XS [rr]' are each one byte.
3. For formats 5 and 6, the destination address 'XL [rr]' and the source address 'XL [rr]' are each two bytes.
4. The DA and r source values for formats 7 and 8 are used to address program memory; the second set of values, used in formats 9 and 10, are used to address data memory.
LDC/LDE — Load Memory

LDC/LDE  (Continued)

Examples:  Given:  R0  =  11H, R1  =  34H, R2  =  01H, R3  =  04H; Program memory locations 0103H  =  4FH, 0104H  =  1A, 0105H  =  6DH, and 1104H  =  88H. External data memory locations 0103H  =  5FH, 0104H  =  2AH, 0105H  =  7DH, and 1104H  =  98H:

LDC  R0,@RR2 ; R0 ← contents of program memory location 0104H  
            ; R0  =  1AH, R2  =  01H, R3  =  04H
LDE  R0,@RR2 ; R0 ← contents of external data memory location 0104H  
            ; R0  =  2AH, R2  =  01H, R3  =  04H
LDC  (note)  @RR2,R0 ; 11H (contents of R0) is loaded into program memory  
            ; location 0104H (RR2),  
            ; working registers R0, R2, R3  →  no change
LDE  @RR2,R0 ; 11H (contents of R0) is loaded into external data memory  
            ; location 0104H (RR2),  
            ; working registers R0, R2, R3  →  no change
LDC  R0,#01H[RR2] ; R0 ← contents of program memory location 0105H  
            ; (01H + RR2),  
            ; R0  =  6DH, R2  =  01H, R3  =  04H
LDE  R0,#01H[RR2] ; R0 ← contents of external data memory location 0105H  
            ; (01H + RR2), R0  =  7DH, R2  =  01H, R3  =  04H
LDC  (note)  #01H[RR2],R0 ; 11H (contents of R0) is loaded into program memory location  
            ; 0105H (01H + 0104H)  
LDE  #01H[RR2],R0 ; 11H (contents of R0) is loaded into external data memory  
            ; location 0105H (01H + 0104H)
LDC  R0,#1000H[RR2] ; R0 ← contents of program memory location 1104H  
            ; (1000H + 0104H), R0  =  88H, R2  =  01H, R3  =  04H
LDE  R0,#1000H[RR2] ; R0 ← contents of external data memory location 1104H  
            ; (1000H + 0104H), R0  =  98H, R2  =  01H, R3  =  04H
LDC  R0,1104H ; R0 ← contents of program memory location 1104H, R0  =  88H
LDE  R0,1104H ; R0 ← contents of external data memory location 1104H,  
            ; R0  =  98H
LDC  (note)  1105H,R0 ; 11H (contents of R0) is loaded into program memory location  
            ; 1105H, (1105H)  ←  11H
LDE  1105H,R0 ; 11H (contents of R0) is loaded into external data memory  
            ; location 1105H, (1105H)  ←  11H

NOTE:  These instructions are not supported by masked ROM type devices.
**LDCD/LDED — Load Memory and Decrement**

**LDCD/LDED**  dst,src

**Operation:**

\[
\text{dst} \leftarrow \text{src} \\
\text{rr} \leftarrow \text{rr} - 1
\]

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then decremented. The contents of the source are unaffected.

LDCD references program memory and LDED references external data memory. The assembler makes 'rr' an even number for program memory and an odd number for data memory.

**Flags:**

No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>src</td>
<td>E2</td>
</tr>
</tbody>
</table>

**Examples:**

Given: \(R6 = 10H, R7 = 33H, R8 = 12H,\) program memory location \(1033H = 0CDH,\) and external data memory location \(1033H = 0DDH:\)

LDCD \(R8, @RR6\) ; 0CDH (contents of program memory location 1033H) is loaded into R8 and RR6 is decremented by one
\(\text{R8} = 0CDH, \text{R6} = 10H, \text{R7} = 32H (RR6 \leftarrow RR6 - 1)\)

LDED \(R8, @RR6\) ; 0DDH (contents of data memory location 1033H) is loaded into R8 and RR6 is decremented by one (RR6 \leftarrow RR6 - 1)
\(\text{R8} = 0DDH, \text{R6} = 10H, \text{R7} = 32H\)
LDCI/LDEI — Load Memory and Increment

LDCI/LDEI  dst,src

Operation:  dst ← src
rr ← rr + 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then incremented automatically. The contents of the source are unaffected.

LDCI refers to program memory and LDEI refers to external data memory. The assembler makes 'Ir' even for program memory and odd for data memory.

Flags:  No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc dst</td>
<td>src</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Examples:  Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory locations 1033H = 0CDH and 1034H = 0C5H; external data memory locations 1033H = 0DDH and 1034H = 0D5H:

LDCI R8,@RR6 ; 0CDH (contents of program memory location 1033H) is loaded into R8 and RR6 is incremented by one (RR6 ← RR6 + 1)
; R8 = 0CDH, R6 = 10H, R7 = 34H

LDEI R8,@RR6 ; 0DDH (contents of data memory location 1033H) is loaded into R8 and RR6 is incremented by one (RR6 ← RR6 + 1)
; R8 = 0DDH, R6 = 10H, R7 = 34H
LDCPD/LDEPD — Load Memory with Pre-Decrement

LDCPD/

LDEPD   dst, src

Operation:  \( \text{id} \leftarrow \text{id} - 1 \)
 dst  \leftarrow\ src

These instructions are used for block transfers of data from program or data memory from the
register file. The address of the memory location is specified by a working register pair and is
first decremented. The contents of the source location are then loaded into the destination
location. The contents of the source are unaffected.

LDCPD refers to program memory and LDEPD refers to external data memory. The assembler
makes 'Irr' an even number for program memory and an odd number for external data memory.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>src</td>
<td>dst</td>
<td>Irr</td>
</tr>
</tbody>
</table>

Examples: Given: R0 = 77H, R6 = 30H, and R7 = 00H:

LDCPD @RR6,R0 ; (RR6 \leftarrow RR6 – 1)
; 77H (contents of R0) is loaded into program memory location
; 2FFFH (3000H – 1H)
; R0 = 77H, R6 = 2FH, R7 = 0FFH

LDEPD @RR6,R0 ; (RR6 \leftarrow RR6 – 1)
; 77H (contents of R0) is loaded into external data memory
; location 2FFFH (3000H – 1H)
; R0 = 77H, R6 = 2FH, R7 = 0FFH
LDCPI/LDEPI — Load Memory with Pre-Increment

LDCPI/

LDEPI       dst,src
Operation:  rr ← rr + 1
dst ← src

These instructions are used for block transfers of data from program or data memory from the register file. The address of the memory location is specified by a working register pair and is first incremented. The contents of the source location are loaded into the destination location. The contents of the source are unaffected.

LDCPI refers to program memory and LDEPI refers to external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>src</td>
<td>dst</td>
<td>src</td>
</tr>
</tbody>
</table>

Examples: Given: R0 = 7FH, R6 = 21H, and R7 = 0FFH:

LDCPI @RR6,R0  ; (RR6 ← RR6 + 1)
; 7FH (contents of R0) is loaded into program memory
; location 2200H (21FFH + 1H)
; R0 = 7FH, R6 = 22H, R7 = 00H

LDEPI @RR6,R0  ; (RR6 ← RR6 + 1)
; 7FH (contents of R0) is loaded into external data memory
; location 2200H (21FFH + 1H)
; R0 = 7FH, R6 = 22H, R7 = 00H
LDW — Load Word

**LDW**  
dst, src

**Operation:**  
dst ← src

The contents of the source (a word) are loaded into the destination. The contents of the source are unaffected.

**Flags:**  
No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>C4</td>
</tr>
<tr>
<td>8</td>
<td>C5</td>
<td>RR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>C6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IML</td>
</tr>
</tbody>
</table>

**Examples:**

Given: R4 = 06H, R5 = 1CH, R6 = 05H, R7 = 02H, register 00H = 1AH, register 01H = 02H, register 02H = 03H, and register 03H = 0FH:

- **LDW** RR6,RR4 → R6 = 06H, R7 = 1CH, R4 = 06H, R5 = 1CH
- **LDW** 00H,02H → Register 00H = 03H, register 01H = 0FH, register 02H = 03H, register 03H = 0FH
- **LDW** RR2,@R7 → R2 = 03H, R3 = 0FH,
- **LDW** 04H,@01H → Register 04H = 03H, register 05H = 0FH
- **LDW** RR6,#1234H → R6 = 12H, R7 = 34H
- **LDW** 02H,#0FEDH → Register 02H = 0FH, register 03H = 0EDH

In the second example, please note that the statement "LDW 00H,02H" loads the contents of the source word 02H, 03H into the destination word 00H, 01H. This leaves the value 03H in general register 00H and the value 0FH in register 01H.

The other examples show how to use the LDW instruction with various addressing modes and formats.
MULT — Multiply (Unsigned)

MULT  dst, src

Operation:  dst ← dst × src

The 8-bit destination operand (even register of the register pair) is multiplied by the source operand (8 bits) and the product (16 bits) is stored in the register pair specified by the destination address. Both operands are treated as unsigned integers.

Flags:
C: Set if result is > 255; cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if MSB of the result is a "1"; cleared otherwise.
V: Cleared.
D: Unaffected.
H: Unaffected.

Format:

<table>
<thead>
<tr>
<th>opc</th>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>22</td>
<td>84</td>
</tr>
<tr>
<td>22</td>
<td>85</td>
<td>RR</td>
</tr>
<tr>
<td>22</td>
<td>86</td>
<td>RR</td>
</tr>
</tbody>
</table>

Examples:  Given: Register 00H = 20H, register 01H = 03H, register 02H = 09H, register 03H = 06H:

MULT 00H, 02H  →  Register 00H = 01H, register 01H = 20H, register 02H = 09H
MULT 00H, @01H  →  Register 00H = 00H, register 01H = 0C0H
MULT 00H, #30H  →  Register 00H = 06H, register 01H = 00H

In the first example, the statement "MULT 00H,02H" multiplies the 8-bit destination operand (in the register 00H of the register pair 00H, 01H) by the source register 02H operand (09H). The 16-bit product, 0120H, is stored in the register pair 00H, 01H.
NEXT — Next

Operation: \[ \text{PC} \leftarrow @ \text{IP} \]
\[ \text{IP} \leftarrow \text{IP} + 2 \]

The NEXT instruction is useful when implementing threaded-code languages. The program memory word that is pointed to by the instruction pointer is loaded into the program counter. The instruction pointer is then incremented by two.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0F</td>
</tr>
</tbody>
</table>

Example: The following diagram shows one example of how to use the NEXT instruction.
NOP — No Operation

**NOP**

**Operation:** No action is performed when the CPU executes this instruction. Typically, one or more NOPs are executed in sequence in order to effect a timing delay of variable duration.

**Flags:** No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>FF</td>
</tr>
</tbody>
</table>

**Example:** When the instruction

NOP

is encountered in a program, no operation occurs. Instead, there is a delay in instruction execution time.
OR — Logical OR

OR dst, src

Operation: dst ← dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are unaffected. The OR operation results in a "1" being stored whenever either of the corresponding bits in the two operands is a "1"; otherwise a "0" is stored.

Flags:
- C: Unaffected.
- Z: Set if the result is "0"; cleared otherwise.
- S: Set if the result bit 7 is set; cleared otherwise.
- V: Always cleared to "0".
- D: Unaffected.
- H: Unaffected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>src</td>
<td>2 4 42 r r</td>
</tr>
<tr>
<td></td>
<td>6 43 r lr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opc</td>
<td>src</td>
<td>dst</td>
<td>3 6 44 R R</td>
</tr>
<tr>
<td></td>
<td>6 45 R IR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opc</td>
<td>dst</td>
<td>src</td>
<td>3 6 46 R IM</td>
</tr>
</tbody>
</table>

Examples:

Given: R0 = 15H, R1 = 2AH, R2 = 01H, register 00H = 08H, register 01H = 37H, and register 08H = 8AH:

OR R0,R1 → R0 = 3FH, R1 = 2AH
OR R0,@R2 → R0 = 37H, R2 = 01H, register 01H = 37H
OR 00H,01H → Register 00H = 3FH, register 01H = 37H
OR 01H,@00H → Register 00H = 08H, register 01H = 0BFH
OR 00H,#02H → Register 00H = 0AH

In the first example, if working register R0 contains the value 15H and register R1 the value 2AH, the statement "OR R0,R1" logical-ORs the R0 and R1 register contents and stores the result (3FH) in destination register R0.

The other examples show the use of the logical OR instruction with the various addressing modes and formats.
**POP — Pop From Stack**

**POP**

**dst**

**Operation:**

\[ \text{dst} \leftarrow @\text{SP} \]

\[ \text{SP} \leftarrow \text{SP} + 1 \]

The contents of the location addressed by the stack pointer are loaded into the destination. The stack pointer is then incremented by one.

**Flags:**

No flags affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>50</td>
<td>R</td>
</tr>
<tr>
<td>8</td>
<td>51</td>
<td></td>
<td>IR</td>
</tr>
</tbody>
</table>

**Examples:**

Given: Register 00H = 01H, register 01H = 1BH, SPH (0D8H) = 00H, SPL (0D9H) = 0FBH, and stack register 0FBH = 55H:

POP 00H  \rightarrow  Register 00H = 55H, SP = 00FCH

POP @00H  \rightarrow  Register 00H = 01H, register 01H = 55H, SP = 00FCH

In the first example, general register 00H contains the value 01H. The statement "POP 00H" loads the contents of location 00FBH (55H) into destination register 00H and then increments the stack pointer by one. Register 00H then contains the value 55H and the SP points to location 00FCH.
### POPUD — Pop User Stack (Decrementing)

**POPUD**

\[ \text{dst,src} \]

**Operation:**

\[ \text{dst} \leftarrow \text{src} \]

\[ \text{IR} \leftarrow \text{IR} - 1 \]

This instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack pointer is then decremented.

**Flags:**

No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>src</td>
<td>dst</td>
<td></td>
</tr>
</tbody>
</table>

| 3 | 8 | 92 | R | IR |

**Example:**

Given: Register 00H = 42H (user stack pointer register), register 42H = 6FH, and register 02H = 70H:

POPUD 02H,@00H → Register 00H = 41H, register 02H = 6FH, register 42H = 6FH

If general register 00H contains the value 42H and register 42H the value 6FH, the statement “POPUD 02H,@00H” loads the contents of register 42H into the destination register 02H. The user stack pointer is then decremented by one, leaving the value 41H.
**POPUI — Pop User Stack (Incrementing)**

**POPUI**

| dst,src |

**Operation:**

```
dst ← src
IR ← IR + 1
```

The POPUI instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack pointer is then incremented.

**Flags:**

No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>src</th>
<th>dst</th>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>93</td>
<td>R</td>
<td>IR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

Given: Register 00H = 01H and register 01H = 70H:

```
POPUI 02H,@00H → Register 00H = 02H, register 01H = 70H, register 02H = 70H
```

If general register 00H contains the value 01H and register 01H the value 70H, the statement "POPUI 02H,@00H" loads the value 70H into the destination general register 02H. The user stack pointer (register 00H) is then incremented by one, changing its value from 01H to 02H.
PUSH — Push To Stack

PUSH src

Operation: SP ← SP – 1
            @SP ← src

A PUSH instruction decrements the stack pointer value and loads the contents of the source (src) into the location addressed by the decremented stack pointer. The operation then adds the new value to the top of the stack.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th></th>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>src</td>
<td>2</td>
<td>8 (internal clock)</td>
<td>70 R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>8 (external clock)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>8 (internal clock)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>8 (external clock)</td>
<td></td>
</tr>
<tr>
<td>opc</td>
<td>src</td>
<td>8</td>
<td>71 IR</td>
<td></td>
</tr>
</tbody>
</table>

Examples: Given: Register 40H = 4FH, register 4FH = 0AAH, SPH = 00H, and SPL = 00H:

PUSH 40H → Register 40H = 4FH, stack register 0FFH = 4FH, SPH = 0FFH, SPL = 0FFH

PUSH @40H → Register 40H = 4FH, register 4FH = 0AAH, stack register 0FFH = 0AAH, SPH = 0FFH, SPL = 0FFH

In the first example, if the stack pointer contains the value 0000H, and general register 40H the value 4FH, the statement "PUSH 40H" decrements the stack pointer from 0000 to 0FFFFFFH. It then loads the contents of register 40H into location 0FFFFFFH and adds this new value to the top of the stack.
**PUSHUD — Push User Stack (Decrementing)**

**PUSHUD**  
dst, src  

**Operation:**  
IR ← IR − 1  
dst ← src  

This instruction is used to address user-defined stacks in the register file. PUSHUD decrements the user stack pointer and loads the contents of the source into the register addressed by the decremented stack pointer.

**Flags:**  
No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>82</td>
<td>IR R</td>
</tr>
</tbody>
</table>

**Example:**  
Given: Register 00H = 03H, register 01H = 05H, and register 02H = 1AH:

PUSHUD @00H,01H  
→  
Register 00H = 02H, register 01H = 05H, register 02H = 05H

If the user stack pointer (register 00H, for example) contains the value 03H, the statement "PUSHUD @00H,01H" decrements the user stack pointer by one, leaving the value 02H. The 01H register value, 05H, is then loaded into the register addressed by the decremented user stack pointer.
**PUSHUI — Push User Stack (Incrementing)**

**PUSHUI**  
dst,src

**Operation:**  
IR ← IR + 1  
dst ← src

This instruction is used for user-defined stacks in the register file. PUSHUI increments the user stack pointer and then loads the contents of the source into the register location addressed by the incremented user stack pointer.

**Flags:**  
No flags are affected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>src</td>
<td></td>
</tr>
</tbody>
</table>

| 3     | 8      | 83           | IR R      |

**Example:**  
Given: Register 00H = 03H, register 01H = 05H, and register 04H = 2AH:

PUSHUI @00H,01H → Register 00H = 04H, register 01H = 05H, register 04H = 05H

If the user stack pointer (register 00H, for example) contains the value 03H, the statement "PUSHUI @00H,01H" increments the user stack pointer by one, leaving the value 04H. The 01H register value, 05H, is then loaded into the location addressed by the incremented user stack pointer.
RCF — Reset Carry Flag

Operation: C ← 0

The carry flag is cleared to logic zero, regardless of its previous value.

Flags: C: Cleared to "0".

No other flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>1</td>
<td>CF</td>
</tr>
</tbody>
</table>

Example: Given: C = "1" or "0":

The instruction RCF clears the carry flag (C) to logic zero.
RET — Return

RET

Operation: \[
\begin{align*}
\text{PC} & \leftarrow @\text{SP} \\
\text{SP} & \leftarrow \text{SP} + 2
\end{align*}
\]

The RET instruction is normally used to return to the previously executing procedure at the end of a procedure entered by a CALL instruction. The contents of the location addressed by the stack pointer are popped into the program counter. The next statement that is executed is the one that is addressed by the new program counter value.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8 (internal stack)</td>
<td>AF</td>
</tr>
<tr>
<td>10</td>
<td>(external stack)</td>
<td></td>
</tr>
</tbody>
</table>

Example:

Given: \( \text{SP} = 00\text{FCH}, (\text{SP}) = 10\text{1AH}, \) and \( \text{PC} = 12\text{34}: \)

\[
\text{RET} \quad \quad \quad \text{PC} = 10\text{1AH}, \text{SP} = 00\text{FEH}
\]

The statement "RET" pops the contents of stack pointer location 00FCH (10H) into the high byte of the program counter. The stack pointer then pops the value in location 00FEH (1AH) into the PC's low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location 00FEH.
RL — Rotate Left

**Operation:**
\[ \text{dst} \leftarrow \text{dst} (7) \]
\[ \text{dst} (0) \leftarrow \text{dst} (7) \]
\[ \text{dst} (n + 1) \leftarrow \text{dst} (n), \ n = 0–6 \]

The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit zero (LSB) position and also replaces the carry flag.

**Flags:**
- **C:** Set if the bit rotated from the most significant bit position (bit 7) was "1".
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result bit 7 is set; cleared otherwise.
- **V:** Set if arithmetic overflow occurred; cleared otherwise.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>2 4 90</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 91</td>
<td>IR</td>
</tr>
</tbody>
</table>

**Examples:**

Given: Register 00H = 0AAH, register 01H = 02H and register 02H = 17H:

- RL 00H → Register 00H = 55H, C = "1"
- RL @01H → Register 01H = 02H, register 02H = 2EH, C = "0"

In the first example, if general register 00H contains the value 0AAH (10101010B), the statement "RL 00H" rotates the 0AAH value left one bit position, leaving the new value 55H (01010101B) and setting the carry and overflow flags.
**RLC — Rotate Left Through Carry**

**Operation:**

- \( \text{dst} (0) \leftarrow C \)
- \( C \leftarrow \text{dst} (7) \)
- \( \text{dst} (n + 1) \leftarrow \text{dst} (n), \ n = 0–6 \)

The contents of the destination operand with the carry flag are rotated left one bit position. The initial value of bit 7 replaces the carry flag (C); the initial value of the carry flag replaces bit zero.

**Flags:**

- \( C: \) Set if the bit rotated from the most significant bit position (bit 7) was "1".
- \( Z: \) Set if the result is "0"; cleared otherwise.
- \( S: \) Set if the result bit 7 is set; cleared otherwise.
- \( V: \) Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
- \( D: \) Unaffected.
- \( H: \) Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{opc} )</td>
<td>( \text{dst} )</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>

**Examples:**

Given: Register 00H = 0AAH, register 01H = 02H, and register 02H = 17H, C = "0":

- RLC 00H \( \rightarrow \) Register 00H = 54H, C = "1"
- RLC @01H \( \rightarrow \) Register 01H = 02H, register 02H = 2EH, C = "0"

In the first example, if general register 00H has the value 0AAH (10101010B), the statement "RLC 00H" rotates 0AAH one bit position to the left. The initial value of bit 7 sets the carry flag and the initial value of the C flag replaces bit zero of register 00H, leaving the value 55H (01010101B). The MSB of register 00H resets the carry flag to "1" and sets the overflow flag.
**RR — Rotate Right**

**Operation:**

\[
C \leftarrow \text{dst} (0) \\
\text{dst} (7) \leftarrow \text{dst} (0) \\
\text{dst} (n) \leftarrow \text{dst} (n + 1), n = 0–6
\]

The contents of the destination operand are rotated right one bit position. The initial value of bit zero (LSB) is moved to bit 7 (MSB) and also replaces the carry flag (C).

**Flags:**

- **C:** Set if the bit rotated from the least significant bit position (bit zero) was "1".
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result bit 7 is set; cleared otherwise.
- **V:** Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>2 4 E0</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 E1</td>
<td>IR</td>
</tr>
</tbody>
</table>

**Examples:**

Given: Register 00H = 31H, register 01H = 02H, and register 02H = 17H:

- **RR 00H** → Register 00H = 98H, C = "1"
- **RR @01H** → Register 01H = 02H, register 02H = 8BH, C = "1"

In the first example, if general register 00H contains the value 31H (00110001B), the statement "RR 00H" rotates this value one bit position to the right. The initial value of bit zero is moved to bit 7, leaving the new value 98H (10011000B) in the destination register. The initial bit zero also resets the C flag to "1" and the sign flag and overflow flag are also set to "1".
**RRC — Rotate Right Through Carry**

**RRC**

dst

**Operation:**
dst (7) ← C
C ← dst (0)
dst (n) ← dst (n + 1), n = 0–6

The contents of the destination operand and the carry flag are rotated right one bit position. The initial value of bit zero (LSB) replaces the carry flag; the initial value of the carry flag replaces bit 7 (MSB).

**Flags:**
- **C:** Set if the bit rotated from the least significant bit position (bit zero) was "1".
- **Z:** Set if the result is "0" cleared otherwise.
- **S:** Set if the result bit 7 is set; cleared otherwise.
- **V:** Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>C0</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C1</td>
<td>IR</td>
</tr>
</tbody>
</table>

**Examples:**

Given: Register 00H = 55H, register 01H = 02H, register 02H = 17H, and C = "0":

RRC 00H → Register 00H = 2AH, C = "1"

RRC @01H → Register 01H = 02H, register 02H = 0BH, C = "1"

In the first example, if general register 00H contains the value 55H (01010101B), the statement "RRC 00H" rotates this value one bit position to the right. The initial value of bit zero ("1") replaces the carry flag and the initial value of the C flag ("1") replaces bit 7. This leaves the new value 2AH (00101010B) in destination register 00H. The sign flag and overflow flag are both cleared to "0".
SB0 — Select Bank 0

SB0

Operation: BANK ← 0

The SB0 instruction clears the bank address flag in the FLAGS register (FLAGS.0) to logic zero, selecting bank 0 register addressing in the set 1 area of the register file.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>1</td>
<td>4F</td>
</tr>
</tbody>
</table>

Example: The statement

SB0

clears FLAGS.0 to "0", selecting bank 0 register addressing.
SB1 — Select Bank 1

SB1

Operation: BANK ← 1

The SB1 instruction sets the bank address flag in the FLAGS register (FLAGS.0) to logic one, selecting bank 1 register addressing in the set 1 area of the register file. (Bank 1 is not implemented in some KS88-series microcontrollers.)

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5F</td>
</tr>
</tbody>
</table>

Example: The statement

SB1

sets FLAGS.0 to "1", selecting bank 1 register addressing, if implemented.
SBC — Subtract With Carry

SBC dst,src

Operation: dst ← dst – src – c

The source operand, along with the current value of the carry flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's-complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of the low-order operands to be subtracted from the subtraction of high-order operands.

Flags:

- C: Set if a borrow occurred (src > dst); cleared otherwise.
- Z: Set if the result is "0"; cleared otherwise.
- S: Set if the result is negative; cleared otherwise.
- V: Set if arithmetic overflow occurred, that is, if the operands were of opposite sign and the sign of the result is the same as the sign of the source; cleared otherwise.
- D: Always set to "1".
- H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise, indicating a "borrow".

Format:

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>r r</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>34</td>
</tr>
<tr>
<td>R R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>R IM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples: Given: R1 = 10H, R2 = 03H, C = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

- SBC R1,R2 → R1 = 0CH, R2 = 03H
- SBC R1,@R2 → R1 = 05H, R2 = 03H, register 03H = 0AH
- SBC 01H,02H → Register 01H = 1CH, register 02H = 03H
- SBC 01H,@02H → Register 01H = 15H, register 02H = 03H, register 03H = 0AH
- SBC 01H,#8AH → Register 01H = 95H; C, S, and V = "1"

In the first example, if working register R1 contains the value 10H and register R2 the value 03H, the statement "SBC R1,R2" subtracts the source value (03H) and the C flag value ("1") from the destination (10H) and then stores the result (0CH) in register R1.
SCF — Set Carry Flag

Operation: \( C \leftarrow 1 \)

The carry flag \( (C) \) is set to logic one, regardless of its previous value.

Flags: \( C: \) Set to "1".

No other flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DF</td>
</tr>
</tbody>
</table>

Example: The statement

SCF

sets the carry flag to logic one.
**SRA — Shift Right Arithmetic**

**SRA** dst

**Operation:**
- dst (7) ← dst (7)
- C ← dst (0)
- dst (n) ← dst (n + 1), n = 0–6

An arithmetic shift-right of one bit position is performed on the destination operand. Bit zero (the LSB) replaces the carry flag. The value of bit 7 (the sign bit) is unchanged and is shifted into bit position 6.

![Shift Right Arithmetic Diagram](attachment:image.png)

**Flags:**
- **C:** Set if the bit shifted from the LSB position (bit zero) was "1".
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result is negative; cleared otherwise.
- **V:** Always cleared to "0".
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Examples:**

Given: Register 00H = 9AH, register 02H = 03H, register 03H = 0BCH, and C = "1":

- **SRA 00H** → Register 00H = 0CD, C = "0"
- **SRA @02H** → Register 02H = 03H, register 03H = 0DEH, C = "0"

In the first example, if general register 00H contains the value 9AH (10011010B), the statement "SRA 00H" shifts the bit values in register 00H right one bit position. Bit zero ("0") clears the C flag and bit 7 ("1") is then shifted into the bit 6 position (bit 7 remains unchanged). This leaves the value 0CDH (11001101B) in destination register 00H.
SRP/SRP0/SRP1 — Set Register Pointer

SRP    src
SRP0   src
SRP1   src

Operation:
If src (1) = 1 and src (0) = 0 then: RP0 (3–7) ← src (3–7)
If src (1) = 0 and src (0) = 1 then: RP1 (3–7) ← src (3–7)
If src (1) = 0 and src (0) = 0 then: RP0 (4–7) ← src (4–7),
    RP0 (3) ← 0
    RP1 (4–7) ← src (4–7),
    RP1 (3) ← 1

The source data bits one and zero (LSB) determine whether to write one or both of the register pointers, RP0 and RP1. Bits 3–7 of the selected register pointer are written unless both register pointers are selected. RP0.3 is then cleared to logic zero and RP1.3 is set to logic one.

Flags:
No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>src</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Examples:
The statement

SRP  #40H

sets register pointer 0 (RP0) at location 0D6H to 40H and register pointer 1 (RP1) at location 0D7H to 48H.

The statement "SRP0  #50H" sets RP0 to 50H, and the statement "SRP1  #68H" sets RP1 to 68H.
STOP — Stop Operation

Operation:
The STOP instruction stops both the CPU clock and system clock and causes the microcontroller to enter Stop mode. During Stop mode, the contents of on-chip CPU registers, peripheral registers, and I/O port control and data registers are retained. Stop mode can be released by an external reset operation or by external interrupts. For the reset operation, the RESET pin must be held to Low level until the required oscillation stabilization interval has elapsed.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>7F</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Example:
The statement

```
STOP
NOP
NOP
NOP

```
halts all microcontroller operations.
SUB — Subtract

**SUB**
dst, src

**Operation:**
dst ← dst − src

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

**Flags:**
- **C:** Set if a "borrow" occurred; cleared otherwise.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result is negative; cleared otherwise.
- **V:** Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is the same as the sign of the source operand; cleared otherwise.
- **D:** Always set to "1".
- **H:** Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow".

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>4</td>
<td>22</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>23</td>
<td>r</td>
<td>Ir</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>6</td>
<td>24</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>25</td>
<td>R</td>
<td>IR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>6</td>
<td>26</td>
<td>R</td>
<td>IM</td>
</tr>
</tbody>
</table>

**Examples:**

Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

- **SUB** R1,R2 → R1 = 0FH, R2 = 03H
- **SUB** R1,@R2 → R1 = 08H, R2 = 03H
- **SUB** 01H,02H → Register 01H = 1EH, register 02H = 03H
- **SUB** 01H,@02H → Register 01H = 17H, register 02H = 03H
- **SUB** 01H,#90H → Register 01H = 91H; C, S, and V = "1"
- **SUB** 01H,#65H → Register 01H = 0BCH; C and S = "1", V = "0"

In the first example, if working register R1 contains the value 12H and if register R2 contains the value 03H, the statement "SUB R1,R2" subtracts the source value (03H) from the destination value (12H) and stores the result (0FH) in destination register R1.
SWAP — Swap Nibbles

**Operation:**
dst (0 – 3) ↔ dst (4 – 7)

The contents of the lower four bits and upper four bits of the destination operand are swapped.

![Diagram showing 8 bits: 7 4 3 0, with arrow indicating swap of lower four bits with upper four bits.]

**Flags:**
- **C:** Undefined.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result bit 7 is set; cleared otherwise.
- **V:** Undefined.
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>dst</td>
<td>2</td>
<td>4</td>
<td>F0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
<td>F1</td>
</tr>
</tbody>
</table>

**Examples:**

Given: Register 00H = 3EH, register 02H = 03H, and register 03H = 0A4H:

```
SWAP 00H → Register 00H = 0E3H
SWAP @02H → Register 02H = 03H, register 03H = 4AH
```

In the first example, if general register 00H contains the value 3EH (00111110B), the statement "SWAP 00H" swaps the lower and upper four bits (nibbles) in the 00H register, leaving the value 0E3H (11100110B).
TCM — Test Complement Under Mask

TCM dst,src

Operation: (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logic one value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

Flags:
C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Always cleared to "0".
D: Unaffected.
H: Unaffected.

Format:

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>62</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>r</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>65</td>
<td>R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>66</td>
</tr>
</tbody>
</table>

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 12H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

TCM R0,R1 → R0 = 0C7H, R1 = 02H, Z = "1"
TCM R0,@R1 → R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
TCM 00H,01H → Register 00H = 2BH, register 01H = 02H, Z = "1"
TCM 00H,@01H → Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "1"
TCM 00H,#34 → Register 00H = 2BH, Z = "0"

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TCM R0,R1" tests bit one in the destination register for a "1" value. Because the mask value corresponds to the test bit, the Z flag is set to logic one and can be tested to determine the result of the TCM operation.
**TM — Test Under Mask**

**Operation:**

This instruction tests selected bits in the destination operand for a logic zero value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

**Flags:**

- **C:** Unaffected.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result bit 7 is set; cleared otherwise.
- **V:** Always reset to "0".
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
<th>Addr Mode</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc dst</td>
<td>src 2 4</td>
<td>72</td>
<td>r r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opc dst</td>
<td>src 2 6</td>
<td>73</td>
<td>r Ir</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opc dst</td>
<td>src 3 6</td>
<td>74</td>
<td>R R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opc dst src</td>
<td>3 6</td>
<td>75</td>
<td>R IR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opc dst src</td>
<td>3 6</td>
<td>76</td>
<td>R IM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Examples:**

Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

- **TM R0,R1** → R0 = 0C7H, R1 = 02H, Z = "0"
- **TM R0,@R1** → R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
- **TM 00H,01H** → Register 00H = 2BH, register 01H = 02H, Z = "0"
- **TM 00H,@01H** → Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "0"
- **TM 00H,#54H** → Register 00H = 2BH, Z = "1"

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TM R0,R1" tests bit one in the destination register for a "0" value. Because the mask value does not match the test bit, the Z flag is cleared to logic zero and can be tested to determine the result of the TM operation.
WFI — Wait For Interrupt

WFI

Operation:
The CPU is effectively halted until an interrupt occurs, except that DMA transfers can still take place during this wait state. The WFI status can be released by an internal interrupt, including a fast interrupt.

Flags: No flags are affected.

Format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Opcode (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>opc</td>
<td>1</td>
<td>4n 3F</td>
</tr>
</tbody>
</table>

Example: The following sample program structure shows the sequence of operations that follow a "WFI" statement:

```
Main program

... (Next instruction)

WFI (Wait for interrupt)

... (Enable global interrupt)

Interrupt occurs

Interrupt service routine

... (Wait for interrupt)

Clear interrupt flag

IRET

Service routine completed
```
**XOR — Logical Exclusive OR**

**XOR**  

dst, src

**Operation:**  
dst ← dst XOR src

The source operand is logically exclusive-ORed with the destination operand and the result is stored in the destination. The exclusive-OR operation results in a "1" bit being stored whenever the corresponding bits in the operands are different; otherwise, a "0" bit is stored.

**Flags:**

- **C:** Unaffected.
- **Z:** Set if the result is "0"; cleared otherwise.
- **S:** Set if the result bit 7 is set; cleared otherwise.
- **V:** Always reset to "0".
- **D:** Unaffected.
- **H:** Unaffected.

**Format:**

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>B3</td>
<td>r</td>
<td>Ir</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>B4</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>B5</td>
<td>R</td>
<td>IR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opc</th>
<th>dst</th>
<th>src</th>
</tr>
</thead>
<tbody>
<tr>
<td>B6</td>
<td>R</td>
<td>IM</td>
</tr>
</tbody>
</table>

**Examples:**  

Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

- XOR R0,R1 → R0 = 0C5H, R1 = 02H
- XOR R0,@R1 → R0 = 0E4H, R1 = 02H, register 02H = 23H
- XOR 00H,01H → Register 00H = 29H, register 01H = 02H
- XOR 00H,@01H → Register 00H = 08H, register 01H = 02H, register 02H = 23H
- XOR 00H,#54H → Register 00H = 7FH

In the first example, if working register R0 contains the value 0C7H and if register R1 contains the value 02H, the statement "XOR R0,R1" logically exclusive-ORs the R1 value with the R0 value and stores the result (0C5H) in the destination register R0.
NOTES
Clock Circuit

RESET and Power-Down

I/O Ports

Basic Timer

Timer M0

Timer M1

Timer M2

Analog-to-Digital Converter

Pulse Width Modulation

Sync Processor

DDC and IIC-Bus Interface

Slave IIC-Bus Interface

Electrical Data

Mechanical Data

S3P863A OTP

Development Tools
OVERVIEW

The clock frequency generated for S3C8639/C863A/C8647 by an external crystal ranges from 8 MHz to 12 MHz. The maximum CPU clock frequency is 12 MHz. The X\text{IN} and X\text{OUT} pins connect the external oscillator or clock source to the on-chip clock circuit.

SYSTEM CLOCK CIRCUIT

The system clock circuit has the following components:

— External crystal or ceramic resonator oscillation source (or an external clock source)
— Oscillator stop and wake-up functions
— Programmable frequency divider for the CPU clock ($f_{\text{OSC}}$ divided by 1, 2, 8, or 16)
— System clock control register, CLKCON

Figure 7-1. Main Oscillator Circuit
(External Crystal or Ceramic Resonator)
CLOCK STATUS DURING POWER-DOWN MODES

The two power-down modes, Stop mode and Idle mode, affect the system clock as follows:

— In Stop mode, the main oscillator is halted. Stop mode is released and the oscillator is started by a reset operation or an external interrupt (with RC delay noise filter).

— In Idle mode, the internal clock signal is gated to the CPU, but not to interrupt structure, timers and timer/counters, and the IIC-bus interface functions. Idle mode is released by a reset or by an external or internal interrupt.

**Figure 7-2. System Clock Circuit Diagram**

**NOTES:**
1. An external interrupt (with RC-delay noise filter) can be used to release Stop mode and "wake up" the main oscillator. In S3C8639/C863A/C8647, the P0.0-P0.2 and external interrupts are of this type.
2. For S3C8639/C863A/C8647, the CLKCON signature code (CLKCON.0-CLKCON.2) should not be "101B" (because no subsystem clock is implemented).
SYSTEM CLOCK CONTROL REGISTER (CLKCON)

The system clock control register, CLKCON, is located in set 1, address D4H. It is read/write addressable and has the following functions:

- Oscillator IRQ wake-up function enable/disable
- Main oscillator stop control
- Oscillator frequency divide-by value
- System clock signal selection

The CLKCON register controls whether or not an external interrupt can be used to trigger a power down mode release. (This is called the "IRQ wake-up" function.) The IRQ wake-up enable bit is CLKCON.7.

After a reset, the external interrupt oscillator wake-up function is enabled, the main oscillator is activated, and the $f_{\text{OSC}}/16$ (the slowest clock speed) is selected as the CPU clock. If necessary, you can raise the CPU clock speed to $f_{\text{OSC}}$, $f_{\text{OSC}}/2$, or $f_{\text{OSC}}/8$.

For the S3C8639/C863A/C8647 microcontrollers, the CLKCON.2–CLKCON.0 system clock signature code must be any value other than "101B". (The "101B" setting is invalid because a subsystem clock is not implemented.) The reset value for the clock signature code is "000B" and should remain so during the normal operation.

