1 PRODUCT OVERVIEW

S3C8-SERIES MICROCONTROLLERS

Samsung's SAM8RC family of 8-bit single-chip CMOS microcontrollers offer a fast and efficient CPU, a wide range of integrated peripherals, and various mask-programmable ROM sizes.
An address/data bus architecture and a large number of bit-configurable I/O ports provide a flexible programming environment for applications with varied memory and I/O requirements. Timer/counters with selectable operating modes are included to support real-time operations.

S3F84B8 MICROCONTROLLER

The S3F84B8 single-chip CMOS microcontrollers are fabricated using the highly advanced CMOS process technology based on Samsung's latest CPU architecture.

The S3F84B8 is a microcontroller with a 8K-byte **full-flash** ROM embedded.

Using a proven modular design approach, Samsung engineers have successfully developed the S3F84B8 by integrating the following peripheral modules with the powerful SAM8 RC core:

— Three configurable I/O ports (18pin)

— Seventeen interrupt sources with Seventeen vectors and Six interrupt level

— One watchdog timer function (Basic Timer)

— One 8-bit basic timer for oscillation stabilization

— Three 8-bit timer/counters with time interval, PWM, and Capture mode

(Timer C and Timer D can be used for 16-bit Timer 0)

— One 16-bit timer/counter with 2 operating modes; Interval timer and PWM mode

(If Timer C and Timer D are used for Timer 0, S3F84B8 has one 16-bit Timer0)

— Analog to digital converter with 8 input channels and 10-bit resolution

— One BUZ for programmable frequency output

— High current LED drive I/O ports (High current output: typical 12 mA)

The S3F84B8 microcontroller is ideal for use in a wide range of home applications requiring simple timer/counter, ADC, etc. They are currently available in 20-pin SOP/DIP package.

FEATURES

CPU

- SAM8RC CPU core

Memory

- 8K-byte internal multi time program memory Full-Flash
  - √ Sector size: 128 Bytes
  - √ 10 Years data retention
  - √ Fast programming time:
    - + Chip erase: 32ms
    - + Sector erase: 12ms
    - + Byte program: 20us
  - √ User programmable by 'LDC' instruction
  - √ Endurance: 10,000 erase/program cycles
  - √ Sector(128-bytes) erase available
  - √ Byte programmable
- 272-byte general-purpose register area

Instruction Set

- 78 instructions
- Idle and Stop instructions added for power-down modes

Instruction Execution Time

- 400 ns at 10 MHz f_{OSC} (minimum)

Interrupts

- 17 Interrupt sources with 17 vectors
- Fast interrupt processing feature

General I/O

- Three I/O ports
- Bit programmable ports

10-bit IH PWM

- 10bit IH specific PWM 1-ch
- Cooperate with CMPs
- Anti-mis-trigger function
- Delay trigger function

Comparators

- 4 integrated comparators

A/D Converter

- Eight analog input pins (MAX)
- 10-bit conversion resolution

OP amplifier

- One integrated op amplifier

Timer/Counters

- One 8-bit basic timer for watchdog function
- One 8-bit timer A
  - Interval mode
  - Capture mode
  - 8bit PWM mode
- One 16-bit timer/counter Timer0
  - Configurable to be two 8-bit timer/counters
  - Interval mode
  - CMP0 event counter mode
  - 6/7/8 bit PWM mode

BUZ

- One programmable Buzzer

Oscillation Frequency

- 1 MHz to 10 MHz external crystal oscillator
- Typical 8MHz external RC oscillator
- Internal RC: 8 MHz (Typ), 0.5 MHz (Typ)
- Maximum 10 MHz CPU clock

Built-in RESET Circuit (LVR)

- Low-Voltage check to reset system
- V_{LVR} = 1.9/2.3 /3.0/3.6/3.9 V (by smart option)

Operating Temperature Range

- – 40°C to + 85°C

Operating Voltage Range

- 1.8 V to 5.5 V @ 0.4-2MHz
- 2.0 V to 5.5 V @ 0.4-4MHz
- 2.7 V to 5.5V @ 0.4-10M Hz

Package Types

- S3F84B8:
  - 20-SOP/20-DIP

BLOCK DIAGRAM

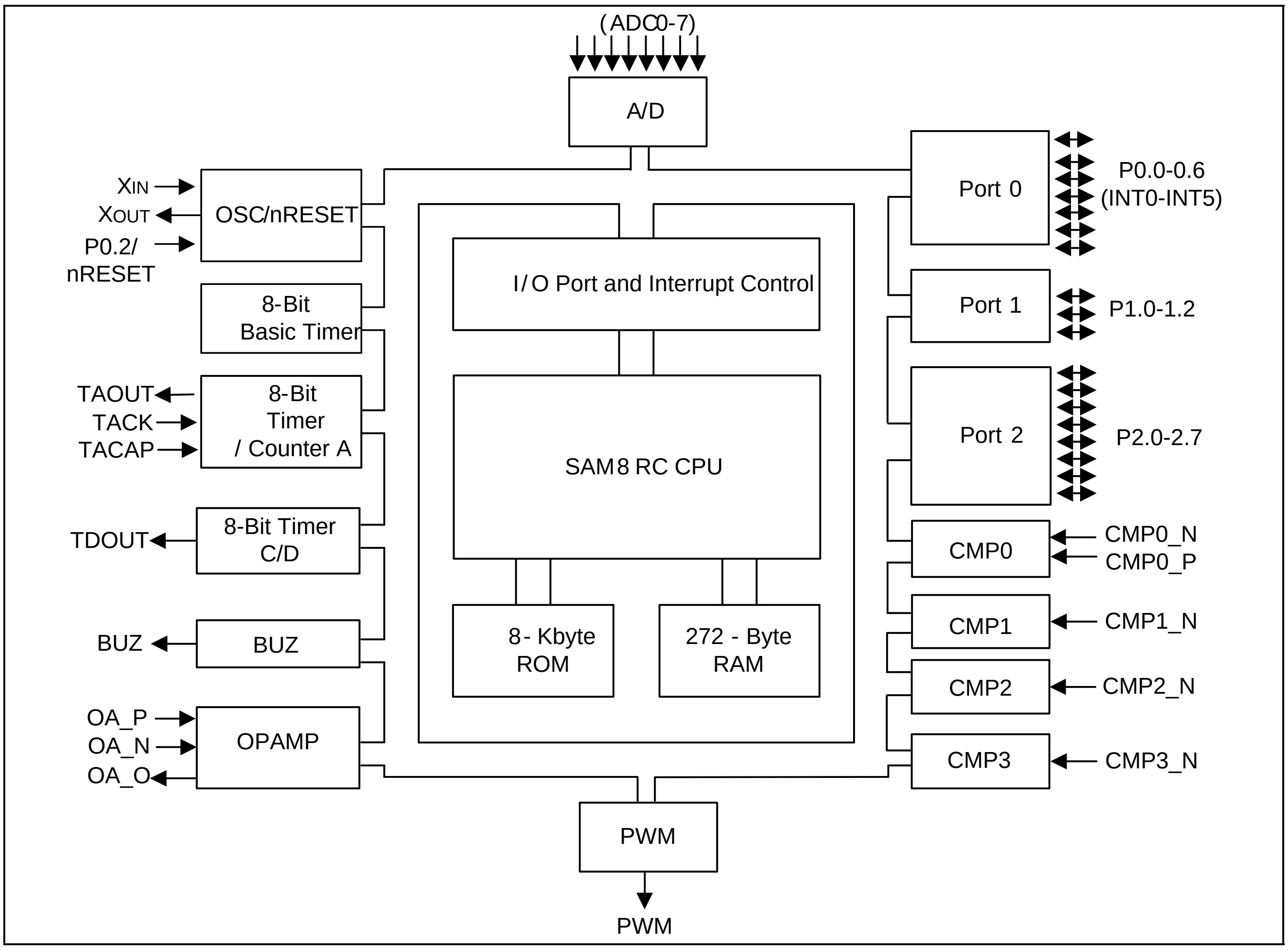


Figure 1-1. S3F84B8 Block Diagram

PIN ASSIGNMENTS

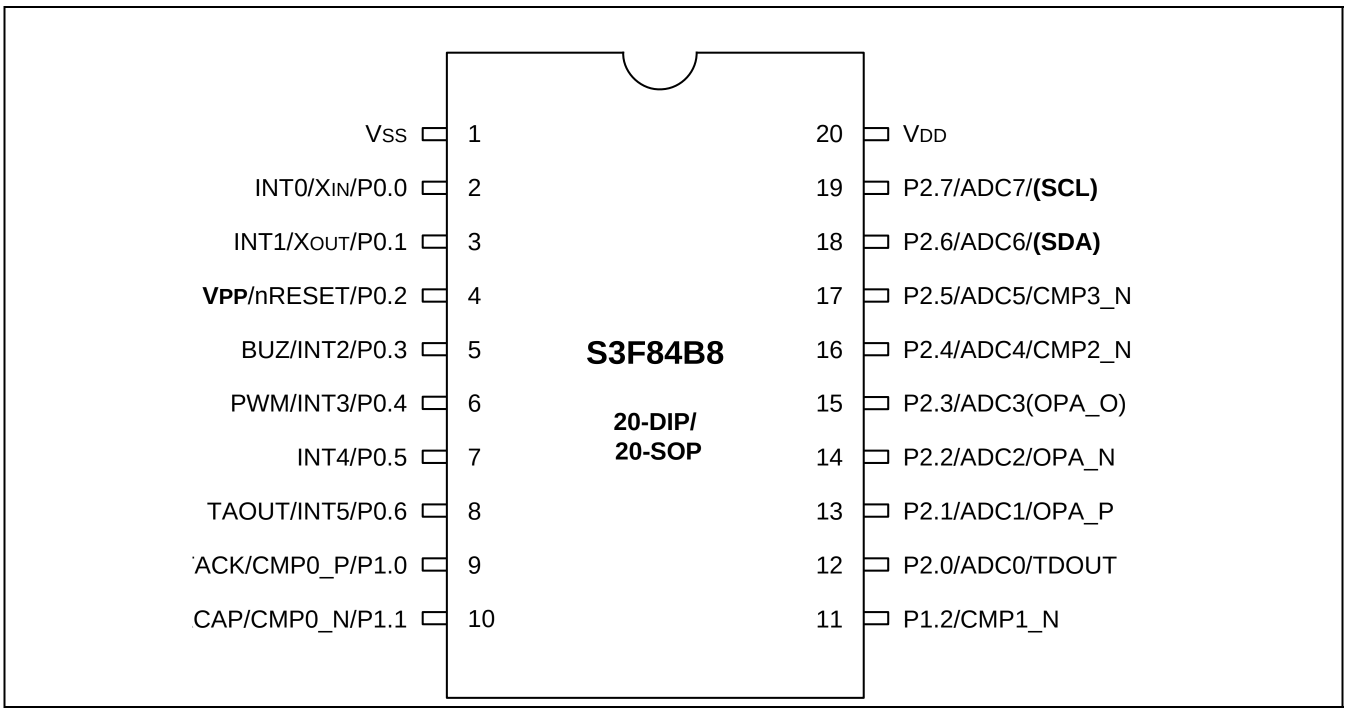


Figure 1-2. S3F84B8 Pin Assignment (20-DIP, 20-SOP)

PIN DESCRIPTIONS

Table 1-1. Pin Descriptions of 20-DIP (20-SOP)

Pin Names	Pin Type	Pin Description	Circuit Type	Pin No.	Shared Pins
INT0-INT5	I	External interrupts	1-3 2-1	2-8	P0.0-P0.6
ADC0-ADC7	I	A/D converter analog input channels	1-1 1-2	12-19	P2.0-2.7
BUZ	O	Frequency output from buzzer	1-3	5	P0.3
PWM	O	PWM output	1-3	6	P0.4
TAOUT	O	Timer/counter(A) match output, or Timer/counter(A) PWM output	1-3	8	P0.6
TACK	I	Timer/counter(A) external clock input	1-1	9	P1.0
TACAP	I	Timer/counter(A) external capture input	1-1	10	P1.1
TDOUT	O	Timer/counter(D) match output, or Timer/counter(D) PWM output	1-1	12	P2.0
CMP0_P	I	Comparater0 positive input pin	1-1	9	P1.0
CMP0_N	I	Comparater0 negative input pin	1-1	10	P1.1
CMP1_N	I	Comparater1 negative input pin	1-1	11	P1.2
CMP2_N	I	Comparater2 negative input pin	1-1	16	P2.4
CMP3_N	I	Comparater3 negative input pin	1-1	17	P2.5
OPA_O	O	Operational amplifier output pin	1-2	15	P2.3
OPA_N	I	Operational amplifier negative input pin	1-1	14	P2.2
OPA_P	I	Operational amplifier positive input pin	1-1	13	P2.1
nRESET	I	Reset Pin	3	4	P0.2

Table 1-2. Descriptions of Pins Used to Read/Write the Flash ROM

Main Chip	During Programming			
Pin	Pin Name	Pin No.	I/O	Function
P2.6	SDA	18	I/O	Serial data pin (output when reading, Input when writing) Input and push-pull output port can be assigned
P2.7	SCL	19	I	Serial clock pin (input only pin)
RESET/P0.2	V_{PP}	4	I	Power supply pin for flash ROM cell writing (indicates that MTP enters into the writing mode). When 11 V is applied, MTP is in writing mode and when 5 V is applied, MTP is in reading mode. (Option)
V_{DD}/V_{SS}	V_{DD}/V_{SS}	20/1	I	Logic power supply pin.

PIN CIRCUITS

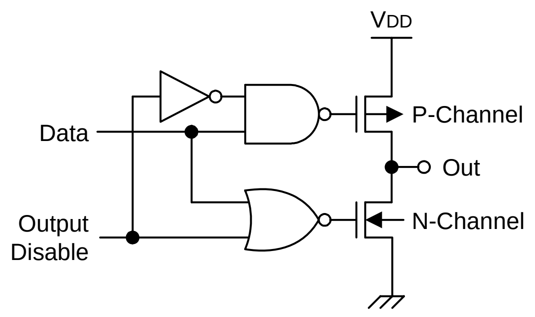


Figure 1-3. Pin Circuit Type 1

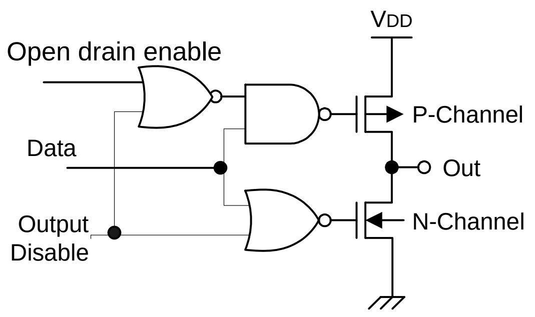


Figure 1-4. Pin Circuit Type 2

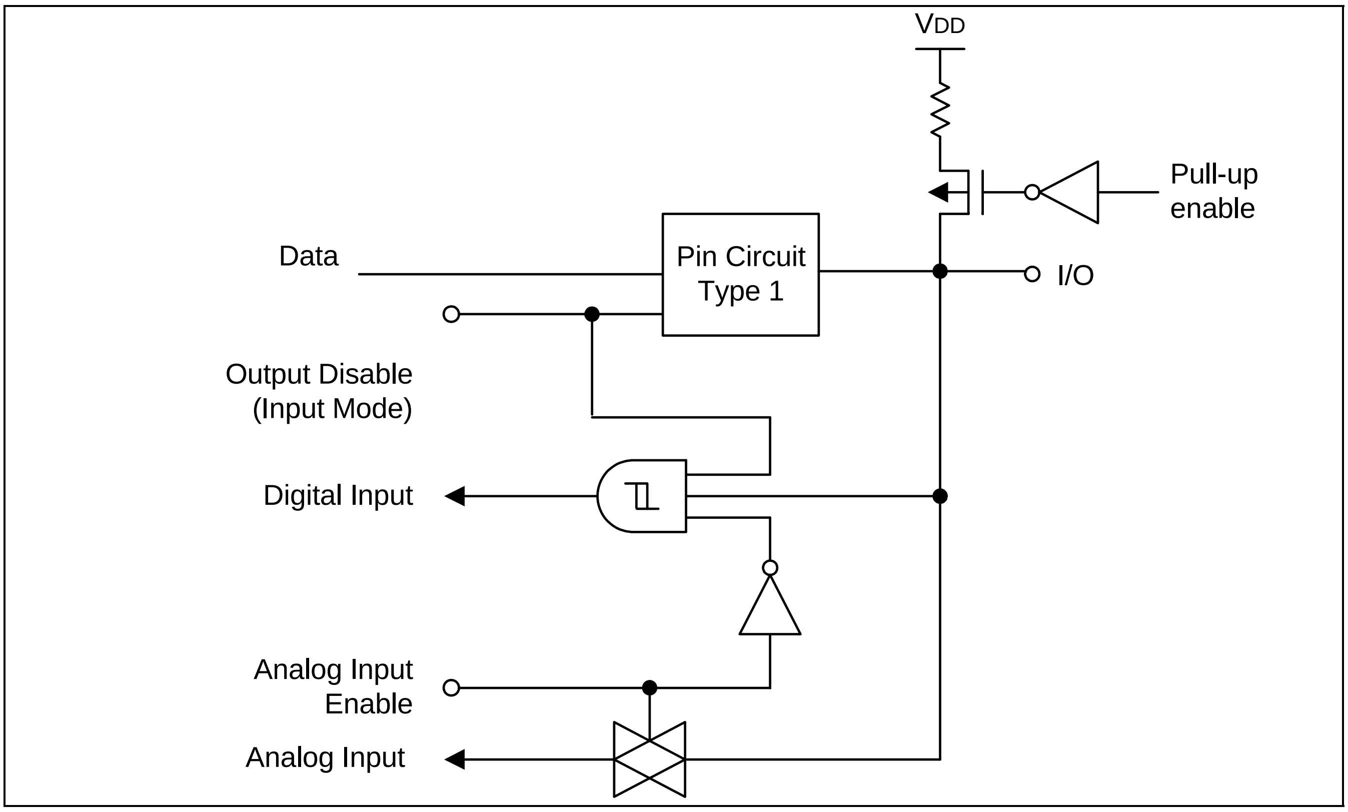


Figure 1-5. Pin Circuit Type 1-1 (P1.0-1.2, P2.0-2.2, P2.4-2.7)

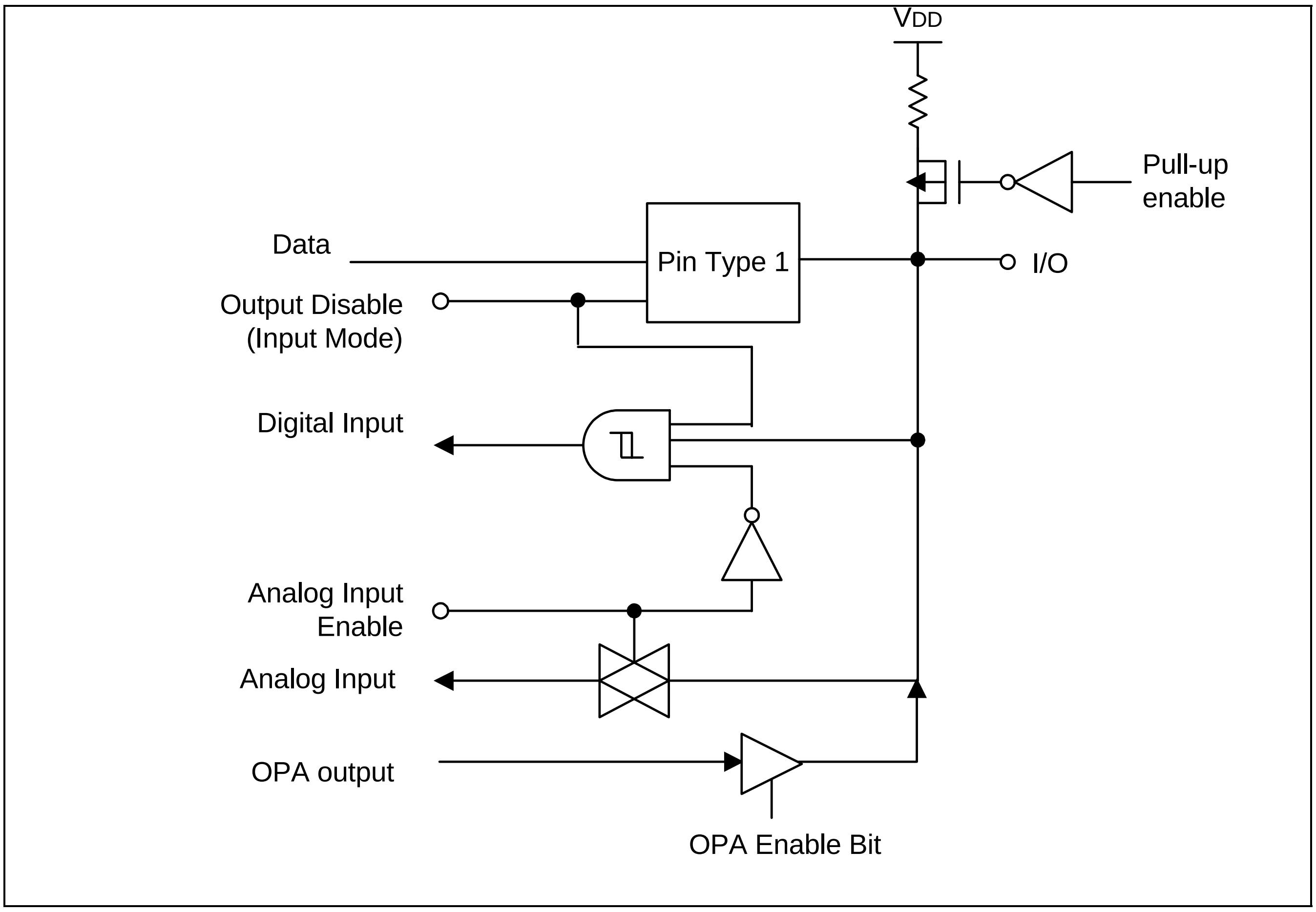


Figure 1-6. Pin Circuit Type 1-2 (P2.3)