<table>
<thead>
<tr>
<th>System Clock Control Register (CLKCON)</th>
<th>D4H, Set 1, R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>.7 .6 .5 .4 .3 .2 .1 .0</td>
</tr>
</tbody>
</table>

**Oscillator IRQ wake-up enable bit:**
- 0 = Enable IRQ for main system oscillator wake-up function in power down mode
- 1 = Disable IRQ for main system oscillator wake-up function in power down mode

**System clock selection bits:**
- 101B = Invalid selection
- Others = Normal operating mode

**Divide-by selection bits for CPU clock frequency:**
- 00 = $f_{\text{OSC}}/16$
- 01 = $f_{\text{OSC}}/8$
- 10 = $f_{\text{OSC}}/2$
- 11 = $f_{\text{OSC}}$ (non-divided)

**Main oscillator stop control bits:**
- 00 = No effect
- 01 = No effect
- 10 = stop main oscillator
- 11 = No effect

![Figure 7-3. System Clock Control Register (CLKCON)](image-url)
NOTES
8

RESET and POWER-DOWN

SYSTEM RESET

OVERVIEW

During a power-on reset, the voltage at $V_{DD}$ goes to High level and the RESET pin is forced to Low level. The RESET signal is input through a schmitt trigger circuit where it is then synchronized with the CPU clock. This procedure brings S3C8639/C863A/C8647 into a known operating status.

To spare time for internal CPU clock oscillation to stabilize, the RESET pin must be held to Low level for a minimum time interval after the power supply comes within tolerance. The minimum required time for oscillation stabilization in a reset operation is 1 millisecond.

Whenever a reset occurs during the normal operation (that is, when both $V_{DD}$ and RESET are at High level), the RESET pin is forced Low and the reset operation starts. All system and peripheral control registers are then reset to their default hardware values (see Tables 8-1, 8-2, and 8-3).

In summary, the following sequence of events occurs during a reset operation:

— All interrupts are disabled.
— The watchdog function (basic timer) is enabled.
— Ports 0–3 are set to input mode.
— Peripheral control and data registers are disabled and reset to their default hardware values.
— The program counter (PC) is loaded with the program reset address in the ROM, 0100H.
— When the programmed oscillation stabilization time interval has elapsed, the instruction stored in the ROM location 0100H (and 0101H) is fetched and executed.

NOTE

To program the duration of the oscillation stabilization interval, you should make the settings appropriate to the basic timer control register, BTCON, before entering Stop mode. Also, if you do not want to use the basic timer watchdog function (which causes a system reset if a basic timer counter overflow occurs), you can disable it by writing “1010B” to the upper nibble of BTCON.
HARDWARE RESET VALUES

Tables 8-1, 8-2, and 8-3 list the reset values for CPU and system registers, peripheral control registers, and peripheral data registers after a reset operation. The following notation is used to represent reset values:

— A "1" or a "0" shows the reset bit value as logic one or logic zero, respectively.
— An "x" means that the bit value is undefined after a reset.
— A dash ("–") means that the bit is either not used or not mapped.

### Table 8-1. Set 1 Register Values After Reset

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Mnemonic</th>
<th>Address</th>
<th>Bit Values After Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer M0 counter register</td>
<td>TM0CNT</td>
<td>208 D0H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Timer M0 data register</td>
<td>TM0DATA</td>
<td>209 D1H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Timer M0 control register</td>
<td>TM0CON</td>
<td>210 D2H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Basic timer control register</td>
<td>BTCON</td>
<td>211 D3H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Clock control register</td>
<td>CLKCON</td>
<td>212 D4H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>System flags register</td>
<td>FLAGS</td>
<td>213 D5H</td>
<td>x x x x x x 0 0</td>
</tr>
<tr>
<td>Register pointer 0</td>
<td>RP0</td>
<td>214 D6H</td>
<td>1 1 0 0 0 0 – – –</td>
</tr>
<tr>
<td>Register pointer 1</td>
<td>RP1</td>
<td>215 D7H</td>
<td>1 1 0 0 1 – – –</td>
</tr>
<tr>
<td>Stack pointer (high byte)</td>
<td>SPH</td>
<td>216 D8H</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>Stack pointer (low byte)</td>
<td>SPL</td>
<td>217 D9H</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>Instruction pointer (high byte)</td>
<td>IPH</td>
<td>218 DAH</td>
<td>x x x x x x x x x x x x</td>
</tr>
<tr>
<td>Instruction pointer (low byte)</td>
<td>IPL</td>
<td>219 DBH</td>
<td>x x x x x x x x x x x x</td>
</tr>
<tr>
<td>Interrupt request register</td>
<td>IRQ</td>
<td>220 DCH</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Interrupt mask register</td>
<td>IMR</td>
<td>221 DDH</td>
<td>x x x x x x x x x x x x</td>
</tr>
<tr>
<td>System mode register</td>
<td>SYM</td>
<td>222 DEH</td>
<td>0 – – x x x 0 0</td>
</tr>
<tr>
<td>Page pointer register</td>
<td>PP</td>
<td>223 DFH</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

NOTES:
1. As the SYM register is not used for S3C8639/C863A/C8647, SYM.5 should always be "0". If you accidentally write a “1” to this bit during the normal operation, a system malfunction may occur.
2. Except for TM0CNT, TM0DATA, and IRQ, all registers in set 1 are read/write addressable.
3. You cannot use a read-only register as a destination field for the instructions OR, AND, LD, and LDB. The read-only registers in the S3C8639/C863A/C8647 register file are: TM0CNT, TM0DATA, IRQ, SYNCRD, TM1CNTH, TM1CNTL, TM1DATAH, TM1DATAL, ADDATA, BTCNT, PWMCNT, and RBDR.
4. Interrupt pending flags that must be cleared by software are noted by shaded table cells.
## Table 8-2. Set 1, Bank 0 Register Values after Reset

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Mnemonic</th>
<th>Address</th>
<th>Bit Values After Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 0 data register</td>
<td>P0</td>
<td>224 E0H</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 1 data register <em>(note)</em></td>
<td>P1</td>
<td>225 E1H</td>
<td>– – – – – – 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 2 data register</td>
<td>P2</td>
<td>226 E2H</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 3 data register</td>
<td>P3</td>
<td>227 E3H</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 0 control register (high byte)</td>
<td>P0CONH</td>
<td>228 E4H</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 0 control register (low byte)</td>
<td>P0CONL</td>
<td>229 E5H</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 1 control register <em>(note)</em></td>
<td>P1CON</td>
<td>230 E6H</td>
<td>– – 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 2 control register (high byte)</td>
<td>P2CONH</td>
<td>231 E7H</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 2 control register (low byte)</td>
<td>P2CONL</td>
<td>232 E8H</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 3 control register (high byte)</td>
<td>P3CONH</td>
<td>233 E9H</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 3 control register (low byte)</td>
<td>P3CONL</td>
<td>234 EAH</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Port 0 external interrupt control register</td>
<td>P0INT</td>
<td>235 EBH</td>
<td>– 0 0 0 – 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Watchdog time control register</td>
<td>WDTCON</td>
<td>236 ECH</td>
<td>– – – – 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Sync control register 0</td>
<td>SYNCON0</td>
<td>237 EDH</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Sync control register 1</td>
<td>SYNCON1</td>
<td>238 EEH</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Sync control register 2</td>
<td>SYNCON2</td>
<td>239 EFH</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Sync port read data register</td>
<td>SYNCRD</td>
<td>240 F0H</td>
<td>– – – – 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

**NOTE:** Not used for the S3C8647.
Table 8-2. Set 1, Bank 0 Register Values after Reset (Continued)

| Register Name           | Mnemonic | Address | Bit Values After Reset | Dec | Hex | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------------------|----------|---------|-------------------------|-----|-----|---|---|---|---|---|---|---|---|---|
| Timer M1 counter register high | TM1CNTH  | 241     | F1H – – – – 0 0 0 0     |     |     |   |   |   |   |   |   |   |   |
| Timer M1 counter register low  | TM1CNTL  | 242     | F2H 0 0 0 0 0 0 0 0     |     |     |   |   |   |   |   |   |   |   |
| Timer M1 data register high   | TM1DATAH | 243     | F3H – – – – 0 0 0 0     |     |     |   |   |   |   |   |   |   |   |
| Timer M1 data register low    | TM1DATAL | 244     | F4H 0 0 0 0 0 0 0 0     |     |     |   |   |   |   |   |   |   |   |
| Timer M1 control register    | TM1CON   | 245     | F5H 0 0 0 0 0 0 0 0     |     |     |   |   |   |   |   |   |   |   |
| Timer M2 control register    | TM2CON   | 246     | F6H 1 1 1 1 1 0 0 0     |     |     |   |   |   |   |   |   |   |   |
| A/D converter control register | ADCON   | 247     | F7H – 0 0 0 0 0 0 0     |     |     |   |   |   |   |   |   |   |   |
| A/D converter data register  | ADDATA   | 248     | F8H x x x x x(4) x(4) x(4) x(4) |     |     |   |   |   |   |   |   |   |   |
| Pseudo Hsync generation register | PHGEN  | 249     | F9H 0 1 0 1 0 0 1 1     |     |     |   |   |   |   |   |   |   |   |
| Pseudo Vsync generation register | PVGEN  | 250     | FAH 0 1 0 1 0 0 1 1     |     |     |   |   |   |   |   |   |   |   |
| Stop control register        | STOPCON  | 251     | FBH 0 0 0 0 0 0 0 0     |     |     |   |   |   |   |   |   |   |   |

Location FCH is not mapped.

Notes:
1. Except for SYNCRD, TM1CNTH, TM1CNTL, TM1DATAH, TM1DATAL, ADDATA, and BTCNT, all registers in set 1, bank 0 are read/write addressable.
2. You cannot use a read-only register as a destination field for the instructions OR, AND, LD, and LDB. The read-only registers in the S3C8639/C863A/C8647 register file are: TM0CNT, TM0DATA, IRQ, SYNCRD, TM1CNTH, TM1CNTL, TM1DATAH, TM1DATAL, ADDATA, BTCNT, PWMCNT, and RBDR.
3. Interrupt pending flags that must be cleared by software are noted by shaded table cells.
4. Not mapped for the S3C8647.
### Table 8-3. Set 1, Bank 1 Register Values after Reset

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Mnemonic</th>
<th>Address Dec</th>
<th>Address Hex</th>
<th>Bit Values After Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM 0 data register</td>
<td>PWM0</td>
<td>224</td>
<td>E0H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>PWM 1 data register</td>
<td>PWM1</td>
<td>225</td>
<td>E1H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>PWM 2 data register</td>
<td>PWM2</td>
<td>226</td>
<td>E2H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>PWM 3 data register</td>
<td>PWM3</td>
<td>227</td>
<td>E3H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>PWM 4 data register</td>
<td>PWM4</td>
<td>228</td>
<td>E4H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>PWM 5 data register</td>
<td>PWM5</td>
<td>229</td>
<td>E5H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>PWM 6 data register (4)</td>
<td>PWM6</td>
<td>230</td>
<td>E6H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>PWM control register</td>
<td>PWMCON</td>
<td>231</td>
<td>E7H</td>
<td>0 0 0 – – – – –</td>
</tr>
<tr>
<td>PWM counter register</td>
<td>PWMCNT</td>
<td>232</td>
<td>E8H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>DDC control register</td>
<td>DCON</td>
<td>233</td>
<td>E9H</td>
<td>– – – – 1 0 0 0</td>
</tr>
<tr>
<td>DDC address register 0</td>
<td>DAR0</td>
<td>234</td>
<td>EAH</td>
<td>1 0 1 0 – – – –</td>
</tr>
<tr>
<td>DDC clock control register</td>
<td>DCCR</td>
<td>235</td>
<td>EBH</td>
<td>0 0 0 0 1 1 1 1</td>
</tr>
<tr>
<td>DDC control/status register 0</td>
<td>DCSR0</td>
<td>236</td>
<td>ECH</td>
<td>0 0 0 0 0 0 0 –</td>
</tr>
<tr>
<td>DDC control/status register 1</td>
<td>DCSR1</td>
<td>237</td>
<td>EDH</td>
<td>– – – – – 0 1 0</td>
</tr>
<tr>
<td>DDC address register 1</td>
<td>DAR1</td>
<td>238</td>
<td>EEH</td>
<td>x x x x x x x x –</td>
</tr>
<tr>
<td>Transmit prebuffer data register</td>
<td>TBDR</td>
<td>239</td>
<td>EFH</td>
<td>x x x x x x x x x</td>
</tr>
<tr>
<td>Receive prebuffer data register</td>
<td>RBDR</td>
<td>240</td>
<td>F0H</td>
<td>x x x x x x x x x</td>
</tr>
<tr>
<td>DDC data shift register</td>
<td>DDSR</td>
<td>241</td>
<td>F1H</td>
<td>x x x x x x x x x</td>
</tr>
<tr>
<td>Slave IIC-bus control/status register (4)</td>
<td>SICSR</td>
<td>242</td>
<td>F2H</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Slave IIC-bus address register (4)</td>
<td>SIAR</td>
<td>243</td>
<td>F3H</td>
<td>x x x x x x x x –</td>
</tr>
<tr>
<td>Slave IIC-bus data shift register (4)</td>
<td>SIDS</td>
<td>244</td>
<td>F4H</td>
<td>x x x x x x x x x</td>
</tr>
</tbody>
</table>

Locations F5H–FFH are not mapped.

**NOTES:**

1. Except for PWMCNT and RBDR, all registers in set 1, bank 1 are read/write addressable.
2. You cannot use a read-only register as a destination field for the instructions OR, AND, LD, and LDB. The read-only registers in the S3C8639/C863A/C8647 register file are: TM0CNT, TM0DATA, IRQ, SYNCRD, TM1CNTL, TM1DATAH, TM1DATAL, ADDATA, BTCNT, PWMCNT, and RBDR.
3. Interrupt pending flags that must be cleared by software are noted by shaded table cells.
4. Not used for the S3C8647.
POWER-DOWN MODES

STOP MODE

Stop mode is invoked by the instruction STOP (opcode 7FH) and the stop control register (STOPCON). In Stop mode, the operation of the CPU and all peripherals is halted. That is, the on-chip main oscillator stops and the supply current is reduced to less than 5 \( \mu \)A. All system functions stop when the clock "freezes," but data stored in the internal register file is retained. Stop mode can be released in one of two ways: by a reset or by an external interrupt (with RC delay).

**NOTE**

Do not use stop mode if you are using an external clock source as \( X_{IN} \) input must be restricted internally to \( V_{SS} \) to reduce current leakage.

**Using RESET to Release Stop Mode**

Stop mode is released when the RESET signal goes active (High level): all system and peripheral control registers are reset to their default hardware values and the contents of all data registers are retained. A reset operation automatically selects a slow clock (1/16) because CLKCON.3 and CLKCON.4 are cleared to "00B". After the programmed oscillation stabilization interval has elapsed, the CPU starts the system initialization routine by fetching the program instruction stored in the ROM location 0100H (and 0101H).

**Using an External Interrupt to Release Stop Mode**

Only external interrupts with an RC-delay noise filter circuit can be used to release Stop mode. Which interrupt you can use to release Stop mode in a given situation depends on the microcontroller's current internal operating mode. The external interrupts in the S3C8639/C863A/C8647 interrupt structure that can be used to release Stop mode are:

- External interrupts P0.0 (INT0), P0.1 (INT1), and P0.2 (INT2)
- Timer M0 capture interrupt in capture mode (with rising or falling edge trigger at the TM0CAP pin and Vsync-O from sync-processor.)

Please note the following conditions for Stop mode release:

- If you release Stop mode using an external interrupt, the current values in system and peripheral control registers are unchanged.
- If you use an external interrupt for Stop mode release, you can also program the duration of the oscillation stabilization interval. To do this, you must make the control and clock settings appropriate before entering Stop mode.
- If you use an interrupt to release Stop mode, the CLKCON.4 and CLKCON.3 bit-pair setting remains unchanged and the currently selected clock value is used.
- The external interrupt is serviced when the Stop mode release occurs. Following the IRET from the service routine, the instruction right next to the one that initiated Stop mode is executed.
IDLE MODE

Idle mode is invoked by the instruction IDLE (opcode 6FH). In Idle mode, CPU operations are halted while some peripherals remain active. In idle mode, the internal clock signal is gated away from the CPU, but all peripherals timers remain active. Port pins retain the mode (input or output) they had at the time Idle mode was entered.

There are two ways to release idle mode:

1. Execute a reset. All system and peripheral control registers are reset to their default values and the contents of all data registers are retained. The reset automatically selects a slow clock (1/16) because CLKCON.4 and CLKCON.3 are cleared to "00B". If interrupts are masked, a reset is the only way to release Idle mode.

2. Activate any enabled interrupt, causing Idle mode to be released. When you use an interrupt to release Idle mode, the CLKCON.4 and CLKCON.3 register values remain unchanged, and the currently selected clock value is used. The interrupt is then serviced. When the return-from-interrupt (IRET) occurs, the instruction right next to the one that initiated Idle mode is executed.

NOTE

Only external interrupts can be used to release Stop mode. To release idle mode, you can use either an internally-generated or externally-generated interrupt.
PROGRAMMING TIP — Sample S3C8639/C863A/C8647 Initialization Routine

The following sample program shows you how to make initial settings for the S3C8639/C863A/C8647 address space, interrupt vectors, and peripheral functions. Program comments guide you through the steps:

; << Base Number Setting >>
DECIMAL

; << Definition >>
TM0_REG EQU 40H
ORG 0000H

; << Interrupt Vector Addresses >>
ORG 00ECH
VECTOR TM0_OVF_INT ; IRQ0
VECTOR TM0_CAP_INT ; IRQ0
VECTOR TM2_INT ; IRQ1
VECTOR TM1_OVF_INT ; IRQ2
VECTOR TM1_CAP_INT ; IRQ2
VECTOR DDC_INT ; IRQ3
VECTOR P00_INT ; IRQ4
VECTOR P01_INT ; IRQ5
VECTOR P02_INT ; IRQ6
VECTOR SIIC_INT ; IRQ7 (used only S3C863X)

; << Initialize System and Peripherals >>
ORG 0100H ; Reset address
LD BTCON,#0A0H ; Disable watchdog timer
LD CLKCON,#10H ; Select divided-by-two oscillator frequency as CPU clock
; Enable IRQ for main system oscillator wake-up

; < System Register Settings >
CLR SYM ; Disable fast interrupts; global interrupt disable
CLR EMT ; No access wait time; select internal stack area
LD SPH,#00H ; Set stack pointer (stack starts from #0FFH)

; < Interrupt Settings >
LD IPR,#8FH ; Set interrupt priorities as follows:
; IRQ3 > IRQ2 > IRQ1 > IRQ0
LD IMR,#0FH ; Enable IRQ levels 0, 1, 2, and 3

; < Timer M0 Settings >
LD TM0CON,#8FH ; Enable timer M0 overflow and capture interrupts
PROGRAMMING TIP — Sample S3C8639/C863A/C8647 Initialization Routine (Continued)

INI_PERI_SET:

SB0 ; Select bank 0
LD P0CONH,#0FFH ; Set port 0 high byte to push-pull output mode
LD P0CONL,#0FFH ; Set port 0 low byte to push-pull output mode
LD P0INT,#00H ; Disable P0.0, P0.1 and P0.2 external interrupts

LD P1CON,#00H ; Set P1.0–P1.2 to input mode
LD P2CONH,#0FFH ; Set port 2 high byte to n-channel open-drain PWM output mode
LD P2CONL,#0FFH ; Set port 2 low byte to push-pull PWM output mode
LD P3CONH,#0AAH ; Set port 3 high byte to push-pull output mode
LD P3CONL,#0AAH ; Set port 3 low byte to push-pull output mode

; < Timer M1 Settings >
LD TM1CON,#2CH ; Enable timer M1 capture and overflow interrupt, Timer M1 clock source is HsyncI from sync processor

; < Timer M2 Settings >
LD TM2CON,#3DH ; Enable timer M2 capture and overflow interrupt

; < Sync Processor Settings >
LD SYNCON0,#20H ; 5 bit counter capture mode
LD SYNCON1,#80H ; Set negative polarity (500 ns at 8 MHz) for clampO
LD SYNCON2,#0A0H ; Pseudo sync output

; < PWM Settings >
SB1 ; Select Bank 1
LD PWMCON,#20H ; Start PWM counter, PWM counter clock is fOSC

; < DDC Tx/Rx Interface Settings >
LD DCON,#0AH ; Select DDC1 Tx mode
LD DCCR,#0A3H ; Enable DDC interrupt, DDC clock is 100 kHz

; << Initialize Data Registers >>
SB0 ; Select bank 0
SRP #0C0H ; Set register pointer

; < Clear all data registers from 00H to FFH >
LD R0,#0FFH ; Enable timer M2 interrupt
RAMCLR: CLR @R0
DJNZ R0,RAMCLR ; Page 0 RAM clear
PROGRAMMING TIP — Sample S3C8639/C863A/C8647 Initialization Routine (Continued)

; < Initialize Other Registers >

; EI ; You must execute an EI instruction in this position
; in the initialization routine to enable servicing of
; external interrupts

; << Main Loop >>

MAIN: NOP ; Start main loop

CALL KEY_SCAN ; Sub-program module

CALL LED_DISPLAY ; Sub-program module

CALL JOB ; Sub-program module

JR t,MAIN ; For main loop

; < Subroutines >

KEY_SCAN:
  NOP
  ...
  RET

LED_DISPLAY:
  NOP
  ...
  RET

JOB:
  NOP
  ...
  RET
PROGRAMMING TIP — Sample S3C8639/C863A/C8647 Initialization Routine (Concluded)

; << Interrupt Service Routines >>

P00_INT:
PUSH RP0 ; Save old RP0 value
SRP0 #60H ; Set RP0 for P0.0 interrupt service routine
•
•
•
POP RP0 ; Restore the RP0 value
IRET ; Return from the interrupt

DDC_INT:
PUSH RP0 ; Save old RP0 value
SRP0 #50H ; Set RP0 for IIC-bus interrupt service routine
•
•
•
SB1
AND DDCR, #11101111B ; Clear DDC interrupt pending bit
SB0
POP RP0 ; Restore the RP0 value
IRET ; Return from the interrupt

TM0_CAP_INT:
PUSH RP0 ; Save old RP0 value
SRP #TM0_REG ; TM0_REG value should be defined
•
•
•
POP RP0 ; Restore the RP0 value
IRET ; Return from the interrupt
•
•
END
I/O PORTS

OVERVIEW

The S3C8639/C863A/C8647 microcontrollers have four I/O ports with a total of 27 pins. And the S3C8647 microcontroller has three I/O port 0 with a total of 19 pins. Each port can be flexibly configured to meet application design requirements. The CPU accesses ports by directly writing or reading port registers. No special I/O instructions are required. Table 9-1 gives you an overview of port functions:

<table>
<thead>
<tr>
<th>Port</th>
<th>Configuration Options</th>
<th>Programmability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8-bit general I/O port. Alternatively used for external interrupt inputs and for timer M0 input function.</td>
<td>Bit programmable</td>
</tr>
<tr>
<td>1</td>
<td>3-bit I/O port for normal I/O or n-channel open drain output. Alternatively used for IIC-bus clock and data I/O.</td>
<td>Bit programmable</td>
</tr>
<tr>
<td>2</td>
<td>8-bit I/O port for normal I/O, PWM push-pull outputs, PWM n-channel open-drain outputs with 5-volt load capability, or Csync signal input.</td>
<td>Bit programmable</td>
</tr>
<tr>
<td>3</td>
<td>8-bit general I/O port. Alternatively used as n-channel open-drain, push-pull outputs with 5-volt load capability or for normal input with pull-up resistor. Multiplexed for alternative use as A/D converter inputs, AD0–AD3.</td>
<td>Bit programmable</td>
</tr>
</tbody>
</table>
PORT DATA REGISTERS

Data registers for ports 0–3 have the format shown in Figure 9-1. Table 9-2 gives you an overview of the port data register locations:

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Mnemonic</th>
<th>Decimal</th>
<th>Hex</th>
<th>Location</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 0 data register</td>
<td>P0</td>
<td>224</td>
<td>E0H</td>
<td>Set 1, bank 0</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 1 data register</td>
<td>P1</td>
<td>225</td>
<td>E1H</td>
<td>Set 1, bank 0</td>
<td>R/W</td>
</tr>
<tr>
<td>(only S3C863X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port 2 data register</td>
<td>P2</td>
<td>226</td>
<td>E2H</td>
<td>Set 1, bank 0</td>
<td>R/W</td>
</tr>
<tr>
<td>Port 3 data register</td>
<td>P3</td>
<td>227</td>
<td>E3H</td>
<td>Set 1, bank 0</td>
<td>R/W</td>
</tr>
</tbody>
</table>

NOTE: Port 1 is a 3-bit port. Only bits P1.2–P1.0 of the port 1 data register are mapped. All the other S3C8639/C863A I/O ports are 8-bit.

Figure 9-1. Port Data Register Format

PORT 0

Port 0 is an 8-bit I/O port with individually configurable pins. You can directly access port 0 pins by writing or reading the port 0 data register, P0 (set 1, bank 0, E0H). You can use port 0 for general I/O, or for the following alternative functions:

— Low-byte pins (P0.3–P0.0) can be configured as push-pull outputs, while P0.2–P0.0 as a multiplexed input pins for external interrupts INT2–INT0 with rising or falling edge detection.
— High-byte pins (P0.7–P0.4) can be configured as multiplexed inputs and push-pull outputs. P0.4 can serve as the timer M0 capture input pin (TM0CAP).

Two 8-bit control registers are used to configure port 0 pins: P0CONH (set 1, bank 0, E4H) for P0.7–P0.4 and P0CONL (set 1, bank 0, E5H) for P0.3–P0.0. Each byte contains four bit-pairs and each bit-pair configures one pin. The low-byte port 0 control register, P0CONL, is also used to enable and disable the external interrupts, INT2–INT0, at pins P0.2–P0.0, respectively.
Port 0 High-Byte Control Register (P0CONH)
The four bit-pairs in the port 0 high-byte control register, P0CONH, have the following functions:

— To configure individual port 0 pins to multiplexed input mode or push-pull output mode.
— To configure alternative input or output functions for P0.7–P0.4.

Bit-pair 1/0 configures the capture signal input pin for timer M0 at P0.4.

![Port 0 Control Register, High Byte (P0CONH)](image)

**NOTE:** Not used for the S3C8647.
Port 0 Low-Byte Control Register (P0CONL)

The low-byte port 0 pins, P0.3–P0.0 can be configured individually as inputs or as push-pull outputs. You can alternatively configure the pins P0.2–P0.0 as external interrupt inputs with rising or falling edge detection.

Figure 9-3. Port 0 Low-Byte Control Register (P0CONL)
Port 0 External Interrupt Control Register (P0INT)

The port 0 external interrupt control register, P0INT, is used to enable and disable the external interrupts INT2–INT0 at P0.2–P0.0, respectively, and also to detect and clear external interrupt pending conditions at these pins.

To selectively enable the external interrupts INT0, INT1, and INT2, you set P0INT.0, P0INT.1, and P0INT.2 to “1”, respectively. The application program can poll the corresponding interrupt pending bits — P0INT.4 for INT0, P0INT.5 for INT1, and P0INT.6 for INT2 — to detect external interrupt pending conditions.

After an external interrupt has been serviced, the service routine must clear the pending condition by writing a “0” to the appropriate pending bit. Writing a “1” to the pending bit has no effect.

![Port 0 External Interrupt Control Register (P0INT)](image)

Figure 9-4. Port 0 External Interrupt Control Register (P0INT)
PORT 1 (Only S3C863X)

Port 1 is an 3-bit port with individually configurable pins. You can directly access it by writing or reading the port 1 data register, P1 (set 1, bank 0, E1H). You can use port 1 for normal output, input mode, or n-channel open-drain output mode.

The port 1 control register, P1CON (set 1, bank 0, E6H) is used to configure port 1 pins. Each byte contains four bit-pairs and each bit-pair configures one pin.

Bit pair 3/2 configures the IIC-bus clock pin for SCL1 at P1.1. Bit pair 1/0 controls P1.0 when it is set to “11B”, the SDA1 is enabled for IIC-bus data pin.

**Port 1 Control Register (P1CON)**

<table>
<thead>
<tr>
<th></th>
<th>E6H, Set 1, Bank 0, R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>.7</td>
<td>0</td>
</tr>
<tr>
<td>.6</td>
<td>0</td>
</tr>
<tr>
<td>.5</td>
<td>0</td>
</tr>
<tr>
<td>.4</td>
<td>0</td>
</tr>
<tr>
<td>.3</td>
<td>0</td>
</tr>
<tr>
<td>.2</td>
<td>0</td>
</tr>
<tr>
<td>.1</td>
<td>0</td>
</tr>
<tr>
<td>.0</td>
<td>0</td>
</tr>
</tbody>
</table>

Not used for S3C8639/C863A/C8647

P1CON Pin Configuration Settings:

- 00: Input mode
- 01: Push-pull output mode
- 10: N-channel open-drain output mode
  (5 V load capability)
- 11: Multiplexed mode (SCL1/SDA1)

**Figure 9-5. Port 1 Control Register (P1CON)**
Port 2 is an 8-bit I/O port with individually configurable pins. You can directly access port 2 pins by writing or reading the port 2 data register, P2 (set 1, bank 0, E2H).

Two 8-bit control registers are used to configure port 2 pins: P2CONH (set 1, bank 0, E7H) which let you select digital input mode (or TTL input mode), normal or PWM push-pull output mode, or n-channel open drain PWM output mode. And you can select digital input mode, normal or PWM push-pull output mode at the P2CONL (set 1, bank 0, E8H).

**Figure 9-6. Port 2 High-Byte Control Register (P2CONH)**

<table>
<thead>
<tr>
<th>bits 5-0</th>
<th>00</th>
<th>Input mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01</td>
<td>Push-pull output mode</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Push-pull PWM output mode (5 V load capability)</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>N-channel open-drain PWM output mode (5 V load capability)</td>
</tr>
</tbody>
</table>

**NOTE:** Not used for the S3C8647.
Port 2 Control Register, Low Byte (P2CONL)
E8H, Set 1, Bank 0, R/W

<table>
<thead>
<tr>
<th>MSB</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P2.1/PWM1</td>
<td></td>
<td></td>
<td>P2.0/PWM0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P2.2/PWM2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P2.3/PWM3</td>
</tr>
</tbody>
</table>

P2CONL Pin Configuration Settings:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x</td>
<td>Input mode</td>
</tr>
<tr>
<td>10</td>
<td>Push-pull output mode</td>
</tr>
<tr>
<td>11</td>
<td>Push-pull PWM output mode (5 V load capability)</td>
</tr>
</tbody>
</table>

Figure 9-7. Port 2 Low-Byte Control Register (P2CONL)
PORT 3

Port 3 is an 8-bit I/O port with individually configurable pins. You can directly access it by writing or reading the port 3 data register, P3 (set 1, bank 0, E3H). You can selectively configure P3 pins to input or output mode. In input mode, you can also select A/D converter input mode (P3.0–P3.3 only) or normal digital input mode (with or without pull-up resistor). Output mode is push-pull mode or n-channel open-drain mode (P3.4–P3.7 only).

Two 8-bit control registers are used to configure port 3 pins: P3CONH (E9H, set 1, bank 0) for P3.7–P3.4 and P3CONL (set 1, bank 0, EAH) for P3.3–P3.0. Each byte contains four bit-pairs and each bit-pair configures one pin.

### Port 3 Control Register, High Byte (P3CONH)

**E9H, Set 1, Bank 0, R/W**

<table>
<thead>
<tr>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.7</td>
<td>P3.6</td>
<td>P3.5</td>
<td>P3.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**P3CONH Pin Configuration Settings:**

<table>
<thead>
<tr>
<th>Bit-Pair</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Input mode</td>
</tr>
<tr>
<td>01</td>
<td>Input mode with pull-up resistor</td>
</tr>
<tr>
<td>10</td>
<td>Push-pull output mode</td>
</tr>
<tr>
<td>11</td>
<td>N-channel open-drain output mode (5 V load capability)</td>
</tr>
</tbody>
</table>

**Figure 9-8. Port 3 High-Byte Control Register (P3CONH)**

### Port 3 Control Register, Low Byte (P3CONL)

**EAH, Set 1, Bank 0, R/W**

<table>
<thead>
<tr>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.3/ADC3</td>
<td>P3.2/ADC2</td>
<td>P3.1/ADC1</td>
<td>P3.0/ADC0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**P3CONL Pin Configuration Settings:**

<table>
<thead>
<tr>
<th>Bit-Pair</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Input mode</td>
</tr>
<tr>
<td>01</td>
<td>Analog input mode</td>
</tr>
<tr>
<td>10</td>
<td>Push-pull output mode</td>
</tr>
<tr>
<td>11</td>
<td>N-channel open-drain output mode</td>
</tr>
</tbody>
</table>

**Figure 9-9. Port 3 Low-Byte Control Register (P3CONL)**

---

**SAMSUNG ELECTRONICS**

9-9
FUNCTION-FIXED PORT

These I/O pins are used only for the input and output of video synchronization signals to the sync processor or DDC & IIC-bus interface. The horizontal and vertical sync signals can be monitored directly through the Sync Port Read Data Register (SYNCRD).

Sync signal ports
- Csync-I: Composite (SOG) synchronization input port (TTL level)
- Hsync-I: Horizontal synchronization input (TTL level)
- Vsync-I: Vertical synchronization input and synchro clock (VCLK) for DDC1 (TTL level)
- Hsync-O: Horizontal synchronization output from the sync processor
- Vsync-O: Vertical synchronization output from the sync processor
- Clamp-O: Clamp signal output with programmable width from the sync processor

DDC and IIC-bus interface ports
- SDA0: DDC and IIC-bus interface serial data
- SCL0: DDC and IIC-bus interface serial clock

Figure 9-10. Sync Port Read Data Register (SYNCRD)
PROGRAMMING TIP — Configuring I/O Port Pins to Specification

The following sample program shows you how to configure the S3C8639/C863A/C8647 I/O ports to specification. The program comments explain the effect of the settings:

- SB0 ; Select bank 0
- LD P0CONH,#0FFH ; Set port 0 high byte to push-pull output mode
- LD P0CONL,#0D5H ; Set P0.3 to push-pull output mode
- ; Set P0.0–P0.2 to rising edge interrupt mode
- LD P0INT,#0FH ; Enable port 0 external interrupt
- LD P1CON,#00H ; Set port 1 to input mode
- LD P2CONH,#3FH ; Set port 2 high byte to PWM n-channel open-drain
- ; output mode (5-volt capability) and Csync input mode
- LD P2CONL,#0FFH ; Set port 2 low byte to PWM push-pull output mode
- LD P3CONH,#0AAH ; Set port 3 high byte to push-pull output mode
- LD P3CONL,#55H ; Set port 3 low byte to analog input mode
OVERVIEW

S3C8639/C863A/C8647 has a default timer: an 8-bit basic timer.

You can use the basic timer (BT) in two different ways:

— As a watchdog timer, it provides an automatic reset mechanism in the event of a system malfunction.
— Signals the end of the required oscillation stabilization interval after a reset or a Stop mode release.

The functional components of the basic timer block are:

— Clock frequency divider (f_{OSC} divided by 4096, 1024, or 128) with multiplexer
— 8-bit basic timer counter, BTCNT (set 1, bank 0, FDH, read-only)
— Basic timer control register, BTCON (set 1, D3H, read/write)
— Watchdog timer control register, WDTCON (set 1, bank 0, ECH, read/write)
BASIC TIMER CONTROL REGISTER (BTCON)

The basic timer control register, BTCON, is used to select the input clock frequency, to clear the basic timer counter and frequency dividers, and to enable or disable the watchdog timer function. It is located in set 1, address D3H, and is read/write addressable using register addressing mode.

A reset clears BTCON to "00H". This enables the watchdog function and selects a basic timer clock frequency of $f_{OSC}/4096$. To disable the watchdog function, you must write the signature code "1010B" to the basic timer register control bits BTCON.7–BTCON.4.

The 8-bit basic timer counter, BTCNT (set 1, bank 0, FDH), can be cleared at any time during the normal operation by writing a "1" to BTCON.1. To clear the frequency dividers for both the basic timer input clock and the timer M0 clock (unless timer M0 uses an external clock source), you should write a "1" to BTCON.0.

![Figure 10-1. Basic Timer Control Register (BTCON)](image-url)
WATCHDOG TIME CONTROL REGISTER (WDTCON)

The watchdog time control register, WDTCON, is used to generate various watchdog time and to select Hsync output. It is located in set 1, bank 0, address ECH, and is read/write addressable using register addressing mode.

![Table of WDTCON Register](image)

<table>
<thead>
<tr>
<th>MSB</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not use for S3C8639/C863A/C8647

Hsync-O divide enable bit:
0 = Hsync-I (Non-divide)
1 = Hsync-I/2

Watchdog time generation control bits:

| 000 | TBTOVF |
| 001 | TBTOVF/2 |
| 010 | TBTOVF/3 |
| 011 | TBTOVF/4 |
| 100 | TBTOVF/5 |
| 101 | TBTOVF/6 |
| 110 | TBTOVF/7 |
| 111 | TBTOVF/8 |

NOTE: \( t_{BTOVF} = \frac{1}{f_{OSC}} \times \text{(divider count of basic timer input clock)} \times 256 \)

Figure 10-2. Watchdog Time Control Register (WDTCON)
BASIC TIMER FUNCTION DESCRIPTION

Watchdog Timer Function
You can program the basic timer overflow signal (BTOVF) to generate a reset by setting BTCON.7–BTCON.4 to any value other than "1010B" (The "1010B" value disables the watchdog function). A reset clears BTCON to "00H", automatically enabling the watchdog timer function. A reset also selects the CPU clock (as determined by the current CLKCON register setting), divided by 4096, as the BT clock.

A reset whenever a basic timer counter overflow occurs. During the normal operation, the application program must prevent the overflow and the accompanying reset operation from occurring. To do this, the BTCNT value must be cleared (by writing a "1" to BTCON.1) at regular intervals. And you can generate the various watchdog time by setting WDTCON.2-WDTCON.0.

If a system malfunction occurs due to circuit noise or some other error condition, the BT counter clear operation will not be executed and a basic timer overflow will occur, initiating a reset. In other words, during the normal operation, the basic timer overflow loop (a bit 7 overflow of the 8-bit basic timer counter, BTCNT) is always broken by a BTCNT clear instruction. If a malfunction does occur, a reset is triggered automatically.

Oscillation Stabilization Interval Timer Function
You can also use the basic timer to program a specific oscillation stabilization interval after a reset or when stop mode has been released by an external interrupt.

In Stop mode, whenever a reset or an external interrupt occurs, the oscillator starts. The BTCNT value then starts increasing at the rate of f_{OSC}/4096 (for reset), or at the rate of the preset clock source (for an external interrupt). When BTCNT.4 overflows, a signal is generated to indicate that the stabilization interval has elapsed and to gate the clock signal off to the CPU so that it can resume the normal operation.

In summary, the following events occur when Stop mode is released:

1. During the stop mode, a power-on reset or an external interrupt occurs to trigger the stop mode release and oscillation starts.
2. If a power-on reset occurred, the basic timer counter would increase at the rate of f_{OSC}/4096. If an external interrupt is used to release Stop mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until bit 4 of the basic timer counter overflows.
4. When a BTCNT.4 overflow occurs, the normal CPU operation resumes.
NOTE: In a power-on reset operation, the CPU is idle during the required oscillation stabilization interval when BTCNT.4 is set after releasing from RESET or STOP mode, CPU clock starts.

Figure 10-3. Basic Timer Block Diagram
PROGRAMMING TIP — Configuring the Basic Timer

This example shows how to configure the basic timer to sample specifications:

```
ORG 0100H
RESET DI ; Disable all interrupts
       SB0 ; Select bank 0
       LD BTCON,#0AAH ; Disable the watchdog timer
       LD CLKCON,#98H ; Non-divided clock
       CLR SYM ; Disable global and fast interrupts
       CLR SPL ; Stack pointer low byte ← "0"
               ; Stack area starts at 0FFH
       SRP #0C0H ; Set register pointer ← 0C0H
       EI ; Enable interrupts

MAIN LD BTCON,#A2H ; Watchdog timer disable
       ; Basic timer/counter clear
       LD BTCON,#52H ; Enable the watchdog timer
       ; Basic timer clock: fosc/4096
       ; Clear basic timer counter
       LD WDTCON,#03H ; Watchdog time: tTOVF/4
       NOP
       NOP
       JP T,MAIN
```
11

TIMER M0

OVERVIEW

The 8-bit timer M0 is for monitor application. Timer M0 includes capture timer mode using the appropriate TM0CON setting.

Timer M0 has the following functional components:

— Clock frequency divider (f_{OSC} divided by 128 or 8) with multiplexer
— 2-bit prescaler for the timer M0 input clock
— 8-bit counter (TM0CNT; set1, D0H, read-only) and 8-bit reference data register (TM0DATA; set1, D1H, read-only)
— Timer M0 capture or overflow interrupt (IRQ0, vector E2H, E0H) generation
— Timer M0 control register, TM0CON (set 1, D2H, read/write)

FUNCTION DESCRIPTION

CAPTURE TIMER FUNCTION

The timer M0 module can generate two interrupts: the timer M0 capture interrupt (TM0INT), and the timer M0 overflow interrupt (TM0OVF). TM0INT belongs to interrupt level IRQ0, and is assigned the separate vector address, E2H. TM0OVF is interrupt level IRQ0, vector E0H.

The TM0INT and TM0OVF pending conditions are automatically cleared by hardware after they are serviced.

In capture timer mode, a signal edge that is detected at the TM0CAP pin opens a gate and loads the current counter value into the timer M0 data register (TM0DATA). You can select rising or falling edge to trigger this operation.

Both kinds of timer M0 interrupts can be used in capture mode: the timer M0 overflow interrupt is generated whenever a counter overflow occurs; the timer M0 capture interrupt is generated whenever the counter value is loaded into the timer M0 data register.

By reading captured data value in TM0DATA, and assuming a specific value for the timer M0 clock frequency, you can calculate the internal time of the signal being input to the TM0CAP pin or the vertical sync output signal being output from the sync-processor module.
Timer M0 Control Register (TM0CON)

You use the timer M0 control register, TM0CON, to

— Select the timer M0 operating mode (capture mode)
— Select the timer M0 input clock frequency
— Clear the timer M0 counter, TM0CNT
— Enable the timer M0 overflow interrupt and timer M0 capture interrupt
— Select a 2-bit prescaler value for the Timer M0 input clock
— Select the timer M0 capture input source

TM0CON is located in set 1, at address D2H, and is read/write addressable using Register addressing mode.

A reset clears TM0CON to "00H". This sets timer M0 to disable capture timer mode, selects an input clock frequency of \( f_{OSC}/128 \), and disables timer M0 overflow and capture interrupts. You can clear the timer M0 counter at any time during the normal operation by writing a "1" to TM0CON.2.

The timer M0 overflow interrupt (TM0OVF) is in the interrupt level IRQ0 and has the vector address E0H. When the timer M0 capture interrupt is disabled, the Timer M0 overflow interrupt by clock \( f_{OSC} \) is possible. When a timer M0 overflow interrupt occurs and is serviced by the CPU, the pending condition is cleared automatically by hardware.

To enable the timer M0 capture interrupt (IRQ0, vector E2H), you must write TM0CON.1 to "1". There is no pending bit cleared by software or static read bit which is H/W pending. After the interrupt request is serviced, the pending condition is automatically cleared by hardware.

<table>
<thead>
<tr>
<th>MSB</th>
<th>0.7</th>
<th>0.6</th>
<th>0.5</th>
<th>0.4</th>
<th>0.3</th>
<th>0.2</th>
<th>0.1</th>
<th>0.0</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer M0 input clock selection bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = ( f_{OSC}/128 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = ( f_{OSC}/8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M0 capture input selection bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = TM0CAP input pin selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = V-sync output path selection from sync-processor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-bit prescaler bits:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 = No division</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 = Divide by 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 = Divide by 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 = Divide by 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M0 capture interrupt enable bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Disable the timer M0 capture interrupt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Enable the timer M0 capture interrupt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M0 capture mode selection bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Capture on rising mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Capture on falling mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M0 overflow interrupt enable bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Disable the timer M0 overflow interrupt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Enable the timer M0 overflow interrupt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer M0 counter clear bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = No effect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Clear timer M0 counter (when write)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: When the captured value is #0FFh, the overflow interrupt does not occur. When the value of capture is changed from #0FFh to #00h, the overflow interrupt always occurs. When the captured value is #00h, the overflow interrupt occurs in advance.

Figure 11-1. Timer M0 Control Register (TM0CON)
Figure 11-2. Timer M0 Functional Block Diagram
12 TIMER M1

OVERVIEW

The 12-bit timer M1 is an 12-bit timer/counter for monitor application. Timer M1 offers capture/overflow timer mode using the appropriate TM1CON setting.

Timer M1 has the following functional components:

— Clock frequency selector as the timer M1 clock (f_{OSC} divided by 512, 128, or 2, Hsync-I or Csync-I from sync-processor) with multiplexer
— Capture signal selector from V-syncO (sync-processor) or the timer M2 interval time
— 12-bit counter (TM1CNTH, TM1CNTL; set1, bank0, F1H, F2H, read-only) and 12-bit reference data register (TM1DATAH, TM1DATAL; set1, bank0, F3H, F4H, read-only)
— Timer M1 capture or overflow interrupt (IRQ2, vector E8H, E6H) generation
— Timer M1 control register, TM1CON (set 1, bank0, F5H, read/write)

FUNCTION DESCRIPTION

Overflow Timer Function

The timer M1 module generates an overflow signal whenever the timer M1 counter overflow occurs. If you set the timer M1 overflow interrupt enable bit, TM1CON.2, to "1", an interrupt is generated whenever an overflow state is detected. After the interrupt request is generated, the counter register value is cleared and counting resumes from "00H".

The timer M1 overflow interrupt pending condition is automatically cleared by hardware when it has been serviced.

Capture Timer Function

The Timer M1 module can generate, the timer M1 capture interrupt (TM1INT). TM1INT belongs to interrupt level IRQ2, and is assigned the vector address, E8H.

In capture timer mode, a capture signal from Vsync-O (sync-processor) or the timer M2 interval timer opens a gate and loads the current counter value into the timer M1 data register (TM1DATA). You can select Vsync-O or the timer M2 interval timer as the capture signal source to trigger this operation.

By reading captured data value in TM1DATAH and TM1DATAL, and assuming a specific value for the timer M1 clock frequency, you can calculate the frequency of the signal being input to the Hsync-I or Csync-I from sync-processor by capture signal.
Timer M1 Control Register (TM1CON)

You use the timer M1 control register, TM1CON, to

- Select the capture signal source
- Select the timer M1 clock input
- Clear the timer M1 counter, TM1CNTH and TM1CNTL
- Enable the timer M1 capture and overflow interrupt
- Clear the timer M1 capture interrupt pending bit
- Select Vsync-O capture edge as capture signal source (When TM1CON.7 = "1")

TM1CON is located in set 1, bank0, at address F5H, and is read/write addressable using Register addressing mode.

The setting for bit-pair TM1CON.0 and TM1CON.1 selects the timer M1 counter clock input. The timer M1 capture and overflow interrupt (TM1INT, TM1OVF) are in the interrupt level IRQ2, but has the different vector address (E8H, E6H respectively).

TM1CON.4 is the interrupt pending flag for the timer M1 capture interrupt. To clear a timer M1 interrupt pending condition, the interrupt service routine must write a "0" to TM1CON.4 after the CPU has acknowledged the request. TM1CON.3 is flag to clear the 12-bit Timer M1 counter.

TM1CON.7 is flag to select the capture signal source (timer M2 interval time or Vsync-O from sync-processor) and TM1CON.6 is flag to select the capture edge as the Vsync-O capture signal source.

A reset operation clears TM1CON to "00H", selecting the Hsync-I or Csync-I from sync-processor are the timer M1 clock and disabling the timer M1 capture and overflow interrupt.
## Timer M1 Control Register (TM1CON)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.7</td>
<td>Timer M1 capture signal source selection bit:</td>
</tr>
<tr>
<td>.6</td>
<td>0 = Signal from the timer M2 interval time</td>
</tr>
<tr>
<td>.5</td>
<td>1 = Vsync-O from sync-processor</td>
</tr>
<tr>
<td>.4</td>
<td>Timer M1 clock input selection bits:</td>
</tr>
<tr>
<td>.3</td>
<td>00 = Hsync-l or Csync-l from sync-processor</td>
</tr>
<tr>
<td>.2</td>
<td>01 = fosc/2</td>
</tr>
<tr>
<td>.1</td>
<td>10 = fosc/128</td>
</tr>
<tr>
<td>.0</td>
<td>11 = fosc/512</td>
</tr>
<tr>
<td>MSB</td>
<td>Timer M1 overflow interrupt enable bit:</td>
</tr>
<tr>
<td>LSB</td>
<td>0 = Disable the timer M1 overflow interrupt</td>
</tr>
<tr>
<td></td>
<td>1 = Enable the timer M1 overflow interrupt</td>
</tr>
<tr>
<td></td>
<td>Vsync-O capture source edge selection bit (when TM1CON.7=1):</td>
</tr>
<tr>
<td></td>
<td>0 = Vsync-O rising edge from sync-processor</td>
</tr>
<tr>
<td></td>
<td>1 = Vsync-O falling edge from sync-processor</td>
</tr>
<tr>
<td></td>
<td>Timer M1 counter clear bit (when write):</td>
</tr>
<tr>
<td></td>
<td>0 = No effect</td>
</tr>
<tr>
<td></td>
<td>1 = Clear timer M1 counter</td>
</tr>
<tr>
<td></td>
<td>Timer M1 capture interrupt enable bit:</td>
</tr>
<tr>
<td></td>
<td>0 = Disable the timer M1 capture interrupt</td>
</tr>
<tr>
<td></td>
<td>1 = Enable the timer M1 capture interrupt</td>
</tr>
<tr>
<td></td>
<td>Timer M1 capture interrupt pending bit:</td>
</tr>
<tr>
<td></td>
<td>0 = Interrupt is not pending (when read)</td>
</tr>
<tr>
<td></td>
<td>1 = No effect (when write)</td>
</tr>
</tbody>
</table>

### Diagram

![Figure 12-1. Timer M1 Control Register (TM1CON)](image-url)
**BLOCK DIAGRAM**

Figure 12-2. Timer M1 Functional Block Diagram

- **TM1CNTL**
- **TM1CNTH**
- **OVF**
- **IRQ2**
- **Clear**
- **CAP**
- **f\_OSC/512**
- **f\_OSC/128**
- **f\_OSC/2**
- **Hsync-I/Csync-I from sync processor**
- **Vsync-O from sync-processor**
- **Capture signal from timer M2 interval time (TM2CON1,0)**

(The TM1CNTL and TM1CNTH registers are read-only.)

(The TM1DATAL and TM1DATAH registers are read-only.)
The interval timer M2 is a no-counter timer for monitor applications. Timer M2 offers interval timer mode using the appropriate TM1CON setting.

Timer M2 has the following functional components:
- 5-bit scaler by $f_{\text{OSC}}/1000$ for timer M2 interval source
- Timer M1 capture interval time source selector (When TM1CON.5 is "1") with 2-bit scaler
- Timer M2 interval interrupt (IRQ1, vector E4H) generation
- Timer M2 control register, TM2CON (set 1, F6H, read/write)

**FUNCTION DESCRIPTION**

**Interval Timer Function**
The timer M2 module generates an interval interrupt whenever the TM2CON.2 is "1". TM2INT belongs to the interrupt level IRQ1, and is assigned the separate vector address, E4H. The TM2INT pending condition is automatically cleared by hardware when it has been serviced.

**Timer M2 Control Register (TM2CON)**
You use the timer M2 control register, TM2CON, to
- Select the interval time signal source by 5-bit scaler
- Enable the timer M2 interval interrupt
- Select timer M1 capture interval time by 2-bit scaler (When TM1CON.5 = "1")

TM2CON is located in set 1, bank0, at address F6H, and is read/write addressable using Register addressing mode.

A reset operation clears TM2CON to "F8H" (11111000B), thereby setting the 5-bit scaler value to be divided by 32.
Timer M2 Control Register (TM2CON)
F6H, Set 1, R/W

<table>
<thead>
<tr>
<th>MSB</th>
<th>.7</th>
<th>.6</th>
<th>.5</th>
<th>.4</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
<th>LSB</th>
</tr>
</thead>
</table>

5-bit scaler bits:
00000 = No division
00001 = Divide by 2
00010 = Divide by 3
...  
11111 = Divide by 32

Timer M1 capture interval time selection bits:
00 = Timer M2 interval (bypass)
01 = Timer M2 interval x 10
10 = Timer M2 interval x 20
11 = Timer M2 interval x 30

Timer M2 interval interrupt enable bit:
0 = Disable the timer M2 interval interrupt
1 = Enable the timer M2 interval interrupt

NOTES:
1. When the timer M1 capture mode is enabled (TM1CON.5 = "1"), the value of 5-bit or 2-bit scaler can be changed only in the timer M1 capture interrupt routine.
2. When the timer M1 capture mode is disabled (TM1CON.5 = "0"), the value of 5-bit scaler can be changed only in the timer M2 interval interrupt routine.

Figure 13-1. Timer M2 Control Register (TM2CON)
Figure 13-2. Timer M2 Functional Block Diagram
ANALOG-TO-DIGITAL CONVERTER

OVERVIEW

The 8-bit A/D converter (ADC) module of S3C8639/C863A/C8647 employs successive approximation logic to convert analog levels entering one of the four input channels to equivalent 8-bit digital values. The analog input level must lie between the $V_{DD2}$ and $V_{SS2}$ values. The A/D converter has the following components:

- Analog comparator with successive approximation logic
- D/A converter logic (resistor string type)
- 8-bit ADC control register (ADCON)
- Four multiplexed analog data input pins (ADC0–ADC3)
- 8-bit A/D conversion data output register (ADDATA) (S3C863X)
- 4-bit A/D conversion data output register (ADDATA) (S3C8647)
- 8-bit digital input port (Alternatively, I/O port)
- $V_{DD2}$ and $V_{SS2}$ pins (S3C863X)

FUNCTION DESCRIPTION

To initiate an analog-to-digital conversion procedure, write the channel selection data in the A/D converter control register ADCON to select one of the four analog input pins (ADCn, n = 0–3) and set the conversion start or enable bit, ADCON.0. The read-write ADCON register is located in set 1, bank0, at address F7H.

During the normal conversion, ADC block initially sets the successive approximation register to 80H (approximately the half-way point of an 8-bit register). This register is then updated automatically in each conversion step. The successive approximation block performs 8-bit conversions for one input channel at a time. You can dynamically select different channels by manipulating the channel selection bit value (ADCON.5–4) in the ADCON register. To start the A/D conversion, you should set, ADCON.0. When a conversion is completed, ADCON.3, the end-of-conversion (EOC) bit, is automatically set to 1 and the result is dumped into the ADDATA register where it can be read. The A/D converter then enters an idle state. Remember to read the contents of ADDATA before another conversion starts. Otherwise, the previous result will be overwritten by the next conversion result.

NOTE

As the A/D converter does not include any sample-and-hold circuitry, it is very important to keep the fluctuation in the analog level at the ADC0–ADC3 input pins to an absolute minimum during the conversion process. Any change in the input level, perhaps due to noise, will invalidate the result. If the chip enters to STOP or IDLE mode in conversion process, there will be a leakage current path in A/D block. You must use STOP or IDLE mode after A/D converting operation is finished.
CONVERSION TIMING

The A/D conversion process requires 4 steps (8 clock edges) to convert each bit. Therefore, a total of 48 clocks are required to complete an 10-bit conversion. With an 8 MHz $f_{OSC}$ clock frequency, one clock cycle is $1 \mu s$ (when ADCON.2, .1 are “01”). If each bit conversion requires 4 clocks, the conversion rate is calculated as follows:

\[
\text{start (4 clocks) + (4 clocks/bit} \times n \text{ bits) + EOC (4 clocks) = } 4(n+2) \text{ clocks, } 1 \mu s \times 4(n+2) = 4(n+2) \mu s \text{ at } 8 \text{ MHz}
\]

where, $n = 4$ (S3C8647), 10 (S3C863x)

A/D CONVERTER CONTROL REGISTER (ADCON)

The A/D converter control register, ADCON, is located at address F7H in set 1, bank0. It has four functions:

— Analog input pin selection (bits 4,5 and 6)
— End-of-conversion status detection (bit 3)
— Clock source selection (bits 2 and 1)
— A/D operation start or enable (bit 0)

After a reset, the ADC0 pin is automatically selected as the analog data input pin, and the start bit is turned off.

You can select only one analog input channel at a time. Other analog input pins (ADC0–ADC3) can be selected dynamically by manipulating the ADCON.6–4 bits.

<table>
<thead>
<tr>
<th>A/D input pin selection bits:</th>
<th>A/D input pin</th>
<th>Clock source select:</th>
<th>End-of conversion bit (read-only):</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 5 4</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0 = Conversion is not complete</td>
</tr>
<tr>
<td></td>
<td>0 0 1</td>
<td>0 1</td>
<td>1 = Conversion is complete</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1 1</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Others</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADC0 (P3.0)</td>
<td>foSC/16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADC1 (P3.1)</td>
<td>foSC/8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADC2 (P3.2)</td>
<td>foSC/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADC3 (P3.3)</td>
<td>foSC</td>
<td></td>
</tr>
</tbody>
</table>

Figure 14-1. A/D Converter Control Register (ADCON)
INTERNAL REFERENCE VOLTAGE LEVELS

In the ADC function block, the analog input voltage level is compared to the reference voltage. The analog input level must remain within the range of $V_{SS}$ ($V_{SS2}$) to $V_{REF}$ ($V_{DD2}$).

Different reference voltage levels are generated internally along the resistor tree during the analog conversion process for each conversion step. The reference voltage level for the first conversion bit is always $1/2 V_{DD2}$.

BLOCK DIAGRAM

![A/D Converter Functional Block Diagram](image)

**Figure 14-2. A/D Converter Functional Block Diagram**

### A/D Converter Data Register (ADDATA)

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3 (note)</th>
<th>Bit 2 (note)</th>
<th>Bit 1 (note)</th>
<th>Bit 0 (note)</th>
</tr>
</thead>
</table>

**NOTE:** Not mapped for the S3C8647.
Table 14-1. A/D Converter Electrical Characteristics (S3C863X)

\(T_A = -40^\circ C \text{ to } +85^\circ C, V_{DD} = 3.0 \text{ V to } 5.5 \text{ V, } V_{SS} = 0 \text{ V}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td></td>
<td></td>
<td>–</td>
<td>8</td>
<td>–</td>
<td>bit</td>
</tr>
<tr>
<td>Total accuracy</td>
<td></td>
<td>(V_{DD} = 5 \text{ V}) Conversion time = 5(\mu)s</td>
<td>–</td>
<td>–</td>
<td>± 2</td>
<td>LSB</td>
</tr>
<tr>
<td>Integral linearity error</td>
<td>ILE</td>
<td>(AV_{REF} = 5 \text{ V})</td>
<td>–</td>
<td>–</td>
<td>± 1</td>
<td></td>
</tr>
<tr>
<td>Differential linearity error</td>
<td>DLE</td>
<td>(AV_{SS} = 0 \text{ V})</td>
<td>–</td>
<td>–</td>
<td>± 1</td>
<td></td>
</tr>
<tr>
<td>Offset error of top</td>
<td>EOT</td>
<td></td>
<td>± 1</td>
<td>± 2</td>
<td>± 2</td>
<td></td>
</tr>
<tr>
<td>Offset error of bottom</td>
<td>EOB</td>
<td></td>
<td>± 0.5</td>
<td>± 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conversion time (1)</td>
<td>(t_{CON})</td>
<td>8-bit conversion 48 (\times) (n/f_{OSC}) (3), (n = 1, 4, 8, 16)</td>
<td>20</td>
<td>–</td>
<td>170</td>
<td>(\mu)s</td>
</tr>
<tr>
<td>Analog input voltage</td>
<td>VI_{AN}</td>
<td></td>
<td>–</td>
<td>(AV_{SS})</td>
<td>–</td>
<td>(AV_{REF})</td>
</tr>
<tr>
<td>Analog input impedance</td>
<td>R_{AN}</td>
<td></td>
<td>2</td>
<td>1000</td>
<td>–</td>
<td>(M\Omega)</td>
</tr>
<tr>
<td>Analog reference voltage</td>
<td>AV_{REF}</td>
<td></td>
<td>2.5</td>
<td>–</td>
<td>(V_{DD})</td>
<td>(V)</td>
</tr>
<tr>
<td>Analog ground</td>
<td>AV_{SS}</td>
<td></td>
<td>–</td>
<td>(V_{SS})</td>
<td>–</td>
<td>(V_{SS} + 0.3)</td>
</tr>
<tr>
<td>Analog input current</td>
<td>I_{ADIN}</td>
<td>(AV_{REF} = V_{DD} = 5\text{ V})</td>
<td>–</td>
<td>–</td>
<td>10</td>
<td>(\mu)A</td>
</tr>
<tr>
<td>Analog block Current (2)</td>
<td>I_{ADC}</td>
<td>(AV_{REF} = V_{DD} = 5\text{ V}) (AV_{REF} = V_{DD} = 3\text{ V}) (AV_{REF} = V_{DD} = 5\text{ V}) When power down mode</td>
<td>0.5</td>
<td>1.5</td>
<td>100</td>
<td>500</td>
</tr>
</tbody>
</table>

NOTES:
1. “Conversion time” is the time required from the moment a conversion operation starts until it ends.
2. \(I_{ADC}\) is an operating current during the A/D conversion.
3. \(f_{OSC}\) is the main oscillator clock.
Table 14-2. A/D Converter Electrical Characteristics (S3C8647)

(T_A = –40°C to +85°C, V_DD = 4.0 V to 5.5 V, V_SS = 0 V)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>4</td>
<td>–</td>
<td>bit</td>
</tr>
<tr>
<td>Absolute accuracy (1)</td>
<td>–</td>
<td>4 bit conversion</td>
<td>–</td>
<td>–</td>
<td>±0.5</td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24 x n_/f_OSC (3), n = 1, 4, 8, 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conversion time (2)</td>
<td>t_CON</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>us</td>
</tr>
<tr>
<td>Analog input voltage</td>
<td>VIAN</td>
<td>–</td>
<td>V_SS</td>
<td>–</td>
<td>V_DD</td>
<td>V</td>
</tr>
<tr>
<td>Analog input impedance</td>
<td>R_AN</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>MΩ</td>
</tr>
</tbody>
</table>

NOTES:
1. Excluding quantization error, absolute accuracy values are within ±0.5 LSB.
2. "Conversion time" is the time required from the moment a conversion operation starts until it ends.
3. f_OSC is the main oscillator clock.
15 PULSE WIDTH MODULATION

PWM MODULE

The S3C8639/C863A/C8647 microcontrollers include seven 8-bit PWM circuits, PWM0–PWM6. The S3C8647 microcontroller includes six 8-bit PWM circuits, PWM0–PWM5. The operation of all PWM circuits is controlled by a single control register, PWMCON.

The PWM counter, a 8-bit incrementing counter, is used by the 8-bit PWM circuits. To start the counter and enable the PWM circuits, set PWMCON.5 to “1”. If the counter is stopped, it retains its current count value. When restarted, it resumes counting from the retained count value.

By modifying the prescaler value, you can divide the input clock by one (non-divided), two, three, or four. The prescaler output is the clock frequency of the PWM counter.

The PWM counter overflows when it reaches “3FH”, and then continues counting from zero.
### PWM CONTROL REGISTER (PWMCON)

The control register for the PWM module, PWMCON, is located in set 1, bank 1, at register address E7H. You use PWMCON bit settings to control the following functions in the 8-bit:

- PWM counter operation: stop/start (or resume counting)

A reset clears PWMCON to "00H", disabling all PWM functions.

**Figure 15-1. PWM Control Register (PWMCON)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PWM counter enable bit: 0 = Stop the PWM counter, 1 = Start the PWM counter</td>
</tr>
<tr>
<td>1</td>
<td>2-bit prescaler for PWM counter clock: 00 = Non-divided, 01 = Divide by 2, 10 = Divide by 3, 11 = Divide by 4</td>
</tr>
<tr>
<td>2</td>
<td>Not used for S3C8639/C863A/C8647</td>
</tr>
</tbody>
</table>

The control register for the PWM module, PWMCON, is located in set 1, bank 1, at register address E7H. You use PWMCON bit settings to control the following functions in the 8-bit:

- PWM counter operation: stop/start (or resume counting)

A reset clears PWMCON to "00H", disabling all PWM functions.
PWM0–PWM6

The S3C8639/C863A/C8647 microcontrollers include seven 8-bit PWM circuits, PWM0–PWM6. The S3C8647 microcontroller include six 8-bit PWM circuits, PWM0–PWM5. Each 8-bit PWM data unit is comprised of an 8-bit basic frame. The 8-bit PWM circuits have the following components:

- 8-bit counter
- 8-bit comparators
- 8-bit PWM data registers (PWM0–PWM5, PWM6 (note))
- PWM output pins (PWM0–PWM5, PWM6 (note))

The PWM0–PWM6 circuits are controlled by the PWMCON register (set 1, bank 1, E7H).

NOTE: Not used for the S3C8647.
PWM0–PWM6 FUNCTION DESCRIPTION

All the seven 8-bit PWM circuits have an identical function and each has its own 8-bit data register and 8-bit comparator. Each circuit compares a unique data register value to the 8-bit PWM counter.

The PWM0–PWM6 data registers are located in set 1, bank 1, at locations E0H–E6H, respectively. These data registers are read/write addressable. By loading specific values into the respective data registers, you can modulate the pulse width at the corresponding PWM output pins, PWM0–PWM6. (PWM0–PWM6 correspond to port 2 pins P2.0–P2.6.)

The level at the output pins toggles High and Low at a frequency equal to the counter clock, divided by 64 ($2^6$). The duty cycle of the 8-bit PWM pins ranges from 0% to 98.44% (63/64), based on the corresponding data register values.

To determine the output duty cycle of an 8-bit PWM circuit, its 8-bit comparator sends the output level High when the data register value is greater than the lower 8-bit count value. The output level is Low when the data register value is less than or equal to the lower 8-bit count value. The output level at the PWM0–PWM6 pins remains at Low level for the first 256 counter clocks. Then, each PWM waveform is repeated continuously, at the same frequency and duty cycle, until one of the following three events occurs:

- The counter is stopped
- The counter clock frequency is changed
- A new value is written to the PWM data register

STAGGERED PWM OUTPUTS

The PWM0–PWM6 outputs are staggered to reduce the overall noise level on the pulse width modulation circuits. If you load the same value to the PWM0–PWM6 data registers, a match condition (data register value is equal to the 8-bit count value) will occur on the same clock cycle for all the seven 8-bit PWM circuits.

For example, the PWM0 output is delayed by one-half of a counter clock, PWM1 output by one-half of a counter clock, PWM2 output by one-half of a counter clock, and so on for the subsequent clock cycles (see Figure 15-4).

NOTE: The S3C8647 microcontroller includes just six 8-bit PWM circuits, PWM0–PWM5.
NOTES:
1. A counter clock value of 8 MHz is assumed for all timing values.
2. 'n' = 0 to 6, for PWM0-PWM5, PWM6 (3).
3. Not used for the S3C8647.

Figure 15-3. PWM Waveforms for PWM0–PWM6
Figure 15-4. PWM Clock to PWM0–PWM6 Output Delays
PWM COUNTER

The PWM counter is an 8-bit incrementing counter. The same 8-bit counter is used by all PWM circuits. To determine the PWM module's base operating frequency, the counter is compared to the PWM data register value.

PWM DATA REGISTERS

A reset operation disables all PWM output. The current counter value is retained when the counter stops. When the counter starts, counting resumes from the retained value.

PWM CLOCK RATE

The timing of the 8-bit output channel is based on the maximum 12 MHz CPU clock frequency. The 2-bit prescaler value in the PWMCON register determines the frequency of the counter clock. You can set PWMCON.6 and PWMCON.7 to divide the CPU clock frequency by one (non-divided), two, three, or four.

As the maximum CPU clock rate for the S3C8639/C863A/C8647 microcontrollers is 12 MHz, the maximum base PWM frequency is 187.5 kHz (12 MHz divided by 64). This assumes a non-divided CPU clock.
PROGRAMMING TIP — Programming PWM0 to Sample Specifications

This sample program executes a test of the PWM block. The program parameters are as follows:

— The oscillation frequency of the main crystal is 8 MHz
— PWM frequency is 125 kHz

RESET:    DI ; Disable global interrupts
       SB0 ; Select bank 0
        •
        •
LD     P2CONH,#11111111B ; Select n-channel open-drain PWM output
LD     P2CONL,#11111111B ; Select push-pull PWM output
SB1 ; Select bank 1
OR    PWMCON,#00100000B ; PWMCON.5 ← 1; start the counter
       ; PWM counter clock is f_{OSC}
SB0 ; Select bank 0
EI ; Enable global interrupts

PWMstart:
SB1 ; Select bank 1
LD     PWM0, #80H ; Load PWM0 data
SB0 ; Select bank 0
RET
OVERVIEW

The S3C8639/C863A/C8647 multi-sync signal processor (sync processor) is designed to process horizontal (Hsync) and vertical (Vsync) signals that are input to a multi-sync monitor. The sync processor can perform the following functions:

— Detect sync input signals (Vsync-I, Hsync-I, and Csync-I, also called Screen-On-Green, or SOG)
— Output a programmable pseudo sync generation signal
— Detect the polarity of sync input signals
— Separate and output sync signals (Hsync-O, Vsync-O, and Clamp-O)

The sync processor circuits are controlled by three control registers: SYNCON0, SYNCON1, and SYNCON2.

Vsync SEPARATION

SYNCON0 register setting controls the output path of the sync processor’s 5-bit counter. Using the 5-bit counter, the sync processor can separate the Vsync signal from composite (H+V) sync signal.

The counter value increments when a High level sync signal is detected and decrements when a Low level signal is detected. No overflow or underflow can occur. That is, the 5-bit counter increments until it reaches the maximum value of 11111B and then stops or decrements until it reaches the minimum value of 00000B. You can select fOSC/2 or fOSC/3 as the counter’s clock input source.

When SYNCON0.5 is "1", a High signal level is output to a multiplexer whenever the counter value reaches 11111B and a Low level is output when the counter value reaches 00000B. The signal level remains constant when the counter value is less than 11111B or greater than 00000B.

CLAMP SIGNAL OUTPUT

SYNCON1 register settings control Clamp signal output and pulse width. Clamp output can be completely inhibited, or it can be generated at two, four, or eight times fOSC. You can specify the signal edge on which the selected Clamp pulse width is to be output (“front porch” or “back porch”). When SYNCON1.7–.6 is set to “00”, the clamp signal output is inhibited. In this case the clamp signal level (Clamp-O) can be either “low” (when SYNCON1.4 is set to “1”) or “high” (when SYNCON1.4 is “0”).
Logic for Detecting Sync-On-Green (SOG)
Special logic in the sync processor block can compare Hsync and Csync input signals to detect Sync-On-Green (SOG). The interrupt SOG through Csync-P port is detected automatically at the SOG detection block. You can confirm to SOG by means of reading SYNCON2.2 (SOG detection bit).

Pseudo Sync Generator
SYNCON2 settings (SYNCON2.4 = "0") control the pseudo Hsync and Vsync generation registers value (See figure 16.1 and 16.2). The polarity of these frequencies is always positive, with pulse width of 2us (eight fsync clock, when fsync is 4 MHz) and 6 x PHGEN periods, respectively. The pseudo sync generator supports factory testing of the sync processor block and also protects a system against the effect of unexpected signals in transition period while mode changing.

Pseudo Hsync Generation Register (PHGEN)
F9H, Set 1, Bank 0, R/W

| .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 | LSB |

Pseudo Hsync generation bits:
- When SYNCON2.4 (generation pseudo H/Vsync generation mode) = "0"
- Positive polarity only
- Pulse width: 2us (eight fsync clock, when fsync is 4 MHz)
- Range: 15.68 kHz (PHGEN = FFh) -400 kHz (PHGEN = 10h)

Figure 16-1. Pseudo Hsync Generation Register (PHGEN)

Pseudo Vsync Generation Register (PVGEN)
FAH, Set 1, Bank 0, R/W

| .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 | LSB |

Pseudo Vsync generation bits:
- When SYNCON2.4 (generation pseudo H/Vsync generation mode) = "0"
- Positive polarity only
- Pulse width: 6 PHGEN periods
- PVGEN value must be in [2-255] range

Figure 16-2. Pseudo Vsync Generation Register (PVGEN)
Hsync & Vsync Polarity Detection, Unmixed Hsync Detection and Hsync Blanking

The polarity of Hsync & Vsync signal input to Hsync-I & Vsync-I pin is automatically detected. If the Hsync polarity is negative, SYNCON1.0 equals to "0". If the Hsync polarity is positive, SYNCON1.0 equals to "1". This polarity detection bit (SYNCON1.0) may be not accurate when the sync level is not in a transitional condition.

And if the Vsync polarity is negative, SYNCON1.1 equals to "1". This polarity detection bit (SYNCON1.1) may be not accurate when the sync level is not in a transitional condition.

In composite sync mode, if SYNCON2.7 is set to "1", the current period of checked Hsync is stable, unmixed with Vsync signal. If SYNCON2.7 is "0", the current period of checked Hsync is mixed with Vsync signal, in which case it is recommended not to calculate the sync frequency. In this mode, the Hsync signal is automatically blanked during the Vsync signal extraction period.

Table 16-1. VESA Monitor Timing Standards & PHGEN/PVGEN Value

<table>
<thead>
<tr>
<th>Standard Hsync Freq. [kHz]</th>
<th>Standard Vsync Freq. [Hz]</th>
<th>Resolution</th>
<th>Line Num. [Hf/Vf]</th>
<th>Pseudo Hf/(PHGEN) [kHz]</th>
<th>Pseudo Vf/(PVGEN) [Hz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.469</td>
<td>59.940</td>
<td>640 × 480</td>
<td>525</td>
<td>31.49 (127)</td>
<td>59.65 (66)</td>
</tr>
<tr>
<td>37.861</td>
<td>72.807</td>
<td>800 × 600</td>
<td>625</td>
<td>38.09 (105)</td>
<td>73.26 (65)</td>
</tr>
<tr>
<td>37.500</td>
<td>75.000</td>
<td>500</td>
<td>625</td>
<td>37.73 (106)</td>
<td>74.87 (63)</td>
</tr>
<tr>
<td>35.156</td>
<td>56.250</td>
<td>800 × 600</td>
<td>625</td>
<td>35.08 (114)</td>
<td>56.23 (78)</td>
</tr>
<tr>
<td>37.879</td>
<td>60.317</td>
<td>628</td>
<td>625</td>
<td>38.09 (105)</td>
<td>60.27 (79)</td>
</tr>
<tr>
<td>48.077</td>
<td>72.188</td>
<td>666</td>
<td>48.19 (83)</td>
<td>48.19 (115)</td>
<td>60.27 (79)</td>
</tr>
<tr>
<td>46.875</td>
<td>75.000</td>
<td>625</td>
<td>47.06 (85)</td>
<td>47.06 (105)</td>
<td>75.41 (78)</td>
</tr>
<tr>
<td>35.522</td>
<td>43.479</td>
<td>1024 × 768</td>
<td>817</td>
<td>35.71 (112)</td>
<td>43.76 (102)</td>
</tr>
<tr>
<td>48.363</td>
<td>60.004</td>
<td>806</td>
<td>48.19 (83)</td>
<td>48.19 (105)</td>
<td>59.64 (101)</td>
</tr>
<tr>
<td>56.476</td>
<td>70.069</td>
<td>806</td>
<td>56.33 (71)</td>
<td>56.33 (105)</td>
<td>69.72 (101)</td>
</tr>
<tr>
<td>60.023</td>
<td>75.029</td>
<td>800</td>
<td>59.70 (67)</td>
<td>59.70 (105)</td>
<td>74.62 (100)</td>
</tr>
<tr>
<td>63.995</td>
<td>70.016</td>
<td>1152 × 864</td>
<td>914</td>
<td>63.49 (63)</td>
<td>70.23 (113)</td>
</tr>
<tr>
<td>77.487</td>
<td>85.057</td>
<td>911</td>
<td>76.92 (52)</td>
<td>76.92 (105)</td>
<td>85.09 (113)</td>
</tr>
<tr>
<td>75.000</td>
<td>75.000</td>
<td>1280 × 960</td>
<td>1000</td>
<td>75.47 (53)</td>
<td>75.47 (125)</td>
</tr>
<tr>
<td>63.974</td>
<td>60.013</td>
<td>1280 × 1024</td>
<td>1066</td>
<td>63.49 (63)</td>
<td>60.12 (132)</td>
</tr>
<tr>
<td>79.976</td>
<td>75.025</td>
<td>1066</td>
<td>80.00 (50)</td>
<td>80.00 (105)</td>
<td>75.18 (133)</td>
</tr>
<tr>
<td>75.000</td>
<td>60.000</td>
<td>1600 × 1200</td>
<td>1250</td>
<td>75.47 (53)</td>
<td>60.08 (157)</td>
</tr>
<tr>
<td>107.043</td>
<td>85.022</td>
<td>1259</td>
<td>108.11 (37)</td>
<td>108.11 (105)</td>
<td>85.52 (158)</td>
</tr>
</tbody>
</table>

NOTE: Pseudo Hsync frequency = fsync/PHGEN value
Pseudo Vsync frequency = Pseudo Hsync frequency/(8 × PVGEN value)
SYNC PROCESSOR CONTROL REGISTER 0 (SYNCON0)

The sync processor control register 0, SYNCON0, is located in set 1, bank 0, at address EDH. It is read/write addressable. SYNCON0 bits 4–0 hold the 5-bit counter value which is used for compare function. Whenever a High signal level is detected, the count value is incremented by one until it reaches the maximum value of "11111B" (No overflow occurs). Whenever a Low signal level is detected, the count value is decremented by one until it reaches the minimum value of "00000B" (No underflow occurs).

NOTE

When the composit sync is inputted, compare mode is also called Vsync separation mode. In this mode, output to the multiplexer is enabled. When the counter value is "11111B", the output is High level; when the counter value is "00000B", the output is Low level. Whenever the counter value is less than ( < ) "11111B", or greater than ( > ) "00000B", the previous output level is retained.

SYNCON0 settings also control the following sync processor functions:

— Horizontal or composite sync input (Hsync-I or Csync-I) selection
— Automatically enable Hsync blanking or Hsync signal bypass
— Vsync port input or 5-bit counter compare mode
— Select the clock source for Vsync-O

See Figure 16-3 for a detailed description of SYNCON0 register settings.