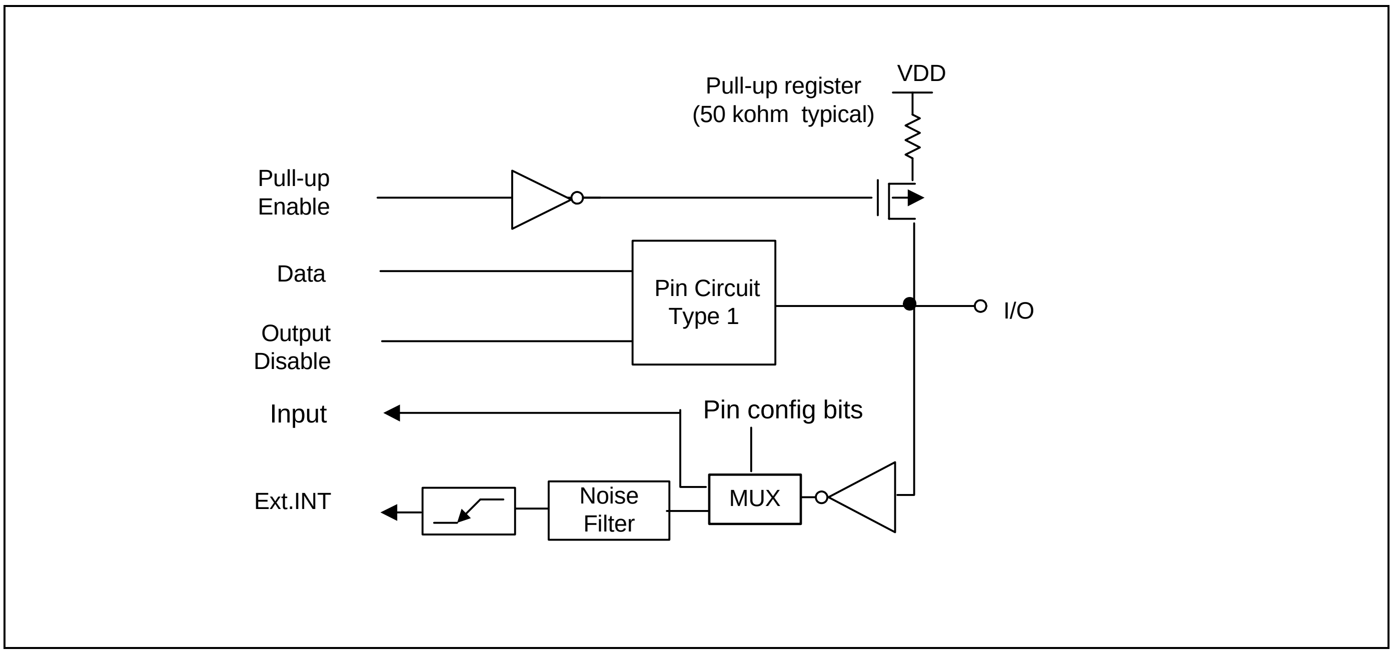


Figure 1-7. Pin Circuit Type 1-3 (P0.3, P0.4, P0.6)

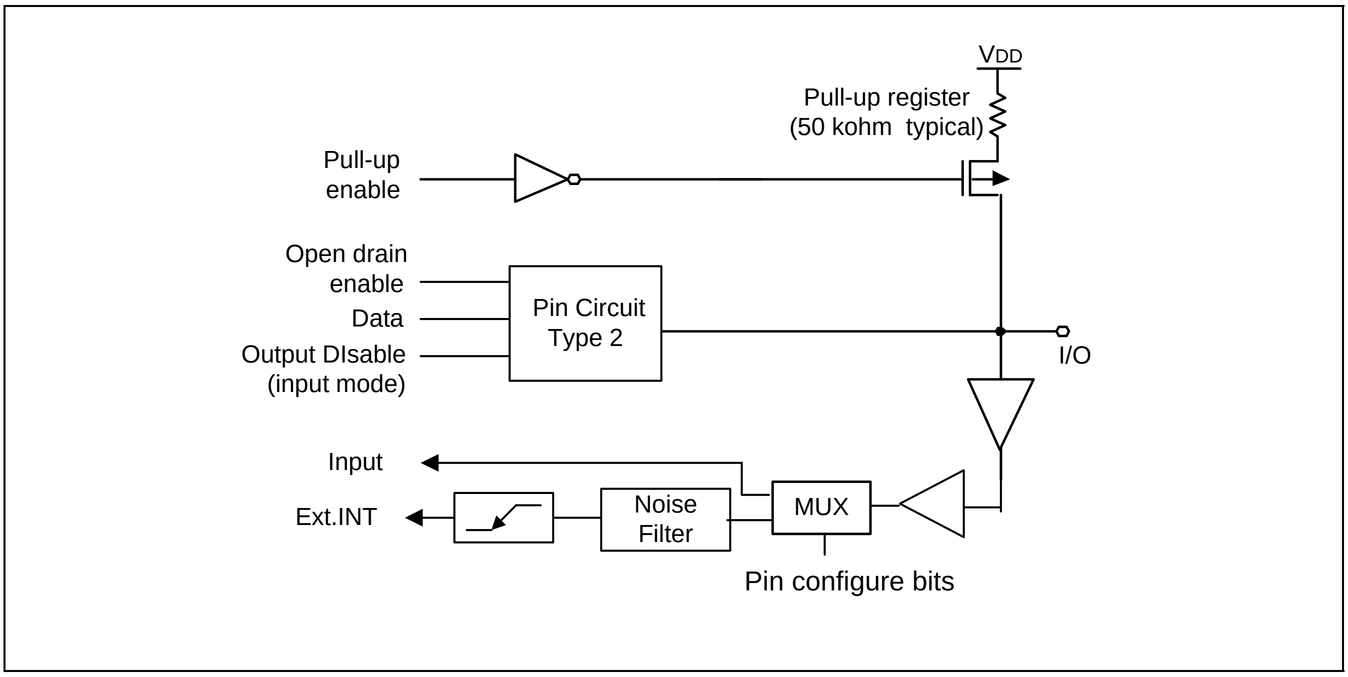


Figure 1-8. Pin Circuit Type 2-1 (P0.5)

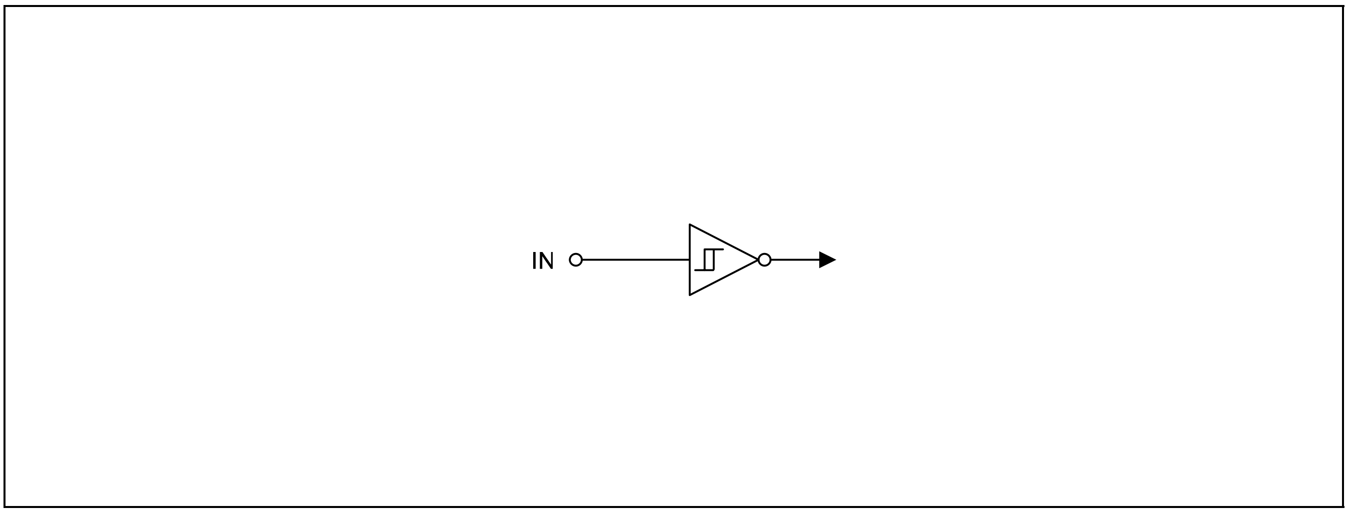


Figure 1-9. Pin Circuit Type 3 (P0.2)

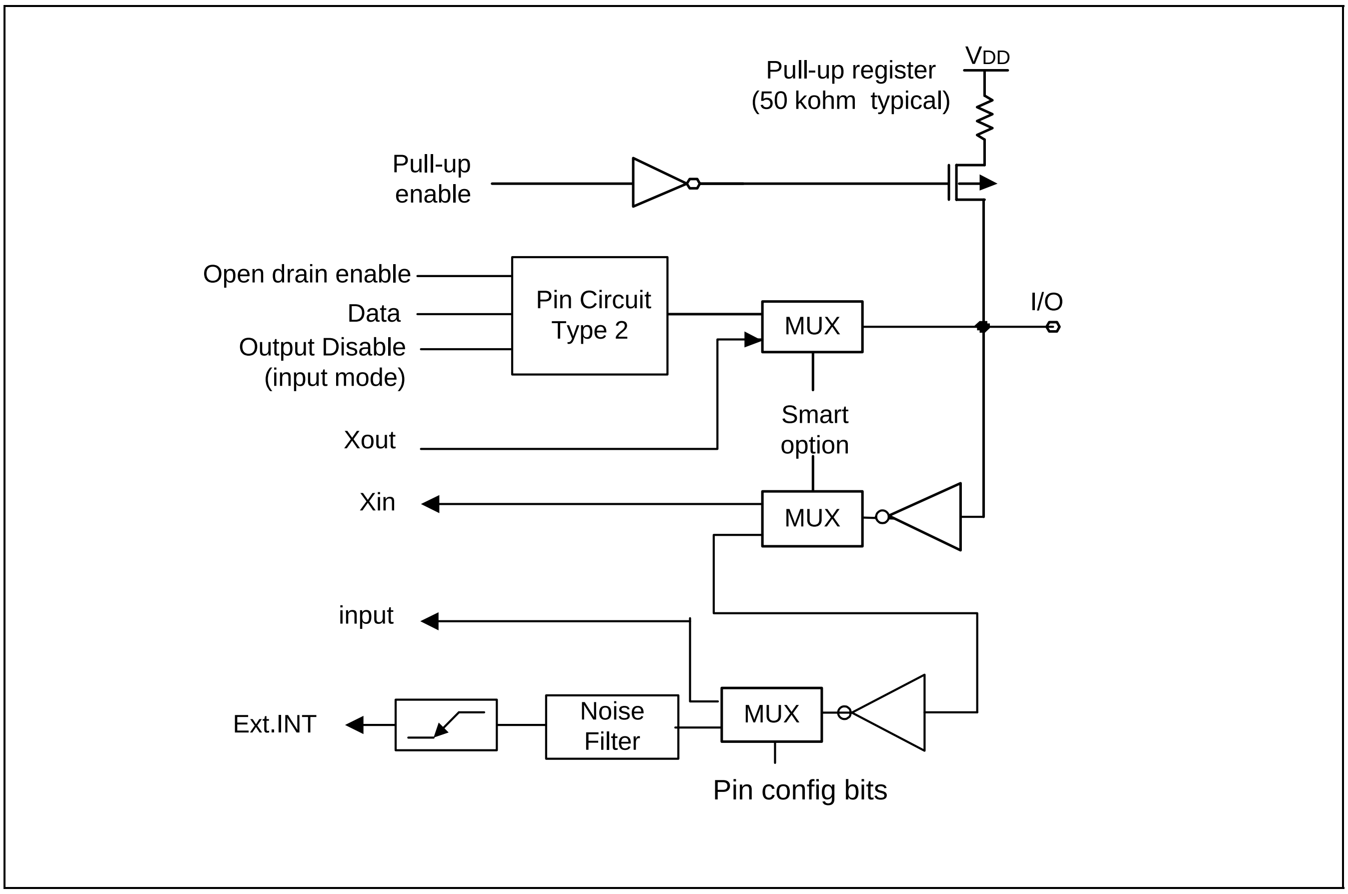


Figure 1-10. Pin Circuit Type 2-2 (P0.0, P0.1)

2 ADDRESS SPACES

OVERVIEW

The S3F84B8 microcontroller has two kinds of address space:

— Internal program memory (ROM)

— Internal register file

A 16-bit address bus supports program memory operations. A separate 8-bit register bus carries addresses and data between the CPU and the internal register file.

The S3F84B8 have 8-Kbytes of on-chip program memory, which is configured as the Internal ROM mode.

The S3F84B8 microcontroller has 272 general-purpose registers in its internal register file. 59 bytes in the register file are mapped for system and peripheral control functions.

PROGRAM MEMORY (ROM)

Program memory (ROM) stores program codes or table data. The S3F84B8 have 8Kbytes of internal multi time programmable (MTP) program memory (see Figure 2-1).

The first 256 bytes of the ROM (0H–0FFH) are reserved for interrupt vector addresses. Unused locations (except 3CH, 3DH, 3EH, 3FH) in this address range can be used as normal program memory. If you use the vector address area to store a program code, be careful not to overwrite the vector addresses stored in these locations.

3CH, 3DH, 3EH, 3FH is used as smart option ROM cell.

The default program Reset address in the ROM is 0100H.

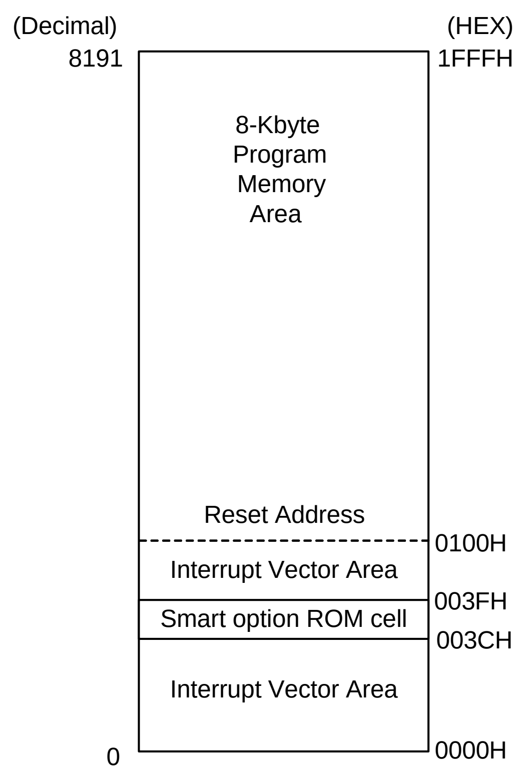
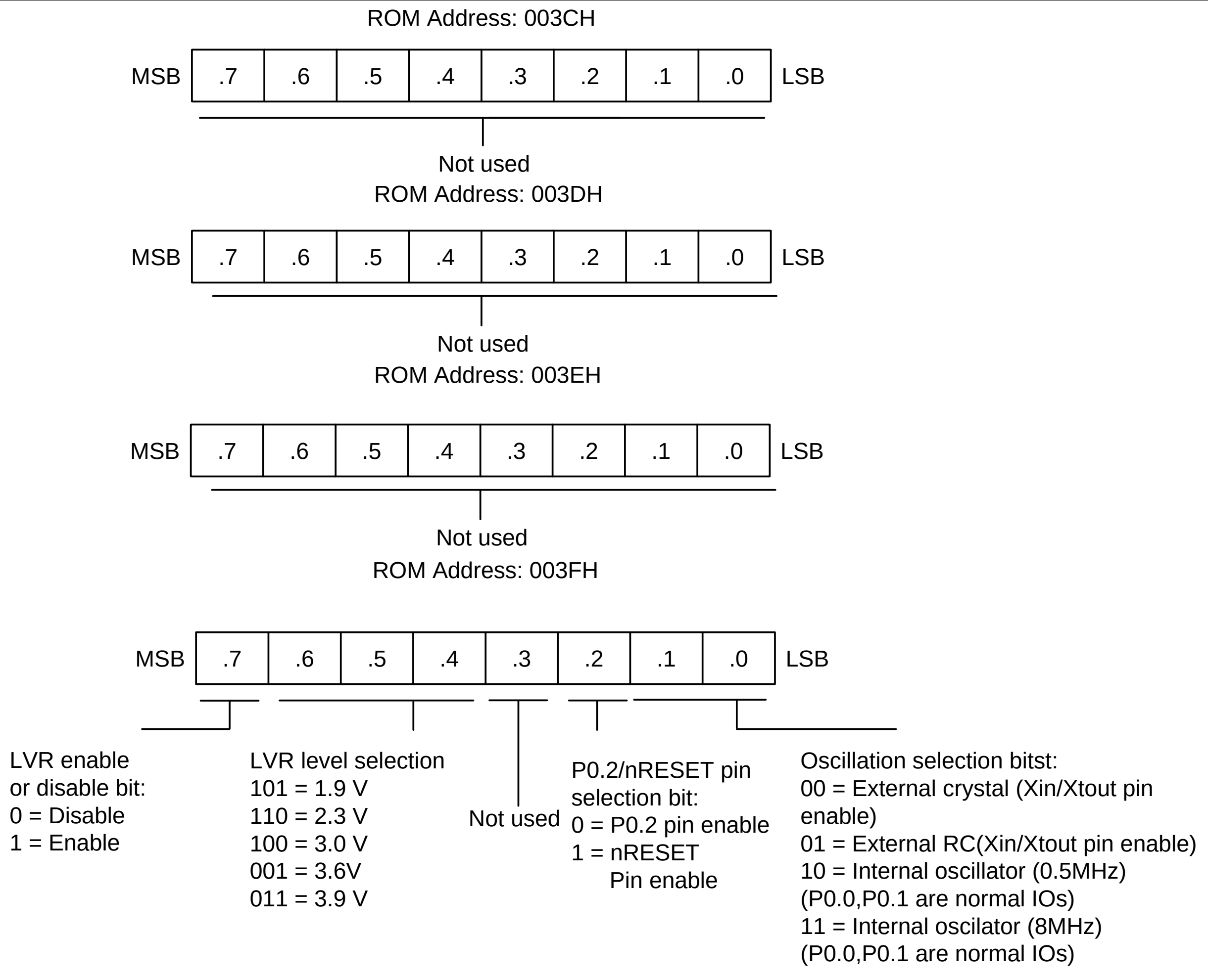


Figure 2-1. Program Memory Address Space

Smart Option


ROM Address: 003CH
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
Not used
ROM Address: 003DH
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
Not used
ROM Address: 003EH
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
Not used
ROM Address: 003FH
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
LVR enable
or disable bit:
0 = Disable
1 = Enable
LVR level selection
101 = 1.9 V
110 = 2.3 V
100 = 3.0 V
001 = 3.6V
011 = 3.9 V
Not used
P0.2/nRESET pin
selection bit:
0 = P0.2 pin enable
1 = nRESET
Pin enable
Oscillation selection bitst:
00 = External crystal (Xin/Xtout pin
enable)
01 = External RC(Xin/Xtout pin enable)
10 = Internal oscillator (0.5MHz)
(P0.0,P0.1 are normal IOs)
11 = Internal oscilator (8MHz)
(P0.0,P0.1 are normal IOs)

NOTES:
1. The unused bits of 3CH, 3DH, 3EH, 3FH must be logic "1".
2. When LVR is enabled, LVR level must be set to appropriate value.
3. P0.2 has only input (without pull-up) function when sets 003F.2 as 0.
4. You must set P0.0,P0.1,P0.2 function on smart option. For example, if you select XIN (P0.0)/XOUT (P0.1)/nRESET(P0.2) function by smart option, you can't change them to Normal I/O after reset operation.

Figure 2-2. Smart Option

Smart option is the ROM option for start condition of the chip. The ROM address used by smart option is from 003EH to 003FH. 003CH and 003DH are not used in S3F84B8.

REGISTER ARCHITECTURE

In the S3F84B8 implementation, the upper 64-byte area of register files is expanded two 64-byte areas, called *set 1* and *set 2*. The upper 32-byte area of set 1 is further expanded two 32-byte register banks (bank 0 and bank 1), and the lower 32-byte area is a single 32-byte common area.

In case of S3F84B8 the total number of addressable 8-bit registers is 336. Of these 336 registers, 15 bytes are for CPU and system control registers, 49 bytes are for peripheral control and data registers, 16 bytes are used as a shared working registers, and 272 registers are for general-purpose use, page 0.

You can always address set 1 register locations, regardless of which of the two register pages is currently selected. Set 1 location, however, can only be addressed using register addressing modes.

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

Specific register types and the area (in bytes) that they occupy in the register file are summarized in Table 2–1.