![Sync Processor Control Register 0 (SYNCON0) Diagram](image-url)
SYNC PROCESSOR CONTROL REGISTER 1 (SYNCON1)

The sync processor control register 1, SYNCON1, is located in set 1, bank 0, at address EEH. It is read/write addressable. Using SYNCON1 settings, you can:

— Generate a clock pulse for Clamp signal output
— Select “front porch” or “back porch” mode for Clamp-O
— Control Clamp-O, Vsync-O, and Hsync-O status
— Detect Hsync & Vsync polarity

See Figure 16-3 for a detailed description of SYNCON1 register settings.

| Clamp output signal generation bits (CSG1,0): |
| 00 = Inhibit clamp signal output  |
| 01 = (fosc x 2) clock pulse (250 ns at 8 MHz fosc)  |
| 10 = (fosc x 4) clock pulse (500 ns at 8 MHz fosc, 333 ns at 12 MHz fosc)  |
| 11 = (fOSC x 8) clock pulse (1 us at 8 MHz, 666 ns at 12 MHz)  |

| Front/back porch clamp-O mode selection bit: |
| 0 = Output clamp signal after rising edge of Hsync-I (front porch)  |
| 1 = Output clamp signal after falling edge of Hsync-I (back porch)  |

| Hsync polarity detection bit: (2) |
| 0 = Negative  |
| 1 = Positive  |

| Hsync output status bit: |
| 0 = Do not invert (by pass)  |
| 1 = Invert Hsync-O signal  |

| Vsync polarity detection bit: (1) |
| 0 = Negative  |
| 1 = Positive  |

| Vsync output status bit: |
| 0 = Do not invert (by pass)  |
| 1 = Invert Vsync-O signal  |

| Clamp signal output status bit: |
| 0 = Negative polarity  |
| 1 = Positive polarity  |

NOTES:
1. To check Hsync/Vsync polarity, it uses 16 clocks of timer M2 (fx/1000). If the Vsync polarity is changing, this bit will be updated after a typical delay of 2 ms, at 8 MHz fosc (1.33 ms at 12 MHz fosc)
2. The SYNCON1.0 may not be accurate when the Hsync-I is composite sync signal input.

Figure 16-4. Sync Processor Control Register 1 (SYNCON1)
SYNC PROCESSOR CONTROL REGISTER 2 (SYNCON2)

The sync processor control register 2, SYNCON2, is located in set 1, bank 0, at address EFH. It is read/write addressable. Using SYNCON2 settings, you can:

— Detect mixed and unmixed Hsync period in composite sync
— Select the pseudo sync generation enable mode
— Select the clock source for the 5-bit counter
— Sync signal output disable or enable
— SOG signal detection
— 5-bit up/down counter latch status changing detection
— $V_{DD}$ level selection for TTL sync input ports

See Figure 16-5 for a detailed description of SYNCON2 register settings.

<table>
<thead>
<tr>
<th>MSB (.7)</th>
<th>(.6)</th>
<th>(.5)</th>
<th>(.4)</th>
<th>(.3)</th>
<th>(.2)</th>
<th>(.1)</th>
<th>(.0) LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unmixed Hsync detection bit (when SYNCON0.5 is &quot;1&quot;, read only)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Mixed Hsync period with Vsync of composite sync input (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Unmixed Hsync periods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not used for KS88C6332/C6348 (only &quot;0&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-bit counter source clock ($fsync$) input selection bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = $f_{osc}$/3 (when $f_{osc}$ is 12 MHz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = $f_{osc}$/2 (when $f_{osc}$ is 8 MHz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Countable maximum Hsync pulse width: 7.85 us (when $f_{sync}$ is 4 MHz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pseudo sync generation disable bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Enable pseudo Hsync/Vsync generation (positive polarity only)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Normal sync-processor operation (by pass)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{DD}$ level selection bit for TTL sync input ports (only S3C863X):</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = When $V_{DD}$ is +5 V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = When $V_{DD}$ is +3 V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-bit up/down counter latch status changing detection bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = When the latch status is not changed or it writes &quot;0&quot; to this bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = When the latch status changing is detected</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOG detection bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = No SOG signal (when read)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Clear SOG detection 5-bit counter (when write)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Csinc-I is SOG signal (to check SOG presence, it uses 64 Csync input edge signal)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sync signal output disable bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Enable sync signal output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Inhibit sync signal output (output level is low)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTES:
1. The SYNCON2.7 is still cleared before read this bit or it has been in mixed Hsync period.
2. The SYNCON2.1 can be used to check the presence of composite sync signal input.

Figure 16-5. Sync Processor Control Register 2 (SYNCON2)
Figure 16-6. Sync Processor Block Diagram
DETECTING SYNC SIGNAL INPUT

You can detect the presence of a sync signal in two ways — directly or indirectly. The direct detection method can be implemented in read port. The indirect detection method is interrupt-driven and uses the S3C8639/C863A/C8647 sync processor hardware. These methods are explained in detail below.

Direct Detection Method

By reading the input status directly on the sync input pins Vsync-I, Hsync-I, Csync-I, you can detect the presence of the incoming sync for a corresponding output.

To enable direct sync input detection, you set SYNCON0.5 to “0” (for Vsync-I), SYNCON0.7 to “0” (for Hsync-I), and SYNCON0.7 to “1” (for Csync-I).

You then read the state of the input pin(s) over a period of time to detect transitions in the signal level(s). If a transition is detected, it can be assumed that a sync signal is present.

Indirect Detection Method

To indirectly detect vertical sync input at the Vsync-I pin, you use register settings to assign either the timer M0 capture interrupt to this pin.

For indirect detection of horizontal or composite input at the Hsync-I or Csync-I pin, you use the timer 1 input clock source to generate a timer M1 capture/overflow interrupt by capture signal from timer M2 interval or Vsync-O from sync-processor, when a signal level transition occurs. Or to detect composite sync, you can confirm to presence with checking SYNCON2.1. (This bit is used to check the presence of composite sync signal input.)

When the correct settings have been made, the application software polls for the respective interrupts to determine the presence of sync input signals, as follows:

— Indirect Vsync input detection
  Check for the occurrence of a timer M0 capture interrupt (IRQ0).

— Indirect Hsync input detection
  Check for the occurrence of a timer M1 capture/overflow interrupt (IRQ2).

— Indirect Csync input detection (SOG)
  Check for the occurrence of a timer M1 capture/overflow interrupt (IRQ2).
AUTO-DETECTING SYNC SIGNAL POLARITY

The S3C8639/C863A/C8647 sync processor lets you detect automatically the polarity of Vsync or Hsync signals by hardware. To check H/Vsync polarity, it uses 16 clocks of timer M2 ($f_{OSC}/1000$).

You can detect the polarity of Hsync signal inputted to Hsync-I port through checking SYNCON1.0 by the 5-bit counter of sync processor. If SYNCON1.0 is “1”, the polarity of the inputting Hsync signal is positive. When SYNCON1.0 is “0”, the polarity of Hsync signal is negative. But when the inputted sync signal to Hsync-I is composite sync signal (H+Vsync signal), the status of SYNCON1.0 may not be accurate.

To detect the polarity of Vsync signal, it uses SYNCON1.1. If SYNCON1.1 is “1”, the polarity of Vsync signal is positive. When the polarity of Vsync signal is negative, SYNCON1.1 is “0”. If Vsync polarity is changing, SYNCON1.1 will be updated after a typical delay of 2ms, at 8 MHz $f_{OSC}$ (1.33ms at 12 MHz $f_{OSC}$).

![Figure 16-7. Vsync Input Timing Diagram](image-url)
Positive Type

Negative Type

Hsync Pulse Width:
Min: 0.5 us
Max: 7.85 us

Hsync Frequency:
Max: 160 kHz (6.25 us)

Figure 16-8. Hsync Input Timing Diagram
EXTRACTING VSYNC OUTPUT

When the Vsync input at Hsync-I or Csync-I (P2.7) also contains Hsync signals, you must extract the Vsync component from the Hsync (or Csync) input. To do this, you use the 5-bit up/down counter.

To extract the Vsync component of the input signal, you first set the 5-bit up/down counter to operate in compare mode (SYNCON0.5 = “1”). Vsync output is enabled only when the minimum or maximum threshold value is reached.

During vertical blanking, the counter decreases until it reaches a minimum value while the Hsync-I or Csync-I signal level is negative. Or, the counter value increases until it reaches a maximum value while the Hsync-I or Csync-I signal level is positive (no overflow or borrow occurs).

The timer M1 capture interrupt (IRQ2) can be enabled to verify that the Vsync signal has been extracted successfully from the mixed input signal.

![Figure 16-9. Vsync Extraction Using an Up/Down Counter](image-url)
CLAMP SIGNAL OUTPUT

Clamp signal output (Clamp-O) must be synchronized with Hsync output. The Clamp-O signal can be transmitted to a vertically or horizontally driven integrated circuit to provide a pedestal level for image signals with programmable pulse width.

The Clamp signal is output on the “front porch” of an Hsync signal (NO SOG condition) or on the "back porch" of an Csync signal (SOG condition). You can control the polarity of clamp output signal with using SYNCON1.4. If you want to the negative pulse of clamp signal, you must set SYNCON1.4 to “0”. If you set SYNCON1.4 to "1", the polarity of clamp output signal is positive.

Figure 16-10. Clamp-O Signal (SOG and NO SOG Condition)
DIFFERENTIATING SOG FROM NO SOG

The pulse width at the Csync-I pin is different in SOG and NO SOG conditions. In a SOG condition, the pulse width at Csync-I and Hsync-I is identical. If a NO SOG condition exists, Csync-I has a wider pulse width than Hsync-I because the Csync-I pulse is truncated at the base of the pedestal level (see Figure 16-10).

To differentiate the Csync pulse, you must delay the Csync-I pulse for about 150 ns and then compare its phase with that of the Hsync-I pin signal.

To indicate a SOG condition, comparator logic for Hsync and Csync sets the SYNCON2.2 flag to “1” whenever Csync status differs from Hsync status more than 32 times at the rising edge of Hsync-I. To perform the comparison, first detect the polarity of the Hsync-I signal. Then configure the pin for positive output. (Csync-I is always positive and requires no special settings.) To recognize the SOG condition, you can poll the SYNCON2.2 status flag to detect when it is set to “1”.

---

**Figure 16-11. Sync Input at the Hsync-I and Csync-I Pins**

**Figure 16-12. Clamp-O Signal Delay Timing**
**PROGRAMMING TIP — Programming the Sync Processor**

This example shows how to program the sync processor to sample specifications. The sample program performs the following actions:

- Confirm the presence of sync signal input
- Detect the polarity of Hsync or Vsync signal input

```c
;*************************************************************/
;/** Title : Definition Flag Ram for user (00h - 0Fh)  *
;/**   *
;*************************************************************/
;
DeGausport equ 3 ;P0.3=Degauss control pin(Active High)
SuspendPort equ 4 ;P0.4=Suspend control pin(High)
Offport equ 5 ;P0.5=Off control pin(Low)
Muteport equ 7 ;P0.7=Video mute control pin(Low)
;
SelfRastport equ 0 ;P1.0=Self-Raster input pin
S1 equ 1 ;P1.1=S-correction 1
S2 equ 2 ;P1.2=S-correction 1
;
;X-Ray equ 0 ;Not used
;Rotation equ 1 ;P2.1=Pwm1 Out (Rotation)
;H-Size equ 2 ;P2.2=Pwm2 Out (H-Size)
;Contrast equ 3 ;Not used
;Brightness equ 4 ;Not used
;ACL equ 5 ;P2.5=Pwm5 Out (ACL)
Hsize_Min equ 6 ;Not used
ModelSelport equ 7 ;P2.7=Model Sel.input pin(14";L/15":H)
;
PwrkeyPort equ 0 ;P3.0=S/W power key input pin
LedPort equ 5 ;P3.5=LED control
SCL equ 6 ;P3.6=SCL(S/W IIC.bus)
SDA equ 7 ;P3.7=SDA
;
;Fixed port.
;H_Input equ 1 ;H-Sync. Input
;V_Input equ 2 ;V-Sync. Input
;Clamp equ 3 ;Clamp Output
;H_Out equ 4 ;H-Sync. Output
;V_Out equ 5 ;V-Sync. Output
;DDC_Clock equ 6 ;DDC Clock
;DDC_Data equ 7 ;DDC Data
;
HsyncIport equ 0 ;SYNCRD.0=HsyncI pin
VsyncIport equ 1 ;SYNCRD.1=VsyncI pin
VsyncOport equ 3 ;SYNCRD.3=VsyncO pin
;
```

---

**SYNC PROCESSOR**

SAMSUNG

ELECTRONICS

16-14
// Define flags

; SyncP_FGR0 EQU 00h ;SYNC-PROCESSOR
HSyncFin_FG equ 7 ;Hsync signal counting every 10ms
VSyncFin_FG equ 6 ;Vsnc signal find flag(Vsync capture interrupt)
VstblFreq_FG equ 5 ;Stable Vsync frequency input status
HstblFreq_FG equ 4 ;Stable Hsync frequency input status
NormSync_FG equ 3 ;Normal sync output mode(No pseudo sync signal)
SetSepSync_FG equ 2 ;Indicate separate sync mode
MixedSync_FG equ 1 ;Mixed sync input period(composite sync input mode)
DdcHighSpd_FG equ 0 ;DDC1 high speed mode(over 400Hz)

; SyncP_FGR1 EQU 01h
HPolarity_FG equ 7 ;Hsync polarity => 1=positive, 0=negative
VPolarity_FG equ 6 ;Vsnc polarity => 1=positive, 0=negative
HNosync_FG equ 5 ;Hsync freq. < 10KHz
VNosync_FG equ 4 ;Vsnc freq. < 40Hz
NoSync_FG equ 3 ;No Vsnc & No Hsync signal
OverHsync_FG equ 2 ;Hsync over range : over 62KHz
OverRange_FG equ 1 ;Vsnc over range : over 135Hz

; Mute_FGR EQU 02h
Vmute_FG equ 7 ;Being video mute
ChkSyncStus_FG equ 6 ;Video mute time end
MuteWaiting_FG equ 5 ;Being video mute extension
MuteRelse_FG equ 4 ;Video mute release
PwrOnWait_FG equ 3 ;Power-on mute delay(2sec)
PsncOut_FG equ 2 ;Pseudo sync output status
NormMwait_FG equ 1 ;Count mute extension time(350ms)

; Time_FGR EQU 03h
KeyDetect_FG equ 7 ;Key detecting per 10ms
DeGTime_FG equ 6 ;Degaussing time(3sec)
ChkPwrKey_FG equ 5 ;Checking power-key status per 10ms

; Status_FGR EQU 04h
SlfRasterIn_FG equ 7 ;Self-Raster mode
Recall_FG equ 6 ;Recall function(Continuous key=3sec)
UserDel_FG equ 5 ;Delete user data in EEPROM(Continuous key=5sec)
FindSyncSrc_FG equ 4

; EepRom_FGR EQU 05h
UserArea_FG equ 7 ;Checking EEPROM user data area
ClosHsync_FG equ 6 ;Searching closest Hsync mode
SavedEep_FG equ 5 ;Factory data saved EEPROM ?
EepDataRd_FG equ 4 ;EEPROM data read after mode changing
NoFactSave_FG equ 3 ;EEPROM data read after mode changing

; Tda9109_FGR EQU 05h
TdaWrite_FG equ 2 ;Write PWM data to TDA9109
TdaRead_FG equ 1 ;Read from TDA9109
Dpms_FGR EQU 09h ;
DpmsStart_FG equ 7 ;DPMS mode start(3sec after abnormal sync signal)
ChkDpmsCon_FG equ 6 ;DPMS condition input(H<10KHz,V<40Hz)
PwrOffIn_FG equ 5 ;Power Saving Mode
OffMode_FG equ 4 ;Off mode
Suspend_FG equ 3 ;Suspend mode
StandBy_FG equ 2 ;Standby mode
;
IIC_FGR EQU 0Ah
DDCCmd_FG equ 7 ;Get DDC2B+ command
LastByte_FG equ 6 ;Last Tx Data
CommFail_FG equ 5 ;Communication fail(Not ACK)
ReWrite_FG equ 4 ;Rewrite to EEPROM
ReRead_FG equ 4
Ddc2mode_FG equ 3
RevA0match_FG equ 2
Ddc2BTxmode_FG equ 1
;
CheckSum EQU 0Bh ;DDC2B+
ByteCnt EQU 0Ch ;Count number of Tx data
SendType EQU 0Dh ;Reply type
NumTxdByte EQU 0Eh ;(#(Total Tx byte -1) in master Tx mode
xXCntr EQU 0Fh ;Number of Rx data
;
 imperative part

>Title : Definition general purpose Ram (10h - BFh) *

TB1mSR EQU 10h ;1ms interval register(basic time reg.)
TB10mSR EQU 11h ;10ms(Count Hsync event signal)
TB100mSR EQU 12h ;100ms(Func valid time => 7sec)
DLY1mSR EQU 13h ;1ms(Cehcking writecycle time => 10ms)
M10mSR EQU 14h ;10ms(Checking mute time => sec,350ms)
S100mSR EQU 15h ;100ms(Saving start time => 2sec)
DG100mSR EQU 16h ;100ms(Degaussing time => 3sec)
DPMS100mSR EQU 17h ;Check DPMS start(after No Sync : 3sec)
ChkSRasTime EQU 18h ;Check self-raster input(maintain 70ms ?) ;

; HCount EQU 20h ;Double byte(even address + odd address)
;
HFreqStCnt EQU 21h
FreqSpCnt EQU 22h
AverageHf EQU 23h
HfHighNew EQU 24h
HfLowNew EQU 25h ;Current value of Hsync freq high byte
HfHighData EQU 26h
HfLowData EQU 27h ;Saved value of Hsync freq high byte

16-16
; ex> if, HfHighData=#x2h, 
; HfLowData=#58h => Hsync frequency = 258h=60.0Khz
Vcount EQU 30h ; total number of timer0 counter within Vsync period
; EQU 31h ; Double Byte
NoVTime EQU 32h ; Checking the sustaining time of no Vsync signal
; if NoVTime > 30ms(under 33Hz) => Mute
VFreqCHigh EQU 33h ; Temp storage of Timer0(Vsync) overflow count
T0OvfCnt EQU 34h ; No vsync-int. service if DDC1 high freq. Mode
VFreqData EQU 35h ; Current Vsync freq
VclkCnt EQU 36h ; Saved Vsync freq.
VpolaCnt EQU 37h ; For auto recovery of DDC mode (DDC2B -> DDC1)
; PolaCntr EQU 38h ; Count number of polarity checking
VPolaCntr EQU 39h ; Increment when positive polarity
; UmemoNo EQU 40h ; Matched number of user mode
FmodeNo EQU 41h ; Factory mode
ReadData EQU 42h ; Readed Data from EEPROM
; EP_BPlus EQU 50h ; EEPROM & RAM data
EP_CONTRAST EQU 51h ; KA2504
EP_RGain EQU 52h
EP_GGain EQU 53h
EP_BGain EQU 54h
EP_CoffBRIGHT EQU 55h ; KA2504
EP_RCutoff EQU 56h
EP_GCutoff EQU 57h
EP_BCutoff EQU 58h
EP_ACL EQU 59h ; PWM5
; EdidAddr EQU 80h ; Page1 RAM register
; ; // WORKING REGISTERS -> GENERAL RAM
; ; R14 EQU EepSubAddr ; Sub address of EEPROM/TDA9109
; R15 EQU EepWrData ; Data to write in EEPROM/TDA9109
; ; ; R14 EQU PreAmpSubAddr ; Sub address
; R15 EQU PreAmpCtrlData ; Data address
; ; // Buffer for DDC2B+ protocol
; MBusBuff EQU 0B0h ; For DDC2B+(00h-0BFh:16-byte)
AbusDstAddr EQU 0B0h
AbusSrcAddr EQU 0B1h
AbusPlength EQU 0B2h
AbusCommand EQU 0B3h
; ; EQU 0BFh ;
SYNC PROCESSOR S3C8639/C863A/P863A/C8647/F8647

;###############################################################################
; DDC EDID AREA : 00h - 7Fh (128-Byte : Page 1)
;###############################################################################

;****************************************************************************
; Title : Define Control register's flags
;****************************************************************************

// IIC.bus Control Register
// DCON
BUFEN equ 3 ;Tx/Rx pre-buffer data register enable(0:Normal, 1: Pre-buffer Mode)
DDC1MAT equ 2 ;DDC address match(0:Not match, 1:match)
DDC1EN equ 1 ;DDC1 Tx mode enable
SCLF equ 0 ;SCL falling edge detection

; DCCR(DDC Clock Control Reg)
DTXACKEN equ 7
DCLKSEL equ 6
DINTEN equ 5
DPND equ 4
CCR3 equ 3
CCR2 equ 2
CCR1 equ 1
CCR0 equ 0

; DCSR0(DDC Control/Status Reg0)
DMTX equ 7
DSTX equ 6
DBB equ 5
DDCEN equ 4
DAL equ 3
DADDMAT equ 2
DRXACK equ 0 ;Not Used for the KS88C6332/48/P6348

; DCSR1(DDC Control/Status Reg1)
STCONDET equ 2 ;IIC-Bus Stop Condition Detect
DBUFEMT equ 1 ;Data buffer empty status
DBUFFUL equ 0 ;Data buffer full status

; TBDR ;Transmit pre-buffer data register
;RBDR ;Receive pre-buffer data register
;DDSR ;DDC data shift register

; Sync-processor control register
// SYNCON0
SIS equ 7 ;Sync input selection(0:Hsync, 1:Csync)
HBLKEN  equ  6 ; Hsync Blanking Enable (0:Bypass, 1:Blanking)

VOSS   equ  5 ; VsyncO source selection (0:Vsync, 1:5-bit compare out)

; // SYNCON1
CLMP1  equ  7 ;
CLMP0  equ  6 ; ClampO pulse width
FBPS   equ  5 ; Front/Back porch selection (0:back, 1:front)
CLMPS  equ  4 ; ClampO polarity control (0:negative, 1:positive)
VOS    equ  3 ; VsyncO polarity control (0:by-pass, 1:invert)
HOS    equ  2 ; HsyncO polarity control (0:by-pass, 1:invert)
VPOL   equ  1 ; Vsync polarity detection (0:negative, 1:positive)
HPOL   equ  0 ; Hsync polarity detection (0:negative, 1:positive)

; // SYNCON2
UNMIXHSYNC equ  7 ; Unmixed Hsync detection
CCSS   equ  5 ; 5-bit counter clock selection (0:fosc/2, 1:fosc/3)
PSGEN  equ  4 ; Pseudo Sync Generation Disable
SYNOD  equ  3 ; Sync Signal Output Disable
SOGI   equ  2 ; SOG check
UP5BSDET equ  1 ; 5-Bit Up/Down Counter Status Changing Det.
VDDLS  equ  0 ; VDD Level Selection (0:VDD=5V, 1:VDD=3V)

; // Watch-dog(Basic) Timer
BTCLR  equ  1 ;

; // Timer M0
T0EDGSEL equ  4 ; Timer M0 Capture Mode Selection
T0CLR   equ  3 ; Counter clear
T0OVINT equ  2 ; Overflow interrupt enable
T0INT   equ  1 ; Capture enable
T0CAPSEL equ  0 ; Capture input selection

; // Timer M1
T1CAPSEL equ  7 ; Capture signal source selection
VEDGSEL equ  6 ; VsyncO capture edge selection (0:rising, 1:falling)
T1CAPEN equ  5 ; Capture interrupt enable
T1PND   equ  4 ; Capture interrupt pending flag
T1CLR   equ  3 ; Counter clear
T1OVFINT equ  2 ; Overflow interrupt enable

; // Timer M2
T2INT   equ  2 ;
CAPINTV1 equ  1 ;
CAPINTV0 equ  0 ;
SYNC PROCESSOR S3C8639/C863A/P863A/C8647/F8647

:// Pre-Amp Sub-address Mapping

; Slave address=DCh

PSubA_Cont EQU 00h ; Contrast control
PSubA_SBnBr EQU 01h ; Bit7=Soft Blanking (1:ON, 0:OFF)
; Bit6-5=Cut-off control offset current switch
; (CS2:160uA, CS1:80uA)
; Bit4-0=Brightness control

PSubA_RGain EQU 02h ; R Gain Control
PSubA_GGain EQU 03h ; G Gain Control
PSubA_BGain EQU 04h ; B Gain Control
PSubA_CoBr EQU 05h ; Cut-off brightness control
PSubA_RCo EQU 06h ; R Cut-off control
PSubA_GCo EQU 07h ; G Cut-off control
PSubA_BCo EQU 08h ; B Cut-off control
PSubA_Sw EQU 0Ah ; Blanking On-Off Control

:// EEPROM Address Mapping

EPA_CoBr EQU 0F6h ; KA2504 Cut-off Brightness
EPA_Cont EQU 0F7h ; KA2504 Cut-off Contrast
EPA_RGain EQU 0F8h ; KA2504 R-Gain
EPA_GGain EQU 0F9h
EPA_BGain EQU 0Fah
EPA_RCo EQU 0FBh ; KA2504 R-Cut off
EPA_GCo EQU 0FCh
EPA_BCo EQU 0FDh
EPA_ACL EQU 0Feh

; DDC2ab communication command code

I_Reset equ 0F0h
I_IdReq equ 0F1h
I_AsgnAdr equ 0F2h
I_CapReq equ 0F3h
I_ApplRprt equ 0F5h
I_Attention equ 0E0h
I_IdReply equ 0E1h
I_CapReply equ 0E3h
I_GetVCP equ 01h
I_VCPFPReply equ 02h
I_SetVCP equ 03h
I_GetTiming equ 07h
I_ResetVCPF equ 09h
I_DisableVCPF equ 0Ah
I_EnableVCPF equ 0Bh
SaveCurrSet equ 58h
I_TimingReport equ 4Eh
I_GetEdid equ 54h
SaveEdid equ 69h
DelUser equ 50h
AllModeSve equ 52h
SaveColorSet equ 7Dh
BrtContMax equ 0D2h

;*******************************************************************************
;** Title : Interrupt Vector Table                          *
;**                                                          *
;*******************************************************************************

ORG 00E0h
VECTOR TM0Ovf_Int ;Timer M0 overflow int.(IRQ0)
VECTOR VSyncDet_Int ;Timer M0 capture int.(IRQ0)
VECTOR TM2Intv_Int ;Timer M2 interval int.(IRQ2)

ORG 00E8h
VECTOR TM1Cap_Int ;Timer M1 capture int.(IRQ1)

ORG 00EAh
VECTOR DDCnFA_Int ;DDC IIC-bus Tx/Rx int.(IRQ3)

;*******************************************************************************
;** Title : Main Program start from here                           *
;**                                                          *
;*******************************************************************************

RESET: DI ;Disable interrupt
CLR PP ;Source, Destination = page0
CLR SYM ;Disable fast interrupt
LD SPL,#0FFh ;Stack pointer
SRP #0C0h ;Working reg. area
LD IMR,#00001111B ;Timer M0,M1,M2, & DDC Int.
               ;bit7 -> not used
               ;bit6 -> not used
               ;bit5 -> not used
; bit4 -> not used
; bit3 -> IRQ3 DDC Int.
; bit2 -> IRQ2 Timer M1 Cap. Int.
; bit1 -> IRQ1 Timer M2 Int.
; bit0 -> IRQ0 Timer M0 Cap. & Ovf. Int.
SB0 ; Select bank 0
CLR EMT ; 0 wait, Internal stack area
LD IPR,#00010001B ; Group priority(int) undefined(A>B>C)
LD CLKCON,#18h ; CPU=fx(no division)
LD BTCON,#0A2h ; WatchDog Timer Disable
LD WDTCON,#00h ; Watchdog Time = tBTOVF
LD STOPCON,#5Ah ; Stop Function Disable

; Initialize Sync-processor control register
LD SYCON2,#10100000B ; Bit7=read only
LD SYCON1,#11000000B ; ClampO=negative polarity
LD SYCON0,#00100000B ; Automatic Hsync blanking,
; syncO source=VsyncI port
LD PHGEN,#83 ; Pseudo Hsync = 48.19KHz
LD PVGEN,#101 ; Pseudo Vsync = 59.64KHz

; Initialize Timer control register
LD TM0CON,#10001111B ; Timer0 clock source=@8MHz/8=1MHz(1us)
; Capture rising mode
; Enable capture/overflow interrupt
; Capture source=Vsync output path
; from sync-processor
LD TM1CON,#00001100B ; source=HsyncI
; Capture disable
; Capture Source=Timer2 interval time*10(10ms)
; Enable capture interrupt
LD TM2CON,#00111101B ; Timer2 interval=@8MHz/(8*1000)=1ms

**************************************************
*******        DDC2Bi service routine         *******
**************************************************
ChkDDC2Bi TM IIC_FGR,#01<<DDCCmd_FG ; DDCCmd_FG=0 ?
JR Z,DDC2BPrtn
SB1
TM DCSR0,#01<<DBB ;End of Rx(Not usy)  
JR NZ,DDC2BPrtn  
SB0  
CLR RxXcntr ;RxXcntr <- #00h  
AND IIC_FGR,#0FFh-(01<<DDCcmd_FG) ;ActIIC_FG <- 0  
CALL DDC2biRxd  
DDC2BPrtn  
SB0  
RET  

;*************************************************************************************  
;*****     DDC mode checking service routine     *****  
;*************************************************************************************  
ChkDdcRecover TM IIC_FGR,#01<<Ddc2BIn_FG ;Already DDC2B mode  
JR NZ,ChkDdcRtn  
SB1  
TM DCON,#01<<SCLF  
JR NZ,ChkDdc2In  
ChkDdcRtn SB0  
RET  

; ChkDdc2In TM DCON,#01<<DDC1EN ;DDC1 Tx mode  
JR NZ,ChkDdcRtn  
CP VclkCnt,#128  
JR ULE,ChkDdcRtn  
OR DCON,#01<<DDC1EN ;Switch back to DDC1 from DDC2B  
AND DCON,#0FFh-(01<<SCLF)  
AND IIC_FGR,#0FFh-(01<<Ddc2BIn_FG)  
CLR VclkCnt  
LD TBDR,#00h  
LD PP,#11h  
LD EdidAddr,#01h  
CLR PP  
JR ChkDdcRtn  

; DDC2BiRxd NOP  
RET  

;*******************************************************************************  
;***** Title : Check H/V presence, a kind of sync source and mode changing  
;************************************************************************************  
ChkHVPres: TM SyncP_FGR0,#01<<HSyncFin_FG ;Checking period of Hsync frequency  
;is 10ms by timer M1/2 interrupt  
JR Z,ChkPresnVsync  
AND SyncP_FGR0,#0FFh-(01<<HSyncFin_FG) ;Every by 10ms  
CALL Chk10msTimer ;Time counter(100ms,1sec,2sec,3sec,7sec)  

;************************************************************************************
\begin{verbatim}
; CheckHV_Range  CALL   ChkHnosyncRange
   JP   C,SyncOffState ;Under 10KHz?
;
; ChkHsyncData  CALL   UHSyncChk ;Check changing rate of Hsync frequency
   JP   C,SyncOffState ;If changing rate > 00Hz
   CALL   UPolaChk ;Check Polarity data
   JP   C,SyncOffState
;
; ChkPresnVsync  CP   NoVTime,#25 ;If NoVTime > 25ms(under 40Hz) => Mute
   JP   UGT,ChkVsyncSrc
   TM   SyncP_FGR0,#01<<VSyncFin_FG ;Being Vsync signal(Vsync interrupt)?
   JP   Z,ChkHVPrtn
   AND   SyncP_FGR0,#0FFh-(01<<VSyncFin_FG)
   TM   Status_FGR,#01<<FindSyncSrc_FG ;Already find sync source?
   JP   NZ,SkipChkVsyncSrc
   OR   Status_FGR,#01<<FindSyncSrc_FG ;First checking source is 
       ;composite sync then seperate sync
   AND   SyncP_FGR0,#0FFh-(01<<SetSepSync_FG)
   JP   ChkSOGsignal
;
; ChkVsyncSrc  CLR   NoVTime
   TM   Status_FGR,#01<<FindSyncSrc_FG ;New sync source?
   JR   NZ,NoPresVsync
   TM   SyncP_FGR0,#01<<SetSepSync_FG ;No Vsync input
   JR   NZ,NoPresVsync
   OR   SyncP_FGR0,#01<<SetSepSync_FG
   AND   SYNCON0,#0FFh-(01<<VOSS) ;Changing to separate-sync mode
   AND   SYNCON0,#0FFh-(01<<HBLKEN)
   OR   TM1CON,#01<<T1CAPEN ;Enable Timer M1 capture mode
   RET
;
; NoPresVsync  CALL   ClrSyncSrcFlag
   OR   SyncP_FGR1,#01<<VNosync_FG ;VNosync_FG <- 1
   OR   Dpms_FGR,#01<<ChkDpmsCon_FG ;ChkDpmsCon_FG <- 1
       ;(Start DPMS check)
   TM   SynccP_FGR1,#01<<HNosync_FG ;No Hsync & No Vsync
   JP   Z,SyncOffState
   OR   SyncP_FGR1,#01<<NoSync_FG
   JP   SyncOffState ;Mute
;
; SkipChkVsyncSrc:
; ChkSOGsignal  TM   SYNCON2,#01<<SOGI ;Check SOG signal input?
   JR   Z,CountVfreq
   AND   SYNCON1,#01<<FBPS ;Back porch
;
; CountVfreq  CALL   NormalVfCnt ;Calculate Vsync freq.
   CALL   UVSyncChk ;Check changing rate of sync frequency
   JP   C,SyncOffState
   CALL   ChkHVRange ;Check video signal range
   JP   C,SyncOffState ;Range Over!
;
; StableFreqIn  TM   Mute_FGR,#01<<MuteRelase_FG
\end{verbatim}
JR Z, ChkMuteTime
RET
;
ChkMuteTime
TM Mute_FGR,#01<<ChkSyncStus_FG ;Checking sync status every Vsync
;signal event
JP Z, VMuteRtn
TM Mute_FGR,#01<<MuteWaiting_FG ;Already previous mute-release step ?
JP NZ, MuteRelease
TM SyncP_FGR0,#01<<HstblFreq_FG ;Stable Hsync input ?
JR NZ, ChkStblVsync
OR Mute_FGR,#01<<ChkSyncStus_FG
RET
;
ChkStblVsync
TM SyncP_FGR0,#01<<VstblFreq_FG ;Stable Vsync input ?
JR NZ, NormalSyncOut
OR Mute_FGR,#01<<ChkSyncStus_FG
RET
;
NormalSyncOut
CALL PosiPolOut ;Adjust polarity to positive
CALL PolaUpDate ;Polarity data update
LD SYNCON2,#10110000B ;Normal sync operation
AND Mute_FGR,#0FFh-(01<<PsyncOut_FG)
OR SyncP_FGR0,#01<<NormSync_FG ;Re-start polarity checking
;
LoadDAC
CALL UpdateHDuty ;Adjust Hsync duty value in KB2511
CALL AdjModeSize ;Adjust mode size according to Hsync freq. range
CALL B_PlusOut ;Adjust B+ reference value in KB2511
CALL S_Correct ;Adjust S-correction port
;
OR EepRom_FGR,#01<<EepDataRd_FG ;Load PWM data
;(processing in 'MAIN' routine)
OR Mute_FGR,#01<<MuteWaiting_FG ;Start time checking for mute extension
AND Mute_FGR,#0FFh-(01<<ChkSyncStus_FG)
TM Mute_FGR,#01<<PwrOnWait_FG
JR NZ, MuteDelay
CLR M10mSR ;2sec
RET
MuteDelay
OR Mute_FGR,#01<<NormMwait_FG ;Load image data -> 350ms delay
; -> Mute release
CLR M10mSR
RET
;
MuteRelease
AND Mute_FGR,#0FFh-(01<<ChkSyncStus_FG)
AND Mute_FGR,#0FFh-(01<<MuteWaiting_FG)
AND Mute_FGR,#0FFh-(01<<Vmute_FG)
OR Mute_FGR,#01<<MuteRelse_FG
OR P0,#01<<Muteport ;P0.7 <- 1 : mute port release
;
LD R14,#PSubA_SBnBr ;Off soft blanking(bit7 <- 0, KA2504)
LD R15,#00h ;R14= device(KA2504) sub-address,
;R15= control data
CALL Preamp_RGB_Drv
SYNC PROCESSOR

RET
SyncOffState:
  TM Status_FGR,#01<<SfRasterIn_FG ;Self-raster mode?
  JR NZ,VMuteRtn
  TM Mute_FGR,#01<<PsyncOut_FG ;Already pseudo sync output mode?
  JR NZ,VMuteRtn

;VideoMute
  AND P0,#0FFh-(01<<Muteport) ;V-Mute(P0.7=Active low) <- 0
  AND SYNCON1,#11110011B ;Pseudo Sync=only positive Pol.
  LD SYNCON2,#101000000B ;Pseudo sync Enable
  LD RHGEN,#83 ;Pseudo Hsync = 48.19KHz
  LD PVGEN,#101 ;Pseudo Vsync = 59.64KHz

;MutePross
  LD R14,#PSUBA_SBnBr ;Soft blanking(bit7 <- 1)
  CALL Preamp_RGB_Drv

; OR P1,#01<<S1 ;P1.2(S1) <- High (Free run=48KHz)
; AND P1,#0FFh-(01<<S2) ;P1.3(S2) <- Low

; OR Mute_FGR,#01<<Vmute_FG ;Set video mute flag
; OR Mute_FGR,#01<<PsyncOut_FG ;Set pseudo sync output flag
AND Mute_FGR,#0FFh-(01<<MuteRelse_FG) ;Clear mute release status flag
AND Mute_FGR,#0FFh-(01<<MuteWaiting_FG) ;Clear mute extension start flag
AND Mute_FGR,#0FFh-(01<<NormMwait_FG)
AND SyncP_FGR0,#0FFh-(01<<NormSync_FG) ;Clear sync relation register data
AND SyncP_FGR0,#0FFh-(01<<HstblFreq_FG)
AND SyncP_FGR0,#0FFh-(01<<VstblFreq_FG)
OR Mute_FGR,#01<<ChkSyncStus_FG
CALL ClrSncSrcFlag ;Return to default sync source checking mode

VMuteRtn RET

;ClrSncSrcFlag
  TM SyncP_FGR0,#01<<DdcHighSpd_FG ;DDC1 high speed(over 400Hz) mode?
  JR NZ,ClrFlagRtn
  OR SYNCON0,#01<<VOSS ;VsyncO=5-bit compare output for composite sync
  OR SYNCON0,#01<<HBLKEN
  AND TM1CON,#0FFh-(01<<T1CAPEN) ;Disable Timer1 capture mode
  AND Status_FGR,#0FFh-(01<<FindSncSrc_FG)
  AND SyncP_FGR0,#0FFh-(01<<SetSepSync_FG)
  CALL ClrSncSrcFlag ;Return to default sync source checking mode

ClrFlagRtn RET ;Sync source checking: composite -> separate ->composite

/**************************************************************************
//** Title : S-Correction
/**************************************************************************

// Hsync freq. < 35KHz => S1=L, S2=L
// (R5=xxKHz) < 40KHz => S1=L, S2=H
// < 49KHz => S1=H, S2=L
// < 60KHz => S1=H, S2=H

//**************************************************************************

H_CountLoad LD R4,HfHighData
AND R4,#00000011B
LD R5,HfLowData

SAMSUNG
ELECTRONICS
DIV RR4,#10
RET
S_Correct CALL H_CountLoad
CP R5,#35 ;Under 35Khz
; RET

;>Title : Adjust Horizontal Duty Cycle(TDA9109)
; Hsync freq. < 35KHz => 00h(TDA9109 address)=48h
; < 41KHz => 00h=49h
; < 46KHz => 00h=4Ah
; < 52KHz => 00h=4Bh
; < 56KHz => 00h=4Ch

UpdateHDuty CALL H_CountLoad
CP R5,#35 ;Under 35KHz
; RET

;>Title : Adjust Mode Size(PWM6, 14/15")
; 15"
; H_sync freq. < 41KHz => PWM6=#1Bh
; (R5=xxKHz) < 46KHz => PWM6=#4Ah
; < 50KHz => PWM6=#50h
; < 56KHz => PWM6=#91h
; < 62KHz => PWM6=#CDh

AdjModeSize CALL H_CountLoad
CP R5,#41 ;Under 41Khz
; RET

;>Title : Adjust B_Plus Output

B_PlusOut LD R6,EP_BPlus ;EP_BPlus=KA2511 B+ referance data
CALL H_CountLoad
; CP R5,#41 ;Under 41Khz
; 
ADD R6,#0 ;51KHz - 55.9KHz
ModeBplusOut LD R14,#0Bh ;B+ sub-address
LD R15,R6 ;B+ data
OR Tda9109_FGR,#01<<TdaWrite_FG

;** Title : Adjust Horizontal Duty Cycle(TDA9109)
; Hsync freq. < 35KHz => 00h(TDA9109 address)=48h
; < 41KHz => 00h=49h
; < 46KHz => 00h=4Ah
; < 52KHz => 00h=4Bh
; < 56KHz => 00h=4Ch

;>Title : Adjust Mode Size(PWM6, 14/15")
; 15"
; H_sync freq. < 41KHz => PWM6=#1Bh
; (R5=xxKHz) < 46KHz => PWM6=#4Ah
; < 50KHz => PWM6=#50h
; < 56KHz => PWM6=#91h
; < 62KHz => PWM6=#CDh

;** Title : Adjust B_Plus Output

;** Title : Adjust Mode Size(PWM6, 14/15")
CALL WriteCycle
RET

;**********************************************************************************************
;** Title : Check mode change and h/v frequency range for mode detection
;** Normal Fh <= 500Hz
;** Normal Fv <= 1Hz
;** if there is in the sync range , Set carry
;** HfHighNew --> bit7---> H-polarity
;** bit6---> V-polarity
;**
;** Inputs: R0,R1
;** Outputs:
;** Preserves:
;** Corrupts:
;**********************************************************************************************

;***************************************************************************
;** Title : Check New polarity and old polarity
;**
;***************************************************************************
; UPolAChk TM SyncP_FGR0,#01<<NormSync_FG ;Pseudo sync output status ?
JR Z,PolChkRtn
;
LD R0,SyncP_FGR1
AND R0,#11000000b
LD R1,HfHighData ;bit 7 & 6 are used for POL.
AND R1,#11000000b
CP R0,R1
JR NE,PolaUpDate ;Compare hv_polarity.
;
PosiPolOut TM SyncP_FGR1,#01<<HPolarity_FG
JR Z,InvHPolA
AND SYNCON1,#0FFh-(01<<HOS) ;HOS=HsyncO status(polarity) control bit

ChkVPola TM SyncP_FGR1,#01<<VPolarity_FG
JR Z,InvVPola
AND SYNCON1,#0FFh-(01<<VOS) ;VOS=VsyncO status(polarity by-pass)

PolChkRtn RCF
RET
;
InvHPolA OR SYNCON1,#01<<HOS ;HsyncO=Invert HsyncI signal
JR ChkVPola

InvVPola OR SYNCON1,#01<<VOS
JR PolChkRtn
;
PolaUpDate LD R4,HfHighNew
AND R4,#00000011b
LD R5,SyncP_FGR1
AND R5,#11000000b ;Masking except polarity flag
OR R4,R5
LD HfHighData,R4
SCF
RET
;
FLICT***************************************************************************
DECL Title : Compare Vertical Frequency
DECL***************************************************************************
DECL***************************************************************************
DECL UVSyncChk:  LD R0,VIcurrNew
DECL CP R0,VFreqData
DECL JR ULT,RevVSub
DECL SUB R0,VFreqData
;
DECL CmpFvRng CP R0,#1 ;Compare 1Hz
DECL JR UGT,UpdateVfData
DECL OR SyncP_FGR0,#01<<VstblFreq_FG ;Set stable Vsync signal input flag
DECL RCF
DECL RET
;
DECL RevVSub LD R0,VFreqData ;VFreqData= saved stable Vsync frequency
DECL SUB R0,VIcurrNew ;VIcurrNew= inputted new Vsync frequency
DECL JR CmpFvRng
;
DECL UpdateVfData AND SyncP_FGR0,#0FFh-(01<<VstblFreq_FG)
DECL LD VFreqData,VIcurrNew ;Refresh v-frequency
DECL SCF
DECL RET
;
DECL***************************************************************************
DECL Title : Compare Horizontal Frequency
DECL***************************************************************************
DECL***************************************************************************
DECL***************************************************************************
DECL UHSyncChk LD R0,HfLowNew
DECL SUB R0,HfLowData ;R0 = |HfLowNew - HfLowData|
DECL LD R1,HfHighNew
DECL LD R2,HfHighData
DECL AND R2,#00000011b ;R1 = |HfHighNew - HfHighData|
DECL SBC R1,R2
DECL JR CmpHrSub
;
DECL CmpFhRng CP R1,#00h ;HfHighNew /= HfHighData ?
DECL JR NE,UpdateHfData
;
DECL TM SYNCON0,#01<<VOSS ;Composite sync signal ?
DECL JR Z,ChkNormHrRng
DECL CP R0,#9 ;Compare 1KHz
DECL JR UGT,UpdateHfData
DECL JR StblHsyncIn
;
DECL ChkNormHrRng CP R0,#4 ;Changing rate < 500Hz ?
DECL JR UGT,UpdateHfData
StblHsyncIn OR SyncP_FGR0,#01<<HstblFreq_FG ;Set stable Hsync signal input flag
LD R4,HfHighNew
LD R5,HfLowNew
DIV RR4,#10
LD AverageHf,R5
RCF
RET

RevHSub
LD R0,HfLowData
SUB R0,HfLowNew
LD R1,HfHighData
LD R2,HfHighNew
AND R1,#00000011b
SBC R1,R2
JR CmpFhRng

UpdateHfData AND SyncP_FGR0,#0FFh-(01<<HstblFreq_FG)
AND HfHighData,#11000000B
OR HfHighData,HfHighNew
LD HfLowData,HfLowNew ;Refresh h-frequency
SCF
RET

;***************************************************************************
;** Title : Normalize vertical Counter
;** counter source = @8M/8 =1us(Timer0 capture mode)
;** x = Vcount(70h,71h)
;** Fv=1000000 / x
;** Inputs: Vcount,Vcount+1(=net #T0CNT)
;** Outputs: VfCurrNew
;** Preserves:
;** Corrupts:
;***************************************************************************
;Calculation method-1.
NormalVfCnt CLR R4 ;Vsync interval time(Vcount=#0XXXXXus)
LD R5,Vcount ;1/#0XXXXX us(frequency) = 1000000/0XXXXX
DIV RR4,#10 ; (1000000/10) / (0XXXXX/10)
LD R6,R5 ;High = (100000 /2) / (00XXXX/2)
LD R5,Vcount+1
DIV RR4,#10 ;Vcount/10
LD R7,R5 ;Low
RCF
RRC R6 ;RR6(time=XX.XXms)/2
RRC R7
LD R4,#0C3h ;#C350=50000
LD R5,#50h
CLR R0

;ContiSub
RCF
SUB R5,R7
SBC R4,R6
```assembly
JR    C,FiniVfCal
ADD   R0,#1 ;R0=xxHz(Frequency=1/Time)
JR    C,NoVsyncSignal ;Overflow(over 256Hz) ?
JR    ContiSub
FiniVfCal
LD    VfCurrNew,R0
RET
;
NoVsyncSignal
CLR   VfCurrNew ;VfCurrNew <- 00Hz
RET
;
.getOrElse(): Check H/V Sync Range
.getOrElse(): 27Khz(10Eh) < Fh < 62KHz(15" 26Ch)
.getOrElse(): 40Hz(28h) < Fv < 135Hz(87h)
.getOrElse():
.getOrElse(): Inputs:
.getOrElse(): Outputs:
.getOrElse(): Preserves:
.getOrElse(): Corrupts:

ChkHVRange:
LD    R2,HfHighNew
LD    R3,HfLowNew
RCF
SUB   R3,#0Eh ;#10Eh=27KHz
SBC   R2,#01
JR    C,OverHfRange
LD    R2,HfHighNew
LD    R3,HfLowNew
RCF
SUB   R3,#6Ch ;#26Ch=62KHz
SBC   R2,#02
JR    NC,OverHfRange
AND   SyncP_FGR1, #0FFh-(01<<NoSync_FG)
AND   SyncP_FGR1, #0FFh-(01<<OverHsync_FG)
;
ChkVfreqRange
CP    VfCurrNew,#40 ;40Hz
JR    ULT,NoVsncIn
CP    VfCurrNew,#135 ;135Hz
JR    UGT,OverRange
;
NormHVsyncIn
AND   SyncP_FGR1, #0FFh-(01<<VNosync_FG)
AND   SyncP_FGR1, #0FFh-(01<<NoSync_FG)
AND   SyncP_FGR1, #0FFh-(01<<OverRange_FG)
AND   P0,#0FFh-(01<<Suspndport) ;Stop Suspend
NOP
NOP
NOP
NOP
OR    P0,#01<<Offport ;Stop Off
;
TM    Dpms_FGR,#01<<PwrOffIn_FG ;Power off/suspend mode ?
JR    Z,ClrDpmsFlags
```
PreAmpRelease CALL CtrlPreAmp ;Control pre-amp
CALL InitialKB2511 ;TDA9109(Initializing) G
;
CALL VideoMute ;Set power on condition
;
ClrDpmsFlags CLR DPMS100mSR ;Clear dpms checking counter
AND Dpms_FGR,#00000011B ;Clear DPMS flags
TM Status_FGR,#01<<SfRasterIn_FG ;Self Raster in ?
JR Z,ChkHVRrtn
CALL SfRasterEnd ;Release self_raster input mode

ChkHVRrtn RCF
RET
;
OverHfRange OR SyncP_FGR1,#01<<OverHsync_FG
OR SyncP_FGR1,#01<<OverRange_FG
JR ChkVfreqRange

OverRange OR SyncP_FGR1,#01<<OverRange_FG
JR OverRngRtn
;
NoVsyncIn OR SyncP_FGR1,#01<<VNosync_FG
OR Dpms_FGR,#01<<ChkDpmsCon_FG ;Start checking the maintaining time of ;dpms mode
TM SyncP_FGR1,#01<<HNosync_FG ;No Hsync & No Vsync ?
JR Z,OverRngRtn
OR SyncP_FGR1,#01<<NoSync_FG

OverRngRtn SCF
RET
;
ChkHNoSyncRange LD R0,HfHighNew
LD R1,HfLowNew
SUB R1,#64h ;#64h=100=10.0KHz
SBC R0,#00
JR c,HNosyncRange
AND SyncP_FGR1,#0FFh-(01<<HNosync_FG)
AND SyncP_FGR1,#0FFh-(01<<NoSync_FG)
RET

HNosyncRange OR SyncP_FGR1,#01<<HNosync_FG ;HNosync_FG <- 1
OR Dpms_FGR,#01<<ChkDpmsCon_FG ;Start DPMS condition counting
RET
//***************************************************************************
//** Title : Check H/V polarity for every 5ms
//**
//** Inputs: Vsync port data, SYNCON1.0
//** Outputs: HPolarity_FG/VPolarity_FG (in SyncP_FGR1)
//** Preserves:
//** Corrupts:
//***************************************************************************

ChkHVPol: TM SYNCON0,#01<<VOSS ;Composite sync(H+V) ?
JR Z,ChkSepSyncPol
TM SYNCON1,#01<<HPOL ;Hsync polarity=positive ?
JR NZ,HVposiPola
AND SyncP_FGR1,#0FFh-(01<<HPolarity_FG)
AND SyncP_FGR1,#0FFh-(01<<VPolarity_FG)
OR TM1CON,#01<<VEDGSEL
OR TM0CON,#01<<T0EDGSEL
JR ChkHVPol_rtn

HVposiPola OR SyncP_FGR1,#01<<HPolarity_FG
OR SyncP_FGR1,#01<<VPolarity_FG
AND TM1CON,#0FFh-(01<<VEDGSEL)
AND TM0CON,#0FFh-(01<<T0EDGSEL)
JR ChkHVPol_rtn

ChkSepSyncPol TM SYNCON1,#01<<HPOL ;Separated sync signal
JR NZ,HposiPola
AND SyncP_FGR1,#0FFh-(01<<HPolarity_FG)
JR ChkVsyncPol

HposiPola OR SyncP_FGR1,#01<<HPolarity_FG

ChkVsyncPol CP SYNCON1,#01<<VPOL
JR Z,VNegaPola
OR SyncP_FGR1,#01<<VPolarity_FG
AND TM1CON,#0FFh-(01<<VEDGSEL)
AND TM0CON,#0FFh-(01<<T0EDGSEL)
RET

VNegaPola AND SyncP_FGR1,#0FFh-(01<<VPolarity_FG)
OR TM1CON,#01<<VEDGSEL
OR TM0CON,#01<<T0EDGSEL
ChkHVPol_rtn RET

;***************************************************************************
;/** Title : Timer 0 overflow interrupt(interval=256us(1us*256))
;**
;** Inputs: fosc(@8MHz)/8=1us(Timer0 clock source)
;** Outputs: Overflow count for Vsync interval
;** Preserves:
;***************************************************************************
//** Corrupts:                                                                                           
//**************************************************************************************************
TM0Ovf_Int: PUSH PP  ;256us interval interrupt
CLR PP
INC VFreqCHigh  ;Overflow counter for Vsync freq. calculation
INC T0OvfCntr
POP PP
IRET

;//**************************************************************************************************

//** Title : Timer 1 capture interrupt(T1DATA=Number of Hsync signal for 10ms)
//**
//** Inputs: Hsync signal(event counter source)
//** Outputs: Number of Hsync signal for 10ms(separate sync input mode)
//** Preserves:
//** Corrupts:
//**************************************************************************************************
TM1Cap_Int: SB0
PUSH PP
CLR PP
AND TM1CON,#0FFh-(01<<T1PND)  ;Pending clear
TM TM1CON,#01<<T1CAPEN  ;Composite sync signal ?
JR Z,T1CapRtn

LD HfHighNew,TM1DATAH  ;TM1DATA is 12-bit event counter for 10ms
LD HfLowNew,TM1DATAL
CLR AverageHf
OR SyncP_FGR0,#01<<HSyncFin_FG  ;Set Hsync signal find flag
OR Time_FGR,#01<<KeyDetect_FG  ;Key scanning(interval time=every 10ms)
OR Time_FGR,#01<<ChkPwrKey_FG  ;Power key checking
T1CapRtn POP PP
IRET

;//******************************************************************************************

//** Title : Timer 2 base time interrupt(interval=1ms)
//**
//** Inputs: fosc(@8MHz)/(1000*8)=1ms interrupt
//** Outputs: Number of Hsync signal for 10ms(composite sync input mode)
//** Preserves:
//** Corrupts:
//******************************************************************************************
T2Intv_Int: SB0  ;1ms interval timer
PUSH PP
PUSH R0
CLR PP
TM SYNCON0,#01<<VOSS  ;Separated sync signal ?
JP Z,T2IntvRtn
CompSyncCalculate  
LD  HFreqStCnt, HFreqSpCnt  
LD  HFreqSpCnt, TM1CNTL  
TM  SYNCN2, #01<<UNMIXHSYNC  ; Mixed sync signal  
     ; (Hsync period with Vsync signal) ?  
JR  Z, MixedSyncInput  
TM  Sync_P_FGR0, #01<<MixedSync_FG  
JR  Z, NormHcount  
;  
MixedSyncInput  
ADD  HCount+1, AverageHf ; if input sync is mixed sync input for this 1ms period  
ADC  HCount, #0  
JR  BaseTimer  
;  
NormHcount  
LD  R0, HFreqSpCnt  
SUB  R0, HFreqStCnt  
ADD  HCount+1, R0 ; R0 = number of Hsync signal for 1ms  
ADC  HCount, #0  
;  
BaseTimer  
TM  SYNCN2, #01<<UNMIXHSYNC ; Not finish mixed-sync period ?  
JR  Z, Check10ms  
AND  SyncP_FGR0, #0FFh-(01<<MixedSync_FG)  
Check10ms  
INC  TB1mSR  
CP  TB1mSR, #10  
JR  ULT, T2IntvI rtn  
;  
LD  HfHighNew, HCount ; Number of Hsync signal for 10ms  
LD  HfLowNew, HCount+1  
CLR  TB1mSR  
LDW  HCount, #00h  
OR  SyncP_FGR0, #01<<HSyncFin_FG ; Set Hsync signal find flag  
OR  Time_FGR, #01<<KeyDetect_FG ; Key scanning  
OR  Time_FGR, #01<<ChkPwrKey_FG ; Power key checking  
T2IntvI rtn  
POP  R0  
POP  PP  
IRET  
;  
/** Title : 10msec time base  
/**  
Chk10msTimer:  
TM  Mute_FGR, #01<<NormMwait_FG  
JR  Z, PwrUpMTime  
INC  M10mSR  
CP  M10mSR, #35 ; Mute delay = 350ms  
JR  UGT, SetMuteChkTime  
JR  TimeB10mS  
;  
PwrUpMTime  
TM  Mute_FGR, #01<<PwrOnWait_FG  
JR  NZ, TimeB10mS  
INC  M10mSR  
CP  M10mSR, #200 ; Power-up mute = 2sec  
JR  ULE, TimeB10mS  
OR  Mute_FGR, #01<<PwrOnWait_FG  
AND  Mute_FGR, #0FFh-(01<<Vmute_FG)
setMuteChkTime AND Mute_FGR,#0FFh-(01<<NormMwait_FG)
OR Mute_FGR,#01<<ChkSyncStus_FG ; ChkSyncStus_FG <- 1

TimeB10mS DEC TB10mSR
JR ne, TBaseRtn
LD TB10mSR,#10

;="/********************************************************************** */
;** Title : 100msec time base
;="/********************************************************************** */

Time100mS CALL ChkDegEndTime
TM SyncP_FGR1,#01<<OverRange_FG ; Over range?
JR NZ, CheckDpmsIn
TM Dpms_FGR,#01<<ChkDpmsCon_FG ; DPMS condition?
JR Z, TBaseRtn

CheckDpmsIn CP DPMS100mSR,#30 ; 3sec?
JR UGE, SetDpmsChk
INC DPMS100mSR
JR TBaseRtn

SetDpmsChk OR Dpms_FGR,#01<<DpmsStart_FG
TBaseRtn RET

;="/********************************************************************** */
;** Title : Degaussing time check
;="/********************************************************************** */

ChkDegEndTime TM Time_FGR,#01<<DeGTime_FG
JR Z, ChkDegEndRet
DEC DG100mSR
JR NE, ChkDegEndRet
AND P0,#0FFh-(01<<DeGausport) ; Degaussing(3sec) Off
AND Time_FGR,#0FFh-(01<<DeGTime_FG)

ChkDegEndRet RET

;="/********************************************************************** */
;** Title : Timer0 Vsync edge interrupt
;="/********************************************************************** */

VSyncDet_Int: PUSH PP
CLR PP
CLR NoVTImE
CP T0OvfCntr,#10 ; Over 400Hz (DDC1 mode)?
JR UGT, VsyncDetSrv
CLR T0OvfCntr
OR SyncP_FGR0,#01<<DdcHighSpd_FG
POP PP
IRET

; VsyncDetSrv
SB0
OR SyncP_FGR0,#01<<VSyncFin_FG ; bit finish vsync counter
LD Vcount,VFreqCHigh ; Number of Timer0 overflow counter
MULT Vcount,#255
ADD Vcount+1,VFreqCHigh ; VFreqCHigh*256
ADC Vcount,#0
ADD Vcount+1,TM0DATA ; Vcount(2-byte)=(#Overflow*256)+T0CNT
ADC Vcount,#0
CLR VFreqCHigh
CLR T0OvfCntr
AND SyncP_FGR0,#0FFh-(01<<DdcHighSpd_FG)

; CheckDdcVclk
TM IIC_FGR,#01<<Ddc2mode_FG ; Correct DDC2B mode ?
JR Z, SetMixSyncFlag
SB1
TM DCON,#01<<DDC1EN ; DDC1 mode ?
JR Z, SetMixSyncFlag
INC VclkCntr ; Increment Vsync counter for DDC recovery

SetMixSyncFlag
SB0
TM SYNCON0,#01<<VOSS
JR Z, VsyncDetIrtn
OR SyncP_FGR0,#01<<MixedSync_FG

VsyncDetIrtn
POP PP
IRET

;***************************************************************************
;//** Title : Interrupt for multi master I2c bus processor
;//**         - Vector address --> 00F8h for Irq1
;//**
;//** Inputs: PC/Control jig -> Monitor (DDC1/2B/2B+)
;//** Outputs: Monitor -> PC/Control jig (DDC1/2B/2B+)
;//** Preserves:
;//** Corrupts:
;//***************************************************************************
;
DDCnFA_Int: 
SB1
PUSH PP
CLR PP

;-------------------------------------------------------------
;----           DDC1 Tx protocol processor      ----- 
;-------------------------------------------------------------
TM DCON,#01<<DDC1EN ; Normal interface mode (No DDC1) ?
JR Z, DDC2Routine
TM DCON,#01<<SCLF ; Is falling edge detected at SCL pin ?
JP Z, EdidTx
LD PP,#11h
CLR	TBDR	;First EDID(#00h)
CLR	EdidAddr
AND	DCON,#0Ffh-(01<<DDC1EN)	;DDC1 -> Normal IIC-bus interface mode
AND	DCCR,#0Ffh-(01<<DPND)	;Clear IIC bus int. pending bit
POP	PP
SB0
IRET
;
;---------------------------------------------------------------
;---          DDC2 protocol mode checking         ---
;---------------------------------------------------------------

DDC2Routine
TM	DCSR0,#01<<DMTX	;Master Tx : DDC2B+ (Monitor -> PC)
JP	NZ,Master
TM	DCSR0,#01<<DSTX	;Slave Tx : DDC2Bi (Monitor <-> PC)
JR	NZ,ChkDDC2mode
TM	DCON,#01<<DDC1MAT	;DDC2B(#A0h) mode ?
JP	NZ,DDC2Bmode
TM	IIC_FGR,#01<<RevA0match_FG	;Already received slave address #A0h ?
JP	NZ,DDC2Bmode
;
OR	IIC_FGR,#01<<Ddc2mode_FG
OR	IIC_FGR,#01<<DDCCmd_FG	;If Rx mode(DDC2B+/Ci), set DDCCmd_FG
PUSH	R0
LD	R0,#MBusBuff	;DDC2B+ : PC -> Monitor
ADD	R0,RxXCntr
LD	@R0,RBDR	;@(MBusBuff+RxXCntr) <- RBDR(=Rx buff)
INC	RxXCntr
POP	R0
CLR	VclkCnter	;DDC error checking timer
AND	DCON,#0FFh-(01<<RevA0match_FG)
TM	DCSR0,#01<<DDC1MAT	;Address match as #A0h(DDC2B mode)
JP	Z,DDC2BiPrss
;
TM	DDC2SiRtn	;Is ACK received ?
JR	NZ,DdcCommFail
;
TM	IIC_FGR,#01<<Ddc2BTxmode_FG	;Set match flag of slave Tx mode
JR	NZ,EdidTx
OR	IIC_FGR,#01<<Ddc2BTxmode_FG	;address(#A1h)
CP	DDSR,#00h	;In this case : A0h -> 00h -> P & S -> A1h -> ...
JR	NE,EdidTx
LD	PP,#11h	;In this case : A0h -> 00h -> S -> A1h ->
INC	EdidAddr	;EdidAddr : 00h -> 01h (Repeat start case)
; EdidTx
  LD PP,#11h
  CP EdidAddr,#7Fh ; EDID=00h~7Fh(Page1)
  JR ULE,PrepNextAddr
  CLR EdidAddr ; First data
  PrepNextAddr
  LD TBDR,@EdidAddr ; TBDR=Tx buffer
  TM DCSR1,#01<<DBUFEMT
  JR NZ,ReloadTxBuff
  INC EdidAddr
  AND DCCR,#0FFh-(01<<DPND) ; Clear IIC.bus int. pending bit
  POP PP
  SB0
  IRET

; PrepNextAddr
  LD TBDR,@EdidAddr
  CLR EdidAddr
  TM DCSR1,#01<<DBUFEMT
  JR NZ,ReloadTxBuff
  INC EdidAddr
  AND DCCR,#0FFh-(01<<DPND)
  POP PP
  SB0
  IRET

; ReloadTxBuff
  INC EdidAddr
  LD TBDR,@EdidAddr ; For keeping normal pre-buffer mode
  INC EdidAddr
  AND DCCR,#0FFh-(01<<DPND)
  POP PP
  SB0
  IRET

; DdcCommFail
  LD PP,#11h
  CLR TBDR ; First data of EDID
  CLR EdidAddr
  AND DCSR0,#0FFh-(01<<DSTX) ; Return slave Rx mode
  AND DCCR,#0FFh-(01<<DPND) ; Clear IIC.bus int. pending bit
  POP PP
  SB0
  IRET

; / DDC1 -> DDC2B mode
  DDC2Bmode
  CLR VclkCntr ; For DDC1 recovery mode
  OR IIC_FGR,#01<<Ddc2mode_FG
  CP RBDR,#0A0h ; Slave address ?
  JR EQ,RevDdc2bAddr
  LD PP,#11h
  CP RBDR,#00h ; Sub-address ?
  JR NE,RandomAddr
  CLR TBDR ; First data of EDID(#00h)
  CLR EdidAddr
  AND DCCR,#0FFh-(01<<DPND) ; Clear IIC.bus int. pending bit
  POP PP
  SB0
  IRET

; RandomAddr
  LD TBDR,@RBDR
  LD EdidAddr,RBDR
  AND DCCR,#0FFh-(01<<DPND) ; Clear IIC.bus int. pending bit
  POP PP
  SB0
  IRET

; RevDdc2bAddr
  OR IIC_FGR,#01<<RevA0match_FG ; Set slave address(#A0h) match flag
AND    IIC_FGR,#0FFh-(01<<Ddc2BTxmode_FG) ;Clear slave Tx address match flag
LD     PP,#11h
CLR    EdidAddr
AND    DCCR,#0FFh-(01<<DPND) ;Clear IIC.bus int. pending bit
POP    PP
SB0
IRET
;
;----------------------------------------------------------------------------
;---   DDC2Bi protocol processor      ---
;----------------------------------------------------------------------------
DDC2BiPrss CLR PP
CP     ByteCnt,#1 ;Check the Number of Tx Data
JR     ULE,CountZero
DEC    ByteCnt
;
PUSH    R0
LD     R0,NumTxdByte
LD     TBDR,@R0 ;Load Tx Data to TBDR
INC    NumTxdByte
POP    R0
JP     DdcSrvRtn
;
CountZero: AND    DCSR0,#0FFh-(01<<DSTX) ;Return Slave Rx Mode
JP     DdcSrvRtn
;
;***************************************************************************
;** Title : Mater transmitter processor
;***************************************************************************
Master:         PUSH    IMR
LD     IMR,#00000011B ;Enable T0/T1/T2 int.
EI
;
TM     DCSR1,#01<<MISPLS
JR     NZ,CommFail ;Mispalced condition error
TM     DCSR0,#01<<DAL
JR     NZ,CommFail ;Bus arbitration failed during communication
TM     DCSR0,#01<<DRXACK
JR     NZ,CommFail ;Not received ACK
;
PUSH    R0
PUSH    R2
PUSH    R3
PUSH    R4
PUSH    R5
PUSH    R6
CALL    TxdComPart ;DDC2B+ communication (Monitor -> PC(Control jig))
POP    R6
POP    R5
POP    R4
POP    R3
POP     R2
POP     R0
DI
POP     IMR
POP     PP
AND     DCCR,#0FFh-(01<<DPND) ;Clear IIC.bus int. pending bit
SB0
IRET

;********************************************************************************
;*****SendType  *****
;***** 1 : Attention *****
;***** 2 : Reply Identify *****
;***** 3 : Reply Capability *****
;***** 4 : Reply Vcp *****
;***** 5 : Reply Timing *****
;***** 6 : Reply All mode save(EDID data dump) *****
;***** 7 : Reply Factory Save *****
;********************************************************************************

;********************************************************************************
;*****Common parts transmit data *****
;***** Master Transmit in IIC Interrupt *****
;********************************************************************************

TxdComPart:  NOP
            NOP
            RET

;********************************************************************************

VcpTbl
;DB V_HPosition  $20 ;0 HPosition Vcp
;DB V_VPosition  $30 ;1 VPosition Vcp
;DB V_HSize     $22 ;2 HSize Vcp
;DB V_VSize     $32 ;3 VSize Vcp
;DB V_Pincushion $24 ;4 Pincushion Vcp
;DB V_Trapezoid $42 ;5 Trapeziod Vcp
;DB V_Parallel  $40 ;6 Parallel Vcp
;DB V_Pinbalance $26 ;7 Pinbalance Vcp
;DB V_VLinearity $3A ;8 VLinearity Vcp
;DB V_Tilt      $44 ;9 Tilt Vcp
;DB V_HSizeMin  $E4 ;10 HSizeMin Vcp
;DB V_SSelect   $3Ch ;11
;DB V_VMoire    $58h ;12
            ; KA2504 Pre-amp
;DB V_Contrast ;$12 ;13
;DB V_RGain ;16h ;14
;DB V_GGain ;18h ;15
;DB V_BGain ;1Ah ;16
;DB V_CoffBright ;10h ;17
;DB V_RCOff ;6Ch ;18
;DB V_GCOff ;6Eh ;19
;DB V_BCOff ;70h ;20
;DB V_ACL ;F6h ;21
;DB V_Degauss ;01h ;22

;********************************************
;***** Pre_Amp Data Transfer Format *****
;***** IIC_P_Amp_Start *****
;***** Slave Address :#0DCh *****
;***** Sub Address :Rgb_Drv_Tbl *****
;***** Data :@DataAddr *****
;***** IIC_P_Amp_Stop *****
;********************************************

Preamp_RGB_Drv: PUSH R0
PUSH R1
PUSH R2
;
SB0
CLR R2
Preamp_One_Drv CALL IICbus_Start
LD R0,#0DCh ;KA5204 slave address(#0DCh)
CALL P_Amp_Drv_Byte
TM IIC_FGR,#01<<CommFail_FG
JR NZ,KA2504Stop
;
LD R0,R14 ;KA2504 sub-address
CALL P_Amp_Drv_Byte
TM IIC_FGR,#01<<CommFail_FG
JR NZ,KA2504Stop
;
LD R0,R15 ;KA2504 control data
CALL P_Amp_Drv_Byte
KA2504Stop CALL IICbus_Stop
TM IIC_FGR,#01<<CommFail_FG
JR Z,PreAmpDrvRtn
AND IIC_FGR,#0FFh-(01<<CommFail_FG)
INC R2
CP R2,#2 ;Error ?
JR ULE,Preamp_One_Drv
;
PreAmpDrvRtn POP R2
POP R1
POP R0
RET
;rgbDrvTbl DB 00h ;Pre_amp (Contrast)
DB 01h ;Pre_amp (Brightness)
DB 02h ;Pre_amp (R_gain)
DB 03h ;Pre_amp (G_gain)
DB 04h ;Pre_amp (B_gain)
DB 05h ;Pre_amp (OSD Contrast)
DB 07h ;Pre_amp (R_cutoff)
DB 08h ;Pre_amp (G_cutoff)
DB 09h ;Pre_amp (B_cutoff)
DB 0Ah ;Switch

;*********************************************************************
;****** PreAmp DISPLAY INITIDAL ******
;****** Mfr. : samsung electronc ******
;****** Type : KA2504 ******
;*********************************************************************
;****** PreAmp Start Condition ******
;*********************************************************************
IICbus_Start OR BTCON,#01<<BTCLR ;Clear Watch-dog timer
OR P3,#11000000B ;P3.7/6 <- High(SDA,SCL)
Call DelayNop ;IIC-Start
AND P3,#0FFh-(01<<SDA)
Call DelayNop
AND P3,#0FFh-(01<<SCL)
RET

;------------------------------------------------------------------------------
;------IIC_Bus Clock Generation -------
;------------------------------------------------------------------------------
IIC_Clock_1Bit OR P3,#01<<SCL ;Clock Generation.
CALL DelayNop
AND P3,#0FFh-(01<<SCL)
CALL DelayNop
RET

; DelayNop NOP
NOP
RET

;----------------------------------------------------------------------
;-------- IIC Stop Condition --------
;----------------------------------------------------------------------
IICbus_Stop AND P3,#0FFh-(01<<SDA)
NOP
NOP
NOP
OR P3,#01<<SCL
CALL DelayNop
OR P3,#01<<SDA ;SDA <- High(Stop condition)
RET

;-------------------------------------------------------------
;--- SHIFT LEFT ONE BYTE --------
;--- Parameter : R10 : Shift Data --------
;--- R11 : Bit Counter --------
;-------------------------------------------------------------

P_Amp_Drv_Bit LD R1,#8
ShiftLeft RLC R0
JR NC,Data_Low
OR P3,#01<<SDA

; Gen_Clock OR P3,#01<<SCL ;Clock Generation.
NOP
NOP
AND P3,#0FFh-(01<<SCL)
DJNZ R1,ShiftLeft ;R1=DataCntr

; ACK_Check AND P3CONH,#00111111B ;SDA(P3.7)=Input
CALL DelayNop
OR P3,#01<<SCL ;Acknowledge clock
NOP
NOP
NOP
TM P3,#01<<SDA ;Ack in ?
JR NZ,ACK_Fail

; ACK_OK OR P3CONH,#11000000B ;SDA(P3.7)=Output
AND P3,#0FFh-(01<<SCL)
AND IIC_FGR,#0FFh-(01<<CommFail_FG)
RET

; ACK_Fail OR P3CONH,#11000000B ;SDA(P3.7)=Output
AND P3,#0FFh-(01<<SCL)
OR IIC_FGR,#01<<CommFail_FG
RET

; Data_Low AND P3,#0FFh-(01<<SDA)
JR GEN_Clock

; CtrlPreAmp CALL TimeDelay
CALL TimeDelay
CALL CtrlPreDrv_Sub

; CALL TimeDelay
CALL TimeDelay
CALL CtrlPreDrv_Sub
RET

; CtrlPreDrv_Sub TM EepRom_FGR,#01<<SavedEep_FG
JR NZ,LdEepRGB

; LD EP_CoffBRIGHT,#0C0h
LD EP_CONTRAST,#0FFh
LD EP_RGain,#3Fh
LD EP_GGain,#39h
LD EP_BGain,#32h
LD EP_RCutoff,#8Ah
LD EP_GCutoff,#80h
LD EP_BCutoff,#0A6h
LD EP_ACL,#76h
JR CtrlKA2504

LD R14,#EPA_CoBr ;Brightness data
CALL ReadEepData
LD EP_CofBRIGHT,ReadData
INC R14 ;Contrast data
CALL ReadEepData
LD EP_CONTRAST,ReadData
INC R14 ;R-Gain
CALL ReadEepData
LD EP_RGain,ReadData
INC R14 ;G-Gain
CALL ReadEepData
LD EP_GGain,ReadData
INC R14 ;B-Gain
CALL ReadEepData
LD EP_BGain,ReadData
INC R14 ;R-CutOff
CALL ReadEepData
LD EP_RCutoff,ReadData
INC R14 ;G-CutOff
CALL ReadEepData
LD EP_GCutoff,ReadData
INC R14 ;B-CutOff
CALL ReadEepData
LD EP_BCutoff,ReadData

LD R14,#EPA_ACL ;ACL data
CALL ReadEepData
LD EP_ACL,ReadData

LD R14,#PSubA_SBnBr ;Brightness & Soft blanking
CALL Preamp_RGB_Drv
SB1
LD PWM5,EP_ACL ;ACL
SB0

LD R14,#PSubA_CoBr
CALL Preamp_RGB_Drv
LD R14,#PSubA_Cont
LD R15,EP_CONTRAST
CALL Preamp_RGB_Drv
LD R14,#PSubA_RGain ;KA2504 sub address
LD R15,EP_RGain ;DataAddr
CALL Preamp_RGB_Drv
LD R14,#PSubA_GGain
LD R15,EP_Ggain
CALL Preamp_RGB_Drv
LD R14,#PSubA_BGain
LD R15,EP_Bgain
CALL Preamp_RGB_Drv
LD R14,#PSubA_RCo
LD R15,EP_Rcutoff
CALL Preamp_RGB_Drv
LD R14,#PSubA_GCo
LD R15,EP_GCutoff
CALL Preamp_RGB_Drv
LD R14,#PSubA_BCo
LD R15,EP_Bcutoff
CALL Preamp_RGB_Drv
RET

:--------------------------------------------------------------------------
:-------- Initialize IIC.bus control register --------
:--------------------------------------------------------------------------
IniDDCmodule SB1
LD DCCR,#10100100b ;Enable Tx ACK signal
:Enable IIC.bus Tx/Rx int.
:Include DDC1 Tx int.
:100KHz clock speed
CLR DCSR0
LD DAR0,#0A0h ;#0A0h=Monitor address(DDC2B)
LD DAR1,#6Eh ;#6Eh=Monitor address(DDC2B+/2Bi)
LD DCSR0,#00010000B ;DCSR.7/6=Master/Slave mode
:DCSR.5=Start/Stop(When write),
:busy signal status(Read)
:DCSR.4=Enable DDC module
:DCSR.3=Arbitration procedure status
:DCSR.2=Address-as-slave status
:DCSR.1=General call
:DCSR.0=ACK bit status
CLR TBDR ;First EDID data
:First EDID data
LD PP,#11h
INC EdidAddr
CLR SB0
RET