Table 2-1. S3F84B8 Register Type Summary

Register Type	Number of Bytes
System and peripheral registers	64
General-purpose registers (including the 16-bit common working register area)	272
Total Addressable Bytes	**336**

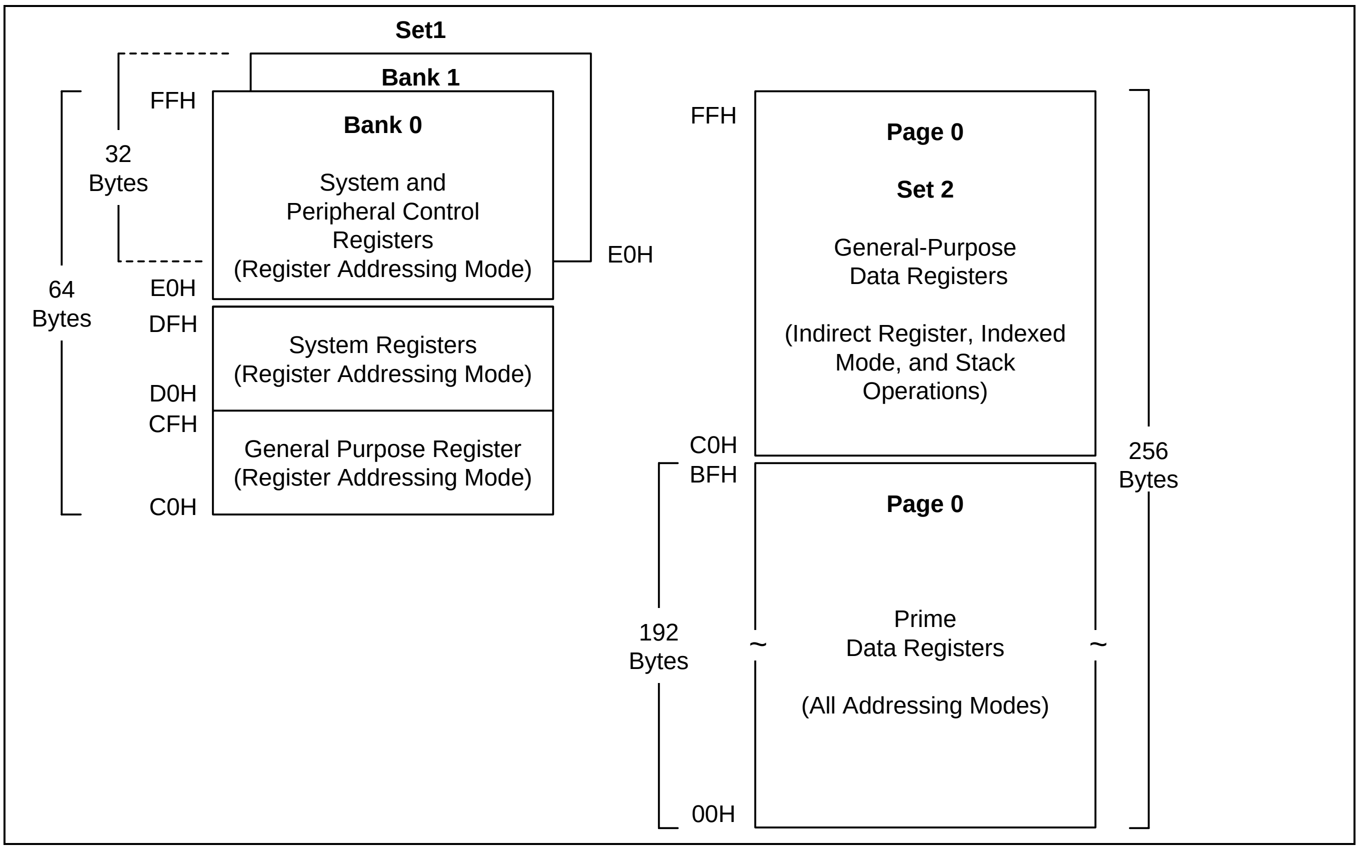
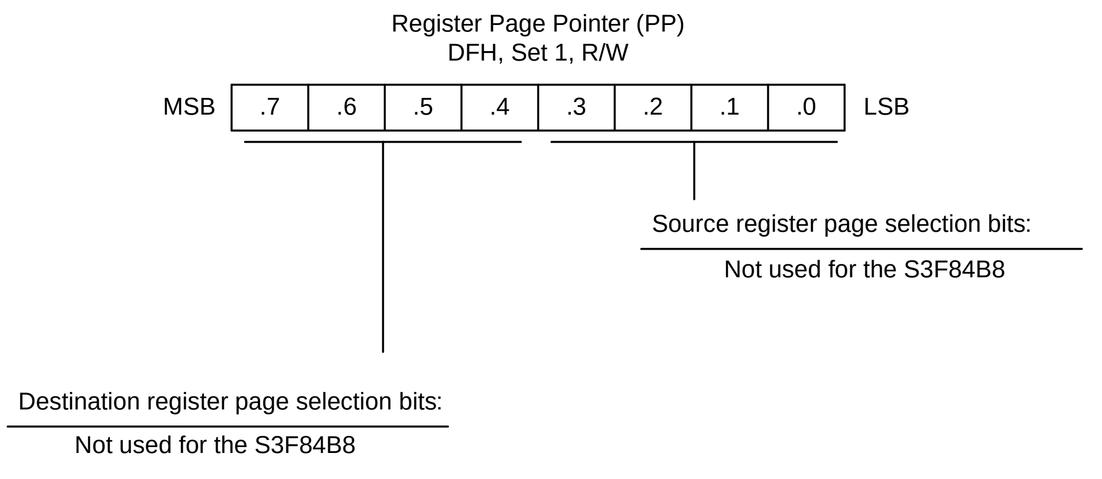

Set1
Bank 1
Bank 0
System and Peripheral Control Registers (Register Addressing Mode)
System Registers (Register Addressing Mode)
General Purpose Register (Register Addressing Mode)
FFH
32 Bytes
64 Bytes
E0H
DFH
D0H
CFH
C0H
E0H
FFH
Page 0
Set 2
General-Purpose Data Registers
(Indirect Register, Indexed Mode, and Stack Operations)
C0H
BFH
Page 0
Prime Data Registers
(All Addressing Modes)
192 Bytes
00H
256 Bytes

Figure 2-3. Internal Register File Organization (S3F84B8)

REGISTER PAGE POINTER (PP)

The S3F8-series architecture supports the logical expansion of the physical 256-byte internal register file (using an 8-bit data bus) into as many as 16 separately addressable register pages. Page addressing is controlled by the register page pointer (PP, DFH).

After a reset, the page pointer's source value (lower nibble) and the destination value (upper nibble) are always "0000", automatically selecting page 0 as the source and destination page for register addressing.


Register Page Pointer (PP)
DFH, Set 1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Source register page selection bits:
Not used for the S3F84B8
Destination register page selection bits:
Not used for the S3F84B8

NOTES:

1. A hardware reset operation writes the 4-bit destination and source values shown above to the register page pointer. These values should be modified to address other pages.

Figure 2-4. Register Page Pointer (PP)

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 expanded two 32-byte register banks, *bank 0* and *bank 1*. The set register bank instructions, SB0 or SB1, are used to address one bank or the other. A hardware reset operation always selects bank 0 addressing.

The upper two 32-byte areas (bank 0 and bank 1) of set 1 (E0H–FFH) contains 46 mapped system and peripheral control registers. The lower 32-byte area contains 14 system registers (D0H–DFH) and a 16-byte common working register area (C0H–CFH). You can use the common working register area as a "scratch" area for data operations being performed in other areas of the register file.

Registers in set 1 locations are directly accessible at any time 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 locations C0H–FFH is logically duplicated to add another 64 bytes of register space. This expanded area of the register file is called set *2*. For S3F84B8, the set 2 address range (C0H–FFH) is accessible on page 0 only. (S3F84B8 has implemented only page 0.)

The logical division of set 1 and set 2 is maintained by means of addressing mode restrictions. You can use only Register addressing mode to access set 1 location. In order to access registers in set 2, you must use Register Indirect addressing mode or Indexed addressing mode.

Set 2 register area is commonly used for stack operations.

PRIME REGISTER SPACE

The lower 192 bytes (00H–BFH) of the S3F84B8's 256-byte register page 0 is called *prime register area.* Prime registers can be accessed using any of the seven addressing modes
(see Chapter 3, "Addressing Modes.")

The prime register area on page 0 is immediately addressable following a reset. In order to address prime registers on pages 0, or 1 you must set the register page pointer (PP) to the appropriate source and destination values.

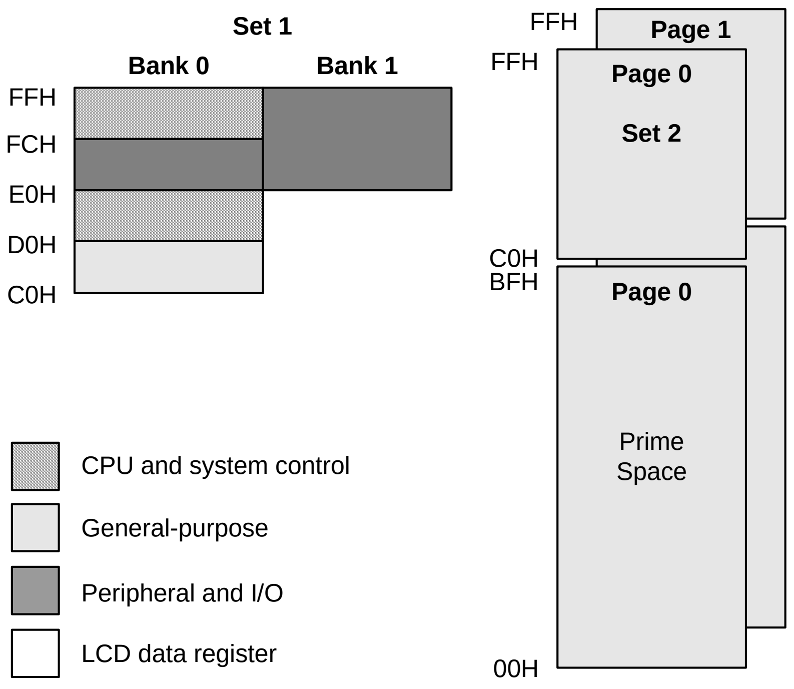


Figure 2-5. Set 1, Set 2, Prime Area Register Map

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.

Using the two 8-bit register pointers, RP1 and RP0, two working register slices can be selected at any one time to form a 16-byte working register block. Using the register pointers, you can move this 16-byte register block anywhere in the addressable register file, except 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 32 slices in the register file. The base addresses for the two selected 8-byte register slices 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).

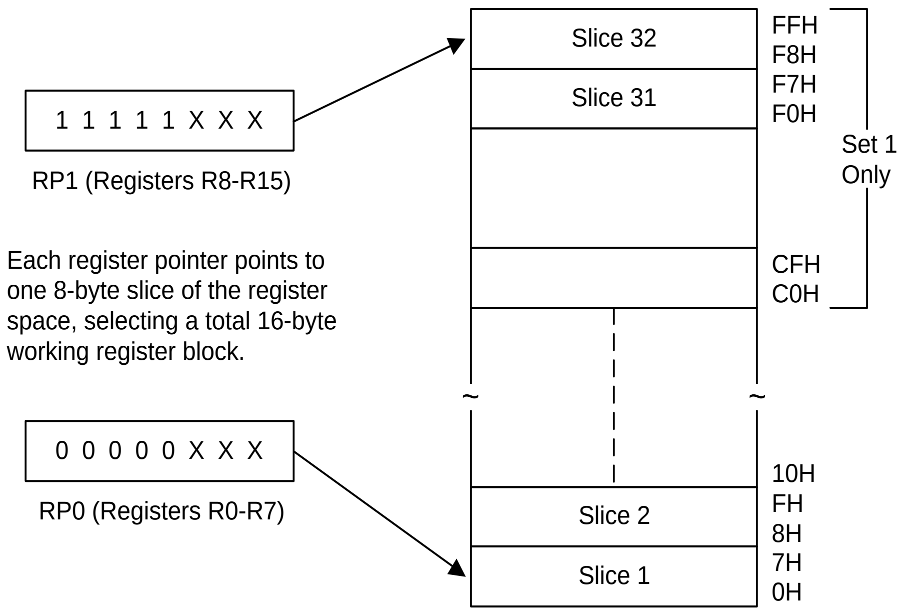


Figure 2-6. 8-Byte Working Register Areas (Slices)

USING THE REGISTER POINTS

Register pointers RP0 and RP1, mapped to 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 addresses C0H–C7H, and RP1 points to addresses C8H–CFH.

To change a register pointer value, you load a new value to RP0 and/or RP1 using an SRP or LD instruction. (see Figures 2-8 and 2-9).

With 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, however, use the register pointers to select a working register space in set 2, C0H–FFH, because these locations can be accessed only using the Indirect Register or Indexed addressing modes.

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

Because 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 program requirements.

PROGRAMMING TIP — Setting the Register Pointers

```
SRP     #70H          ; RP0 ← 70H, RP1 ← 78H
SRP1    #48H          ; RP0 ← no change, RP1 ← 48H,
SRP0    #0A0H         ; RP0 ← A0H, RP1 ← no change
CLR     RP0           ; RP0 ← 00H, RP1 ← no change
LD      RP1, #0F8H    ; RP0 ← no change, RP1 ← 0F8H
```

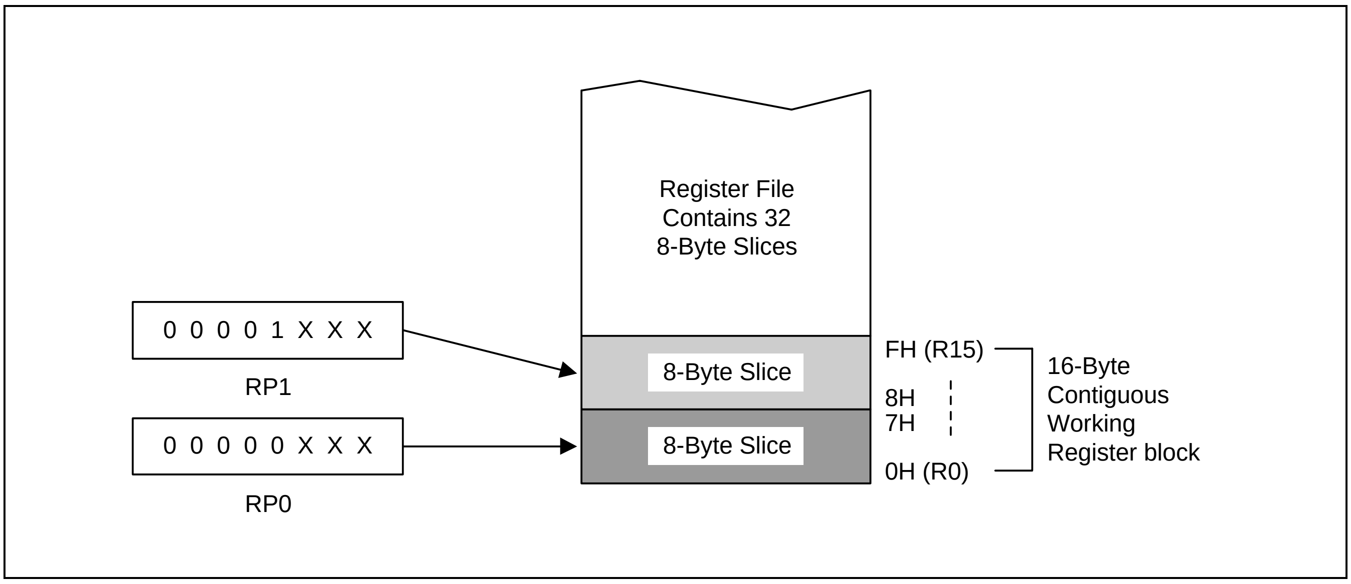


Figure 2-7. Contiguous 16-Byte Working Register Block

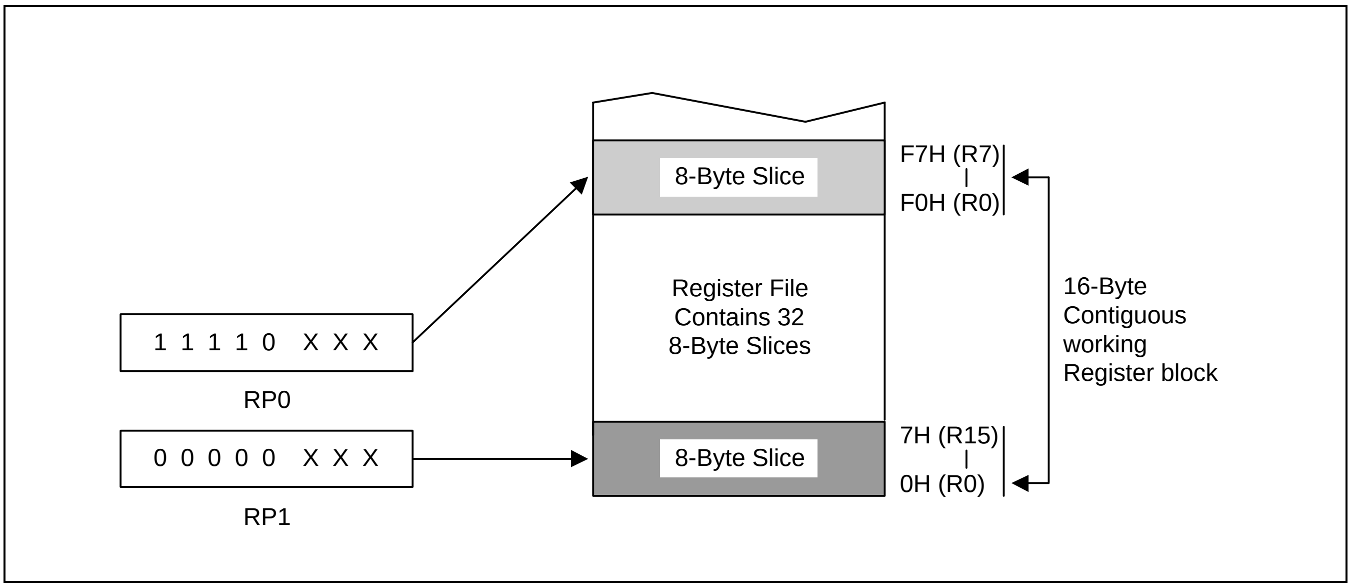


Figure 2-8. Non-Contiguous 16-Byte Working Register Block

PROGRAMMING TIP — Using the RPs to Calculate the Sum of a Series of Registers

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

```
SRP0      #80H           ;  RP0 ← 80H
ADD       R0,R1          ;  R0 ← R0 + R1
ADC       R0,R2          ;  R0 ← R0 + R2 + C
ADC       R0,R3          ;  R0 ← R0 + R3 + C
ADC       R0,R4          ;  R0 ← R0 + R4 + C
ADC       R0,R5          ;  R0 ← R0 + R5 + C
```

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 36 cycles. If the register pointer is not used to calculate the sum of these registers, the following instruction sequence would have to be used:

```
ADD       80H,81H        ;  80H ← (80H) + (81H)
ADC       80H,82H        ;  80H ← (80H) + (82H) + C
ADC       80H,83H        ;  80H ← (80H) + (83H) + C
ADC       80H,84H        ;  80H ← (80H) + (84H) + C
ADC       80H,85H        ;  80H ← (80H) + (85H) + C
```

Now, the sum of the six registers is also located in register 80H. However, this instruction string takes 15 bytes of instruction code rather than 12 bytes, and its execution time is 50 cycles rather than 36 cycles.

REGISTER ADDRESSING

The S3C8-series 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 any location 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, and the least significant byte is always stored in the next (+1) odd-numbered register.

Working register addressing differs from Register addressing as it uses a register pointer to identify a specific 8-byte working register space in the internal register file and a specific 8-bit register within that space.

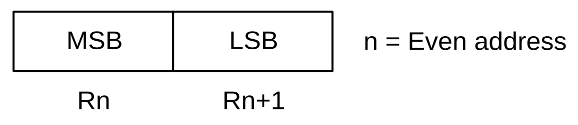


Figure 2-9. 16-Bit Register Pair

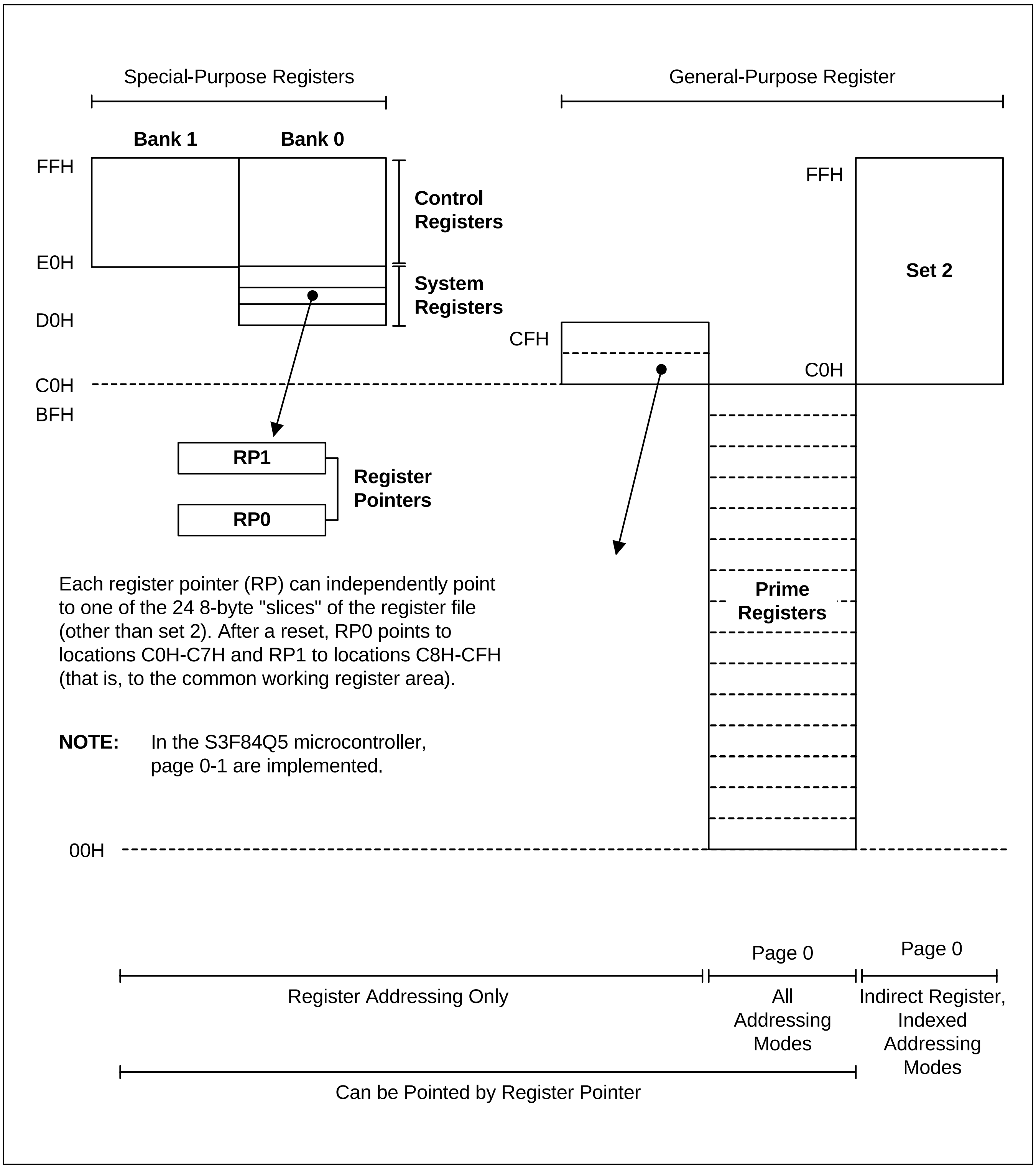

Special-Purpose Registers
General-Purpose Register
Bank 1
Bank 0
FFH
E0H
D0H
C0H
BFH
Control Registers
System Registers
CFH
FFH
C0H
Set 2
RP1
RP0
Register Pointers
Each register pointer (RP) can independently point to one of the 24 8-byte "slices" of the register file (other than set 2). After a reset, RP0 points to locations C0H-C7H and RP1 to locations C8H-CFH (that is, to the common working register area).
NOTE: In the S3F84Q5 microcontroller, page 0-1 are implemented.
Prime Registers
00H
Page 0
Page 0
Register Addressing Only
All Addressing Modes
Indirect Register, Indexed Addressing Modes
Can be Pointed by Register Pointer

Figure 2-10. Register File Addressing

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:

RP0 $\rightarrow$ C0H–C7H

RP1 $\rightarrow$ C8H–CFH

This 16-byte address range is called *common 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.