:******************************************************************************
:** Read 1Byte in EEPROM ***
:by S/W IIC.bus interface **********
******************************************************************************
:Read1Byte: PUSH R0
PUSH R1
PUSH R2
CALL IICbus_Start ;IIC.bus protocol start
TM Tda9109_FGR,#01<<TdaRead_FG
JR NZ,ldTdaSlave
LD R0,#0A0h ;Random read= #A0h -> Slave -> S -> #A1h -> READ
CLR R2
JR ShiftStart

ldTdaSlave
LD R0,#8Dh ;#8C/8Dh=TDA9109 salve address
LD R2,#2
AND Tda9109_FGR,#0FFh-(01<<TdaRead_FG)
ShiftStart
LD R1,#8 ;1byte
DataShift
RLC R0 ;Rotate left SDADATA(=R0)
JP C,Data1
AND P3,#0FFh-(01<<SDA) ;Data 0
OR P3,#01<<SCL ;Clock Generation.
NOP
NOP
NOP
AND P3,#0FFh-(01<<SCL)
SDA8bit
DJNZ R1,DataShift
AND P3CONH,#00111111B ;SDA(P3.7)=Input
OR P3,#01<<SCL ;Acknowledge clock
NOP
NOP
NOP
TM P3,#01<<SDA ;Ack in ?
JP NZ,CommuniFail
OR P3CONH,#11000000B ;SDA(P3.7)=Output
AND P3,#0FFh-(01<<SCL)
;
CP R2,#02 ;SDACNTR=R2
JR UGE,DataRxStart
CP R2,#01
JR UGE,ReStartSignal
;
LD R0,R14 ;SDADATA <- R14
INC R2 ;SDACNTR++
JR ShiftStart
;
ReStartSignal
CALL IICbus_Start
LD R0,#0A1h ;SDADATA <- #0A1h
INC R2 ;SDACNTR++
JR ShiftStart
;
DataRxStart
AND P3CONH,#00111111B ;SDA(P3.7)=Input
NOP
LD R1,#8
RotateConti
OR P3,#01<<SCL ;SCL <- High
TM P3,#01<<SDA ;Data value check
JR NZ,SetCF
RCF
JR DataRotate
SetCF
SCF
RLC R0
AND P3,#0FFh-(01<<SCL) ;SCL <- Low
DJNZ R1,RotateConti ;End of 1byte ?
LD ReadData,R0

No_ACK
OR P3CONH,#11000000B ;SDA(P3.7)=Output
OR P3,#01<<SDA ;SDA <- High(ACK=High):communication end
OR P3,#01<<SCL ;SCL <- High(9th clock)
NOP
NOP
AND P3,#0FFh-(01<<SCL) ;SCL <- Low

GenIicStop
ALL IICbus_Stop
POP R2
POP R1
POP R0
RET

Data1
OR P3,#01<<SDA
OR P3,#01<<SCL ;Clock Generation.
NOP
NOP
NOP
AND P3,#0FFh-(01<<SCL)
JP SDA8bit

ReadEepData:
PUSH R3
PUSH R4
PUSH R5
PUSH R6
CLR R3
ContiDataRead CALL Read1Byte
CP R3,#2
JR UGT,CmpReadData
CP R3,#1
JR UGT,Read3rd
CP R3,#0
JR UGT,Read2nd
LD R4,ReadData ;R3=0
XOR R5,R4
XOR R6,R5
INC R3
JR ContiDataRead

Read2nd
LD R5,ReadData ;R3=1
INC R3
JR ContiDataRead

Read3rd
LD R6,ReadData ;R3=2
INC R3
JR ContiDataRead

CmpReadData
CP R4,R5
JR EQ,RdDataRtn
CP R4,R6
JR EQ,RdDataRtn
CP R5,R6
JR EQ,RdDataRtn
TM IIC_FGR,#01<<ReRead_FG
JR NZ,RdDataRtn
OR IIC_FGR,#01<<ReRead_FG
CLR R3
JR ContiDataRead
;
RdDataRtn AND IIC_FGR,#0FFh-(01<<ReRead_FG)
POP R6
POP R5
POP R4
POP R3
RET
;
;******************************************************
;******     Write 1Byte in EEPROM *******
;******          by S/W IIC.bus interface *******
;******************************************************
;
Write1Byte: PUSH R0
PUSH R1
PUSH R2
CALL IICbus_Start ;IIC.bus protocol start
;
TM Tda9109_FGR,#01<<TdaWrite_FG
JR NZ,WriteTDA
LD R0,#0A0h ;Write : #A0h -> Sub -> Data
CLR R2
JR WriteStart
;
WriteTDA LD R0,#8Ch ;#8Ch=TDA9109 salve address
CLR R2
WriteStart LD R1,#8
TxDataShift RLC R0
JR C,TxData1
AND P3,#0FFh-(01<<SDA) ;Data 0
AND P3,#0FFh-(01<<SDA) ;Data 0
OR P3,#01<<SCL ;Clock Generation.
NOP
NOP
AND P3,#0FFh-(01<<SCL)
Tx1Byte DJNZ R1,TxDataShift ;Acknowledge check
;
AND P3CONH,#00111111B ;SDA(P3.7)=Input
OR P3,#01<<SCL ;Acknowledge clock
NOP
NOP
NOP
TM P3,#01<<SDA ;Ack in ?
JR NZ,CommuniFail
NextBWrite OR P3CONH,#11000000B ;SDA(P3.7)=Output
AND P3,#0FFh-(01<<SCL)
CP R2,#2
JR UGE,EepWriteEnd
CP R2,#1
JR UGE,DataTxStart
LD R0,R14 ;SDADATA <- R14(Address)
INC R2 ;SDACNTR++
JR WriteStart

DataTxStart LD R0,R15 ;SDADATA <- R15(Data)
INC R2 ;SDACNTR++
JR WriteStart

TxData1 OR P3,#01<<SDA
OR P3,#01<<SCL ;Clock Generation.
NOP
NOP
AND P3,#0FFh-(01<<SCL)
AND P3,#0FFh-(01<<SDA)
JR Tx1Byte

CommuniFail OR P3CONH,#11000000B ;SDA(P3.7)=Output
OR IIC_FGR,#01<<CommFail_FG
JP GenlicStop
EepWriteEnd AND IIC_FGR,#0FFh-(01<<CommFail_FG)
JP GenlicStop

;********************************************************************
;******     Write 4Byte in EEPROM ******
;******          by S/W IIC.bus interface ******
;********************************************************************
WriteNByte: PUSH R0
PUSH R1
PUSH R2
CALL IICbus_Start ;IIC.bus protocol start

LD R0,#0A0h ;Page write= #A0h->Word->D1>D2->D3->D4->STOP
LD R2,#6 ;Word addr -> D1 -> D2 -> D3 -> D4
NextTx1Byte LD R1,#8
TxNDataShift RLC R0
JR C,TxNDat1
AND P3,#0FFh-(01<<SDA) ;Data 0
OR P3,#01<<SCL ;Clock Generation
NOP
NOP
AND P3,#0FFh-(01<<SCL)
TxN1Byte DJNZ R1,TxNDataShift

; OR P3,#01<<SCL ;Acknowled clock
AND P3CONH,#00111111B ;SDA(P3.7)=Input
TM P3,#01<<SDA ;Ack in ?
JR NZ,CommuniFail ;Fail
OR  P3CONH,#11000000B ;SDA(P3.7)=Output
AND  P3,#0FFh-(01<<SCL)
LD   R0,R14 ;SDADATA <- R14(Address)
CP   R2,#5
JR   ULE,DataTxNStart
DEC  R2
JR   NextTx1Byte
;
DataTxNStart
LD   R0,R15 ;SDADATA <- R15(Data=#0FFh)
DJNZ R2,NextTx1Byte
JR   EepWriteEnd ;Stop condition
;
TxNData1
OR   P3,#01<<SDA
OR   P3,#01<<SCL ;Clock Generation.
NOP
NOP
AND  P3,#0FFh-(01<<SCL)
AND  P3,#0FFh-(01<<SDA)
JR   TxN1Byte
;
WriteCycle:
SB0  CALL  Write1Byte
TM   Tda9109_FGR,#01<<TdaWrite_FG
JR   NZ,ChkReWrite
CLR  DLY1mSR
CALL  WriteWait ;Check write cycle(Typ=6ms, Max=10ms)
;
ChkReWrite
TM   IIC_FGR,#01<<ReWrite_FG ;ReWrite ?
JR   NZ,CrReWrite
TM   IIC_FGR,#01<<CommFail_FG
JR   Z,CrReWrite
OR   IIC_FGR,#01<<ReWrite_FG
JR   WriteCycle
;
CrReWrite
AND  IIC_FGR,#0FFh-(01<<ReWrite_FG)
AND  IIC_FGR,#0FFh-(01<<CommFail_FG)
WrCycleRtn
AND  Tda9109_FGR,#0FFh-(01<<TdaWrite_FG)
RET
;
;****************************************************************************
//** Title : Waiting for write time ***
;****************************************************************************
;
WriteWait:
PUSH R0 ;Check write cycle
PUSH R1
IICbusRestart CALL  IICbus_Start ;IIC.bus protocol start
;
LD   R0,#0A0h
LD   R1,#8 ;1byte
SlaveA0h
RLC  R0 ;Rotate left SDADATA(=R0)
JP   C,ACKData1
AND  P3,#0FFh-(01<<SDA) ;Data 0
SYNC PROCESSOR S3C8639/C863A/P863A/C8647/F8647

OR P3,#01<<SCL ;Clock Generation.
NOP
NOP
AND P3,#0FFh-(01<<SCL)
TXA0hData DJNZ R1,SlaveA0h
AND P3CONH,#00111111B ;SDA(P3.7)=Input
OR P3,#01<<SCL ;Acknowledge clock
NOP
NOP
NOP
TM P3,#01<<SDA ;Ack in ?
JR NZ,RechkWrCycle
ENDWRcycle AND P3,#0FFh-(01<<SCL)
OR P3CONH,#11000000B ;SDA(P3.7)=Output
CALL IICbus_Stop
POP R1
POP R0
RET

; RechkWrCycle
CP DLY1mSR,#10 ;Over 10ms ?
JR UGE,ENDWRcycle
AND P3,#0FFh-(01<<SCL)
OR P3CONH,#11000000B ;SDA(P3.7)=Output
CALL IICbus_Stop
JR IICbusRestart

; ACKData1
OR P3,#01<<SDA
OR P3,#01<<SCL ;Clock Generation.
NOP
NOP
AND P3,#0FFh-(01<<SCL)
JR TXA0hData

; //10ms delay routine
;
TImeDelay PUSH R4
PUSH R5
LD R4,#9
WaitLoop0 LD R5,#250 ;Write-waiting time=10ms
WaitLoop1 NOP
NOP
DJNZ R5,WaitLoop1
ContiDec DJNZ R4,WaitLoop0
POP R5
POP R4
RET

; < EDID Data >
DDCDUMP: LD PP,#11h
LDW RR2,#DDCData
LD R4,#00h ;R4=Address(00h-7Fh:128-Byte)
WriteEDID

LD @R4,R5
INC R4
CP R4,#80h
JR ULT,WriteEDID
CLR PP
RET

;------ -----------------

DDCData

DB 00h,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,00h
DB 0AH,05H,01H,00H,2EH,1FH,71H,0E8H,07H,65H,0A0H,57H,46H,9AH,26H
DB 10H,48H,4CH,0FFh,0FEH,00H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H
DB 01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H,01H

; < EDID Data >

EDIDtoRAM:
PUSH R0 ;Write EDID(128-byte) to EEPROM page0
PUSH R1
PUSH R2
PUSH PP
LD PP,#11h
LD R4,#00h ;R4=RAM address(50h-CFh:128-Byte)
LD R14,#00h ;R14=Start address of EDID

CALL IICbus_Start ;IIC.bus protocol start

LD R0,#0A0h ;Sequential read operation
CLR R2 ; <= #A0h(Page0) -> Word -> S -> #A1h -> EAD....
Shift1Byte
LD R1,#8 ;1byte

RotateData
RLC R0 ;Rotate left SDADATA(=R0)
JP C,SeqData1
AND P3,#0FFh-(01<<SDA) ;Data 0
OR P3,#01<<SCL ;Clock Generation.
NOP
NOP

AND P3,#0FFh-(01<<SCL)
DJNZ R1,RotateData

Chk1ByteEnd
AND P3CONH,#11110011B ;SDA(P3.5)=Input
OR P3,#01<<SCL ;Acknowledge clock
NOP
NOP

TM P3,#01<<SDA ;Ack in ?
JP NZ,CommuniFail
OR P3CONH,#00001100B ;SDA(P3.5)=Output
AND P3,#0FFh-(01<<SCL)

CP R2,#02 ;SDACNTR=R2
JR UGE,DataRx
CP R2,#01
JR UGE, ReStart
LD R0, R14 ; SDADATA <- R14
INC R2 ; SDACNTR++
JR Shift1Byte
;
ReStart CALL IICbus_Start
LD R0, #0A1h ; SDADATA <- #0A1h
INC R2 ; SDACNTR++
JR Shift1Byte
;
DataRx AND P3CONH, #11110011B ; SDA(P3.5) = Input
NOP
LD R1, #8
DataRead OR P3, #01<<SCL ; SCL <- High
TM P3, #01<<SDA ; Data value check
JR NZ, SetCFlag
RCF
JR RxRotate
SetCFlag SCF
RxRotate RLC R0
AND P3, #0FFh-(01<<SCL) ; SCL <- Low
DJNZ R1, DataRead ; End of 1byte ?
;
OR P3CONH, #00001100B ; SDA(P3.5) = Output
AND P3, #0FFh-(01<<SDA) ; ACK generation
OR P3, #01<<SCL ; SCL <- High(9th clock)
NOP
NOP
AND P3, #0FFh-(01<<SCL) ; SCL <- Low
LD @R4, R0 ; R4=50h~CFh, R0=Read data
INC R4
CP R4, #80h ; 00~7Fh : EDID
JR ULE, DataRx
POP PP
JP No_ACK ; Communication stop
;
SeqData1 OR P3, #01<<SDA
OR P3, #01<<SCL ; Clock Generation.
NOP
NOP
AND P3, #0FFh-(01<<SCL)
JP Chk1ByteEnd
;
;*********************************************************************
****** TDA9109 Initializing *******
;*********************************************************************
InitialKB2511: CLR R14 ; R14=Sub address
Conti2511Ini LDW RR2, #TdaFRunTbl
ADD R3, R14
LDC R15, @RR2 ; R15=Data
OR Tda9109_FGR, #01<<TdaWrite_FG
CALL WriteCycle
INC R14
CP R14,#0Fh ;00-0Fh
JR ULE,Conti2511Ini
RET

; // KB2511 Default Data
; H_Duty(40) / H_posi(40) / Free Run(00) / HFocus(90)
; HFocusKey(10) / Vramp(C0) / VPosi(40) / S Correct(20)
; C Correct(20) / Keystone(A0) / EW Size(C0) / B Plus(40)
; V Moire(00) / Side Pin(A0) / Parallel(A0) / VFocus(20)
;
TdaFRunTbl DB 4Bh,40h,15h,9Fh,14h,0Ch,40h,16h ;9109(0-7)
DB 32h,0Ah,0Ch,30h,00h,0Ah,0Ah,20h ;48KHz free running
;
//************************************************************************************
//********** The H/W IIC Read/Write Programming Tip ************
//************************************************************************************

Rxmode equ 6 ;IIC Receive Mode Flag
RxACK equ 0 ;Acknowledgement Check Flag
;
IIC_FGR EQU 10h ;IIC Status Control Check Register
CommFail_FG equ 3 ;IIC Communication Fail Check Flag
EepromWri_FG equ 2 ;Eeprom Writing Flag
IICRead_FG equ 1 ;IIC Reading Flag
RW_End_FG equ 0 ;IIC Read/Write Ending Check Flag
;
RxTemp EQU 20h ;Temporary Receiving Data Register
TxTemp EQU 21h ;Temporary Transmitting Data Register
IICCNTR EQU 22h ;IIC Read/Write Counter Register
Sub_Addr EQU 23h ;Slave Device Sub-address
Trans_Data EQU 80h ;Transmitting Data
Rx_Data EQU 90h ;Receiving Data
;
//************************************************************************************
//********** < N-Byte Write Program > ************
//************************************************************************************

://(Start) -> A0h -> SubAddress -> N-ByteData -> P(Stop)
Write_NByte: LD Sub_Addr,#10h ;Sub_Addr = Subaddress
LD Trans_Data,#01h ;Trans_Data = Tx Data(8-Byte)
LD Trans_Data+1,#23h
LD Trans_Data+2,#45h
LD Trans_Data+3,#67h
LD Trans_Data+4,#89h
LD Trans_Data+5,#0Abh
LD Trans_Data+6,#0CDh
LD Trans_Data+7,#0Efh
LD TxTemp,#80h
OR IIC_FGR,#01<<EepromWri_FG ;Enable Eeprom Write
AND IIC_FGR,#0FFh-(01<<IICRead_FG)
CALL Write_Cycle
RET
Write_Cycle:
SB1
LD DCON,#08h
LD DCCR,#00100101b
OR DCSR0,#11010000b
LD TBDR,#0A0h
OR DCSR0,#00100000b
SB0
;
IIC_Write:
TM IIC_FGR,#01<<RW_End_FG
JR NZ,Write_Rtn
TM IIC_FGR,#01<<CommFail_FG
JR Z,IIC_Write
CLR IICCNTR
JR W_Rtn
;
Write_Rtn
AND IIC_FGR,#0FFh-(01<<RW_End_FG)
TM IIC_FGR,#01<<EepromWri_FG
JR Z,W_Rtn
CALL WriteWait
;
W_Rtn
AND IIC_FGR,#0FFh-(01<<CommFail_FG)
AND IIC_FGR,#0FFh-(01<<EepromWri_FG)
RET
;
WriteWait:
PUSH R4
PUSH R5
LD R4,#15 ;Write-waiting time=10ms
W_Loop0
LD R5,#250 ;at Fosc=12MHz
W_Loop1
NOP
NOP
NOP
DJNZ R5,W_Loop1
Conti_Dec
DJNZ R4,W_Loop0
POP R5
POP R4
RET
;
IICBUS_INT:
SB1
TM DCSR0,#01<<RxACK
JR NZ,Comm_Fail
;
CP IICCNR,#0
JR EQ,WriteAddr
;
CP IICCNR,#1
JR EQ,WriteData
;
TM DCSR0,#01<<RxMode
;
JR Z, IICReadMode

CP IICCNR,#9 ;IICCNR(2-9) = 8-Byte Write
JR ULT, Conti_Wri
AND DCSR0,#11011111b ;Stop Signal Output
CLR IICCNR ;Clear Tx Counter
OR IIC_FGR,#01<<RW_End_FG ;Write Ending Flag Set
JR IIC_Rtn

Conti_Wri INC IICCNR
INC TbTemp
LD TBDR,@TxTemp ;@TxTemp = Transmitting Data
IIC_Rtn AND DCSR0,#11101111b ;IIC-Bus Int. Pending Bit Clear
SB0 ;Select Bank 0
IRET ;Interrupt Return

IICReadMode: CP IICCNR,#2
JR EQ, ModeChange

CP IICCNR,#10 ;IICCNR(3-10) => 8-Byte Read
JR ULT, Conti_Read

AND DCSR0,#11011111b ;Stop Signal Output
CLR IICCNR ;Clear Rx Counter
LD @RxTemp,RBDR ;@RxTemp = Last Rx Data
OR IIC_FGR,#01<<RW_End_FG ;Read Ending Flag Set
JR IIC_Rtn

Conti_Read CP IICCNR,#9
JR EQ, DisA_IIC

ComRead INC IICCNR
LD Rx_Temp,RBDR ;@Rx_Temp = Received Data
INC Rx_Temp
JR IIC_Rtn

DisA_IIC AND DCCR,#01111111b ;Disable ACK Signal
JR ComRead

ModeChange INC IICCNR ;#0A1h(Read Mode) Write
JR IIC_Rtn

WriteAddr LD TBDR, Sub_Addr ;TBDR <- Slave Device Subaddress
INC IICCNR
JR IIC_Rtn

WriteData TM IIC_FGR,#01<<IICRead_FG ;Read Mode Check
JR NZ, Read_Mode
LD TBDR, Trans_Data ;TBDR <- First Tx Data
INC IICCNR
JR IIC_Rtn

Com_Fail: OR IIC_FGR,#01<<CommFail_FG ;IIC Comm. Fail
AND DCSR0,#11011111b  ;Stop Signal Output
JR IIC_Rtn

Read_Mode
LD TBDR,#0A1h  ;Read Mode Slave Address
INC IICCNR  ;Change to Read Mode
OR DCSR0,#10000000b
AND DCSR0,#10111111b  ;Master Receive Mode
OR DCSR0,#00100000b  ;IIC Restart Signal Output
JR IIC_Rtn
;
//************************************************************************************
//**********<  N-Byte Read Program >              ***********
//************************************************************************************
;S(Start) -> A0h -> Sub Address -> RS(Restart) -> A1h -> N-Byte Read -> P(Stop)
Read_1Byte:
LD Sub_Addr,#10h  ;Slave Device Subaddress
LD RxTemp,#90h
OR IIC_FGR,#01<<IICRead_FG  ;Read Mode Flag Set
CALL ReadCycle
RET
;
ReadCycle:
SB1  ;Select bank 1
LD DCON,#08h  ;Enable Prebuffer Register
LD DCCR,#10100101b  ;Enable IIC-Bus Interrupt
OR DCSR0,#11010000b  ;Master Tx Mode & IIC Module Enable
LD TBDR,#0A0h  ;#0A0h=Slave Device Address
OR DCSR0,#00100000b  ;IIC Start Signal Generation
SB0  ;Select Bank 0
;
IIC_Read
TM IIC_FGR,#01<<RW_End_FG
JR NZ,R_Rtn
TM IIC_FGR,#01<<CommFail_FG  ;IIC Comm. Fail Check
JR Z,IIC_Read
CLR IICCNR
;
R_Rtn:
AND IIC_FGR,#0FFh-(01<<RW_End_FG)
AND IIC_FGR,#0FFh-(01<<CommFail_FG)
AND IIC_FGR,#0FFh-(01<<IICRead_FG)
AND IIC_FGR,#0FFh-(01<<EepromWri_FG)
RET
//include "S3C863A.h"
//include "insam8.h"

// type definition
typedef unsigned char   usch;
typedef unsigned int      usin;

// ******************
// macro definition
// ******************
#define BitTru(sfr,bit) (sfr & (1<<bit))
#define BitFals(sfr,bit) (!(sfr & (1<<bit)))
#define BitSet(sfr,bit) (sfr |= (1<<bit))
#define BitClr(sfr,bit) (sfr &= ~(1<<bit))
#define BitTgg(sfr,bit) (sfr ^= (1<<bit))

// ******************
// interrupt vector
// ******************
#define t2intv_int (0xee)
#define t1cap_int (0xf6)
#define ddcnfa_int (0xf8)
#define t0ovf_int (0xfa)
#define vsync_int (0xfc)                // Timer0 capture interrupt

// ******************
// Port function definition
// ******************
// port0
#define MUTEPORT    0
#define STBYPORT    0
#define SUSPNDPORT  1
#define OFFPORT     2
#define LEDPORT     3

// port3
#define DEGAUSPORT  0
#define CS1         3
#define CS2         4
#define CS3         5
#define SCL         6
#define SDA         7
// Control register definition

// sync-processor part

// SYNCON0
#define HIPORT 7 // Hsync input selection (or Csync-I)
define HBLKEN 6 // Enable Hsync blanking
define UDCNTOUT 5 // VsyncI port selection (or 5-bit compare output)

// SYNCON1
#define CLMP1 7 // Clamp generation
define CLMP0 6
define BPORCH 5 // Back porch clamp signal (or front porch)
define CLMPPOL 4 // Clamp signal polarity
define INVTVPOL 3 // Invert Vsync-O signal (or by-pass)
define INVTCHPOL 2 // Invert Hsync-O signal (or by-pass)
define POSIVPOL 1 // Positive Vsync-I polarity (or negative)
define POSIHPOL 0 // Positive Hsync-I polarity (or negative)

// SYNCON2
#define UNMIXHPERI 7 // Unmixed Hsync periods
define CTNT5SRC 5 // 5-bit counter source
define DISPSEUDO 4 // Disable pseudo sync
define DISSYNOUT 3 // Inhibit sync signal output
define SOGI 2 // SOG detection
define COMPSYNC 1 // Composite sync detection
define SYNCSRC 1
define VDD3VSEL 0 // When Vdd=3V

// DDC(IIC) part

// DCON (DDC Control Reg.)
define PREBUFEN 3 // Enable pre-buffer data register
define DDC1MAT 2 // DAR0 address match
define DDC1EN 1 // Enable DDC1 module
define SCLF 0 // Detect falling edge of SCL line

// DCCR (DDC Clock Control Reg.)
define DTXACKEN 7 // Enable transmit acknowledge
define DCLKSEL 6 // Tx clock source selection
define ENDDCINT 5 // Enable DDC module interrupt
define DDCPND 4 // DDC module interrupt pending

// DCSR0 (DDC Control/Status Reg.0)
define MST 7 // 1=master, 0=slave mode
define TXD 6 // 1=transmit, 0=receive mode
define BUSSTSP 5 // 1=bus busy or start signal
define ENDDC 4 // DDC module enable
define AL 3 // Arbitration lose
define DATAFLD 2 // 1=data field, 0=address field
define NACK 0 // Not received acknowledge

// DCSR1 (DDC Control/Status Reg.1)
define STOPDET 2 // Stop condition detection
define BUFEMT 1 // Data buffer empty
define BUFFUL 0 // Data buffer full
// Timer part
// BTCON (Watch-dog timer)
#define WDCLR 1 // Clear Basic timer counter

// TM0CON (Timer0)
#define CAPFALL 4 // Capture on falling mode
#define T0CLR 3 // Clear Timer0 counter
#define T0OVFINT 2 // Enable Timer0 overflow interrupt
#define T0CAPINT 1 // Enable Timer0 capture interrupt
#define T0CAPVS 0 // Capture source selection (1: Vsync, 0: T0CAP)

// TM1CON (Timer1)
#define T1CAPVS 7 // Capture source selection (1: Vsync, 0: T2 interval)
#define T1CAPFLEG 6 // VsyncO capture edge selection
#define T1CAPINT 5 // Enable Timer1 capture
#define T1PND 4 // Timer1 pending bit
#define T1CLR 3 // Clear Timer1 counter
#define T1OVFINT 2 // Enable Timer1 overflow interrupt

// TM2CON (Timer2)
#define T2INT 2 // Enable Timer2 interrupt

/******************************************************************************
Definition of Slave Address
******************************************************************************/
#define DEFL 0x8C // Deflection processor
#define EEP 0xA0 // EEPROM
#define PREAMP 0xDC // Video Amplifier
#define OSD 0xBA // OSD processor

/******************************************************************************
// General Registers definition
******************************************************************************

#define bit0 0

struct reg00 {
    usin keydetect : 1;
    usin mvaccel : 1;
    usin chkhfreq : 1;
    usin keyscan : 1;
    usin degaussing : 1;
    usin keyactive : 1;
    usin chksvtime : 1;
} ; // time_fgr

struct reg01 {
    usin pwronmute : 1;
    usin selfrasin : 1;
    usin recall : 1;
} ;
usin userdel : 1;
usin powerdown : 1;
usin overrange : 1;
usin vsyncdet : 1;
} ; // status_fgr

struct reg02 {
    usin ddc2b : 1;
    usin ddccmd : 1;
    usin ddcciTxd : 1;
} ; // ddc_fgr

struct reg03 {
    usin novsync : 1;
    usin nohsync : 1;
    usin nohvsync : 1;
    usin dpmsstart : 1;
    usin dpmscond : 1;
} ; // dpms_fgr

struct reg04 {
    usin dataread : 1;
    usin datasave : 1;
    usin usedeeprom : 1;
    usin nearhfreq : 1;
    usin endmodesrch : 1;
    usin nofactmode : 1;
} ; // eeprom_fgr

//**********************************************************************************

// code usch edid_tbl[0x80]= {
    0x00,0xff,0xff,0xff,0xff,0xff,0xff,0x00,
    0x4c,0x2d,0x70,0x4d,0x00,0x00,0x00,0x00,
    0x0a,0x05,0x01,0x00,0x2e,0x1f,0x17,0x71,
    0xe8,0x07,0x65,0xa0,0x57,0x46,0x9a,0x26,
    0x10,0x48,0x4c,0xff,0xfe,0x00,0x01,0x01,
    0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
    0x00,0x80,0x51,0x00,0x24,0x40,0x30,0x90,
    0x33,0x00,0x32,0xe6,0x10,0x00,0x00,0x18,
    0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
    0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
    0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
    0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
    0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
    0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
};

****
#include <S3C863A.h>
#include <insam8.h>
#include <define.h>

#define COMP_RANGE 10 // KHz (tolerance of composite-sync)
#define SEP_RANGE 5 // KHz
#define NoH_RANGE 10 // under 10KHz
#define HF_MIN 28 // normal hsync range=28KHz-96KHz
#define HF_MAX 96
#define VF_MIN 40 // normal vsync range=40Hz-160Hz
#define VF_MAX 160

usch delta_hf; // output to Timer2 interrupt routine
usin hfreq_save; // output
usch vfreq_save;
extern usin hf_new; // from Timer1(sep-sync)/Timer2(comp-sync) interrupt
extern usin vcount; // from Vsync interrupt
extern usch novsynctime; // from Vsync interrupt
extern usch tdpms100ms;

extern struct reg00 time_fgr;
extern struct reg01 status_fgr;
extern struct reg03 dpms_fgr;
extern struct reg04 eeprom_fgr;

static usin hf_old;
static usch vf_new;
static usch vf_old;
static usch tmute10ms;

static struct reg0 {
    usin stbvfreq : 1;
    usin stbhfreq : 1;
    usin ddchighspd : 1;
    usin : 3; // not used
    usin posihsync : 1;
    usin posivsync : 1;
} syncp_fgr0, *psyncp_fgr0;

static struct reg1 {
    usin scrnmute : 1;
    usin muterelse : 1;
    usin psyncout : 1;
    usin endmute : 1;
    usin normmute : 1;
    usin quitpsync : 1;
} syncp_fgr1;
extern void selfraster_end(void);  // 'main.c'
extern void osd_off(void);  // 'osd_drv.c'
extern void deflect_ini(void);  // 'initial.c'
extern void ctrl_preamp(void);  // 'initial.c'
extern void timedelay_ms10(void);  // 'initial.c'
extern void write_swiic(usch, usch, usch);  // 'swiic.c'

usch chk_hnosync_range(void);
usch chk_hf_change(void);
usch chk_pol_change(void);
usch chk_vf_change(void);
usch chk_hv_range(void);
void pseudosync_gen(void);
void chng_vsproc_src_sep(void);
void chng_vsproc_src_comp(void);
void mute_release(void);
void quit_vsproc_out(void);
void s_correct(void);
void h_lin_out(void);
void update_hsync(usin hfnew);
void stable_hsync(usin hfave);
void pola_update(void);
void set_posi_pola(void);
void chkmutsime(time(void);

void pseudosync_gen(void);

void chng_vsync_src_sep(void);
void chng_vsync_src_comp(void);
void mute_release(void);
void quit_vsproc_out(void);
void s_correct(void);
void h_lin_out(void);
void update_hsync(usin hfnew);
void stable_hsync(usin hfave);
void pola_update(void);
void set_posi_pola(void);
void chkmutsime(void);

// Strat sync-processor function
void syncprocess(void)
{
    psyncp_fgr0 = &syncp_fgr0;

    // check hsync frequency & polarity
    if(chk_pol_change())
        pseudosync_gen();
    else if(time_fgr.chkhfreq==1) {  // 10ms flag
        time_fgr.chkhfreq=0;
        chkmutsime();
        if(chk_hnosync_range() || chk_hf_change())
            pseudosync_gen();  // video mute
    }
    // check vsync source, frequency & status
    // Vsync freq. is under 40Hz
    if(novsyncntime>25) {
        novsynctime=0;
        dpms_fgr.novsync=1;
        if(dpms_fgr.nohsync==1)
            dpms_fgr.nohvsync=1;
        if(BitFals(SYNCON2,COMPSYNC))
            chng_vsproc_src_sep();  // Change Vsync input source to Vsync-I port
        else
            chng_vsproc_src_comp();  // Vsync-I port -> 5-bit U/D counter output
    }
}
} // Vsync freq. is over 40Hz
else if(status_fgr.vsyncdet==1) {
    dpms_fgr.novsync=0;
    dpms_fgr.nohvsync=0;
    status_fgr.vsyncdet=0;
    if(BitTru(SYNCON2,SOGI)) {
        BitSet(SYNCON1,BPORCH);
        BitClr(SYNCON0,HIPORT);
    }
} // Calculate Vsync freq.
if(vcount) // 'vcount' is an interval time

    vf_new=500000/vcount;

if(chk_vf_change() || chk_hv_range())
    pseudosync_gen(); // change rate of Vfreq > 1Hz, or Over frequency range

// Normal H/Vsync signal input
if(syncp_fgr1.muterelse==1) // after video-mute has been released
    ;
else if(syncp_fgr1.endmute==1) // 'endmute' flag is set after 'quit_psync_out()'
    // and if mute-delay time is passed.
    mute_release();
else if(syncp_fgr0.stbhfreq==1
    && syncp_fgr0.stbvfreq==1) // output : pseudo-sync -> input sync-signal
    quit_psync_out();
}

} // This function is executed when
void pseudosync_gen(void)
{
    if(status_fgr.selfrasin==0
        && syncp_fgr1.psyncout==0)
        // self-raster mode or already mute processing ?

        BitClr(P0,MUTEPORT); // active low
        BitSet(SYNCON1,POSIVPOL); // pseudo-Vsync polarity is positive
        BitSet(SYNCON1,POSIHPOL);
        BitClr(SYNCON2,DISPSEUDO);
        BitSet(P3,CS1);
        osd_off(); // OSD window off

syncp_fgr1.scrnmute=1;
syncp_fgr1.psyncout=1;
syncp_fgr1.quitpsync=0;
syncp_fgr1.muterelse=0;
syncp_fgr1.endmute=0;
syncp_fgr0.stbvfreq=0;
syncp_fgr0.stbhfreq=0;
eeprom_fgr.datasave=0;
time_fgr.chksvtime=0;
}
}

// Change Vsync input source : Vsync-I port <-> 5-bit up/down counter output
// void chng_vsync_src_sep(void)
{
  BitClr(SYNCON0,UDCNTOUT);  // Change Vsync input source to Vsync-I port
  BitClr(SYNCON0,HBLKEN);    // Disable Hsync blanking
  BitSet(TM1CON,T1CAPINT);   // Enable Timer1 capture mode
}

void chng_vsync_src_comp(void)
{
  BitSet(SYNCON0,UDCNTOUT);  // Input source: Vsync-I -> 5-bit u/d counter output
  BitClr(TM1CON,T1CAPINT);   // Disable T1 capture interrupt
  BitClr(SYNCON2,COMPSYNC);  // Change calculation method of Hsync frequency
  BitClr(SYNCON2,SOGI);      // => 10ms interval -> sum(each 1ms counter by 10)
                             // after stable sync signal input

  BitSet(SYNCON2,DISPSEUDO); // Clear latch status of 5-bit u/d couner
  BitClr(SYNCON2,SOGI);      // Clear SOG detection counter
}

// void quit_psync_out(void)
{
  pola_update();            // setting positive polarity for H/Vsync-O
  set_posi_pola();
  if(syncp_fgr1.quitpsync==0) {
    BitSet(SYNCON2,DISPSEUDO);  // quit pseudo-sync gen.
    syncp_fgr1.psyncout=0;
    syncp_fgr1.quitpsync=1;

    s_correct();
    h_lin_out();

    eeprom_fgr.dataread=1;      // loading PWM data from EEPROM in 'eeprom_rdwr.c'

    if(status_fgr.pwronmute==1) {
      syncp_fgr1.normmute=1;     // power-on muting time:2sec
      tmute10ms=0;
    }
  }
}
void mute_release(void)
{
    syncp_fgr1.scrnmute=0;
    //syncp_fgr1.endmute=0;
    syncp_fgr1.muterelse=1;
    BitSet(P0,MUTEPORT); // release mute-port(P0.0)
}

usch hfkhz_load(usin hfreq)
{
    usin hfreq_khz;
    hfreq_khz=hfreq;
    hfreq_khz &= 0x03ff; // hsync range is under 100KHz
    hfreq_khz /= 10;
    return hfreq_khz;
}

usch chk_hnosync_range(void)
{
    usch hf_khz;
    hf_khz=hfkhz_load(hf_new);
    if(hf_khz < NoH_RANGE) {
        // under 10KHz
        dpms_fgr.nohsync=1;
        return 1;
    }
    else {
        dpms_fgr.nohsync=0;
        dpms_fgr.nohvsync=0;
        return 0;
    }
}

usch chk_hf_change(void)
{
    usin hfreq, hf_ave, temp;
    hf_ave=(hf_new+hf_old)/2;
    hf_old=hf_new;
    delta_hf=hf_ave/10; // KHz
    _DI();
    temp=hfreq_save&0x03ff;
hfreq=(temp>=hf_new)? (temp-hf_new):(hf_new-temp)
_EI(); // always positive value
if((BitTru(SYNCON0,UDCNTOUT) && (hfreq>COMP_RANGE)) {
    update_hsync(hfreq); // update freq. of hsync input signal
    return 1;
}
else if((BitFlas(SYNCON0,UDCNTOUT) && (hfreq>SEP_RANGE)) {
    update_hsync(hfreq);
    return 1;
}
else
    stable_hsync(hf_ave); // stable state of hsync input
    return 0;
}

// void update_hsync(usin hfnew)
{
    syncp_fgr0.stbhfreq=0;
    hfreq_save &= 0xc000; // bit 15,14=polarity
    hfreq_save |= hfnew;
}

// void stable_hsync(usin hfave)
{
    syncp_fgr0.stbhfreq=1;
    hfreq_save &= 0xc000;
    hfreq_save |= hfave;
}

// **********************************************
// Check changing rate of Vsync frequency
// Tolerance of stable Vsync signal is under 1Hz
// **********************************************
usch chk_vf_change(void)
{
    usch temp;
    vf_old=vf_new;
    vfreq_save=vf_new;

temp=(vf_old>=vf_new)? (vf_old-vf_new):(vf_new-vf_old);
if((temp>1) { // temp=|vf_old-vf_new|
    syncp_fgr0.stbvfreq=0;
    return 1;
}
else { // stable Vsync signal
    syncp_fgr0.stbvfreq=1;
    return 0;
}
}
// Checking polarity change
// *******************************
usch chk_pol_change(void)
{
    if(syncp_fgr1.psyncout==0) {
        if(syncp_fgr0.posivsync != BitTru(SYNCON1,POSIVPOL)) {
            pola_update();
            return 1;
        }
    } else if(BitFals(SYNCON0,UDCNTOUT)) {
        // separate-sync
        if(syncp_fgr0.posihsync != BitTru(SYNCON1,POSIHPOL)) {
            pola_update();
            return 1;
        } else {
            set_posi_pola();
            return 0;
        }
    }
    return 0;
}

// update polarity flags
void pola_update(void)
{
    usin hf_temp, pola_temp;

    if(BitTru(SYNCON0,UDCNTOUT)) {
        // composite-sync
        if(BitTru(SYNCON1,POSIVPOL)) {
            syncp_fgr0.posivsync=1;
            syncp_fgr0.posihsync=1;
        } else {
            syncp_fgr0.posivsync=0;
            syncp_fgr0.posihsync=0;
        }
    } else {
        // separate-sync
        if(BitTru(SYNCON1,POSIVPOL))
            syncp_fgr0.posivsync=1;
        else
            syncp_fgr0.posivsync=0;
        if(BitTru(SYNCON1,POSIHPOL))
            syncp_fgr0.posihsync=1;
        else
            syncp_fgr0.posihsync=0;
    }
    hf_temp=hf_new;
hf_temp &= 0x03ff;
pola_temp = *(usin *)psyncp_fgr0; // masking except polarity flags
pola_temp <<= 8;
pola_temp &= 0xc000;
hf_temp |= pola_temp;
hfreq_save = hf_temp; // bit15/14=polarity, bit9~0=Hsync frequency
}

void set_posi_pola(void) {
  if(syncp_fgr0.posivsync==1) BitClr(SYNCON1,INVTVPOL); // by-pass Vsync signal
  else BitSet(SYNCON1,INVTVPOL); // inverting Vsync signal
  if(syncp_fgr0.posihsync==1) BitClr(SYNCON1,INVTHPOL);
  else BitSet(SYNCON1,INVTHPOL);
}

// Check range of the H/Vsync signal
// Check range of the H/Vsync signal
usch chk_hv_range(void) {
  usch hf_khz;
  hf_khz = hfkhz_load(hf_new);
  // checking range of Hsync/Vsync signal
  if(hf_khz<HF_MIN || hf_khz>HF_MAX
    || vf_new<VF_MIN || vf_new>VF_MAX) {
    status_fgr.overrange = 1;
    return 1;
  }
  else {
    // normal H/Vsync signal (28KHz<Hf<95KHz, 40Hz<Vf<160Hz)
    status_fgr.overrange = 0;
    if(status_fgr.powerdown==1) {
      // off(DPMS) mode -> Normal sync mode
      BitClr(P0,SUSPNDPORT); // release suspand port(12V line)
      timedelay_ms10();
      BitSet(P0,OFFPORT); // release off port(5V line)

      write_swiic(DEFL,HDUTY,0); // h-duty off
      deflect_ini(); // free running
      ctrl_preamp();
      tmute10ms=0;
      pseudosync_gen(); // video-mute
      status_fgr.pwronmute=0; // waiting time=2sec
      status_fgr.powerdown=0;
    }
    tdpms100ms=0;
  }
}
dpms_fgr.dpmsstart=0;
if(status_fgr.selfrasin==1)
    selfraster_end();
return 0;
}

// control s-correction cap.
void s_correction(void)
{
    usch hf_khz;
    hf_khz=hfkHz_load(hfreq_save);
    if(hf_khz<33) {
    :
    }
    else if(hf_khz<36) {
    :
    }
    :
}

// Control H-linearity with PWM6
void h_lin_out(void)
{
    usch hf_khz;
    hf_khz=hfkHz_load(hfreq_save);
    if(hf_khz<=30) {
    :
    }
    :
}

// ************************************
// 10ms timer for video-mute
// ************************************
void chkmutetime(void)
{
    if(syncp_fgr1.normmute==1) {
        if(++tmute10ms>30) { // mute delay time=300ms
            syncp_fgr1.normmute=0;
            syncp_fgr1.endmute=1;
        }
    }
    else if(status_fgr.pwronmute==0) {
        if(++tmute10ms>200) { // mute delay time=2sec
            status_fgr.pwronmute=1;
            syncp_fgr1.endmute=1;
        }
    }
}
#include <S3C863A.h>
#include <insam8.h>
#include <define.h>

usch novsynctime // to "syncproc.c"
usin vcount; // to "syncproc.c"
usch vclkcntr; // to "syncproc.c"
usin hf_new; // to "syncproc.c"
usch tbase10ms;
usch tkeyact100ms;
usch tdgaus100ms;
usch tsave100ms;
usch tdpms100ms;
usch DDC_rxtxbuf[32]; // ddc comm. buffer.
  // 1'st byte = dest. address (2Bi: 6Eh(Host to Display), 6Fh(DtoH))
  // 2'nd byte = src. address (2Bi: 51h(HtoD), 6Eh(DtoH))
  // 3'rd byte = length
  // 4'th byte = command
extern usch delta_hf; // from "syncproc.c"
extern usch wrcycletime; // from "swiic.c"
extern usch *txdata; // from "ddc2bci.c"
extern usch t0ovfcnt;

extern struct reg00 time_fgr;
extern struct reg01 status_fgr;
extern struct reg02 ddc_fgr;
extern struct reg03 dpms_fgr;
extern struct reg04 eeprom_fgr;

eextern tinyt usch tinyt *edidaddr; ;DDC
extern tinyt usch DDC_page1[0x80];

#define END_DDC 0x7f

void ms10timer(void);
void ddc2bi(void);

// Timer0 overflow interrupt (count Vsync interval)
// interrupt [t0ovf_int] void t0ovf_interrupt(void)
{
  usch pp_copy;
  pp_copy=PP; // push pp
  PP=0;
}
t0ovfcnt++;  
PP=pp_copy;
}

// Timer0 capture interrupt (capture Vsync signal)
// interrupt vsync_int void vsync_interrupt(void)
{
  usch pp_copy;
  _SB0();
  pp_copy=PP;
  PP=0;
  novsynctime=0;
  if(t0ovfcnt>10) {        // Under 195Hz (256*2*10 us) ?
    status_fgr.vsyncdet=1;
    vcount=(TM0DATA+(t0ovfcnt*256));
    t0ovfcnt=0;
    // increment vsync counter for DDC1 recovery
    _SB1();
    if(ddc_fgr.ddc2b==0 & BitFals(DCON,DDC1EN))
      vckcntr++;
    _SB0();
  } else
    t0ovfcnt=0;
  PP=pp_copy;
}

// Timer1 capture interrupt (event counter for Hsync signal(seperate-sync))
// interrupt t1cap_int void t1cap_interrupt(void)
{
  usin temp, pp_copy;
  _SB0();
  pp_copy=PP;
  PP=0;
  BitClr(TM1CON,T1PND);     // clear pending bit
  if(BitTru(TM1CON,T1CAPINT)) {
    temp=(usin)TM1DATAH;
    temp <<= 8;
    temp += (usin)TM1DATAL;
    hf_new=temp;  // Hsync frequency for 10ms(Timer2 interval * 10)
    delta_hf=0;
  }
  PP=pp_copy;
}
// Timer2 interval interrupt (1ms interval int. & Calcu. Hsync freq.(comp-sync))
//
interrupt [t2intv_int] void t2intv_interrupt(void)
{
    usin hf_cnt;
    usch pp_copy;
    static usch hf_startcnt;
    static usch hf_stopcnt;
    static usin hcount;
    static usch tbase1ms;
    static usch tbase10ms;

    _SB0();
    pp_copy=PP;
    PP=0;

    if(BitTru(SYNCON0,UDCNTOUT)) {
        hf_startcnt=hf_stopcnt;
        hf_stopcnt=TM1CNTL;
        if(BitFals(SYNCON2,UNMIXHPERI))
            hcount += delta_hf;
        else {
            hf_cnt=hf_stopcnt-hf_startcnt;
            hcount += hf_cnt;
        }
    }

    if((++tbase1ms)>=10) {// Over 10ms ?
        tbase1ms=0;
        time_fgr.chkfreq=1;
        time_fgr.keyscan=1;
        ms10timer(); // set reletive reg. to time
        if(BitTru(SYNCON0,UDCNTOUT))
            hf_new=hcount; // Hsync freq.(number of Hsync event for 10ms)
    }
    novsynctime++;
    PP=pp_copy;
}

void ms10timer(void)
{
    wrcycletime++;
    if(++tbase10ms>10) {
        tbase10ms=0;
        // Check degaussing time
        if(time_fgr.degaussing==1 && !(--tdgaus100ms)) {
            BitClr(P3,DEGAUSPORT);
            time_fgr.degaussing=0;
        }
        if(dpms_fgr.dpmscond==1) {

if(++tdpms100ms>30) // 3sec
dpms_fgr.dpmsstart=1;
} else if(time_fgr.chksvtime==1 && !(--tsave100ms)) {
time_fgr.chksvtime=0; // 2sec
eeprom_fgr.datasave=1;
} if(time_fgr.keyactive==1 && !(--tkeyact100ms))
time_fgr.keyactive=0; // 7sec

// Multi-master IIC.bus interrupt (DDC & FA)

interrupt [ddcnfa_int] void multiIIC_interrupt(void)
{
  usch pp_copy, *pt;
  int edid_addtemp, ddc_addtemp;
  static usch *rbuf_addr;
  static usch rxcntr;

  static struct reg {
    usin revA0address : 1;
  } iic_fgr;

  _SB1();
  pp_copy=PP;
  PP=0;

  edid_addtemp=(int)edidaddr;
  ddc_addtemp=(int)DDC_page1; // start address of ram buffer with EDID

  if(BitFals(DCON,DDC1EN)) {
    // DDC2 mode
    if(BitTru(DCSR0,MST)) {
      // master mode
      _NOP();
    }
  } else if(BitTru(DCSR0,TXD)) {
    // slave Tx mode
    iic_fgr.revA0address=0;
    if(BitFals(DCON,DDC1MAT)) // DDC1 match mode
      ddc2bi();
    else if(BitTru(DCSR0,NACK)) {
      // DDC communication error
      TBDR=0;
      edidaddr=DDC_page1; // edid <- start address
      BitClr(DCSR0,TXD); // return slave Rx mode
    } else {
      // 2sec
      eeprom_fgr.datasave=1;
    }
  }

// transmit EDID data
if(edid_addtemp > (ddc_addtemp+END_DDC))
    edidaddr=DDC_page1;
    TBDR=*edidaddr;
    edidaddr++;
}
// slave receive mode
else if(BitTru(DCON,DDC1MAT) || iic_fgr.revA0address==1) {
    // slave address = A0h
    if(BitTru(DCSR0,DATAFLD)) {
        if(RBDR==0x00) { // sub-address=00h ?
            TBDR=0;
edidaddr=DDC_page1;
        }
        else {
            pt=(usch*)RBDR; // random addressing case
            TBDR=*pt;
edidaddr=DDC_page1+RBDR;
        }
    }
    else // address field
        iic_fgr.revA0address=1;
} else {
    // slave address = 6Eh
ddc_fgr.ddccmd=1;
    if(rxcntr++ < 32) // check buffer overflow
        rxbuf_addr=DDC_rxtxbuf+rxcntr;
        *rxbuf_addr=RBDR; // receive ddc command/data
}
  vclkcntr=0; // DDC1 recover timer
ddc_fgr.ddc2b=1; // change DDC1 to DDC2 mode
// not yet changed to DDC2B (still DDC1 mode)
else if(BitFals(DCON,SCLF)) {
    // EDID Tx mode
    if(edid_addtemp > (ddc_addtemp+END_DDC))
        edidaddr=DDC_page1;
        TBDR=*edidaddr;
edidaddr++;
} else {
    TBDR=0;
edidaddr=DDC_page1;
    BitClr(DCON,DDC1EN); // DDC -> normal IIC
}
BitClr(DCCR,DDCPND); // clear pending bit
PP=pp_copy;
_SB0();

// DDC2Bi protocol service
void ddc2bi(void) {
    if (bytecnt-- > 1) {
        Tx buffer pointer
        TBDR = *txdata;
        txdata++;
    }
    else
        BitClr(DCSR0,TxD); // return slave Rx mode
}
OVERVIEW

The S3C8639/C863A/C8647 microcontroller supports the DDC (Display Data Channel) interface. A pair of serial data (SDA0) and serial clock (SCL0) line (except DDC1 mode) is provided to carry information between the master and peripheral that are connected to the bus. The SDA0 and SCL0 lines are bi-directional. The DDC1 mode uses vertical sync input at the Vsync-I or VCLK (VCLK is input-only). DDC1 is implemented physically using VCLK input and SDA0 output.

Protocols for the DDC2B, DDC2Bi, and DDC2B+ are supported in hardware by multi-master IIC-bus logic and in software by the EDID (Extended Display Identification) and VDIF (Video Display Interface) formats.

To control DDC interface, you write values to the following registers:

- DDC Control Register, DCON
- DDC Clock Control Register, DCCR
- DDC Control/Status Registers 0,1, DCSR0,1
- DDC Data Shift Register, DDSR
- DDC Address Registers 0,1, DAR0,1
- Transmit Pre-buffer Data Register, TBDR
- Receive Pre-buffer Data Register, RBDR
**DDC CONTROL REGISTER (DCON)**

The programmable DCON register to control the DDC is located at E9H in set 1, bank 1. It is read/write addressable. Only four bits are mapped in this register.

The DCON.0 setting lets you detect falling edges at the serial clock, SCL0. If the DCON.0 is set to "0", the SCL0 (serial clock) is still high after reset (when read), or the bit can be cleared by S/W written "0" (when write). If the DCON.1 is set to "1", falling edge is detected at SCL0 pin after RESET or after this bit is cleared by S/W.

**NOTE**

When the DDC interrupt is occurred, SCL0 line is not pull-down at the following cases:

— DDC1 mode
— Tx/Rx pre-buffer data registers ‘enable’ bit, DCON.3 is "1" (only slave mode).

The DCON.1 setting lets you select normal IIC-bus interface mode or DDC1 transmit mode. If you select normal IIC-bus interface mode (DCON.1 = "0"), SCL0 pin is selected for clock line and the SCL0 falling edge (SCLF) interrupt is disabled. Or if you select DDC1 transmit mode (DCON.1 = "1"), VCLK pin is selected for clock line and the SCLF interrupt is enable.

The DCON.2 is a DDC address match bit and read-only. When the received DDC address matches to DAR0 register, DCON.2 is "1". And when it is start, stop or reset condition, DCON.2 is "0". To enable transmit or receive pre-buffer data register, DCON.3 is used. When the transmit or receive pre-buffer data register is not used, DCON.3 is "0" (normal IIC-bus mode). DCON.3 is set by writing One to it or by reset. If DCON.3 is "1", the transmit or receive pre-buffer data register is enable.

---

**DDC Control Register (DCON)**

E9H, Set 1, Bank 1, R/W (Bit 2 is read-only)

<table>
<thead>
<tr>
<th>MSB</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>.3</th>
<th>.2</th>
<th>.1</th>
<th>.0</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCL0 (Serial Clock) falling edge detection bit (SCLF):</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = SCL0 is high after RESET (when read)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Cleared by S/W written “0” (when write)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Falling edge is detected (when read)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = No effect (when write)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmit or receive pre-buffer data register enable bit:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = Normal IIC-bus mode (Pre-buffer data registers are not used)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Pre-buffer data registers enable mode (This bit is set by writing one to it or by reset)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDC address match bit (read-only):</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 = When start or stop or reset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = When the received DDC address matches to DAR0 register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 17-1. DDC Control Register (DCON)**
DDC Clock Control Register (DCCR)

The DDC clock control register, DCCR, is located at EBH in set 1, bank 1. It is read/write addressable. DCCR settings control the following functions:

- CPU acknowledge signal (ACK) enable or suppress
- DDC clock source selection (fOSC/10 or fOSC/256)
- DDC interrupt enable or disable
- DDC interrupt pending control
- 4-bit prescaler for the serial clock (SCL0)

When DCCR.7 bit is set to "1", it is enable to acknowledgment signal. DCCR.6 is bit for transmit clock source selection by fOSC/10 or fOSC/256. DCCR.3–DCCR.0 bits (CCR3–CCR0) are 4-bit prescaler for the transmit clock (SCL0). The SCL0 clock may be "Stretched" if a slow slave device holds the clock for clock synchronization.

In the S3C8639/C863A/C8647 interrupt structure, the DDC interrupt is assigned level IRQ3, vector EAH. To enable this interrupt, you set DCCR.5 to "1". Program software can then poll the DDC interrupt pending bit(DCCR.4) to detect DDC interrupt request. When the CPU acknowledges the interrupt request from the DDC, the interrupt service routine must clear the interrupt pending condition by writing a "0" to DCCR.4.

```
<table>
<thead>
<tr>
<th>DDC Clock Control Register (DCCR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBH, Set 1, Bank 1, R/W</td>
</tr>
<tr>
<td>MSB</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Transmit acknowledge (ACK)</td>
</tr>
<tr>
<td>enable bit:</td>
</tr>
<tr>
<td>0 = Disable ACK generation</td>
</tr>
<tr>
<td>1 = Enable ACK generation</td>
</tr>
<tr>
<td>Transmit clock source selection bit:</td>
</tr>
<tr>
<td>0 = fosc/10</td>
</tr>
<tr>
<td>1 = fosc/256</td>
</tr>
<tr>
<td>DDC module interrupt enable bit:</td>
</tr>
<tr>
<td>0 = Disable DDC interrupt</td>
</tr>
<tr>
<td>1 = Enable DDC interrupt</td>
</tr>
<tr>
<td>DDC module interrupt pending flag:</td>
</tr>
<tr>
<td>0 = When write &quot;0&quot; to this bit (write &quot;1&quot; has no effect)</td>
</tr>
<tr>
<td>0 = When DCSR.4 is &quot;0&quot;</td>
</tr>
<tr>
<td>1 = When slave address match occurred</td>
</tr>
<tr>
<td>1 = When arbitration lost (master mode)</td>
</tr>
<tr>
<td>1 = When a 1-byte transmit or receive operation is terminated</td>
</tr>
<tr>
<td>1 = As soon as the DDC1 mode is enable after the prebuffer is used</td>
</tr>
</tbody>
</table>

Figure 17-2. DDC Clock Control Register (DCCR)
```
Table 17-1. Sample Timing Calculations for the DDC Transmit Clock (SCL0)

<table>
<thead>
<tr>
<th>DCCR.3–DCCR.0 Value (IICLK = 4 MHz)</th>
<th>IICLK (DCCR.3–DCCR.0 Settings + 1)</th>
<th>(f_{OSC} = 8 MHz) DCCR.6 = 0 (f_{OSC}/10) IICLK = 400 kHz</th>
<th>(f_{OSC} = 8 MHz) DCCR.6 = 1 (f_{OSC}/256) IICLK = 15.625 kHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>IICLK/1</td>
<td>400 kHz</td>
<td>15.625 kHz</td>
</tr>
<tr>
<td>0001</td>
<td>IICLK/2</td>
<td>200 kHz</td>
<td>7.1825 kHz</td>
</tr>
<tr>
<td>0010</td>
<td>IICLK/3</td>
<td>133.3 kHz</td>
<td>5.2038 kHz</td>
</tr>
<tr>
<td>0011</td>
<td>IICLK/4</td>
<td>100 kHz</td>
<td>3.9063 kHz</td>
</tr>
<tr>
<td>0100</td>
<td>IICLK/5</td>
<td>80.0 kHz</td>
<td>3.1250 kHz</td>
</tr>
<tr>
<td>0101</td>
<td>IICLK/6</td>
<td>66.7 kHz</td>
<td>2.6042 kHz</td>
</tr>
<tr>
<td>0110</td>
<td>IICLK/7</td>
<td>57.1 kHz</td>
<td>2.2321 kHz</td>
</tr>
<tr>
<td>0111</td>
<td>IICLK/8</td>
<td>50.0 kHz</td>
<td>1.9531 kHz</td>
</tr>
<tr>
<td>1000</td>
<td>IICLK/9</td>
<td>44.4 kHz</td>
<td>1.7361 kHz</td>
</tr>
<tr>
<td>1001</td>
<td>IICLK/10</td>
<td>40.0 kHz</td>
<td>1.5625 kHz</td>
</tr>
<tr>
<td>1010</td>
<td>IICLK/11</td>
<td>36.4 kHz</td>
<td>1.4205 kHz</td>
</tr>
<tr>
<td>1011</td>
<td>IICLK/12</td>
<td>33.3 kHz</td>
<td>1.3021 kHz</td>
</tr>
<tr>
<td>1100</td>
<td>IICLK/13</td>
<td>30.8 kHz</td>
<td>1.2019 kHz</td>
</tr>
<tr>
<td>1101</td>
<td>IICLK/14</td>
<td>28.7 kHz</td>
<td>1.1160 kHz</td>
</tr>
<tr>
<td>1110</td>
<td>IICLK/15</td>
<td>26.7 kHz</td>
<td>1.0417 kHz</td>
</tr>
<tr>
<td>1111</td>
<td>IICLK/16</td>
<td>25.0 kHz</td>
<td>0.9766 kHz</td>
</tr>
</tbody>
</table>
The DDC control/status register 0, DCSR0, is located at ECH in set 1, bank 1. It is read/write addressable. Although the DCSR0 register is read/write addressable, four bits are read only: DCSR0.3–DCSR0.0.

DDSR0 register settings are used to control or monitor the following functions:

- Master/slave transmit or receive mode selection
- Bus busy status flag
- DDC module enable or disable
- Failed bus arbitration procedure status flag
- Received address register match status flag
- Last received bit status flag (No ACK = "1", ACK = "0")

DDSR0.3 is automatically set to "1" when a bus arbitration procedure fails over serial I/O interface, while the IIC-bus is set to master mode. If slave mode is selected, DCSR0.3 is automatically set to "1" if the value of DCSR0.7–.4 are changed by program when the busy signal bit, DCSR0.5 is "1", and the DDC address/data field classification bit, DCSR0.2 is "0". When the DDC module is transmitting a One to SDA0 line but detected a Zero from SDA0 line in master mode at the slave mode, DCSR0.3 is set.
DDC CONTROL/STATUS REGISTER 1 (DCSR1)

The DDC control/status register 1, called DCSR1, is located at EDH in set 1, bank 1. It is read/write addressable. Only three bits are mapped in this register. Two bits are read-only: DCSR1.1 and DCSR1.0.

DCSR1 register settings are used to control or monitor the following functions:

- Stop condition detection flag
- Data buffer empty status flag
- Data buffer full status flag

---

**Figure 17-4. DDC Control/Status Register 1 (DCSR1)**
DDC DATA SHIFT REGISTER (DDSR)

The DDC data shift register for DDC interface, called DDSR, is located at F1H in set 1, bank 1. It is read/write addressable. The transmitted data output serially from most significant bit (MSB) after writing a data to DDSR. In addition, the received data from the IIC-bus input to DDSR serially from least significant bit (LSB). DDSR register capable to write while DCSR0.4 is set to "1" and DCON.3 is set to "0", and to read anytime regardless of ICSR0.4.

Figure 17-5. DDC Data Shift Register (DDSR)
**DDC ADDRESS REGISTER 0 (DAR0)**

The DDC address register 0 for DDC interface, called DAR0, is located at EAH in set 1, bank 1. It is read/write addressable. This register is consisted of 4-bit slave address latch (DAR0.3–DAR0.0 is not mapped at the S3C8639/C863A/C8647). DAR0 register is capable to write when DCSR0.4 is "0", and to read anytime regardless of DCSR0.4. 4-bits of the DAR0 register are operate only when receive the slave address.

![Figure 17-6. DDC Address Register 0 (DAR0)](image)

**DDC ADDRESS REGISTER 1 (DAR1)**

The DDC address register 1 for DDC interface, called DAR1, is located at EEH in set 1, bank 1. It is read/write addressable. This register is consisted of 7-bit slave address latch (DAR1.0 is not mapped at the S3C8639/C863A/C8647). DAR1 register is capable to write when DCSR0.4 is "0", and to read anytime regardless of DCSR0.4. 7-bits of the DAR1 register are operate only when receive the slave address.

![Figure 17-7. DDC Address Register 1 (DAR1)](image)
TRANSMIT PRE-BUFFER DATA REGISTER (TBDR)

The transmit pre-buffer data register, called TBDR, is located at EFH in set 1, bank 1. It is read/write addressable. TBDR register is capable to write when DCSR0.4 is "1", and to read anytime regardless of DCSR0.4.

When DCON.3 (TBDR enable bit) = "1" and DCSR1.1 = "0", the data written into this register will be automatically downloaded to the DDC data shift register (DDSR) and generate the interrupt request when the module detects the calling address is matched and the bit 0 of the received data is "1" (DCSR0.7-6 = "01") and when the data in the DDSR register has been transmitted with received acknowledge bit, DCSR0.0 = "0".

At this interrupt service routine, the CPU must write the next data to the TBDR register to clear DCSR1.1 and for the auto downloading of data to the DDSR register after the data in the DDSR register is transmitted over again with DCSR0.0 = "0". When DCON.3 = "1" and DCSR1.1 = "1", the data stored in this register will not be downloaded to the module detects the calling address is matched and the bit 0 of the received data is "1".

At this interrupt service routine, the CPU must write the current data and rewrite the next data to the TBDR register to clear DCSR1.1. If the master receiver doesn't acknowledge the transmitted data, DCSR0.0 = "1", the module will release the SDA line for master to generate STOP or repeated START conditions. If DCON.3 (TBDR enable bit) is "0", the module will pull-down the SCL line in the IIC-bus interrupt service routine when the DCSR0.2 is "1". And the module will release the SCL line if the CPU writes a data to the DDSR registers and the interrupt pending bit is cleared.

<table>
<thead>
<tr>
<th>Transmit Pre-buffer Data Register (TBDR)</th>
<th>EFH, Set 1, Bank 1, R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>8-bit transmit pre-buffer data register: Write enable when DCSR0.4 is &quot;1&quot;. Read enable anytime.</td>
</tr>
<tr>
<td>.7 .6 .5 .4 .3 .2 .1 LSB</td>
<td></td>
</tr>
</tbody>
</table>
RECEIVE PRE-BUFFER DATA REGISTER (RBDR)

The receive pre-buffer data register, called RBDR, is located at F0H in set 1, bank 1. It is read-only addressable. RBDR register is capable to read anytime.

RBDR register will be updated after a data byte is received when the DCSR0.2 is "1" and the DCSR1.0 will be "1". The read operation of RBDR register will clear the DCSR1.0. After the DCSR1.0 is cleared, the register can load the received data again and set the DCSR1.0.

![Receive Pre-buffer Data Register (RBDR)](image)

**Figure 17-9. Receive Pre-buffer Data Register (RBDR)**

![DDC1 Mode Timing Diagram (One-Byte Transfer)](image)

**Figure 17-10. DDC1 Mode Timing Diagram (One-Byte Transfer)**

Where,

- **tv** = Data valid time (min. 30 us)
- **tH** = VCLK high pulse width (min. 20 us)
- **tL** = VCLK low pulse width (min. 20 us)

(Max. VCLK input frequency = 25 kHz)

**NOTE:** MSB (Most Significant Bit) first output in each bytes.
Figure 17-11. DDC Module Block Diagram
THE DDC INTERFACE

DDC2BI MODE

Overview
DDC2B capable graphic hosts have limited and mono-directional communications with the display devices. At the contrary, DDC2Bi mode is an extension of the DDC2B level in order to offer a bi-direction communication between the computer graphic host and the display device. DDC2Bi brings DDC2B+ functionality to DDC2B graphic hosts using a simple S/W driver. So DDC2Bi display device is made by simple S/W upgrade to DDC2B+ capable displays. DDC2Bi protocol relies on the DDC2B H/W definition and the Access bus messages protocol. The Graphic host behaves as an IIC single master host, and the display device behaves as an IIC slave device. The DDC2Bi is a modification of the Access bus multi-master protocol to fit single master communication.

DDC2Bi Host and Display Device
DDC2Bi host is considered as an IIC single master capable device. The virtual IIC slave address of the host is 50/51H. But DDC2Bi display device is considered as a fixed address display device (6E/6F), and uses only IIC slave mode to communicate with the host.

A display dependent devices are geographically located around the display and follow the same DDC2Bi data protocol than the display device. And fixed address IIC slave devices group all the existing stand-alone and brain-less IIC slave device. These devices can coexist and be connected to the DDC/IIC-bus.

DDC2Bi S/W Implementation
In order to describe the display that the received message is of DDC2Bi type, the source address byte bit 0 is set. And when the host expects an answer from the display, the host reads the answer message at the display device slave address 6FH. The checksum is still computed by using the 50H, virtual host address.

A null message can be defined as an Access bus message without any data byte. The null message is used in the following cases:

— To detect that the display is DDC2Bi capable by reading it at 6FH, IIC slave address.
— To describe the host that the display does not have any answer to give to the host
— The enable application report has not been sent prior application messages exchange with the host

DDC2Bi Communication
In the DDC2Bi communication, it is capable to retrials when a communication fails (bus error or bad checksum). So the host is responsible for resending its message and trying to get an answer from the display again. When the communication fail is occurred, the DDC2Bi devices must answer by the retry of host.

The DDC2Bi capable device must properly send and receive all its supported messages. This determines the maximum internal data communication buffer required size for proper display operation. If the device receive a message which size is lager than the maximum supported by the device, the message be accepted entirely by the device, but does not need to be supported internally, and then be discarded. Therefore the DDC2Bi capable device must acknowledge all received data bytes from the host.
THE IIC-BUS INTERFACE

The S3C8639/C863A/C8647 IIC-bus interface has four operating modes:

- Master transmitter mode
- Master receive mode
- Slave transmitter mode
- Slave receive mode

Functional relationships between these operating modes are described below.

START AND STOP CONDITIONS

When the IIC-bus interface is inactive, it is in slave mode. The interface is therefore always in slave mode when a start condition is detected on the SDA line. (A start condition is a High-to-Low transition of the SDA line while the clock signal, SCL, is High level.) When the interface enters master mode, it initiates a data transfer and generates the SCL signal.

A start condition initiates a one-byte serial data transfer over the SDA line and a stop condition ends the transfer. (A stop condition is a Low-to-High transition of the SDA line while SCL is High level.) Start and stop conditions are always generated by the master. The IIC-bus is “busy” when a start condition is generated. A few clocks after a stop condition is generated, the IIC-bus is again “free”.

When a master initiates a start condition, it sends its slave address onto the bus. The address byte consists of a 7-bit address and a 1-bit transfer direction indicator (that is, write or read). If bit 8 is “0”, a transmit operation (write) is indicated; if bit 8 is “1”, a request for data (read) is indicated.

The master ends the indicated transfer operation by transmitting a stop condition. If the master wants to continue sending data over the bus, it can generate another start condition and another slave address. In this way, read-write operations can be performed in various formats.
Figure 17-12. Start and Stop Conditions

Figure 17-13. Input Data Protocol
DATA TRANSFER FORMATS

Every byte put on the SDA line must be eight bits in length. The number of bytes which can be transmitted per transfer is unlimited. The first byte following a start condition is the address byte. This address byte is transmitted by the master when the IIC-bus is operating in master mode. Each byte must be followed by an acknowledge (ACK) bit. Serial data and addresses are always sent MSB first.

### Single Byte Write Mode Format

```
S  Slave Address  W  A  DATA  A  P  
```

“0” (write)  Data Transferred (Data + Acknowledge)

### Multigle Byte Write Mode Format

```
S  Slave Address  W  A  Sub Address  A  DATA  A  DATA  A  P  
```

“0” (write)  Data Transferred (Data n + Acknowledge)

Auto Increment of Sub Address

### Single Byte Read Mode Format

```
S  Slave Address  R  A  DATA  A  P  
```

“1” (read)  Data Transferred (Data + Acknowledge)

### Multigle Byte Read Mode Format

```
S  Slave Address  W  A  Sub Address  A  S  Slave Address  R  A  DATA  A  DATA  A  P  
```

“0” (write)  Data Transferred (Data n + Acknowledge)

### NOTES:
1.  S: start, A: acknowledge, P: stop
2.  The "Sub Address" indicates the internal address of the slave device.

Figure 17-14. IIC-Bus Interface Data Formats
ACK SIGNAL TRANSMISSION

To complete a one-byte transfer operation, the receiver must send an ACK bit to the transmitter. The ACK pulse occurs at the ninth clock of the SCL line (eight clocks are required to complete the one-byte transfer). The clock pulse required for the transmission of the ACK bit is always generated by the master.

The transmitter releases the SDA line (that is, it sends the SDA line High) when the ACK clock pulse is received. The receiver must drive the SDA line Low during the ACK clock pulse so that SDA is Low during the High period of the ninth SCL pulse.

The ACK bit transmit function can be enabled and disabled by software (DCCR.7). However, the ACK pulse on the ninth clock of SCL is required to complete a one-byte data transfer operation.

Figure 17-15. Acknowledge Response from Receiver
READ-WRITE OPERATIONS

When operating in transmitter mode, the IIC-bus interface interrupt routine waits for the master (the KS88C6332/C6348) to write a data byte into the IIC-bus data shift register (DDSR). To do this, it holds the SCL line Low prior to transmission.

In receive mode, the IIC-bus interface waits for the master to read the byte from the IIC-bus data shift register (DDSR). It does this by holding the SCL line Low following the complete reception of a data byte.

BUS ARBITRATION PROCEDURES

Arbitration takes place on the SDA line to prevent contention on the bus between two masters. If a master with a SDA High level detects another master with an SDA active Low level, it will not initiate a data transfer because the current level on the bus does not correspond to its own. The master which loses the arbitration can generate SCL pulses only until the end of the last-transmitted data byte. The arbitration procedure can continue while data continues to be transferred over the bus.

The first stage of arbitration is the comparison of address bits. If a master loses the arbitration during the addressing stage of a data transfer, it is possible that the master which won the arbitration is attempting to address the master which lost. In this case, the losing master must immediately switch to slave receiver mode.

ABORT CONDITIONS

If a slave receiver does not acknowledge the slave address, it must hold the level of the SDA line High. This signals the master to generate a stop condition and to abort the transfer.

If a master receiver is involved in the aborted transfer, it must also signal the end of the slave transmit operation. It does this by not generating an ACK after the last data byte received from the slave. The slave transmitter must then release the SDA to allow a master to generate a stop condition.

CONFIGURING THE IIC-BUS

To control the frequency of the serial clock (SCL), you program the 4-bit prescaler value in the DCCR register. The IIC-bus interface address is stored in IIC-bus address register, DIAR0/DAR1. (By default, the IIC-bus interface address is an unknown value.)
SLAVE IIC-BUS INTERFACE (Only S3C863X)

OVERVIEW

The S3C8639/C863A microcontroller supports a slave only IIC-bus serial interface.

A dedicated serial data line (SDA) and a serial clock line (SCL) carry information between bus master and slave devices which are connected to the IIC-bus. The SDA is bi-directional. But in the S3C8639/C863A/C8647, the SCL line is uni-directional (input only).

S3C8639/C863A microcontroller can receive and transmit serial data to and from master. When the IIC-bus is free, the SDA and SCL lines are both at high level.

To control slave-only IIC-bus operations, you write values to the following registers:

- Slave only IIC-bus control/status register, SICSR
- Slave only IIC-bus Tx/Rx data shift register, SIDSR
- Slave only IIC-bus address register, SIAR

Start and Stop conditions are always generated by the master. A 7-bit address value in the first data byte that is put onto the bus after the Start condition is initiated determines which slave device the bus master selects. The 8th bit determines the direction of the transfer (read or write).

Every data byte that is put onto the SDA line must total eight bits. The number of bytes which can be sent or received per bus transfer operation is unlimited.

Refer to the IIC-bus interface (slave Tx/Rx) of chapter 17 for the protocol of the slave IIC-bus at the S3C8639/C863A.
SLAVE ONLY IIC-BUS CONTROL/STATUS REGISTER (SICSR)

The slave only IIC-bus control/status register, SICSR, is located in set 1, bank 1, at address F2H. SICSR register settings are used to control or monitor the following slave IIC-bus functions (see figure 18-4):

— Slave IIC-bus acknowledgement (ACK) signal generation enable or suppress
— Slave IIC-bus module enable
— Slave IIC-bus Tx/Rx interrupt enable
— Slave IIC-bus Tx/Rx interrupt pending condition control
— Slave IIC-bus Tx/Rx mode status detect/control
— Slave IIC-bus busy status detect
— Slave IIC-bus address match status detect
— Received acknowledge signal detect (No ACK = “1”, ACK = “0”)

![Slave Only IIC-Bus Control/Status Register (SICSR)](image)

Figure 18-1. Slave only IIC-Bus Control/Status Register (SICSR)
SLAVE ONLY IIC-BUS TRANSMIT/RECEIVE DATA SHIFT REGISTER (SIDSR)

The slave IIC-bus data shift register, SIDSR, is located in set 1, bank 1, at address F4H. In a transmit operation, data that is written to the IIC is transmitted serially.

The SICSR.6 setting enables or disables serial transmit/receive operations. When SICSR.6 = “1”, data can be written to the shift register. The slave IIC-bus shift register can, however, be read at any time, regardless of the current SICSR.6 setting.

<table>
<thead>
<tr>
<th>LSbild</th>
<th>MSbild</th>
<th>8-bit data shift register for slave IIC-bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>.0</td>
<td>.1</td>
<td>.2</td>
</tr>
<tr>
<td>.3</td>
<td>.4</td>
<td>.5</td>
</tr>
<tr>
<td>.6</td>
<td>.7</td>
<td></td>
</tr>
</tbody>
</table>

When SICS, 6 = “0”, write operation is enabled. You can read the SIDSR data value at anytime, regardless of the current SICS,6 setting.

SLAVE ONLY IIC-BUS ADDRESS REGISTER (SIAR)

The address register for the IIC-bus interface, SIAR, is located, in set 1, bank 1, at address F3H. It is used to store a latched 7-bit slave address. This address is mapped to IAR.7–IAR.1; bit 0 is not used (see figure 18-3).

The latched slave address is compared to the next received slave address.

<table>
<thead>
<tr>
<th>LSbild</th>
<th>MSbild</th>
<th>7-bit slave address, latched from the IIC-bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>.0</td>
<td>.1</td>
<td>.2</td>
</tr>
<tr>
<td>.3</td>
<td>.4</td>
<td>.5</td>
</tr>
<tr>
<td>.6</td>
<td>.7</td>
<td></td>
</tr>
</tbody>
</table>

These bits are operate only when receive the slave address. When SICS,6 = “0”, read operation is enabled. You can read the SIDSR data value at any time, regardless of the current SICS,6 setting.

Figure 18-2. Slave Only IIC-Bus Tx/Rx Data Shift Register (SIDSR)

Figure 18-3. Slave only IIC-Bus Address Register (SIAR)
NOTE: The IIC-bus interrupt (IRQ7) is generated when a 1-byte receive or transmit operation is terminated before the shift operation has been completed.