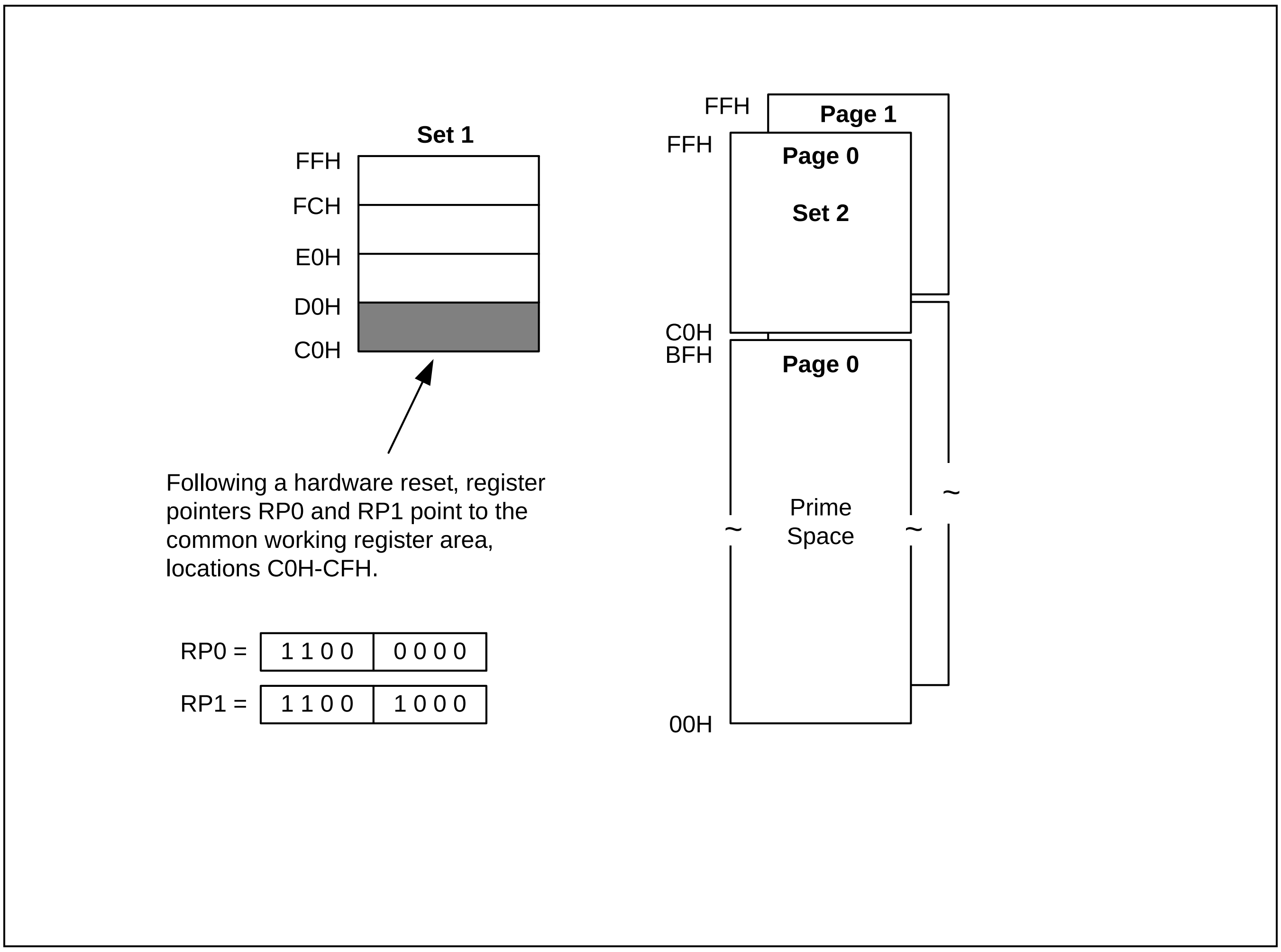

Set 1
FFH
FCH
E0H
D0H
C0H
Following a hardware reset, register pointers RP0 and RP1 point to the common working register area, locations C0H-CFH.
RP0 =
1 1 0 0
0 0 0 0
RP1 =
1 1 0 0
1 0 0 0
FFH
Page 1
FFH
Page 0
Set 2
C0H
BFH
Page 0
Prime Space
00H

Figure 2-11. Common Working Register Area

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 2-12, 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 2-14 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).

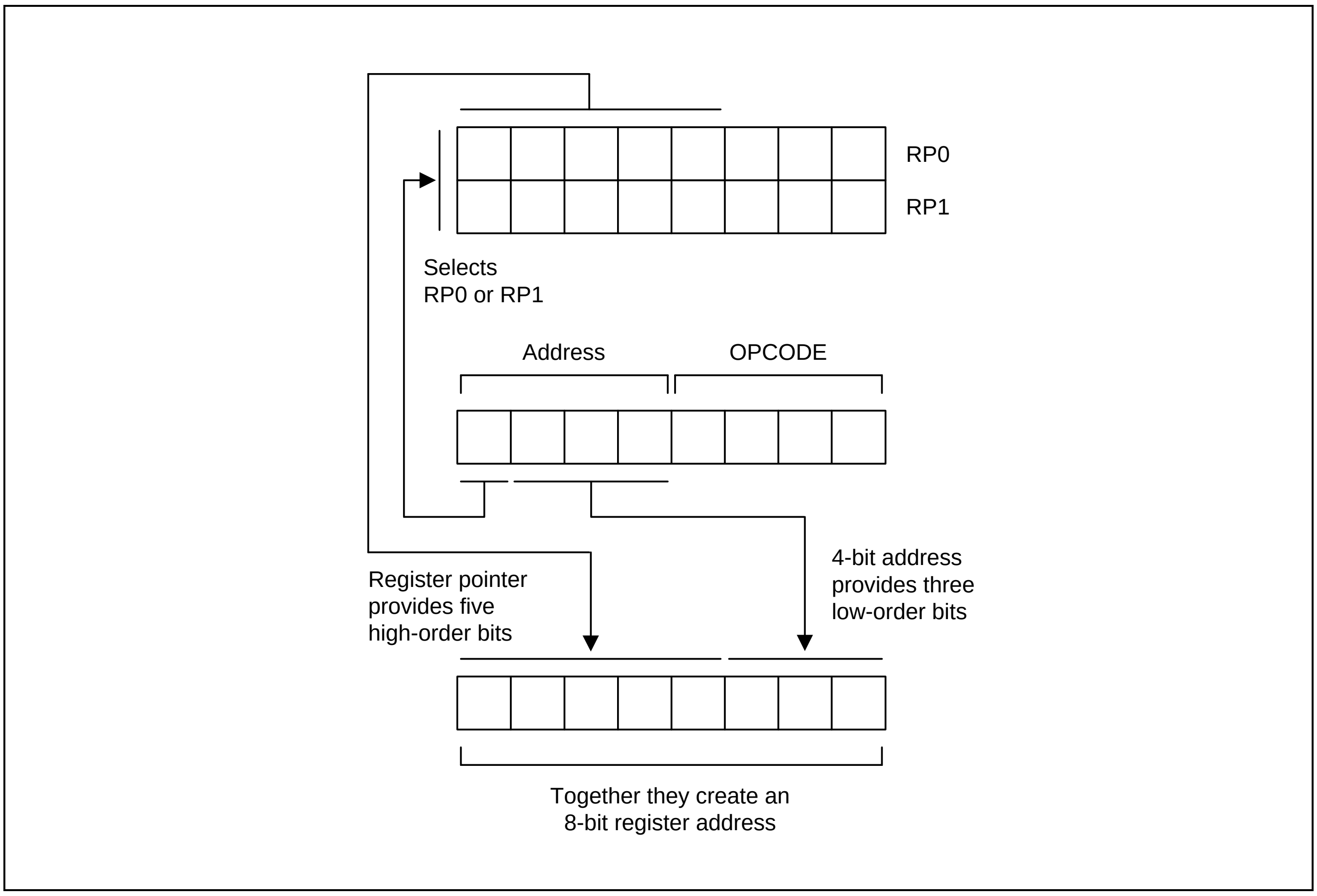

RP0
RP1
Selects RP0 or RP1
Address
OPCODE
Register pointer provides five high-order bits
4-bit address provides three low-order bits
Together they create an 8-bit register address

Figure 2-12. 4-Bit Working Register Addressing

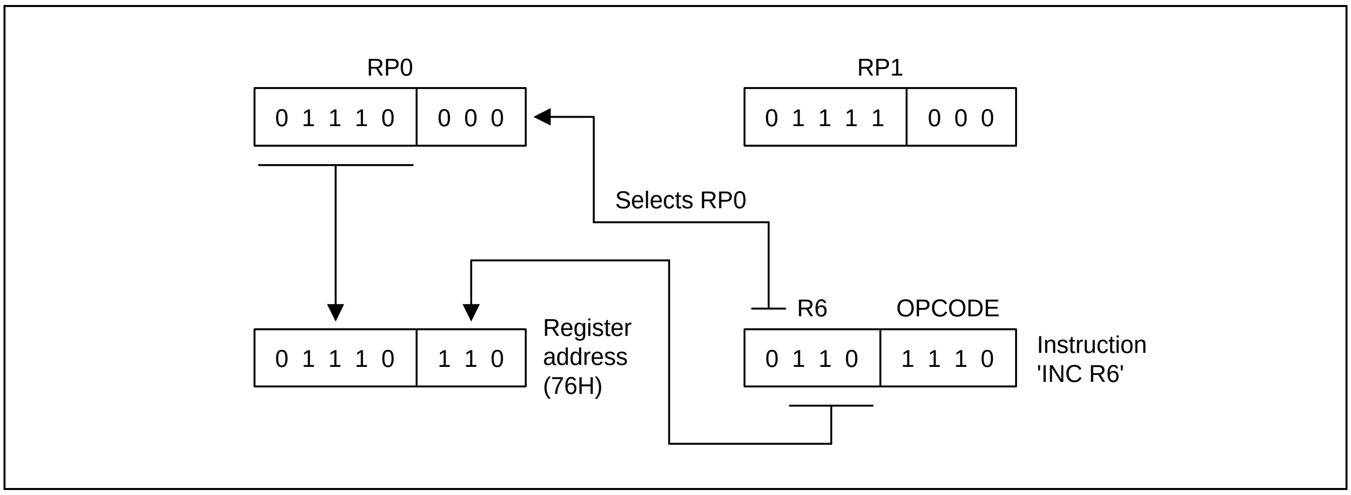


Figure 2-13. 4-Bit Working Register Addressing Example

8-BIT WORKING REGISTER ADDRESSING

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 "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 2-14, 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 2-15 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).

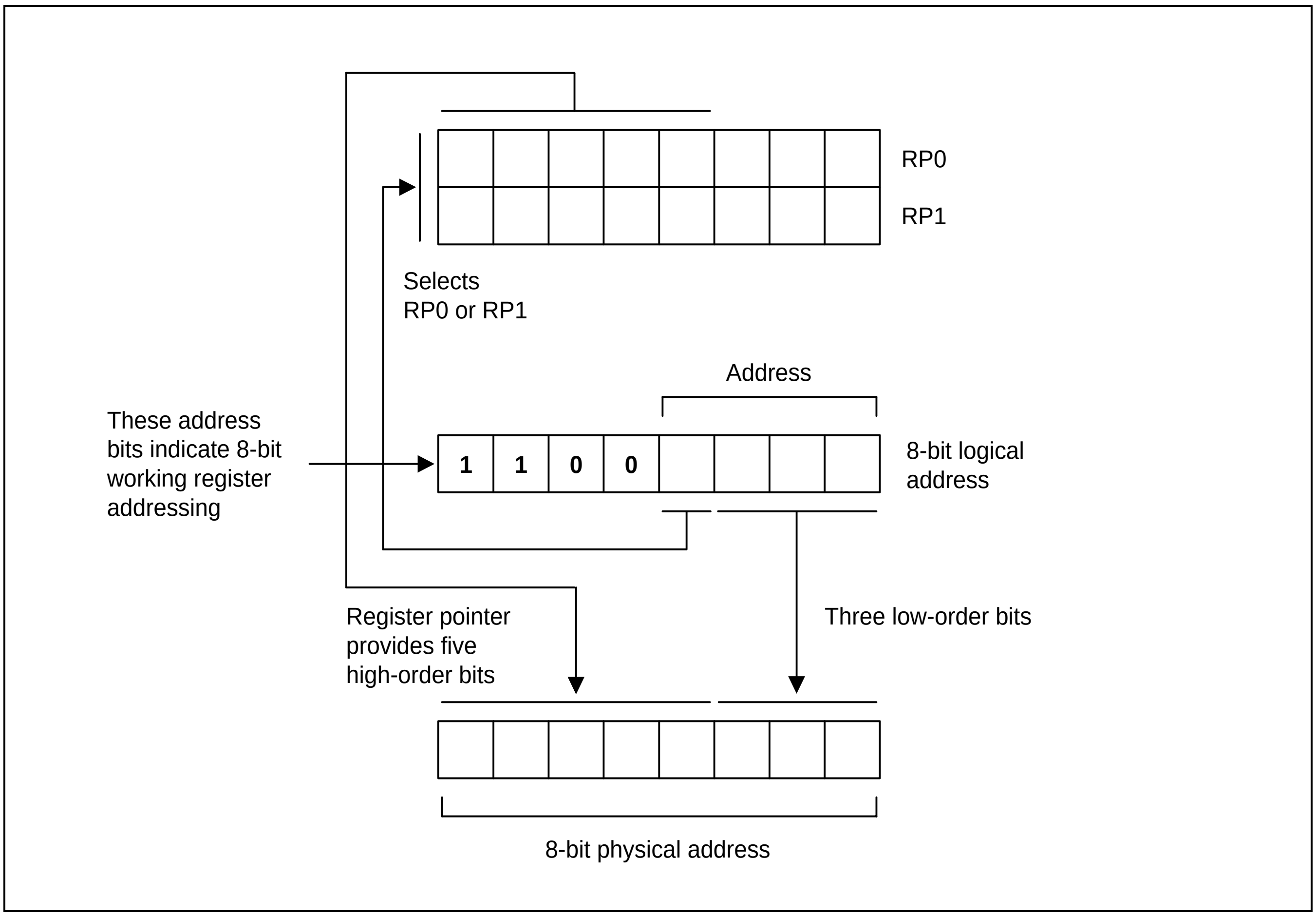


Figure 2-14. 8-Bit Working Register Addressing

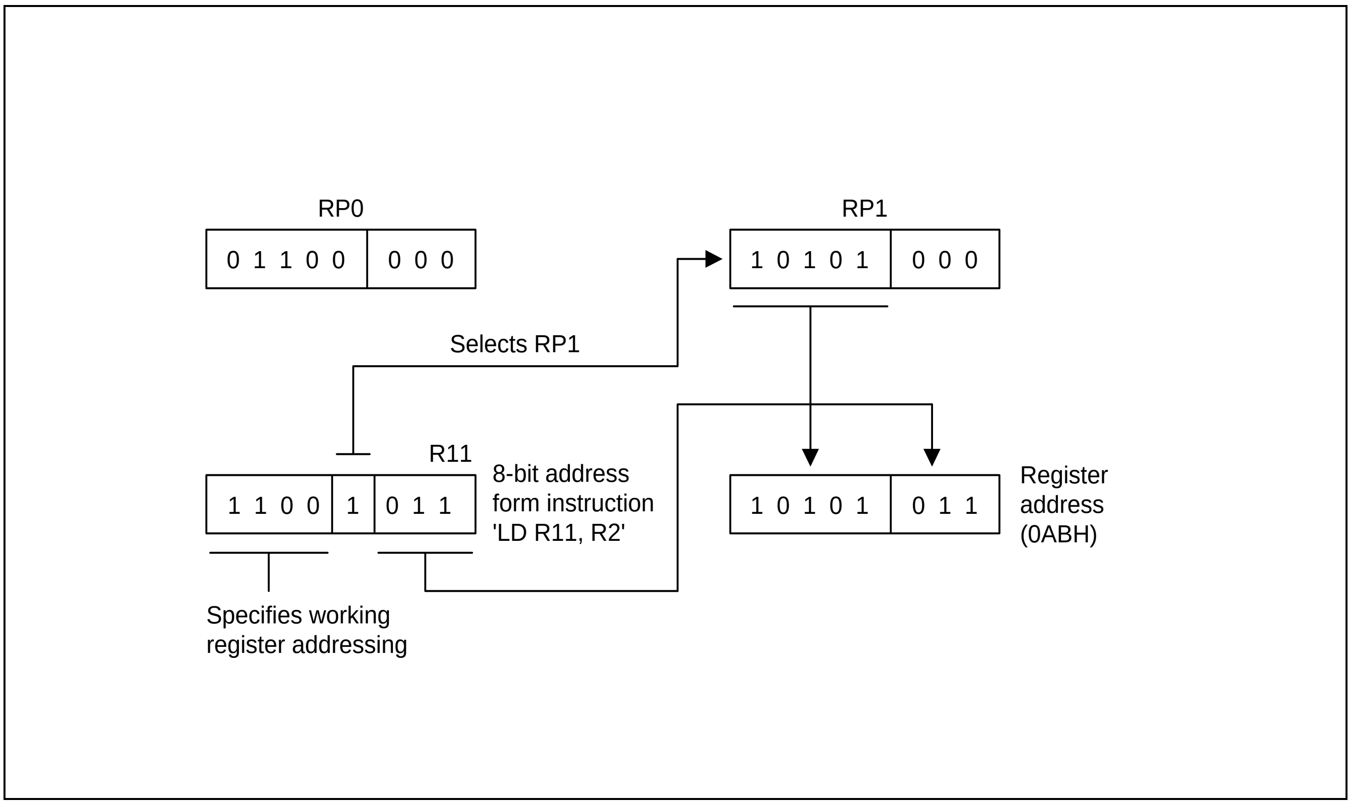


Figure 2-15. 8-Bit Working Register Addressing Example

SYSTEM AND USER STACK

The S3C8-series microcontrollers use the system stack for data storage, subroutine calls and returns. The PUSH and POP instructions are used to control system stack operations. The S3F84B8 architecture supports stack operations in the internal register file.

Stack Operations

Return addresses for procedure calls, 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 value is always decreased by one before a push operation and increased by one 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 2-16.