Figure 18-4. IIC-Bus Block Diagram
OVERVIEW

In this section, S3C8639/C863A/C8647 electrical characteristics are presented in tables and graphs. The information is arranged in the following order:

— Absolute maximum ratings
— D.C. electrical characteristics
— Data retention supply voltage in stop mode
— Stop mode release timing when initiated by a reset
— I/O capacitance
— A/D Converter electrical characteristics
— A.C. electrical characteristics
— Input timing measurement points for P0.0–P0.2 and TM0CAP
— Oscillation characteristics
— Oscillation stabilization time
— Clock timing measurement points for $X_{\text{IN}}$
— Schmitt trigger characteristics
— Power-on reset circuit characteristics
Table 19-1. Absolute Maximum Ratings

\( T_A = 25°C \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Rating</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply voltage</td>
<td>( V_{DD} )</td>
<td>–</td>
<td>– 0.3 to + 6.5</td>
<td>V</td>
</tr>
<tr>
<td>Input voltage</td>
<td>( V_{I1} )</td>
<td>Type G-3 (n-channel open drain)</td>
<td>– 0.3 to + 7.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{I2} )</td>
<td>All port pins except ( V_{I1} )</td>
<td>– 0.3 to ( V_{DD} + 0.3 )</td>
<td></td>
</tr>
<tr>
<td>Output voltage</td>
<td>( V_O )</td>
<td>All output pins</td>
<td>– 0.3 to ( V_{DD} + 0.3 )</td>
<td></td>
</tr>
<tr>
<td>Output current High</td>
<td>( I_{OH} )</td>
<td>One I/O pin active</td>
<td>– 10 mA</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All I/O pins active</td>
<td>– 60 mA</td>
<td></td>
</tr>
<tr>
<td>Output current Low</td>
<td>( I_{OL} )</td>
<td>One I/O pin active</td>
<td>+ 30 mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total pin current except port 3</td>
<td>+ 100 mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sync-processor I/O pins and IIC-bus clock and data pins</td>
<td>+ 150 mA</td>
<td></td>
</tr>
<tr>
<td>Operating temperature</td>
<td>( T_A )</td>
<td>–</td>
<td>– 40 to + 85°C</td>
<td>°C</td>
</tr>
<tr>
<td>Storage temperature</td>
<td>( T_{STG} )</td>
<td>–</td>
<td>– 65 to + 150°C</td>
<td></td>
</tr>
</tbody>
</table>

Table 19-2. D.C. Electrical Characteristics

\( T_A = -40°C \) to + 85°C, \( V_{DD} = 3.0 \) V to 5.5 V (S3C863X), \( V_{DD} = 4.0 \) V to 5.5 V (S3C8647)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input High voltage</td>
<td>( V_{IH1} )</td>
<td>All input pins except ( V_{IH2}, V_{IH3} ) and ( V_{IH4} )</td>
<td>0.8 ( V_{DD} )</td>
<td>–</td>
<td>( V_{DD} )</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>( V_{IH2} )</td>
<td>( X_{IN} )</td>
<td>( V_{DD} - 0.5 )</td>
<td></td>
<td>( V_{DD} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{IH3} )</td>
<td>TTL input (Hsync-I, Vsync-I, and Csync-I)</td>
<td>2.0</td>
<td></td>
<td>( V_{DD} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{IH4} )</td>
<td>SCL0/SDA0, SCL1/SDA1</td>
<td>0.7( V_{DD} )</td>
<td></td>
<td>( V_{DD} )</td>
<td></td>
</tr>
<tr>
<td>Input Low voltage</td>
<td>( V_{IL1} )</td>
<td>All input pins except ( V_{IL2}, V_{IL3} )</td>
<td>–</td>
<td></td>
<td>0.2 ( V_{DD} )</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>( V_{IL2} )</td>
<td>( X_{IN} )</td>
<td></td>
<td></td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{IL3} )</td>
<td>TTL input (Hsync-I, Vsync-I, and Csync-I)</td>
<td></td>
<td></td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{IL4} )</td>
<td>SCL0/SDA0, SCL1/SDA1</td>
<td></td>
<td></td>
<td>0.3( V_{DD} )</td>
<td></td>
</tr>
<tr>
<td>Output High voltage</td>
<td>( V_{OH1} )</td>
<td>( V_{DD} = 5 ) V ± 10%; ( I_{OH} = -15 ) mA (S3C863x), ( I_{OH} = -14 ) mA (S3C8647); Port 3.6–3.7</td>
<td>( V_{DD} - 1.2 )</td>
<td></td>
<td>–</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{OH2} )</td>
<td>( V_{DD} = 5 ) V ± 10%; ( I_{OH} = -4 ) mA (S3C863x), ( I_{OH} = -3.6 ) mA (S3C8647); Port 1.2, Port 3.0–3.5</td>
<td>( V_{DD} - 1.0 )</td>
<td></td>
<td>–</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{OH3} )</td>
<td>( V_{DD} = 5 ) V ± 10%; ( I_{OH} = -2 ) mA; Port 0, 2, Clamp-O, H, and Vsync-O</td>
<td>( V_{DD} - 1.2 )</td>
<td></td>
<td>–</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{OH4} )</td>
<td>( V_{DD} = 5 ) V ± 10%; ( I_{OH} = 6 ) mA; Port 1.0–P1.1, SCL0 and SDA0</td>
<td>( V_{DD} - 1.0 )</td>
<td></td>
<td>–</td>
<td></td>
</tr>
</tbody>
</table>
Table 19-2. D.C. Electrical Characteristics (Continued)

\(T_A = -40^\circ C \text{ to } +85^\circ C, V_{DD} = 3.0 \text{ V to } 5.5 \text{ V (S3C863X), } V_{DD} = 4.0 \text{ V to } 5.5 \text{ V (S3C8647)}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Low voltage</td>
<td>(V_{OL1})</td>
<td>(V_{DD} = 5 \text{ V } \pm 10% ; I_{OL} = 15 \text{ mA} ) Port 3.6–3.7</td>
<td>–</td>
<td>–</td>
<td>0.4</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>(V_{OL2})</td>
<td>(V_{DD} = 5 \text{ V } \pm 10% ; I_{OL} = 4 \text{ mA} ) Port 3.0–3.5 and Port 1.2</td>
<td>–</td>
<td>–</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(V_{OL3})</td>
<td>(V_{DD} = 5 \text{ V } \pm 10% ; I_{OL} = 2 \text{ mA} ) Port 0, 2, Clamp-O, H, and Vsync-O</td>
<td>–</td>
<td>–</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(V_{OL4})</td>
<td>(V_{DD} = 5 \text{ V } \pm 10% ; I_{OL} = 6 \text{ mA} ) Port 1.0–1.1; SCL0 and SDA0</td>
<td>–</td>
<td>–</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>Input High leakage current</td>
<td>(I_{LH1})</td>
<td>(V_{IN} = V_{DD}) All input pins except (X_{IN}, X_{OUT})</td>
<td>–</td>
<td>–</td>
<td>3</td>
<td>(\mu)A</td>
</tr>
<tr>
<td></td>
<td>(I_{LH2})</td>
<td>(V_{IN} = V_{DD}; X_{OUT}) only</td>
<td>–</td>
<td>–</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(I_{LH3})</td>
<td>(V_{IN} = V_{DD}; X_{IN}) only</td>
<td>2.5</td>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Input Low leakage current</td>
<td>(I_{LIL1})</td>
<td>(V_{IN} = 0 \text{ V}; All input pins except } X_{IN}, X_{OUT}, X_{OUT}, \text{ RESET, Hsync, &amp; Vsync} )</td>
<td>–</td>
<td>–</td>
<td>–3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(I_{LIL2})</td>
<td>(V_{IN} = 0 \text{ V}; X_{OUT}) only</td>
<td>–</td>
<td>–</td>
<td>–20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(I_{LIL3})</td>
<td>(V_{IN} = 0 \text{ V}; X_{IN}) only</td>
<td>–2.5</td>
<td>–6</td>
<td>–20</td>
<td></td>
</tr>
<tr>
<td>Output High leakage current</td>
<td>(I_{LOH1})</td>
<td>(V_{OUT} = V_{DD})</td>
<td>–</td>
<td>–</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Output Low leakage current</td>
<td>(I_{LOL1})</td>
<td>(V_{OUT} = 0 \text{ V})</td>
<td>–</td>
<td>–</td>
<td>–3</td>
<td></td>
</tr>
<tr>
<td>Pull-up resistor</td>
<td>(R_{U1})</td>
<td>(V_{IN} = 0 \text{ V}; V_{DD} = 5 \text{ V } \pm 10% ) Ports 3.7–3.4</td>
<td>20</td>
<td>47</td>
<td>80</td>
<td>k\Omega</td>
</tr>
<tr>
<td></td>
<td>(R_{U2})</td>
<td>(V_{IN} = 0 \text{ V}; V_{DD} = 5 \text{ V } \pm 10% ) RESET only</td>
<td>150</td>
<td>280</td>
<td>480</td>
<td></td>
</tr>
<tr>
<td>Pull-down resistor</td>
<td>(R_{D})</td>
<td>(V_{IN} = 0 \text{ V}; V_{DD} = 5 \text{ V } \pm 10% ) Hsync &amp; Vsync</td>
<td>150</td>
<td>300</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>Supply current (note)</td>
<td>(I_{DD1})</td>
<td>(V_{DD} = 5 \text{ V } \pm 10% ) Operation mode; 12 MHz crystal (C_1 = C_2 = 22\text{pF})</td>
<td>–</td>
<td>10</td>
<td>20</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td>(I_{DD2})</td>
<td>(V_{DD} = 5 \text{ V } \pm 10% ) Idle mode; 12 MHz crystal (C_1 = C_2 = 22\text{pF})</td>
<td>–</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(I_{DD3})</td>
<td>(V_{DD} = 5 \text{ V } \pm 10% ) Stop mode</td>
<td>100</td>
<td>150</td>
<td>50</td>
<td>(\mu)A</td>
</tr>
</tbody>
</table>

**NOTE:** Supply current does not include drawn internal pull-up/pull-down resistors and external loads of output.
Table 19-3. Data Retention Supply Voltage in Stop Mode

\( (T_A = -40 \, ^\circ C \text{ to } +85 \, ^\circ C) \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data retention supply voltage</td>
<td>( V_{DDDR} )</td>
<td>Stop mode</td>
<td>2</td>
<td>–</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>Data retention supply current</td>
<td>( I_{DDDR} )</td>
<td>Stop mode, ( V_{DDDR} = 2.0 , V )</td>
<td>–</td>
<td>–</td>
<td>5</td>
<td>( \mu A )</td>
</tr>
</tbody>
</table>

NOTES:
1. During the oscillator stabilization wait time (\( t_{WAIT} \)), all CPU operations must be stopped.
2. Supply current does not include drawn through internal pull-up resistors and external output current loads.

![Figure 19-1. Stop Mode Release Timing When Initiated by a Reset](image-url)

Table 19-4. Input/Output Capacitance

\( (T_A = -40 \, ^\circ C \text{ to } +85 \, ^\circ C, \, V_{DD} = 0 \, V) \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input capacitance</td>
<td>( C_{IN} )</td>
<td>( f = 1 , MHz; \text{ unmeasured pins are connected to } V_{SS} )</td>
<td>–</td>
<td>–</td>
<td>10</td>
<td>pF</td>
</tr>
<tr>
<td>Output capacitance</td>
<td>( C_{OUT} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O capacitance</td>
<td>( C_{IO} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 19-5. A/D Converter Electrical Characteristics (S3C863X)

\(T_A = -40^\circ C \text{ to } +85^\circ C, \ V_{DD} = 3.0 \text{ V to } 5.5 \text{ V, } V_{SS} = 0 \text{ V}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td></td>
<td></td>
<td>–</td>
<td>8</td>
<td>–</td>
<td>bit</td>
</tr>
<tr>
<td>Total accuracy</td>
<td></td>
<td>(V_{DD} = 5 \text{ V} )</td>
<td>–</td>
<td>–</td>
<td>± 2</td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conversion time = 5 (\mu)s</td>
<td>–</td>
<td>–</td>
<td>± 2</td>
<td>LSB</td>
</tr>
<tr>
<td>Integral linearity error</td>
<td>ILE</td>
<td>AVREF = 5 \text{ V}</td>
<td>–</td>
<td>± 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differential linearity error</td>
<td>DLE</td>
<td>AVREF = 0 \text{ V}</td>
<td>± 1</td>
<td>± 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offset error of top</td>
<td>EOT</td>
<td></td>
<td>± 1</td>
<td>± 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offset error of bottom</td>
<td>EOB</td>
<td></td>
<td>± 0.5</td>
<td>± 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conversion time (^{(1)})</td>
<td>(t_{CON})</td>
<td>8-bit conversion</td>
<td>20</td>
<td>–</td>
<td>170</td>
<td>(\mu)s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(48 \times n/f_{OSC}^{(3)}),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(n = 1, 4, 8, 16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analog input voltage</td>
<td>(V_{IAN})</td>
<td></td>
<td>–</td>
<td>AVSS</td>
<td>AVREF</td>
<td>V</td>
</tr>
<tr>
<td>Analog input impedance</td>
<td>(R_{AN})</td>
<td></td>
<td>2</td>
<td>1000</td>
<td>–</td>
<td>M(\Omega)</td>
</tr>
<tr>
<td>Analog reference voltage</td>
<td>AVREF</td>
<td></td>
<td>2.5</td>
<td>–</td>
<td>(V_{DD})</td>
<td>V</td>
</tr>
<tr>
<td>Analog ground</td>
<td>AVSS</td>
<td></td>
<td>–</td>
<td>(V_{SS})</td>
<td>(V_{SS} + 0.3)</td>
<td>V</td>
</tr>
<tr>
<td>Analog input current</td>
<td>IADIN</td>
<td>AVREF = (V_{DD} = 5\text{ V})</td>
<td>–</td>
<td>–</td>
<td>10</td>
<td>(\mu)A</td>
</tr>
<tr>
<td>Analog block Current (^{(2)})</td>
<td>IADC</td>
<td>AVREF = (V_{DD} = 5\text{ V})</td>
<td>–</td>
<td>1</td>
<td>3</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AVREF = (V_{DD} = 3\text{ V})</td>
<td>0.5</td>
<td>1.5</td>
<td></td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AVREF = (V_{DD} = 5\text{ V})</td>
<td>100</td>
<td>500</td>
<td></td>
<td>nA</td>
</tr>
</tbody>
</table>

**NOTES:**

1. “Conversion time” is the time required from the moment a conversion operation starts until it ends.
2. \(I_{ADC}\) is an operating current during the A/D conversion.
3. \(f_{OSC}\) is the main oscillator clock.
Table 19-6. A/D Converter Electrical Characteristics (S3C8647)

(T<sub>A</sub> = –40°C to +85°C, V<sub>DD</sub> = 4.0 V to 5.5 V, V<sub>SS</sub> = 0 V)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>4</td>
<td>–</td>
<td>bit</td>
</tr>
<tr>
<td>Absolute accuracy (1)</td>
<td>–</td>
<td>4 bit conversion 24 x n/f&lt;sub&gt;OSC&lt;/sub&gt; (3), n = 1, 4, 8, 16</td>
<td>–</td>
<td>–</td>
<td>± 0.5</td>
<td>LSB</td>
</tr>
<tr>
<td>Conversion time (2)</td>
<td>t&lt;sub&gt;CON&lt;/sub&gt;</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>us</td>
</tr>
<tr>
<td>Analog input voltage</td>
<td>V&lt;sub&gt;IAN&lt;/sub&gt;</td>
<td>–</td>
<td>V&lt;sub&gt;SS&lt;/sub&gt;</td>
<td>–</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt;</td>
<td>V</td>
</tr>
<tr>
<td>Analog input impedance</td>
<td>R&lt;sub&gt;IAN&lt;/sub&gt;</td>
<td>–</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>MΩ</td>
</tr>
</tbody>
</table>

NOTES:
1. Excluding quantization error, absolute accuracy values are within ± 0.5 LSB.
2. “Conversion time” is the time required from the moment a conversion operation starts until it ends.
3. f<sub>OSC</sub> is the mean oscillator clock.
### Table 19-7. A.C. Electrical Characteristics

\( T_A = -40 \, ^\circ C \) to \( +85 \, ^\circ C \), \( V_{DD} = 3.0 \, V \) to \( 5.5 \, V \) (S3C863X), \( V_{DD} = 4.0 \, V \) to \( 5.5 \, V \) (S3C8647)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noise Filter</td>
<td>( t_{NF1H} )</td>
<td>INTO0–2 and TM0CAP (RC delay)</td>
<td>300</td>
<td>–</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>( t_{NF1L} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( t_{NF2} )</td>
<td>RESET only (RC delay)</td>
<td>1000</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
</tbody>
</table>

![Input Timing Measurement Points for P0.0–P0.2 and TM0CAP](image)

Figure 19-2. Input Timing Measurement Points for P0.0–P0.2 and TM0CAP
Table 19-8. Oscillation Characteristics

\( (T_A = -40 ^\circ C + 85 ^\circ C) \)

<table>
<thead>
<tr>
<th>Oscillator</th>
<th>Clock Circuit</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main crystal or ceramic</td>
<td></td>
<td>( V_{DD} = 3.0 ) V to 5.5 V</td>
<td>8</td>
<td></td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S3C863X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( V_{DD} = 4.0 ) V to 5.5 V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S3C8647)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External clock (main)</td>
<td></td>
<td>( V_{DD} = 3.0 ) V to 5.5 V</td>
<td>8</td>
<td></td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S3C863X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( V_{DD} = 4.0 ) V to 5.5 V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S3C8647)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: The maximum oscillator frequency is 12 MHz. If you use an oscillator frequency higher than 12 MHz, you cannot select a non-divided CPU clock using CLKCON settings. That is, you must select one of the divide-by values.

Table 19-9. Oscillation Stabilization Time

\( (T_A = -40 ^\circ C \text{ to } +85 ^\circ C, V_{DD} = 3.0 \) V to 5.5 V (S3C863X), \( V_{DD} = 4.0 \) V to 5.5 V (S3C8647))

<table>
<thead>
<tr>
<th>Oscillator</th>
<th>Test Condition</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crystal</td>
<td>( V_{DD} = 3.0 ) V (or 4.0 V) to 5.5 V</td>
<td>–</td>
<td>–</td>
<td>20</td>
<td>ms</td>
</tr>
<tr>
<td>Ceramic</td>
<td>( V_{DD} = 3.0 ) V (4.0 V) to 5.5 V</td>
<td>–</td>
<td>–</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>External clock</td>
<td>( X_{IN} ) input high and low level width</td>
<td>25</td>
<td>–</td>
<td>500</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>( (t_{XH}, t_{XL}) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: Oscillation stabilization time is the time required for the CPU clock to return to its normal oscillation frequency after a power-on occurs, or when Stop mode is released.

Figure 19-3. Clock Timing Measurement Points for \( X_{IN} \)
Figure 19-4. Schmitt Trigger Characteristics (Normal Port; except TTL Input)

Table 19-10. Power-on Reset Circuit Characteristics

\(T_A = -40 \, ^\circ C \text{ to } +85 \, ^\circ C, V_{DD} = 3.0 \text{ V to } 5.5 \text{ V (S3C863X), } V_{DD} = 4.0 \text{ V to } 5.5 \text{ V (S3C8647)}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power-on reset release voltage</td>
<td>(V_{DDLVD})</td>
<td></td>
<td>2.3</td>
<td>2.65</td>
<td>3.0</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2)</td>
<td>3.1</td>
<td>3.4</td>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td>Power-on reset detection voltage</td>
<td>(V_{LVD})</td>
<td></td>
<td>2.3</td>
<td>2.65</td>
<td>3.0</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2)</td>
<td>3.1</td>
<td>3.4</td>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td>Power supply voltage off time</td>
<td>(t_{off})</td>
<td></td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>ms</td>
</tr>
<tr>
<td>Power-on reset circuit consumption current</td>
<td>(I_{DDPR})</td>
<td>(V_{DD} = 5 \text{ V } \pm 10%)</td>
<td>100</td>
<td>150</td>
<td></td>
<td>(\mu A)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(V_{DD} = 3.3 \text{ V})</td>
<td>60</td>
<td>100</td>
<td></td>
<td>(\mu A)</td>
</tr>
</tbody>
</table>

NOTES:
1. Current contained when power-on reset circuit is provided internally.
2. Only S3C8647.
Figure 19-5. Power-on Reset Timing
OVERVIEW

The S3C8639/C863A/C8647 microcontroller is available in a 42-pin SDIP package (Samsung part number 42-SDIP-600) and a 44-QFP package (Samsung part number 44-QFP-1010B).

NOTE: Dimensions are in millimeters.

Figure 20-1. 42-Pin SDIP Package Dimensions (42-SDIP-600)
NOTE: Dimensions are in millimeters.

Figure 20-2. 44-Pin QFP Package Dimensions (44-QFP-1010B)
Figure 20-3. 32-Pin SDIP Package Dimensions (32-SDIP-400)
NOTE: Dimensions are in millimeters.

Figure 20-4. 32-Pin SOP Package Dimensions (32-SOP-450A)
S3P863A OTP

OVERVIEW

The S3P863A single-chip CMOS microcontroller is the OTP (One Time Programmable) version of the S3C8639/C863A microcontrollers. It has an on-chip EPROM instead of masked ROM. The EPROM is accessed by serial data format.

The S3P863A is fully compatible with the S3C8639/C863A, both in function and in pin configuration. Because of its simple programming requirements, the S3P863A is ideal for use as an evaluation chip for the S3C8639/C863A.

Figure 21-1. S3P863A Pin Assignments (42-SDIP Package)
NOTES: The bolds indicate an OTP pin name.

Figure 21-2. S3P863A Pin Assignments (44-QFP Package)
### Table 21-1. Descriptions of Pins Used to Read/Write the EPROM

<table>
<thead>
<tr>
<th>Main Chip</th>
<th>During Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin Name</td>
<td>Pin Name</td>
</tr>
<tr>
<td>P1.0</td>
<td>SDAT</td>
</tr>
<tr>
<td>P1.1</td>
<td>SCLK</td>
</tr>
<tr>
<td>TEST</td>
<td>V&lt;sub&gt;PP&lt;/sub&gt; (TEST)</td>
</tr>
<tr>
<td>RESET</td>
<td>RESET</td>
</tr>
<tr>
<td>V&lt;sub&gt;DD&lt;/sub&gt;/V&lt;sub&gt;SS&lt;/sub&gt;</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt;/V&lt;sub&gt;SS&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

**NOTE:** Parentheses indicate 44-QFP OTP pin number.

### Table 21-2. Comparison of S3P863A and S3C8639/C863A Features

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>S3P863A</th>
<th>S3C8639/C863A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Memory</td>
<td>48-Kbyte EPROM</td>
<td>32/48-Kbyte mask ROM</td>
</tr>
<tr>
<td>Operating Voltage (V&lt;sub&gt;DD&lt;/sub&gt;)</td>
<td>3.0 V to 5.5 V</td>
<td>3.0 V to 5.5V</td>
</tr>
<tr>
<td>OTP Programming Mode</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = 5 V, V&lt;sub&gt;PP&lt;/sub&gt; (TEST) = 12.5V</td>
<td></td>
</tr>
<tr>
<td>Pin Configuration</td>
<td>42 SDIP, 44 QFP</td>
<td>42 SDIP, 44 QFP</td>
</tr>
<tr>
<td>EPROM Programmability</td>
<td>User Program 1 time</td>
<td>Programmed at the factory</td>
</tr>
</tbody>
</table>

### OPERATING MODE CHARACTERISTICS

When 12.5 V is supplied to the V<sub>PP</sub> (TEST) pin of the S3P863A, the EPROM programming mode is entered. The operating mode (read, write, or read protection) is selected according to the input signals to the pins listed in Table 21-3 below.

### Table 21-3. Operating Mode Selection Criteria

<table>
<thead>
<tr>
<th>V&lt;sub&gt;DD&lt;/sub&gt;</th>
<th>V&lt;sub&gt;PP&lt;/sub&gt; (TEST)</th>
<th>REG/MEM</th>
<th>Address (A15–A0)</th>
<th>R/W</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 V</td>
<td>5 V</td>
<td>0</td>
<td>0000H</td>
<td>1</td>
<td>EPROM read</td>
</tr>
<tr>
<td>12.5 V</td>
<td>0</td>
<td>0</td>
<td>0000H</td>
<td>0</td>
<td>EPROM program</td>
</tr>
<tr>
<td>12.5 V</td>
<td>0</td>
<td>0</td>
<td>0000H</td>
<td>1</td>
<td>EPROM verify</td>
</tr>
<tr>
<td>12.5 V</td>
<td>1</td>
<td>0</td>
<td>0E3FH</td>
<td>0</td>
<td>EPROM read protection</td>
</tr>
</tbody>
</table>

**NOTE:** "0" means Low level; "1" means High level.
# D.C. ELECTRICAL CHARACTERISTICS

Table 21-4. D.C. Electrical Characteristics

\((T_A = -40 \, ^\circ C \, \text{to} \, +85 \, ^\circ C, \, V_{DD} = 3.0 \, V \, \text{to} \, 5.5 \, V)\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input High leakage current</td>
<td>(I_{LH1})</td>
<td>(V_{IN} = V_{DD}) (\text{All input pins except } X_{IN}, , X_{OUT})</td>
<td>–</td>
<td>–</td>
<td>3</td>
<td>(\mu A)</td>
</tr>
<tr>
<td></td>
<td>(I_{LH2})</td>
<td>(V_{IN} = V_{DD}; , X_{OUT}) only</td>
<td>–</td>
<td>–</td>
<td>20</td>
<td>(\mu A)</td>
</tr>
<tr>
<td></td>
<td>(I_{LH3})</td>
<td>(V_{IN} = V_{DD}; , X_{IN}) only</td>
<td>2.5</td>
<td>6</td>
<td>20</td>
<td>(\mu A)</td>
</tr>
<tr>
<td>Input Low leakage current</td>
<td>(I_{LIL1})</td>
<td>(V_{IN} = 0 , V; , \text{All input pins except } X_{IN}, , X_{OUT}, , \text{RESET }, , \text{Hsync-I and Vsync-I})</td>
<td>–</td>
<td>–</td>
<td>3</td>
<td>(\mu A)</td>
</tr>
<tr>
<td></td>
<td>(I_{LIL2})</td>
<td>(V_{IN} = 0 , V; , X_{OUT}) only</td>
<td>–</td>
<td>–</td>
<td>20</td>
<td>(\mu A)</td>
</tr>
<tr>
<td></td>
<td>(I_{LIL3})</td>
<td>(V_{IN} = 0 , V; , X_{IN}) only</td>
<td>– 2.5</td>
<td>– 6</td>
<td>– 20</td>
<td>(\mu A)</td>
</tr>
<tr>
<td>Output High leakage current</td>
<td>(I_{LOH1})</td>
<td>(V_{OUT} = V_{DD})</td>
<td>–</td>
<td>–</td>
<td>3</td>
<td>(\mu A)</td>
</tr>
<tr>
<td>Output Low leakage current</td>
<td>(I_{LOL1})</td>
<td>(V_{OUT} = 0 , V)</td>
<td>–</td>
<td>–</td>
<td>– 3</td>
<td>(\mu A)</td>
</tr>
<tr>
<td>Pull-up resistor</td>
<td>(R_{U1})</td>
<td>(V_{IN} = 0 , V; , V_{DD} = 5 , V \pm 10%) (\text{Port 3.7–3.4})</td>
<td>20</td>
<td>47</td>
<td>80</td>
<td>(k\Omega)</td>
</tr>
<tr>
<td></td>
<td>(R_{U2})</td>
<td>(V_{IN} = 0 , V; , V_{DD} = 5 , V \pm 10%) (\text{RESET only})</td>
<td>150</td>
<td>280</td>
<td>480</td>
<td>(k\Omega)</td>
</tr>
<tr>
<td>Pull-down resistor</td>
<td>(R_{D})</td>
<td>(V_{IN} = 0 , V; , V_{DD} = 5 , V \pm 10%) (\text{Hsync-I and Vsync-I})</td>
<td>150</td>
<td>300</td>
<td>500</td>
<td>(k\Omega)</td>
</tr>
<tr>
<td>Supply current (note)</td>
<td>(I_{DD1})</td>
<td>(V_{DD} = 5 , V \pm 10%) (\text{Operation mode; 12 MHz crystal})</td>
<td>–</td>
<td>10</td>
<td>20</td>
<td>(mA)</td>
</tr>
<tr>
<td></td>
<td>(I_{DD2})</td>
<td>(V_{DD} = 5 , V \pm 10%) (\text{Idle mode; 12 MHz crystal})</td>
<td>4</td>
<td>8</td>
<td></td>
<td>(mA)</td>
</tr>
<tr>
<td></td>
<td>(I_{DD3})</td>
<td>(V_{DD} = 5 , V \pm 10%) (\text{Stop mode})</td>
<td>100</td>
<td>150</td>
<td></td>
<td>(\mu A)</td>
</tr>
</tbody>
</table>

**NOTE:** Supply current does not include drawn internal pull-up/pull-down resistors and external loads of output.
OVERVIEW

The S3F8647 single-chip CMOS microcontroller is the FLASH version of the S3C8647 microcontrollers. It has an on-chip FLASH ROM instead of masked ROM. The FLASH ROM is accessed in serial data format.

The S3F8647 is fully compatible with the S3C8647, both in function and in pin configuration. Because of its simple programming requirements, the S3F8647 is ideal for use as an evaluation chip for the S3C8647.

Figure 22-1. S3F8647 Pin Assignments (32-SDIP Package)
Figure 22-2. S3F8647 Pin Assignments (32-SOP Package)
### Table 22-1. Descriptions of Pins Used to Read/Write the FLASH ROM

<table>
<thead>
<tr>
<th>Main Chip</th>
<th>Pin Name</th>
<th>Pin Number</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.6</td>
<td>SDAT</td>
<td>30</td>
<td>I/O</td>
<td>Serial data pin. Output port when reading and input port when writing. Can be assigned as an input/push-pull output port.</td>
</tr>
<tr>
<td>P3.7</td>
<td>SCLK</td>
<td>31</td>
<td>I</td>
<td>Serial clock pin. Input only pin.</td>
</tr>
<tr>
<td>TEST</td>
<td>V_{PP} (TEST)</td>
<td>4</td>
<td>I</td>
<td>Power supply pin for EPROM cell writing (indicates that OTP enters into the writing mode). When 12.5 V is applied, OTP is in writing mode and when 5 V is applied, OTP is in reading mode. (Option)</td>
</tr>
<tr>
<td>RESET</td>
<td>RESET</td>
<td>7</td>
<td>I</td>
<td>Chip Initialization</td>
</tr>
<tr>
<td>V_{DD}/V_{SS}</td>
<td>V_{DD}/V_{SS}</td>
<td>32/1</td>
<td>I</td>
<td>Logic power supply pin. V_{DD} should be tied to +5 V during programming.</td>
</tr>
</tbody>
</table>

### Table 22-2. Comparison of S3F8647 and S3C8647 Features

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>S3F8647</th>
<th>S3C8647</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Memory</td>
<td>24-Kbyte flash ROM</td>
<td>24-Kbyte mask ROM</td>
</tr>
<tr>
<td>Operating Voltage (V_{DD})</td>
<td>4.0 V to 5.5 V</td>
<td>4.0 V to 5.5V</td>
</tr>
<tr>
<td>OTP Programming Mode</td>
<td>V_{DD} = 5 V, V_{PP} (TEST) = 12.5V</td>
<td></td>
</tr>
<tr>
<td>Pin Configuration</td>
<td>32 SDIP</td>
<td>32 SDIP</td>
</tr>
<tr>
<td>EPROM Programmability</td>
<td>User Program 1 time</td>
<td>Programmed at the factory</td>
</tr>
</tbody>
</table>
## D.C. ELECTRICAL CHARACTERISTICS

### Table 22-3. D.C. Electrical Characteristics

(V\(_{DD}\) = 4.0 V to 5.5 V)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input High leakage current</td>
<td>I(_{LIH1})</td>
<td>(V_{IN} = V_{DD})</td>
<td>All input pins except X(<em>{IN}), X(</em>{OUT})</td>
<td>–</td>
<td>–</td>
<td>3</td>
<td>µA</td>
</tr>
<tr>
<td></td>
<td>I(_{LIH2})</td>
<td>(V_{IN} = V_{DD}); X(_{OUT}) only</td>
<td></td>
<td>–</td>
<td>–</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(_{LIH3})</td>
<td>(V_{IN} = V_{DD}); X(_{IN}) only</td>
<td></td>
<td>2.5</td>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Input Low leakage current</td>
<td>I(_{LIL1})</td>
<td>(V_{IN} = 0) V; All input pins except X(<em>{IN}), X(</em>{OUT}), RESET, Hsync-I and Vsync-I</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(_{LIL2})</td>
<td>(V_{IN} = 0) V; X(_{OUT}) only</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(_{LIL3})</td>
<td>(V_{IN} = 0) V; X(_{IN}) only</td>
<td></td>
<td>–2.5</td>
<td>–6</td>
<td>–20</td>
<td></td>
</tr>
<tr>
<td>Output High leakage current</td>
<td>I(_{LOH1})</td>
<td>(V_{OUT} = V_{DD})</td>
<td></td>
<td>–</td>
<td>–</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Output Low leakage current</td>
<td>I(_{LOL1})</td>
<td>(V_{OUT} = 0) V</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–3</td>
<td></td>
</tr>
<tr>
<td>Pull-up resistor</td>
<td>R(_{U1})</td>
<td>(V_{IN} = 0) V; (V_{DD} = 5) V ± 10%</td>
<td>Port 3.7–3.4</td>
<td>20</td>
<td>47</td>
<td>80</td>
<td>kΩ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R(_{U2})</td>
<td>(V_{IN} = 0) V; (V_{DD} = 5) V ± 10%</td>
<td>RESET only</td>
<td>150</td>
<td>280</td>
<td>480</td>
<td></td>
</tr>
<tr>
<td>Pull-down resistor</td>
<td>R(_{D})</td>
<td>(V_{IN} = 0) V; (V_{DD} = 5) V ± 10%</td>
<td>Hsync-I and Vsync-I</td>
<td>150</td>
<td>300</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>Supply current (note)</td>
<td>I(_{DD1})</td>
<td>(V_{DD} = 5) V ± 10%</td>
<td>Operation mode; 12 MHz crystal C1 = C2 = 22pF</td>
<td>–</td>
<td>10</td>
<td>20</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td>I(_{DD2})</td>
<td>(V_{DD} = 5) V ± 10%</td>
<td>Idle mode; 12 MHz crystal C1 = C2 = 22pF</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(_{DD3})</td>
<td>(V_{DD} = 5) V ± 10%</td>
<td>Stop mode</td>
<td>100</td>
<td>150</td>
<td></td>
<td>µA</td>
</tr>
</tbody>
</table>

**NOTE:** Supply current does not include drawn internal pull-up/pull-down resistors and external loads of output.
NOTES
OVERVIEW

Samsung provides a powerful and easy-to-use development support system in turnkey form. The development support system is configured with a host system, debugging tools, and support software. For the host system, any standard computer that operates with MS-DOS as its operating system can be used. One type of debugging tool including hardware and software is provided: the sophisticated and powerful in-circuit emulator, SMDS2+, for S3C7, S3C9, S3C8 families of microcontrollers. The SMDS2+ is a new and improved version of SMDS2. Samsung also offers support software that includes debugger, assembler, and a program for setting options.

SHINE

Samsung Host Interface for In-Circuit Emulator, SHINE, is a multi-window based debugger for SMDS2+. SHINE provides pull-down and pop-up menus, mouse support, function/hot keys, and context-sensitive hyper-linked help. It has an advanced, multiple-windowed user interface that emphasizes ease of use. Each window can be sized, moved, scrolled, highlighted, added, or removed completely.

SAMA ASSEMBLER

The Samsung Arrangeable Microcontroller (SAM) Assembler, SAMA, is a universal assembler, and generates object code in standard hexadecimal format. Assembled program code includes the object code that is used for ROM data and required SMDS program control data. To assemble programs, SAMA requires a source file and an auxiliary definition (DEF) file with device specific information.

SASM88

The SASM88 is a relocatable assembler for Samsung's S3C8-series microcontrollers. The SASM88 takes a source file containing assembly language statements and translates into a corresponding source code, object code and comments. The SASM88 supports macros and conditional assembly. It runs on the MS-DOS operating system. It produces the relocatable object code only, so the user should link object file. Object files can be linked with other object files and loaded into memory.

HEX2ROM

HEX2ROM file generates ROM code from HEX file which has been produced by assembler. ROM code must be needed to fabricate a microcontroller which has a mask ROM. When generating the ROM code (.OBJ file) by HEX2ROM, the value "FF" is filled into the unused ROM area up to the maximum ROM size of the target device automatically.

TARGET BOARDS

Target boards are available for all S3C8-series microcontrollers. All required target system cables and adapters are included with the device-specific target board.
OTPs

One time programmable microcontroller (OTP) for the S3C8639/C863A microcontroller and OTP programmer (Gang) are now available.

Figure 23-1. SMDS Product Configuration (SMDS2+)
TB886332B/6348B (TB8639/863A) TARGET BOARD

The TB886332B/6348B (TB8639/863A) target board is used for the S3C8639/C863A microcontroller. It is supported by the SMDS2+ development system.

Figure 23-2. TB886332B/6348B (TB8639/863A) Target Board Configuration
TB886424A (TB8647) TARGET BOARD

The TB886424A (TB8647) target board is used for the S3C8647 microcontroller. It is supported by the SMDS2+ development system.

Figure 23-3. TB886424A (TB8647) Target Board Configuration
### Table 23-1. Power Selection Settings for TB886332B/TB886348B (TB8639/863A)

<table>
<thead>
<tr>
<th>&quot;To User_Vcc&quot; Settings</th>
<th>Operating Mode</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF ON</td>
<td>VCC VSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TB886332B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TB886348B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(TB8639/863A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VCC VSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMDS2/SMDS2+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Target System</td>
<td></td>
</tr>
</tbody>
</table>

The SMDS2/SMDS2+ supplies \( V_{CC} \) to the target board (evaluation chip) and the target system.

<table>
<thead>
<tr>
<th>&quot;To User_Vcc&quot; Settings</th>
<th>Operating Mode</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF ON</td>
<td>VCC VSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TB886332B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TB886348B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(TB8639/863A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VCC VSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMDS2/SMDS2+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Target System</td>
<td></td>
</tr>
</tbody>
</table>

The SMDS2/SMDS2+ supplies \( V_{CC} \) only to the target board (evaluation chip). The target system must have its own power supply.

### Table 23-2. Power Selection Settings for TB886424A (TB8647)

<table>
<thead>
<tr>
<th>&quot;To User_Vcc&quot; Settings</th>
<th>Operating Mode</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF ON</td>
<td>VCC VSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TB886424A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(TB8647)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VCC VSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMDS2/SMDS2+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Target System</td>
<td></td>
</tr>
</tbody>
</table>

The SMDS2/SMDS2+ supplies \( V_{CC} \) to the target board (evaluation chip) and the target system.

<table>
<thead>
<tr>
<th>&quot;To User_Vcc&quot; Settings</th>
<th>Operating Mode</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF ON</td>
<td>VCC VSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TB886424A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(TB8647)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VCC VSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMDS2/SMDS2+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Target System</td>
<td></td>
</tr>
</tbody>
</table>

The SMDS2/SMDS2+ supplies \( V_{CC} \) only to the target board (evaluation chip). The target system must have its own power supply.
SMDS2+ SELECTION (SAM8)

In order to write data into program memory that is available in SMDS2+, the target board should be selected to be for SMDS2+ through a switch as follows. Otherwise, the program memory writing function is not available.

**Table 23-3. The SMDS2+ Tool Selection Setting**

<table>
<thead>
<tr>
<th>&quot;SW1&quot; Setting</th>
<th>Operating Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMDS2SMDS2+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R/W*</td>
</tr>
<tr>
<td></td>
<td>R/W*</td>
</tr>
<tr>
<td></td>
<td>TARGET BOARD</td>
</tr>
</tbody>
</table>

**Table 23-4. Using Single Header Pins as the Input Path for External Trigger Sources**

<table>
<thead>
<tr>
<th>Target Board Part</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTERNAL TRIGGERS</td>
<td>Connector from external trigger sources of the application system</td>
</tr>
<tr>
<td>CH1</td>
<td></td>
</tr>
<tr>
<td>CH2</td>
<td></td>
</tr>
</tbody>
</table>

You can connect an external trigger source to one of the two external trigger channels (CH1 or CH2) for the SMDS2+ breakpoint and trace functions.

**IDLE LED**

This LED is ON when the evaluation chip (S3E8630) is in idle mode.

**STOP LED**

This LED is ON when the evaluation chip (S3E8630) is in stop mode.
Figure 23-4. 40-Pin Connector for TB886332B/6348B (TB8639/A)

Figure 23-5. 30-Pin Connector for TB886424A (TB8647)
Figure 23-6. TB886332B/6348B (TB8639/A) Adapter Cable for 42-SDIP Package

Figure 23-7. TB886424A (TB8647) Adapter Cable for 32-SDIP Package