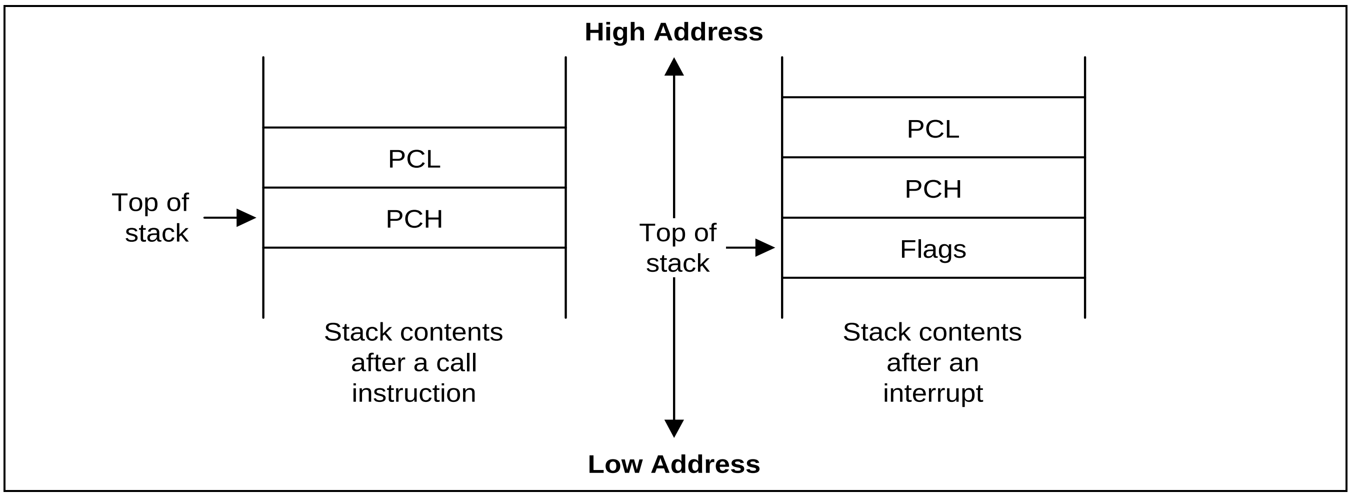

High Address
Top of stack
PCL
PCH
Stack contents after a call instruction
Top of stack
PCL
PCH
Flags
Stack contents after an interrupt
Low Address

Figure 2-16. Stack Operations

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 the S3F84B8, the SPL must be initialized to an 8-bit value in the range 00H–FFH. The SPH register is not needed 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 a result of increasing or decreasing the stack address value in the SPL register during normal stack operations, the value in the SPL register will overflow (or underflow) to the SPH register, overwriting any other data that is currently stored there. To avoid overwriting data in the SPH register, you can initialize the SPL value to "FFH" instead of "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
```

NOTES

3 ADDRESSING MODES

OVERVIEW

Instructions that are stored in program memory are fetched for execution using the program counter. Instructions indicate the operation to be performed and the data to be operated on. Addressing mode is the method used to determine the location of the data operand. The operands specified in SAM8RC instructions may be condition codes, immediate data, or a location in the register file, program memory, or data memory.

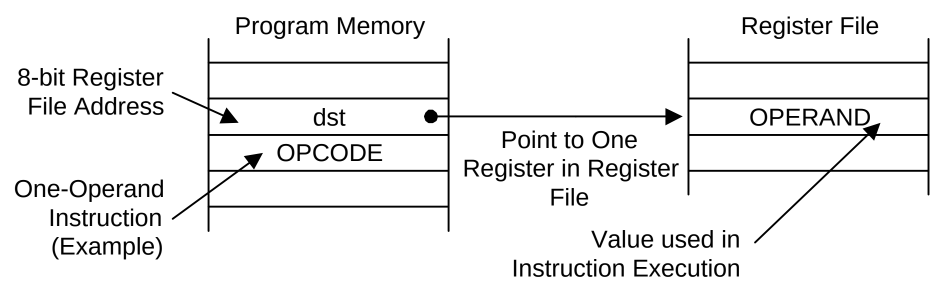
The S3C-series instruction set supports seven explicit addressing modes. Not all of these addressing modes are available for each instruction. The seven addressing modes and their symbols are:

- 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 (R), the operand value is the content of a specified register or register pair (see Figure 3-1).

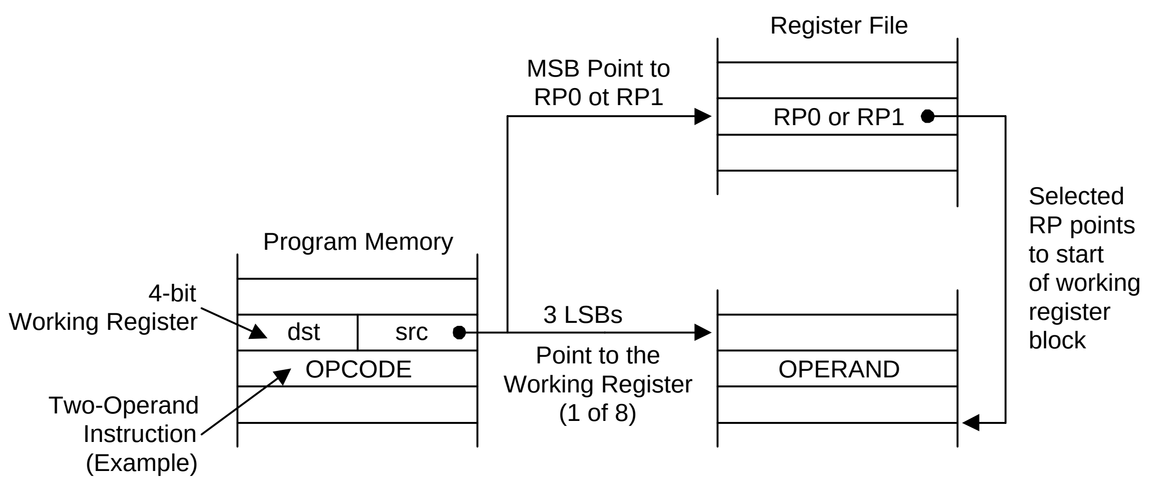
Working register addressing differs from Register addressing in that it uses a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space (see Figure 3-2).


Program Memory
Register File
8-bit Register File Address
dst
OPCODE
OPERAND
Point to One Register in Register File
One-Operand Instruction (Example)
Value used in Instruction Execution

Sample Instruction:

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

Figure 3-1. Register Addressing


Register File
MSB Point to RP0 ot RP1
RP0 or RP1
Selected RP points to start of working register block
Program Memory
4-bit Working Register
dst
src
3 LSBs
OPCODE
Point to the Working Register (1 of 8)
OPERAND
Two-Operand Instruction (Example)

Sample Instruction:

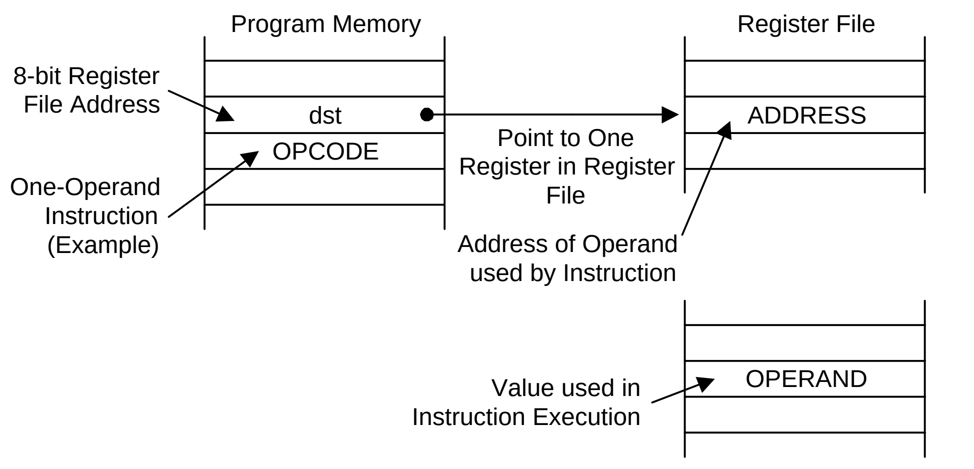
ADD R1, R2 ; Where R1 and R2 are registers in the currently selected working register area.

Figure 3-2. Working Register Addressing

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 (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. Please note, however, that you cannot access locations C0H–FFH in set 1 using the Indirect Register addressing mode.


Program Memory
Register File
8-bit Register File Address
dst
OPCODE
One-Operand Instruction (Example)
Point to One Register in Register File
ADDRESS
Address of Operand used by Instruction
Value used in Instruction Execution
OPERAND

Sample Instruction:

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)

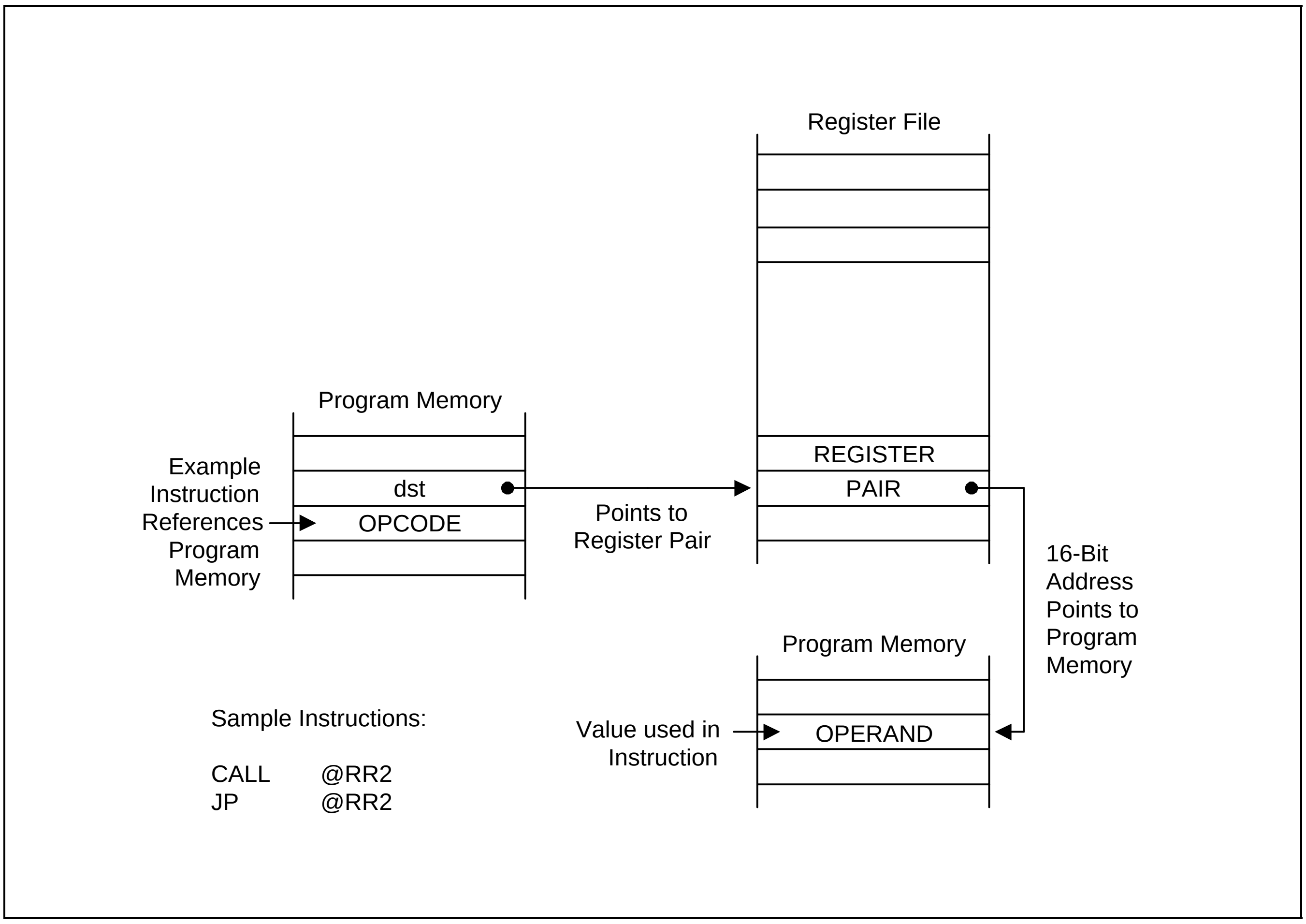


Figure 3-4. Indirect Register Addressing to Program Memory

INDIRECT REGISTER ADDRESSING MODE (Continued)

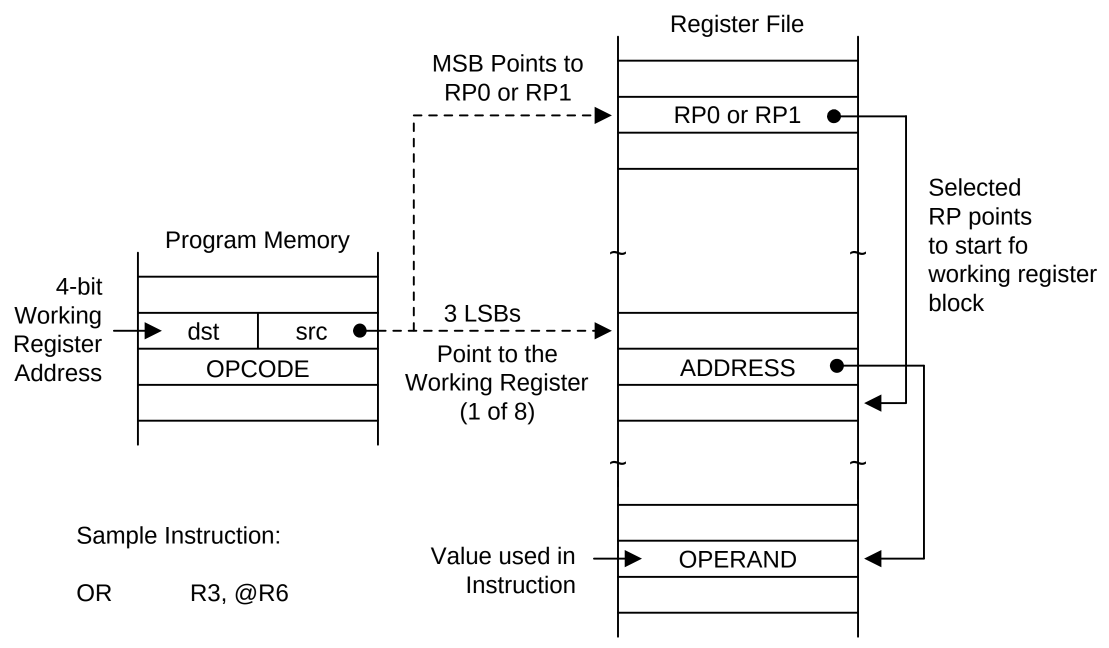
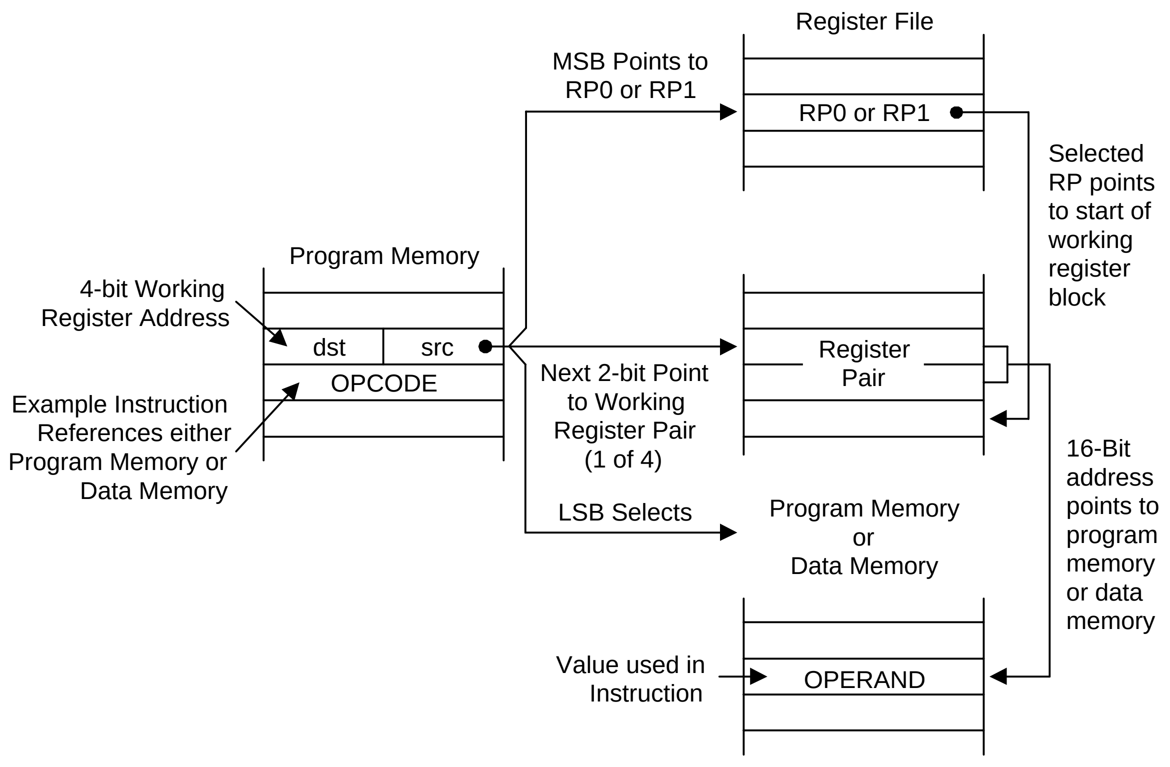

Register File
MSB Points to RP0 or RP1
RP0 or RP1
Selected RP points to start fo working register block
Program Memory
4-bit Working Register Address
dst
src
3 LSBs
Point to the Working Register (1 of 8)
OPCODE
ADDRESS
Sample Instruction:
OR R3, @R6
Value used in Instruction
OPERAND

Figure 3-5. Indirect Working Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Concluded)


Register File
MSB Points to RP0 or RP1
RP0 or RP1
Selected RP points to start of working register block
Program Memory
4-bit Working Register Address
dst
src
OPCODE
Example Instruction References either Program Memory or Data Memory
Next 2-bit Point to Working Register Pair (1 of 4)
Register Pair
16-Bit address points to program memory or data memory
LSB Selects
Program Memory or Data Memory
Value used in Instruction
OPERAND

Sample Instructions:

```
LCD     R5,@RR6        ;  Program memory access
LDE     R3,@RR14       ;  External data memory access
LDE     @RR4, R8       ;  External data memory access
```

Figure 3-6. Indirect Working Register Addressing to Program or Data Memory

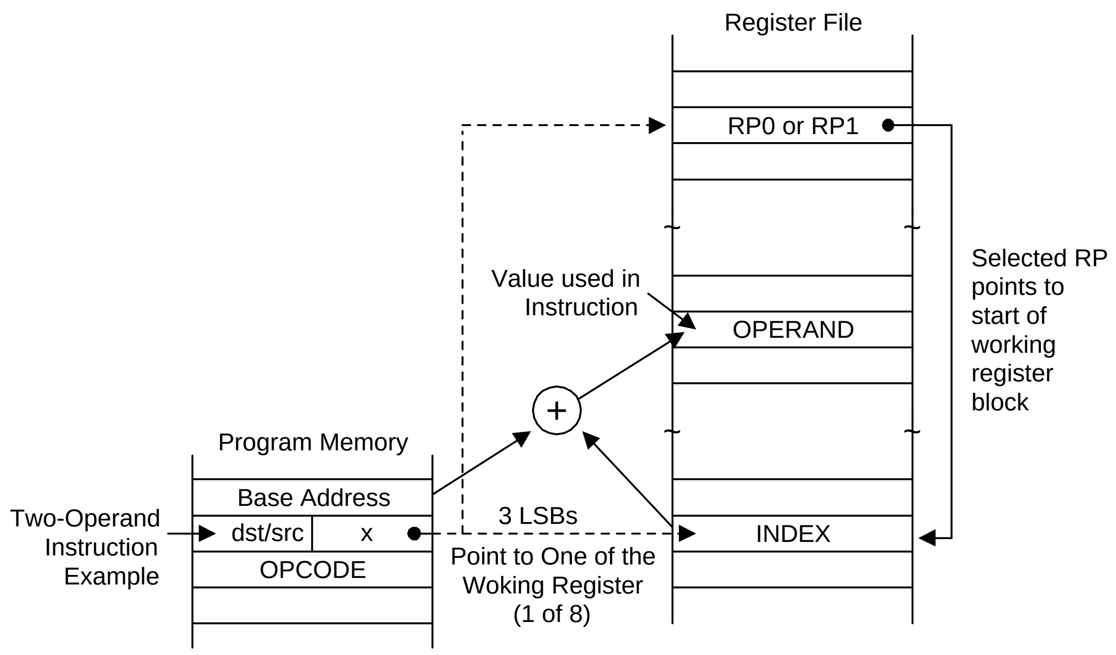
INDEXED ADDRESSING MODE (X)

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. Please note, however, that you cannot access locations C0H–FFH in set 1 using Indexed addressing mode.

In short offset Indexed addressing mode, the 8-bit displacement is treated as a signed integer in the range –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 accesses, 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 that 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, when implemented.

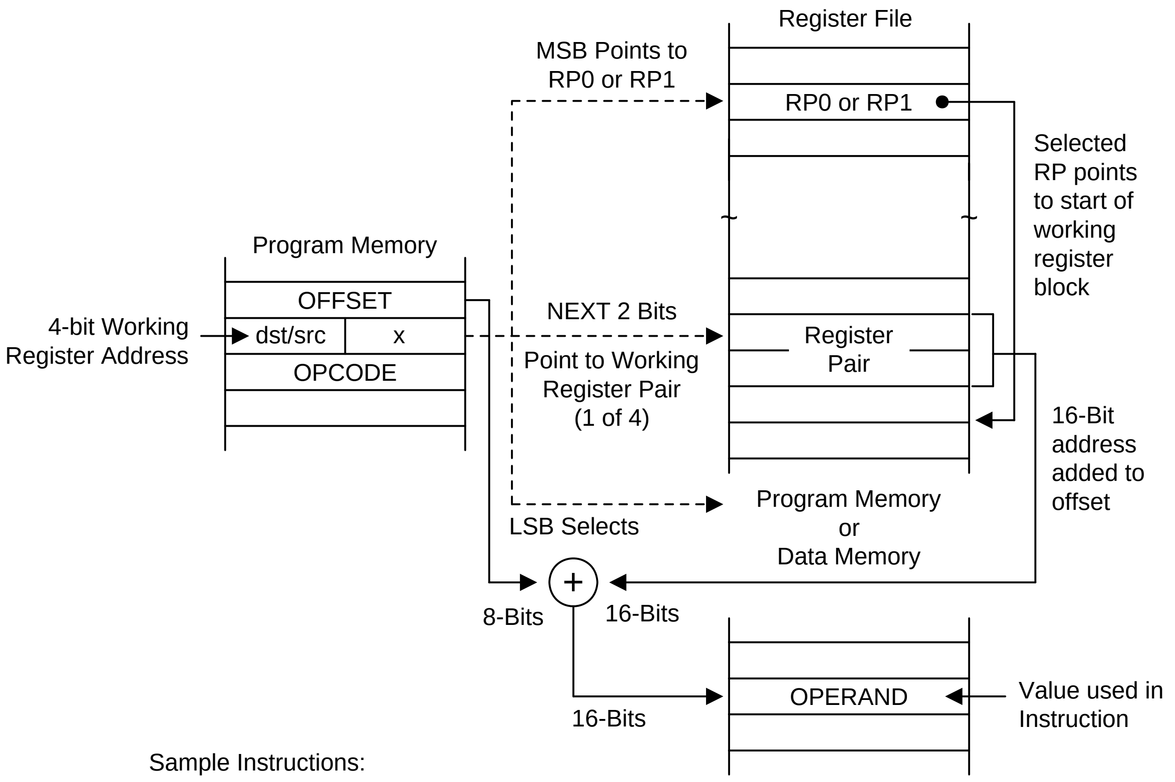

Register File
RP0 or RP1
Value used in Instruction
OPERAND
Selected RP points to start of working register block
Program Memory
Base Address
Two-Operand Instruction Example
dst/src
x
OPCODE
3 LSBs
INDEX
Point to One of the Woking Register (1 of 8)
+

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)

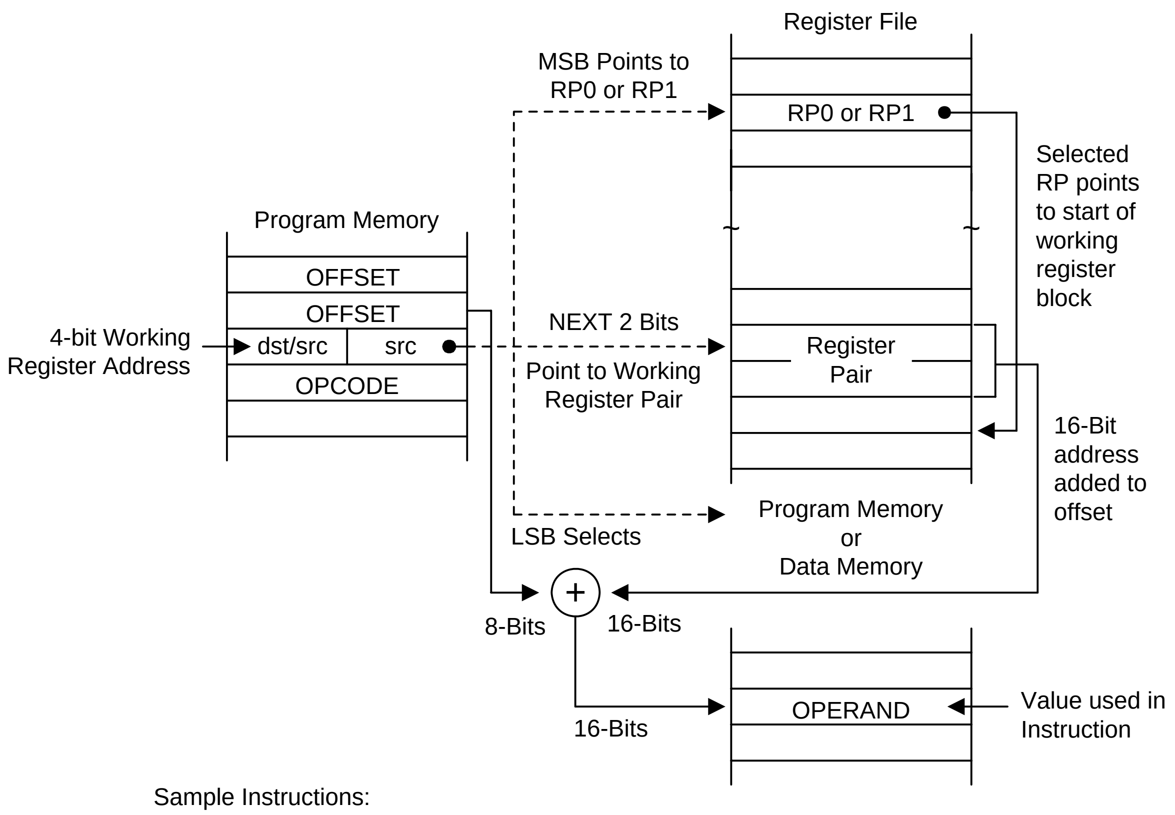

Register File
MSB Points to RP0 or RP1
RP0 or RP1
Selected RP points to start of working register block
Program Memory
OFFSET
4-bit Working Register Address
dst/src
x
OPCODE
NEXT 2 Bits
Point to Working Register Pair (1 of 4)
Register Pair
16-Bit address added to offset
LSB Selects
Program Memory or Data Memory
8-Bits
16-Bits
OPERAND
Value used in Instruction
16-Bits

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)


Register File
MSB Points to RP0 or RP1
RP0 or RP1
Selected RP points to start of working register block
Program Memory
OFFSET
OFFSET
4-bit Working Register Address
dst/src
src
OPCODE
NEXT 2 Bits Point to Working Register Pair
Register Pair
16-Bit address added to offset
LSB Selects
Program Memory or Data Memory
+
8-Bits
16-Bits
OPERAND
Value used in Instruction
16-Bits

Sample Instructions:

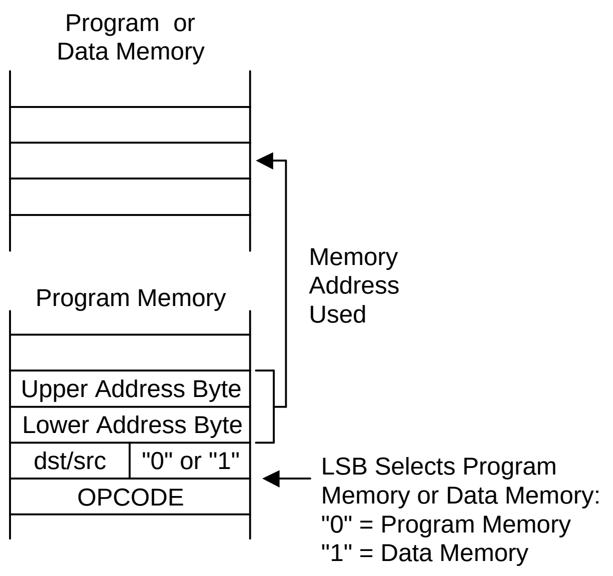
LDC	R4, #1000H[RR2]	; The values in the program address (RR2 + 1000H) are loaded into register R4.
LDE	R4,#1000H[RR2]	; Identical operation to LDC example, except that external program memory is accessed.

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.

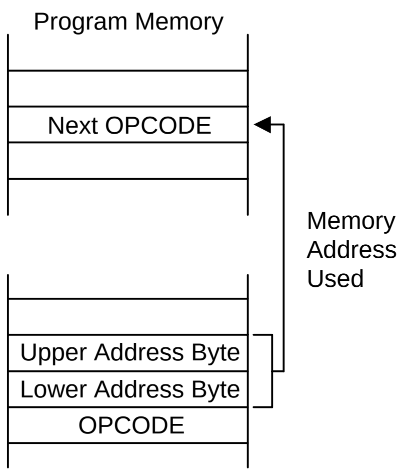


Sample Instructions:

LDC	R5,1234H	;	The values in the program address (1234H) are loaded into register R5.
LDE	R5,1234H	;	Identical operation to LDC example, except that external program memory is accessed.

Figure 3-10. Direct Addressing for Load Instructions

DIRECT ADDRESS MODE (Continued)



Sample Instructions:

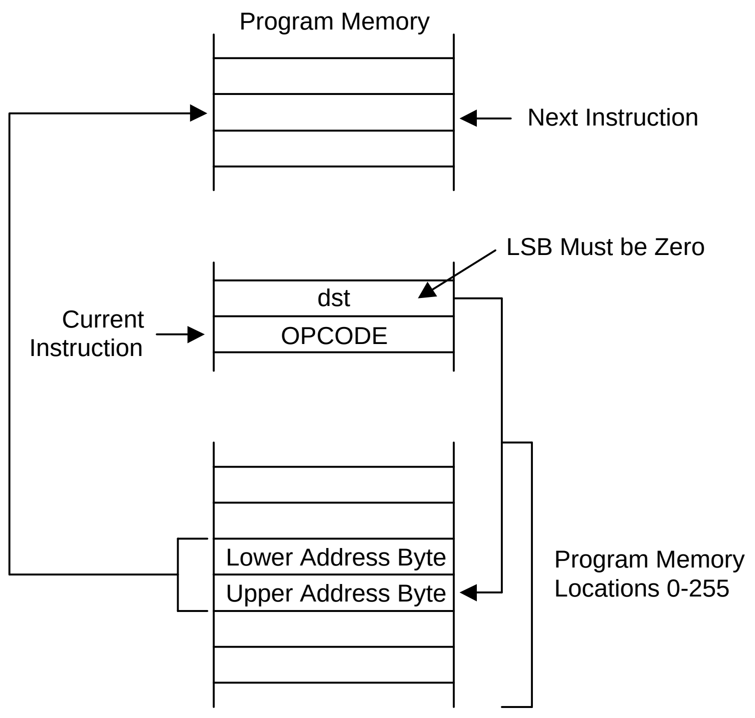
```
JP      C,JOB1        ;   Where JOB1 is a 16-bit immediate address
CALL    DISPLAY       ;   Where DISPLAY is a 16-bit immediate address
```

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 the Indirect Address mode.

Because the 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:

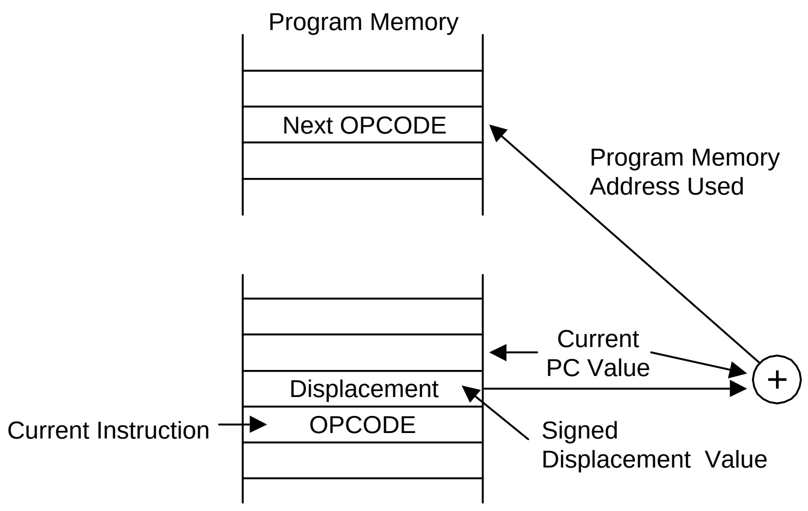
```
CALL    #40H          ;  The 16-bit value in program memory addresses 40H
                         and 41H is the subroutine start address.
```

Figure 3-12. Indirect Addressing

RELATIVE ADDRESS MODE (RA)

In Relative Address (RA) mode, a twos-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 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.



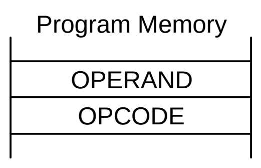
Sample Instructions:

JR ULT,$+OFFSET ; Where OFFSET is a value in the range +127 to -128

Figure 3-13. Relative Addressing

IMMEDIATE MODE (IM)

In Immediate (IM) addressing 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.



(The Operand value is in the instruction)

Sample Instruction:

```
LD      R0,#0AAH
```

Figure 3-14. Immediate Addressing

NOTES

4 CONTROL REGISTERS

OVERVIEW

In this section, detailed descriptions of the S3F84B8 control registers are presented in an easy-to-read format. These descriptions will help familiarize you with the mapped locations in the register file. You can also use them as a quick-reference source when writing application programs.

System and peripheral registers are summarized in Table 4-1. 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 information about control registers is presented in the context of the various peripheral hardware descriptions in Part II of this manual.

Table 4-1. System and Peripheral Control Registers Set1 Bank0

Register name	Mnemonic	Address & Location		RESET value (Bit)							
		Address	R/W	7	6	5	4	3	2	1	0
Locations D0-D2H are not mapped											
Basic timer control register	BTCON	D3H	R/W	0	0	0	0	0	0	0	0
Clock control register	CLKCON	D4H	R/W	0	–	–	0	0	–	–	–
System flags register	FLAGS	D5H	R/W	x	x	x	x	x	x	0	0
Register Pointer 0	RP0	D6H	R/W	1	1	0	0	0	–	–	–
Register Pointer 1	RP1	D7H	R/W	1	1	0	0	1	–	–	–
Location D8H is not mapped											
Stack Pointer register	SPL	D9H	R/W	x	x	x	x	x	x	x	x
Instruction Pointer (High Byte)	IPH	DAH	R/W	x	x	x	x	x	x	x	x
Instruction Pointer (Low Byte)	IPL	DBH	R/W	x	x	x	x	x	x	x	x
Interrupt Request register	IRQ	DCH	R	0	0	0	0	0	0	0	0
Interrupt Mask Register	IMR	DDH	R/W	x	x	x	x	x	x	x	x
System Mode Register	SYM	DEH	R/W	0	–	–	x	x	x	0	0
Register Page Pointer	PP	DFH	R/W	0	0	0	0	0	0	0	0

NOTE: – : Not mapped or not used, x: Undefined

Table 4-1. System and Peripheral Control Registers Set1 Bank 0(Continued)

Register Name	Mnemonic	Address	R/W	Bit Values After RESET							
		Hex		7	6	5	4	3	2	1	0
Port 0 data register	P0	E0H	R/W	–	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	R/W	–	–	–	–	–	0	0	0
Port 2 data register	P2	E2H	R/W	0	0	0	0	0	0	0	0
Port 0 interrupt control register	P0INT	E3H	R/W	–	0	0	0	0	–	0	0
Port 0 control register (High byte)	P0CONH	E4H	R/W	–	–	0	0	0	0	0	0
Port 0 control register (Low byte)	P0CONL	E5H	R/W	0	0	–	–	0	0	0	0
Port 0 interrupt pending register	P0PND	E6H	R/W	–	0	0	0	0	–	0	0
Port 1 control register	P1CON	E7H	R/W	–	–	0	0	0	0	0	0
Port 2 control register (High byte)	P2CONH	E8H	R/W	0	0	0	0	0	0	0	0
Port 2 control register (Low byte)	P2CONL	E9H	R/W	0	0	0	0	0	0	0	0
Comparator 0 control register	CMP0CON	EAH	R/W	–	–	–	0	0	0	1	0
Comparator 1 control register	CMP1CON	EBH	R/W	0	0	0	0	0	0	1	0
Comparator 2 control register	CMP2CON	ECH	R/W	0	0	0	0	0	0	1	0
Comparator 3 control register	CMP3CON	EDH	R/W	0	0	0	0	0	0	1	0
Comparator interrupt control register	CMPINT	EEH	R/W	1	1	1	1	1	1	1	1
PWM control register	PWMCON	EFH	R/W	0	0	0	0	0	0	0	0
PWM CMP register	PWMCCON	F0H	R/W	–	–	–	–	0	0	0	0
PWM delay trigger data register	PWMDL	F1H	R/W	0	0	0	0	0	0	0	0
PWM preset data register (High byte)	PWMPDATAH	F2H	R/W	0	0	0	0	0	0	0	0
PWM preset data register (Low byte)	PWMPDATAL	F3H	R/W	–	–	–	–	–	–	0	0
PWM data register (High byte)	PWMDATAH	F4H	R/W	0	0	0	0	0	0	0	0
PWM data register (Low byte)	PWMDATAL	F5H	R/W	–	–	–	–	–	–	0	0
Anti-mis-trigger data register	AMTDATA	F6H	R/W	1	1	1	1	1	1	1	1
Buzzer control register	BUZCON	F7H	R/W	0	0	0	0	0	0	0	0
A/D converter data register (High byte)	ADDATAH	F8H	R	x	x	x	x	x	x	x	x
A/D converter data register (Low byte)	ADDDATAL	F9H	R	–	–	–	–	–	–	x	x
A/D control register	ADCON	FAH	R/W	0	0	0	0	0	0	0	0
Locations FB-FCH are not mapped											
Basic timer counter	BTCNT	FDH	R	0	0	0	0	0	0	0	0
Location FEH is not mapped											
Interrupt priority register	IPR	FFH	R/W	x	x	x	x	x	x	x	x

NOTES:1. – : Not mapped or not used, x: Undefined,

Table 4-1. System and Peripheral Control Registers Set1 Bank1

Register name	Mnemonic	Address & Location		RESET value (Bit)							
		Address	**R/W**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Operational Amplifier control register	OPACON	E0H	R/W	–	–	–	–	–	–	0	0
Timer A control register	TACON	E1H	R/W	0	0	0	0	0	0	0	0
Timer A clock pre-scalar	TAPS	E2H	R/W	0	–	–	–	0	0	0	0
Timer A data register	TADATA	E3H	R/W	1	1	1	1	1	1	1	1
Timer A counter register	TACNT	E4H	R	0	0	0	0	0	0	0	0
Timer C control register	TCCON	E5H	R/W	0	–	0	0	0	–	0	–
Timer C clock pre-scalar	TCPS	E6H	R/W	0	–	–	–	0	0	0	0
Timer C data register	TCDATA	E7H	R/W	1	1	1	1	1	1	1	1
Timer C counter register	TCCNT	E8H	R	x	x	x	x	x	x	x	x
Timer D control register	TDCON	E9H	R/W	0	0	0	0	0	0	0	0
Timer D clock pre-scalar	TDPS	EAH	R/W	0	–	–	–	0	0	0	0
Timer D data register	TDDATA	EBH	R/W	1	1	1	1	1	1	1	1
Timer D counter register	TDCNT	ECH	R	x	x	x	x	x	x	x	x
Locations EDH-F1H are not mapped											
Reset source indicating register	RESETID	F2H	RW	Refer to the detail description							
Location F3H is not mapped											
STOP control register	STOPCON	F4H	R/W	0	0	0	0	0	0	0	0
Flash memory control register	FMCON	F5H	R/W	0	0	0	0	0	–	–	0
Flash memory user programming enable register	FMUSR	F6H	R/W	0	0	0	0	0	0	0	0
Flash memory sector address register (high byte)	FMSECH	F7H	R/W	0	0	0	0	0	0	0	0
Flash memory sector address register (low byte)	FMSECL	F8H	R/W	0	0	0	0	0	0	0	0
Locations F9H – FFH are not mapped											

NOTE: – : Not mapped or not used, x: Undefined

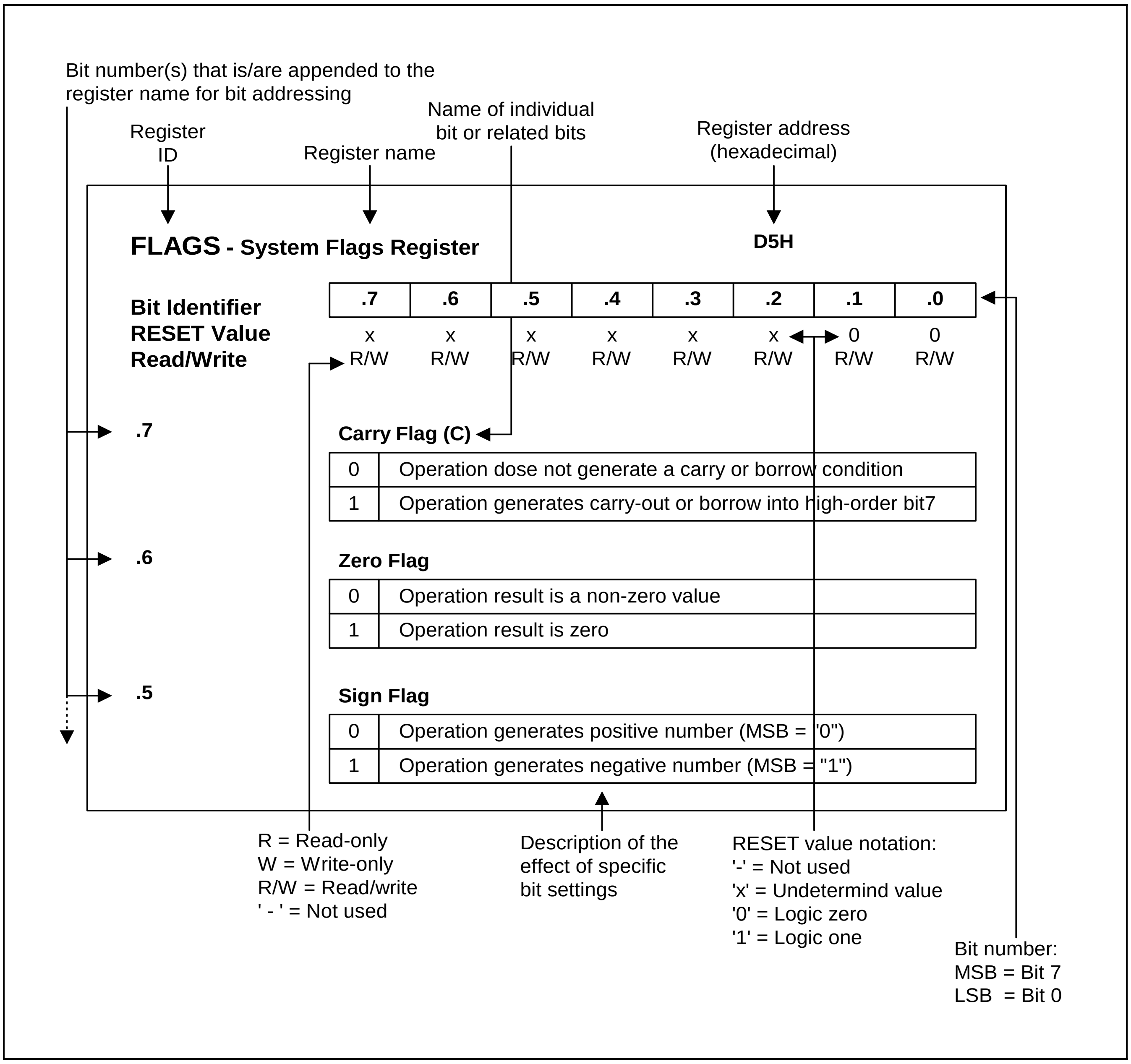

Bit number(s) that is/are appended to the register name for bit addressing
Register ID
Register name
Name of individual bit or related bits
Register address (hexadecimal)
FLAGS - System Flags Register
D5H
Bit Identifier
RESET Value
Read/Write
.7 .6 .5 .4 .3 .2 .1 .0
x x x x x x 0 0
R/W R/W R/W R/W R/W R/W R/W R/W
.7
Carry Flag (C)
0 Operation dose not generate a carry or borrow condition
1 Operation generates carry-out or borrow into high-order bit7
.6
Zero Flag
0 Operation result is a non-zero value
1 Operation result is zero
.5
Sign Flag
0 Operation generates positive number (MSB = "0")
1 Operation generates negative number (MSB = "1")
R = Read-only
W = Write-only
R/W = Read/write
' - ' = Not used
Description of the effect of specific bit settings
RESET value notation:
'-' = Not used
'x' = Undetermind value
'0' = Logic zero
'1' = Logic one
Bit number:
MSB = Bit 7
LSB = Bit 0

Figure 4-1. Register Description Format

ADCON — A/D Converter Control Register FAH, BANK0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
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

.7–.5 **A/D Converter Input Pin Selection Bits**

0	0	0	ADC0 (P2.0)
0	0	1	ADC1 (P2.1)
0	1	0	ADC2 (P2.2)
0	1	1	ADC3 (P2.3)
1	0	0	ADC4 (P2.4)
1	0	1	ADC5 (P2.5)
1	1	0	ADC6 (P2.6)
1	1	1	ADC7 (P2.7)

.4 **AD Conversion completion Interrupt Enable Bit**

0	Disable ADC Interrupt
1	Enable ADC Interrupt

.3 **A/DC Interrupt Pending Bit (EOC)**

0	No interrupt pending, conversion is in progress (clear pending bit when write)
1	Interrupt pending, conversion has completed (no effect when write)

.2–.1 **Clock Source Selection Bit** (NOTE)

0	0	$f_{OSC}/8$ ($f_{OSC} \leq 10$ MHz)
0	1	$f_{OSC}/4$ ($f_{OSC} \leq 10$ MHz)
1	0	$f_{OSC}/2$ ($f_{OSC} \leq 8$ MHz)
1	1	$f_{OSC}/1$ ($f_{OSC} \leq 4$ MHz)

.0 **Conversion Start Bit**

0	No meaning
1	A/D conversion start

NOTE: Maximum ADC clock input = 4 MHz.

AMTDATA —Anti-mis-trigger Data Register F6H, BANK0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
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–.0	Anti-mis-trigger time= (AMTDATA*4)/fpwmclk + T_{ST}

NOTE: 0< T_{ST}（setting time）<4/fpwmclk

BTCON — Basic Timer Control Register D3H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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

.7–.4 **Watchdog Timer Function Enable Bit**

1	0	1	0	Disable watchdog timer function
Others				Enable watchdog timer function

.3–.2 **Basic Timer Input Clock Selection Code**

0	0	f_{OSC}/4096
0	1	f_{OSC}/1024
1	0	f_{OSC}/128
1	1	Invalid setting

.1 **Basic Timer 8-Bit Counter Clear Bit**

0	No effect
1	Clear the basic timer counter value

.0 **Basic Timer Divider Clear Bit**

0	No effect
1	Clear both dividers

NOTE: When you write a "1" to BTCON.0 (or BTCON.1), the basic timer divider (or basic timer counter) is cleared. The bit is then cleared automatically to "0".

BUZCON — BUZ control Register F7H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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-.6 BUZ Input Clock Selection Code

0	0	fosc/16
0	1	fosc/32
1	0	fosc/64
1	1	fosc/128

.5 BUZ enable bit

0	Disable BUZ
1	Enable BUZ

.4-.0 BUZ Frequency = f_{BUZ}/[(BUZCON.4-0)+1]*2

CLKCON — Clock Control Register D4H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	–	–	0	0	–	–	–
Read/Write	R/W	–	–	R/W	R/W	–	–	–

.7 **Oscillator IRQ Wake-up Function Enable Bit**

0	Enable IRQ for main system oscillator wake-up function
1	Disable IRQ for main system oscillator wake-up function

.6–.5 **Not used for S3F84B8**

.4–.3 **Divided by Selection Bits for CPU Clock frequency**

0	0	Divide by 16 (f_{OSC}/16)
0	1	Divide by 8 (f_{OSC}/8)
1	0	Divide by 2 (f_{OSC}/2)
1	1	Non-divided clock (f_{OSC})

.2–.0 **Not used for S3F84B8**

CMP0CON — Comparator0 Control Register EAH, BANK0

	.7	.6	.5	.4	.3	.2	.1	.0
Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	–	–	0	0	0	1	0
Read/Write	–	–	–	R/W	R/W	R/W	R	R/W

.7-.5 Not used for S3F84B8

.4 Comparator0 output polarity select bit (NOTE1)

0	CMP0 output is not inverted
1	CMP0 output is inverted

.3 Comparator0 enable bit (NOTE2)

0	Disable CMP0
1	Enable CMP0

.2 Comparator0 interrupt enable bit

0	Disable CMP0 interrupt
1	Enable CMP0 interrupt

.1 Comparator0 status bit

0	CMP0_N > CMP0_P
1	CMP0_N < CMP0_P

.0 Comparator0 pending bit

0	No interrupt pending (clear pending bit when write)
1	CMP0 interrupt is pending

NOTE:
1 Polarity selection bit (CMP0CON.4) will not affect interrupt generation logic.
2 Refer to Ch14 programming tip for proper configuration sequence

CMP1CON — Comparator1 Control Register EBH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	1	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W

.7-.5 Comparator 1 reference level selection bit

0	0	0	0.45VDD
0	0	1	0.50VDD
0	1	0	0.55VDD
0	1	1	0.60VDD
1	0	0	0.65VDD
1	0	1	0.70VDD
1	1	0	0.75VDD
1	1	1	0.80VDD

.4 Comparator1 output polarity select bit

0	CMP1 output is not inverted
1	CMP1 output is inverted

.3 Comparator1 enable bit

0	Disable CMP1
1	Enable CMP1

.2 Comparator1 interrupt enable bit

0	Disable CMP1 interrupt
1	Enable CMP1 interrupt

.1 Comparator1 status bit

0	CMP1_N > CMP1_P
1	CMP1_N < CMP1_P

.0 Comparator1 pending bit

0	No interrupt pending.(clear pending bit when write)
1	CMP1 interrupt is pending

NOTE:
1 Polarity selection bit (CMP1CON.4) will not affect interrupt generation logic.
2 Refer to Ch14 programming tip for proper configuration sequence.

CMP2CON — Comparator1 Control Register ECH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	1	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W

.7-.5 **Comparator 2 reference level selection bit**

0	0	0	0.45VDD
0	0	1	0.50VDD
0	1	0	0.55VDD
0	1	1	0.60VDD
1	0	0	0.65VDD
1	0	1	0.70VDD
1	1	0	0.75VDD
1	1	1	0.80VDD

.4 **Comparator2 output polarity select bit**

0	CMP2 output is not inverted
1	CMP2 output is inverted

.3 **Comparator2 enable bit**

0	Disable CMP1
1	Enable CMP1

.2 **Comparator2 interrupt enable bit**

0	Disable CMP1 interrupt
1	Enable CMP1 interrupt

.1 **Comparator2 status bit**

0	CMP2_N > CMP2_P
1	CMP2_N < CMP2_P

.0 **Comparator2 pending bit**

0	No interrupt pending.(clear pending bit when write)
1	CMP2 interrupt is pending

NOTE:
1 Polarity selection bit (CMP2CON.4) will not affect interrupt generation logic.
2 Refer to Ch14 programming tip for proper configuration sequence.

CMP3CON — Comparator1 Control Register

EDH, BANK0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	0	0	0	0	0	0	1	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W

.7-.5 **Comparator3 reference level selection bit**

0	0	0	0.45VDD
0	0	1	0.50VDD
0	1	0	0.55VDD
0	1	1	0.60VDD
1	0	0	0.65VDD
1	0	1	0.70VDD
1	1	0	0.75VDD
1	1	1	0.80VDD

.4 **Comparator3 output polarity select bit**

0	CMP3 output is not inverted
1	CMP3 output is inverted

.3 **Comparator3 enable bit**

0	Disable comparator3
1	Enable comparator3

.2 **Comparator3 interrupt enable bit**

0	Disable CMP3 interrupt
1	Enable CMP3 interrupt

.1 **Comparator3 status bit**

0	CMP3_N > CMP3_P
1	CMP3_N < CMP3_P

.0 **Comparator3 pending bit**

0	No interrupt pending.(clear pending bit when write)
1	CMP3 interrupt is pending

NOTE:
1 Polarity selection bit (CMP3CON.4) will not affect interrupt generation logic.
2 Refer to Ch14 programming tip for proper configuration sequence.

CMPINT— Comparator Interrupt Mode Control Register EEH, BANK0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	1	1	1	1	1	1	1	1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7-.6 **CMP3 Interrupt mode selection bit**

0	0	Invalid setting
0	1	Falling edge Interrupt
1	0	Rising edge Interrupt
1	1	Falling and rising edge Interrupt

.5-.4 **CMP2 Interrupt mode selection bit**

0	0	Invalid setting
0	1	Falling edge Interrupt
1	0	Rising edge Interrupt
1	1	Falling and rising edge Interrupt

.3-.2 **CMP1 Interrupt mode selection bit**

0	0	Invalid setting
0	1	Falling edge Interrupt
1	0	Rising edge Interrupt
1	1	Falling and rising edge Interrupt

.1-.0 **CMP0 Interrupt mode selection bit**

0	0	Invalid setting
0	1	Falling edge Interrupt
1	0	Rising edge Interrupt
1	1	Falling and rising edge Interrupt

NOTE: When CMP0/1/2/3 Interrupt is used, register CMPINT must be set to appropriate value before enabling CMP0/1/2/3.

FLAGS — System Flags Register D5H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	x	x	x	x	x	x	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Addressing Mode	Register addressing mode only							

.7 **Carry Flag (C)**

0	Operation does not generate a carry or borrow condition
1	Operation generates a carry-out or borrow into high-order bit 7

.6 **Zero Flag (Z)**

0	Operation result is a non-zero value
1	Operation result is zero

.5 **Sign Flag (S)**

0	Operation generates a positive number (MSB = "0")
1	Operation generates a negative number (MSB = "1")

.4 **Overflow Flag (V)**

0	Operation result is $\leq$ +127 and > –128
1	Operation result is > +127 or < –128

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

0	Add operation completed
1	Subtraction operation completed

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

0	No carry-out of bit 3 or no borrow into bit 3 by addition or subtraction
1	Addition generated carry-out of bit 3 or subtraction generated borrow into bit 3

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

0	Interrupt return (IRET) in progress (when read)
1	Fast interrupt service routine in progress (when read)

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

0	Bank 0 is selected
1	Bank 1 is selected

FMCON — Flash Memory Control Register F5H, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	0	0	0	0	0	–	–	0
Read/Write	R/W	R/W	R/W	R/W	R	–	–	R/W
Addressing Mode	Register addressing mode only							

.7–.4 **Flash Memory Mode Selection Bits**

0	1	0	1	Programming mode
1	0	1	0	Sector erase mode
0	1	1	0	Hard lock mode
Other values				Not available

.3 **Sector Erase Status Bit**

0	Success sector erase
1	Fail sector erase

.2–.1 Not used for the S3F84B8

.0 **Flash Operation Start Bit**

0	Operation stop
1	Operation start (This bit will be cleared automatically just after the corresponding operator completed).

FMSECH — Flash Memory Sector Address Register (High Byte) F7H, BANK1

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
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–.0	**Flash Memory Sector Address Bits (High Byte)**
	The 15th - 8th bits to select a sector of flash ROM

NOTE: The high-byte flash memory sector address pointer value is the higher eight bits of the 16-bit pointer address.

FMSECL — Flash Memory Sector Address Register (Low Byte) F8H, BANK1

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
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	**Flash Memory Sector Address Bit (Low Byte)**
	The 7th bit to select a sector of flash ROM
.6–.0	**Bits 6–0**
	Don't care

NOTE: The low-byte flash memory sector address pointer value is the lower eight bits of the 16-bit pointer address.

FMUSR — Flash Memory User Programming Enable Register F6H, BANK1

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
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–.0 **Flash Memory User Programming Enable Bits**

1 0 1 0 0 1 0 1	Enable user programming mode
Other values	Disable user programming mode

IMR — Interrupt Mask Register DDH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7 **Interrupt Level 7 (IRQ7)**

0	Disable (mask)
1	Enable (unmask)

.6 **Interrupt Level 6 (IRQ6)**

0	Disable (mask)
1	Enable (unmask)

.5 **Interrupt Level 5 (IRQ5)**

0	Disable (mask)
1	Enable (unmask)

.4 **Interrupt Level 4 (IRQ4)**

0	Disable (mask)
1	Enable (unmask)

.3 **Interrupt Level 3 (IRQ3)**

0	Disable (mask)
1	Enable (unmask)

.2 **Interrupt Level 2 (IRQ2)**

0	Disable (mask)
1	Enable (unmask)

.1 **Interrupt Level 1 (IRQ1)**

0	Disable (mask)
1	Enable (unmask)

.0 **Interrupt Level 0 (IRQ0)**

0	Disable (mask)
1	Enable (unmask)

NOTE: When an interrupt level is masked, the CPU does not recognize any interrupt requests that may be issued.

IPH — Instruction Pointer (High Byte) DAH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7–.0 **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) DBH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7–.0 **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 FFH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7, .4, and .1 **Priority Control Bits for Interrupt Groups A, B, and C** (NOTE)

0	0	0	Group priority undefined
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	Group priority undefined

.6 **Interrupt Subgroup C Priority Control Bit**

0	IRQ6 > IRQ7
1	IRQ7 > IRQ6

.5 **Interrupt Group C Priority Control Bit**

0	IRQ5 > (IRQ6, IRQ7)
1	(IRQ6, IRQ7) > IRQ5

.3 **Interrupt Subgroup B Priority Control Bit**

0	IRQ3 > IRQ4
1	IRQ4 > IRQ3

.2 **Interrupt Group B Priority Control Bit**

0	IRQ2 > (IRQ3, IRQ4)
1	(IRQ3, IRQ4) > IRQ2

.0 **Interrupt Group A Priority Control Bit**

0	IRQ0 > IRQ1
1	IRQ1 > IRQ0

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

IRQ — Interrupt Request Register DCH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R

.7 **Level 7 (IRQ7) Request Pending Bit;**

0	Not pending
1	Pending

.6 **Level 6 (IRQ6) Request Pending Bit;**

0	Not pending
1	Pending

.5 **Level 5 (IRQ5) Request Pending Bit;**

0	Not pending
1	Pending

.4 **Level 4 (IRQ4) Request Pending Bit;**

0	Not pending
1	Pending

.3 **Level 3 (IRQ3) Request Pending Bit;**

0	Not pending
1	Pending

.2 **Level 2 (IRQ2) Request Pending Bit;**

0	Not pending
1	Pending

.1 **Level 1 (IRQ1) Request Pending Bit;**

0	Not pending
1	Pending

.0 **Level 0 (IRQ0) Request Pending Bit;**

0	Not pending
1	Pending

OPACON — OP AMP Control Register E0H, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	–	–	–	–	–	0	0
Read/Write	–	–	–	–	–	–	R/W	R/W

.7–.2 **Not used for S3F84B8**

.1 **OP AMP Mode Select Bit**

0	Off chip mode (external positive input)
1	On chip mode (Internal Ground level positive input)

.0 **OP AMP Enable Bit**

0	Disable OP AMP
1	Enable OP AMP

P0CONH — Port 0 Control Register (High Byte) E4H, Bank0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	–	0	0	0	0	0	0
Read/Write	–	–	R/W	R/W	R/W	R/W	R/W	R/W

.7–.6 **Not used for S3F84B8**

.5–.4 **Port 0, P0.6/INT5/TAOUT Configuration Bits**

0	0	Input mode/INT5 falling edge interrupt
0	1	Input mode with pull-up /INT5 falling edge interrupt
1	0	Push-pull output
1	1	Alternative function: TAOUT

.3–.2 **Port 0, P0.5/INT4 Configuration Bits**

0	0	Input mode/INT4 falling edge interrupt
0	1	Input mode with pull-up /INT4 falling edge interrupt
1	0	Push-pull output
1	1	Open-drain output

.1-.0 **Port 0, P0.4/INT3/PWM Configuration Bits**

0	0	Input mode/INT3 falling edge interrupt
0	1	Input mode with pull-up /INT3 falling edge interrupt
1	0	Push-pull output
1	1	Alternative function: PWM output

P0CONL — Port 0 Control Register (Low Byte) E5H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	–	–	0	0	0	0
Read/Write	R/W	R/W	–	–	R/W	R/W	R/W	R/W

7–.6 **Port 0, P0.3/INT2/BUZ Configuration Bits**

0	0	Input mode/INT2 falling edge interrupt
0	1	Input mode with pull-up /INT2 falling edge interrupt
1	0	Push-pull output
1	1	Alternative function: BUZ

.5–.4 **Not used for S3F84B8**

.3–.2 **Port 0, P0.1/INT1 Configuration Bits**

0	0	Input mode/INT1 falling edge interrupt
0	1	Input mode with pull-up /INT1 falling edge interrupt
1	0	Push-pull output
1	1	Open-drain output

.1–.0 **Port 0, P0.0/INT0 Configuration Bits**

0	0	Input mode/INT0 falling edge interrupt
0	1	Input mode with pull-up /INT0 falling edge interrupt
1	0	Push-pull output
1	1	Open-drain output

P0INT — Port 0 Interrupt Control Register E3H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	0	0	0	0	–	0	0
Read/Write	–	R/W	R/W	R/W	R/W	–	R/W	R/W

.7 **Not used for S3F84B8**

.6 **P0.6/ INT5 Interrupt Enable/Disable Selection Bits**

0	Interrupt Disable
1	Interrupt Enable

.5 **P0.5/ INT4 Interrupt Enable/Disable Selection Bits**

0	Interrupt Disable
1	Interrupt Enable

.4 **P0.4/ INT3 Interrupt Enable/Disable Selection Bits**

0	Interrupt Disable
1	Interrupt Enable

.3 **P0.3/ INT2 Interrupt Enable/Disable Selection Bits**

0	Interrupt Disable
1	Interrupt Enable

.2 **Not used for S3F84B8**

.1 **P0.1/ INT1 Interrupt Enable/Disable Selection Bits**

0	Interrupt Disable
1	Interrupt Enable

.0 **P0.0 / INT0 Interrupt Enable/Disable Selection Bits**

0	Interrupt Disable
1	Interrupt Enable

P0PND — Port 0 Interrupt Pending Register E6H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	0	0	0	0	–	0	0
Read/Write	–	R/W	R/W	R/W	R/W	–	R/W	R/W

.7 **Not used for S3F84B8**

.6 **Port 0.6/INT5 Interrupt Pending Bit**

0	No interrupt pending (when read); Pending bit clear (when write)
1	Interrupt is pending (when read); No effect (when write)

.5 **Port 0.5/INT4 Interrupt Pending Bit**

0	No interrupt pending (when read); Pending bit clear (when write)
1	Interrupt is pending (when read); No effect (when write)

.4 **Port 0.4/INT3 Interrupt Pending Bit**

0	No interrupt pending (when read); Pending bit clear (when write)
1	Interrupt is pending (when read); No effect (when write)

.3 **Port 0.3/INT2 Interrupt Pending Bit**

0	No interrupt pending (when read); Pending bit clear (when write)
1	Interrupt is pending (when read); No effect (when write)

.2 **Not used for S3F84B8**

.1 **Port 0.1/INT1 Interrupt Pending Bit**

0	No interrupt pending (when read); Pending bit clear (when write)
1	Interrupt is pending (when read); No effect (when write)

.0 **Port 0.0/INT0 Interrupt Pending Bit**

0	No interrupt pending (when read); Pending bit clear (when write)
1	Interrupt is pending (when read); No effect (when write)

P1CON — Port 1 Control Register E7H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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

.7–.6 **Not used for S3F84B8**

.5–.4 **Port 1, P1.2/CMP1_N Configuration Bits**

0	0	Schmitt trigger input;
0	1	Schmitt trigger input; pull-up enable;
1	0	Push pull output
1	1	Alternative function: comparator 1 negative input

.3–.2 **Port 1, P1.1/CMP0_N/TACAP Configuration Bits**

0	0	Schmitt trigger input; TACAP input
0	1	Schmitt trigger input; pull-up enable; TACAP input
1	0	Push pull output
1	1	Alternative function: comparator 0 negative input

.1–.0 **Port 1, P1.0/CMP0_P/TACK Configuration Bits**

0	0	Schmitt trigger input; TACK input
0	1	Schmitt trigger input; pull-up enable; TACK input
1	0	Push pull Output
1	1	Alternative function: comparator 0 positive input

P2CONH — Port 2 Control Register (High Byte) E8H, BANK0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
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

.7–.6 **Port2, P2.7/ADC7 Configuration Bits**

0	0	Schmitt trigger input;
0	1	Schmitt trigger input; pull-up enable
1	0	Push pull output
1	1	Alternative function: ADC7 input

.5–.4 **Port 2, P2.6/ADC6 Configuration Bits**

0	0	Schmitt trigger input;
0	1	Schmitt trigger input; pull-up enable
1	0	Push pull output
1	1	Alternative function: ADC6 input

.3–.2 **Port 2, P2.5/ADC5/CMP3_N Configuration Bits**

0	0	Schmitt trigger input;
0	1	Alternative function: Comparator 3 negative input
1	0	Push pull output
1	1	Alternative function: ADC5 input

.1–.0 **Port 2, P2.4/ADC4/CMP2_N Configuration Bits**

0	0	Schmitt trigger input;
0	1	Alternative function: Comparator 2 negative input
1	0	Push pull output
1	1	Alternative function: ADC4 input

P2CONL — Port 2 Control Register (Low Byte) E9H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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

.7–.6 **Part 2, P2.3/ADC3/(OA_O) Configuration Bits**

0	0	Schmitt trigger input;
0	1	Schmitt trigger input; pull-up enable
1	0	Push-pull output
1	1	Alternative function: ADC3 input *

.5–.4 **Port 2, P2.2/ADC2/OA_N Configuration Bits**

0	0	Schmitt trigger input
0	1	Alternative function: OPAMP negative input
1	0	Push-pull output
1	1	Alternative function: ADC2 input

.3–.2 **Port 2, P2.1/ADC1/OP_P Configuration Bits**

0	0	Schmitt trigger input
0	1	Alternative function: OPAMP positive input
1	0	Push-pull output
1	1	Alternative function: ADC1 input

.1–.0 **Port 2, P2.0/ADC0/TDOUT Configuration Bits**

0	0	Schmitt trigger input
0	1	Alternative function: TDOUT
1	0	Push-pull output
1	1	Alternative function: ADC0 input

NOTE: when OP AMP is enabled, P2CON.3 must be configured as ADC input no matter you want to use internal ADC module or not.

PWMCON — PWM Control Register EFH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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

.7–.6 **PWM Input Clock Select Bits**

0	0	$f_{OSC}/64$
0	1	$f_{OSC}/8$
1	0	$f_{OSC}/2$
1	1	$f_{OSC}/1$

.5 **PWM Output Polarity Select Bit**

0	Non-inverting output
1	Inverting output

.4 **PWM Counter Clear Bit**

0	No effect
1	Clear the PWM counter (when write)

.3 **PWM Counter Enable Bit**

0	Stop counter
1	Start counter (unlock operation)

.2 **Anti-Mis-Trigger Enable Bit**

0	Disable anti-mis-trigger function
1	Enable anti-mis-trigger function

.1 **PWM Overflow Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.0 **PWM Overflow Interrupt Pending Bit**

0	No interrupt pending, *Clear pending bit (when write)*
1	Interrupt is pending, *No effect (when write)*

NOTE: To use anti-mis-trigger function, user must enable the linkage of CMP0 and PWM by setting PWMCCON.0 = 1

PWMCCON — PWM CMP Control Register F0H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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

.7-.6 **CMP3 PWM linkage mode selection bits**

X	0	Disable linkage
0	1	Soft Lock
1	1	Hard lock

.5-.4 **CMP2 PWM linkage mode selection bit**

X	0	Disable linkage
0	1	Soft Lock
1	1	Hard lock

.3-.2 **CMP1 PWM Lock mode selection bit**

X	0	Disable linkage
0	1	Soft Lock
1	1	Hard lock

.1-.0 **CMP0 PWM Trigger mode selection bit**

X	0	Disable linkage
0	1	Normal trigger
1	1	Delay trigger

NOTE: When CMP-PWM linkage is used, PWMCCON must be set to appropriate value before enabling PWM.

PWMDL — Comparator0 Output Delay Register F5H, Bank0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	–	–	–	–	0	0	0	0
Read/Write	–	–	–	–	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.4	**Not used for S3F84B8**
.3–.0	**Delay Time= (PWMDL+1)*4/fpwmclk + T_{ST}**

NOTE: 0< T_{ST}（setting time）<4/fpwmclk

PP — Register Page Pointer DFH, BANK0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
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

.7–.0	**Not used for the S3F84B8.**

NOTE: In S3F84B8, only page 0 settings are valid. Register page pointer values for the source and destination register page are automatically set to '00F' following a hardware reset. These values should not be changed during normal operation.

RESETID — Reset Source Indicating Register F2H, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Read/Write	–	–	–	R/W	–	R/W	R/W	–

Addressing Mode Register addressing mode only

.7 – .5 **Not used for S3F84B8**

.4 **nReset pin Indicating Bit**

0	Reset is not generated by nReset pin (when read)
1	Reset is generated by nReset pin (when read)

.3 **Not used for S3F84B8**

.2 **WDT Reset Indicating Bit**

0	Reset is not generated by WDT (when read)
1	Reset is generated by WDT (when read)

.1 **LVR Reset Indicating Bit**

0	Reset is not generated by LVR (when read)
1	Reset is generated by LVR (when read)

.0 **Not used for S3F84B8**

State of RESETID depends on reset source

	.7	.6	.5	.4	.3	.2	.1	.0
LVR	–	–	–	0	–	0	1	–
WDT, or nReset pin	–	–	–	NOTE4	–	NOTE4	NOTE3	–

NOTES:

1. When LVR is disabled (Smart Option 3FH.7 = 0), RESETID.1 is invalid; when P0.2 is set to be IO (Smart Option 3FH.2 = 0), RESETID.4 is invalid.
2. To clear an indicating register, write "0" to indicating flag bit (writing "1" to reset indicating bits has no effect).
3. Once a LVR reset happens, RESETID.1 will be set and all the other bits will be cleared to "0" at the same time.
4. Once a WDT or nRESET pin reset happens, corresponding bit will be set, but leave all other indicating bits unchanged.

RP0 — Register Pointer 0 D6H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	1	1	0	0	0	–	–	–
Read/Write	R/W	R/W	R/W	R/W	R/W	–	–	–

.7–.3 **Register Pointer 0 Address Value**

Register pointer 0 can independently point to one of the 208-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, selecting the 8-byte working register slice C0H–C7H.

.2–.0 Not used for the S3F84B8

RP1 — Register Pointer 1 D7H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	1	1	0	0	1	–	–	–
Read/Write	R/W	R/W	R/W	R/W	R/W	–	–	–

.7 – .3 **Register Pointer 1 Address Value**

Register pointer 1 can independently point to one of the 208-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, selecting the 8-byte working register slice C8H–CFH.

.2 – .0 Not used for the S3F84B8

SPL — Stack Pointer D9H, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	x	x	x	x	x	x	x	X
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7–.0 **Stack Pointer Address (Low Byte)**

The SP value is undefined following a reset.

STOPCON — STOP Mode Control Register F4H, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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

.7–.0 **Watchdog Timer Function Enable Bit**

10100101	Enable STOP instruction
Other value	Disable STOP instruction

NOTES:
1. Before execute the STOP instruction, set this STPCON register as "10100101b".
2. When STOPCON register is not #0A5H value, if you use STOP instruction, PC is changed to reset address.

SYM — System Mode Register DEH, BANK0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	0	–	–	x	x	x	0	0
Read/Write	R/W	–	–	R/W	R/W	R/W	R/W	R/W

.7 **Tri-state External Interface Control Bit** (1)

0	Normal operation (disable tri-state operation)
1	Set external interface lines to high impedance (enable tri-state operation)

.6–.5 Not used for the S3F84B8

.4–.2 **Fast Interrupt Level Selection Bits** (2)

0	0	0	IRQ0
0	0	1	IRQ1
0	1	0	IRQ2
0	1	1	IRQ3
1	0	0	IRQ4
1	0	1	IRQ5
1	1	0	IRQ6
1	1	1	IRQ7

.1 **Fast Interrupt Enable Bit** (3)

0	Disable fast interrupt processing
1	Enable fast interrupt processing

.0 **Global Interrupt Enable Bit** (4)

0	Disable all interrupt processing
1	Enable all interrupt processing

NOTES:
1. Because an external interface is not implemented, 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. Following a reset, you must enable global interrupt processing by executing an EI instruction (not by writing a "1" to SYM.0).

TACON — Timer A Control Register E1H, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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

.7-.6 **Timer A Operating Mode Selection Bits**

0	0	Internal mode (TAOUT mode)
0	1	Capture mode (capture on rising edge, counter running, OVF can occur)
1	0	Capture mode (capture on falling edge, counter running, OVF can occur)
1	1	PWM mode (OVF interrupt can occur)

.5 **Timer A Counter Clear Bit**

0	No effect
1	Clear the timer A counter (After clearing, return to zero)

.4 **Timer A Start/Stop Bit**

0	Stop Timer A
1	Start Timer A

.3 **Timer A Match/Capture Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.2 **Timer A Overflow Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.1 **Timer A Match Interrupt pending Bit**

0	No interrupt pending (*Clear pending bit when write)*
1	Interrupt pending

.0 **Timer A Overflow Interrupt pending Bit**

0	No interrupt pending (*Clear pending bit when write)*
1	Interrupt pending

TAPS —TA Pre-scalar Register

E2H, BANK1

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	0	–	–	–	0	0	0	0
Read/Write	R/W	–	–	–	R/W	R/W	R/W	R/W

.7 **Timer A clock source selection**

0	Internal clock source
1	External clock source from TACK

.6-.5 Not used for S3F84B8

.3–.0 **Timer A pre-scalar bits**

TAPS = TA clock/ ($2^{TAPS[3-0]}$) Pre-scalar values above 12 are invalid

TCCON — Timer C Control Register E5H, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	–	0	0	0	–	0	–
Read/Write	R/W	–	R/W	R/W	R/W	–	R/W	–

.7 **Timer 0 operation mode selection bit**

0	Two 8-bit timers mode (Timer C/D)
1	One 16-bit timer mode (Timer 0)

.6 **Not used for S3F84B8**

.5 **Timer C Counter Clear Bit**

0	No effect
1	Clear the timer C counter (After clearing, return to zero)

.4 **Timer C Start/Stop Bit**

0	Stop Timer C
1	Start Timer C

.3 **Timer C Match Interrupt Enable Bit**

0	Disable Interrupt
1	Enable Interrupt

.2 **Not used for S3F84B8**

.1 **Timer C Match Interrupt Pending Bit**

0	No interrupt pending (*Clear pending bit when write)*
1	Interrupt pending

.0 **Not used for S3F84B8**

TCPS —TC Pre-scalar Register E6H, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	–	–	–	0	0	0	0
Read/Write	R/W	–	–	–	R/W	R/W	R/W	R/W

.7 **Timer C clock source selection**

0	Internal clock source
1	CMP0 output

.6-.4 **Not used for S3F84B8**

.3-.0 **Timer C pre-scalar bits**

TC CLK = TC CLK/(2^{TCPS})	Pre-scalar values above 12 are invalid

NOTE: when Timer 0 is working in one 16-bit timer mode, the clock is determined by TCPS.

SAMSUNG ELECTRONICS

TDCON — Timer D Control Register E9H, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
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

.7–.6 **Timer D operating Mode Selection Bits**

0	0	Interval mode
0	1	6-bit PWM mode (OVF interrupt can occur)
1	0	7-bit PWM mode (OVF interrupt can occur)
1	1	8-bit PWM mode (OVF interrupt can occur)

5 **Timer D Counter Clear Bit**

0	No effect
1	Clear the timer D counter (when write)

.4 **Timer D Start/Stop Bit**

0	Stop Timer D
1	Start Timer D

.3 **Timer D Match Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.2 **Timer D Overflow interrupt enable bit**

0	Disable interrupt
1	Enable interrupt

.1 **Timer D Match Interrupt pending Bit**

0	No interrupt pending (*Clear pending bit when write)*
1	Interrupt pending

.0 **Timer D Overflow Interrupt pending Bit**

0	No interrupt pending (*Clear pending bit when write)*
1	Interrupt pending

TDPS —TD Pre-scalar Register EAH, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	–	–	–	0	0	0	0
Read/Write	–	–	–	–	R/W	R/W	R/W	R/W

.7-.4 **Not used for S3F84B8**

.3–.0 **Timer D pre-scalar bits**

TD CLK = TD CLK / ($2^{TDPS[.3-.0]}$)	Pre-scalar values above 12 are invalid

5 INTERRUPT STRUCTURE

OVERVIEW

The S3C8/S3F8-series interrupt structure has three basic components: levels, vectors, and sources. The SAM8RC CPU recognizes up to 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. A vector address can be assigned to 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 possible 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 S3F84B8 interrupt structure recognizes eight interrupt levels.

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 let 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/S3F8-series devices is always much smaller). If an interrupt level has more than one vector address, the vector priorities are set in hardware. S3F84B8 uses 17 vectors.

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 S3F84B8 interrupt structure there are 17 possible interrupt sources, that means every source has its own vector.

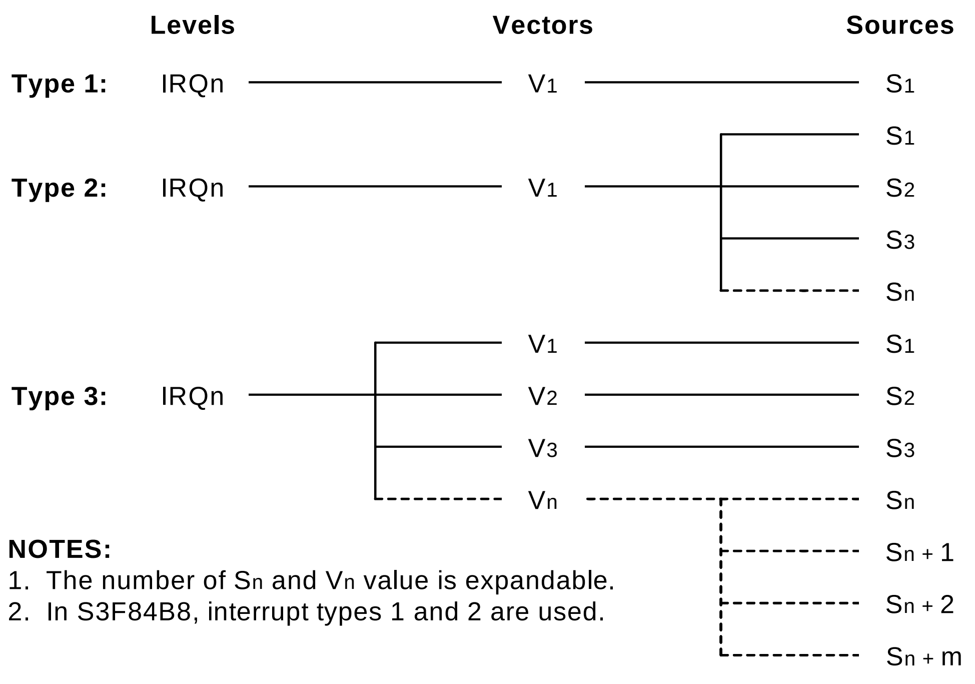
When a service routine starts, the respective pending bit should be either cleared automatically by hardware or cleared "manually" by software. The characteristics of the source's pending mechanism determine which method would be used to clear its respective pending bit.

INTERRUPT TYPES

The three components of the S3C8/S3F8 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 (V_1) + one source (S_1)
Type 2:	One level (IRQn) + one vector (V_1) + multiple sources ($S_1 – S_n$)
Type 3:	One level (IRQn) + multiple vectors ($V_1 – V_n$) + multiple sources ($S_1 – S_n$, $S_{n+1} – S_{n+m}$)

In the S3F84B8 microcontroller, two interrupt types are implemented.



NOTES:
1. The number of Sn and Vn value is expandable.
2. In S3F84B8, interrupt types 1 and 2 are used.

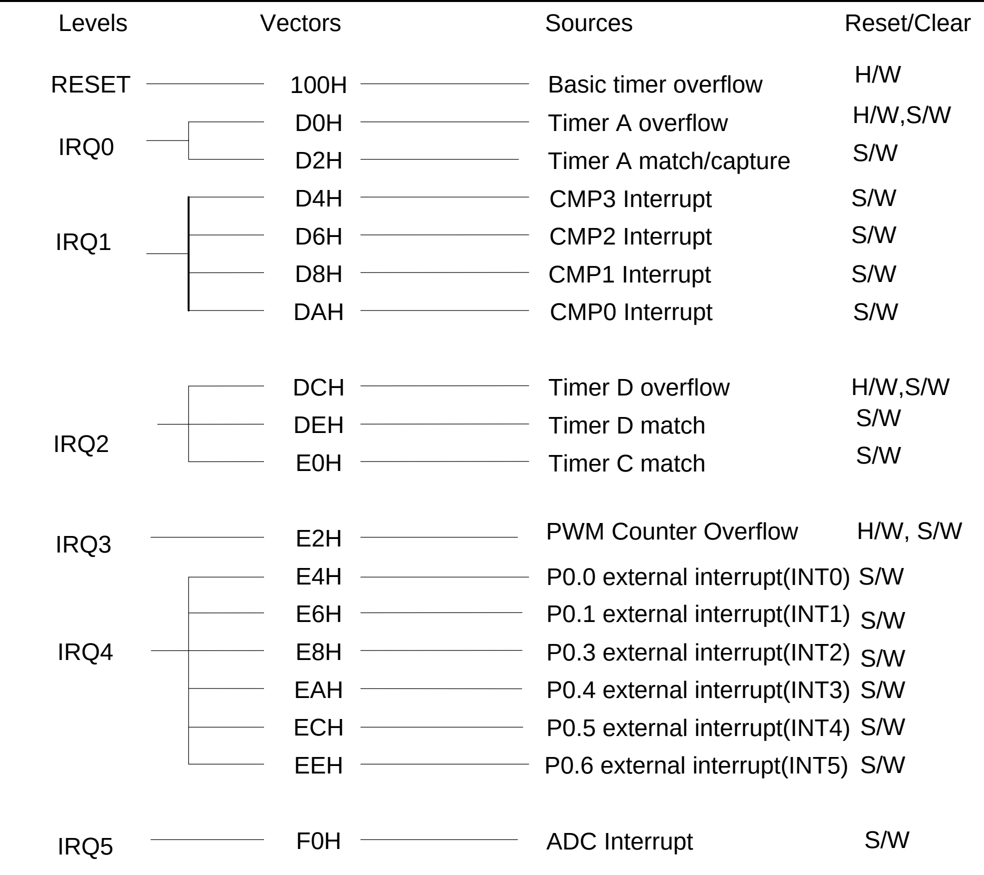
Figure 5-1. S3C8/S3F8-Series Interrupt Types

S3F84B8 INTERRUPT STRUCTURE

The S3F84B8 microcontroller supports 17 interrupt sources. Every interrupt source has a corresponding interrupt address. Eight interrupt levels are recognized by the CPU in this device-specific interrupt structure, as 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.



NOTES:
1. Within a given interrupt level, the low vector address has high priority.
   For example, D0H has higher priority than D2H within the level IRQ0. The priorities within each level are set at the factory.

Figure 5-2. S3F84B8 Interrupt Structure

Interrupt Vector Addresses

All interrupt vector addresses for the S3F84B8 interrupt structure is stored in the vector address area of the first 256 bytes of the program memory (ROM).

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.

The default program reset address in the ROM is 0100H.

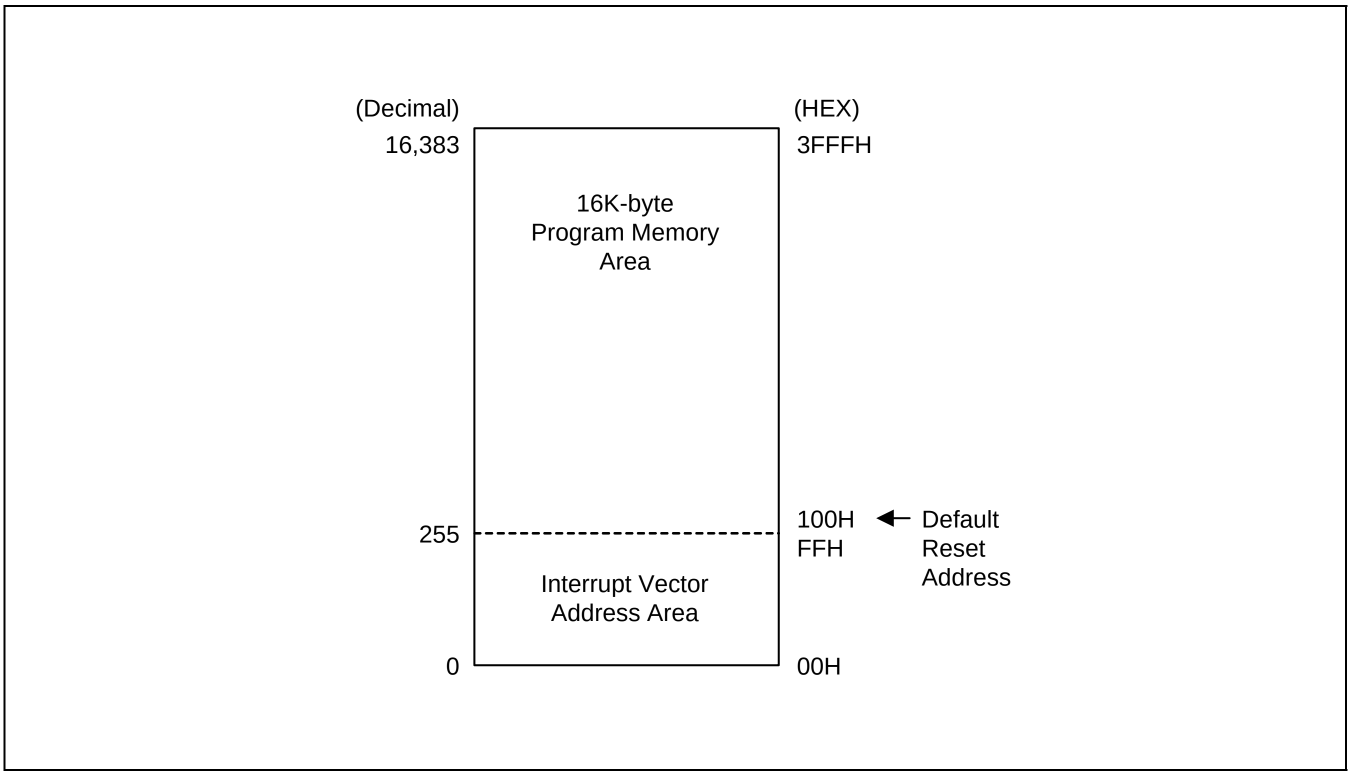


Figure 5-3. ROM Vector Address Area

Enable/Disable Interrupt Instructions (EI, DI)

Executing the Enable Interrupts (EI) instruction globally 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 to globally enable the interrupt structure.

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.

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 5-1. Interrupt Control Register Overview

Control Register	ID	R/W	Function Description
Interrupt mask register	IMR	R/W	Bit settings in the IMR register enable or disable interrupt processing for each of the eight interrupt levels: IRQ0–IRQ7.
Interrupt priority register	IPR	R/W	Controls the relative processing priorities of the interrupt levels. The eight levels of S3F84B8 are organized into three groups: A, B, and C. Group A is IRQ0 and IRQ1, group B is IRQ2, IRQ3 and IRQ4, and group C is IRQ5, IRQ6, and IRQ7.
Interrupt request register	IRQ	R	This register contains a request pending bit for each interrupt level.
System mode register	SYM	R/W	This register enables/disables fast interrupt processing, and dynamic global interrupt processing.

NOTE: All interrupts must be disabled before IMR register is changed to any value. Using DI instruction is recommended.

INTERRUPT PROCESSING CONTROL POINTS

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

— Global interrupt enable and disable (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.

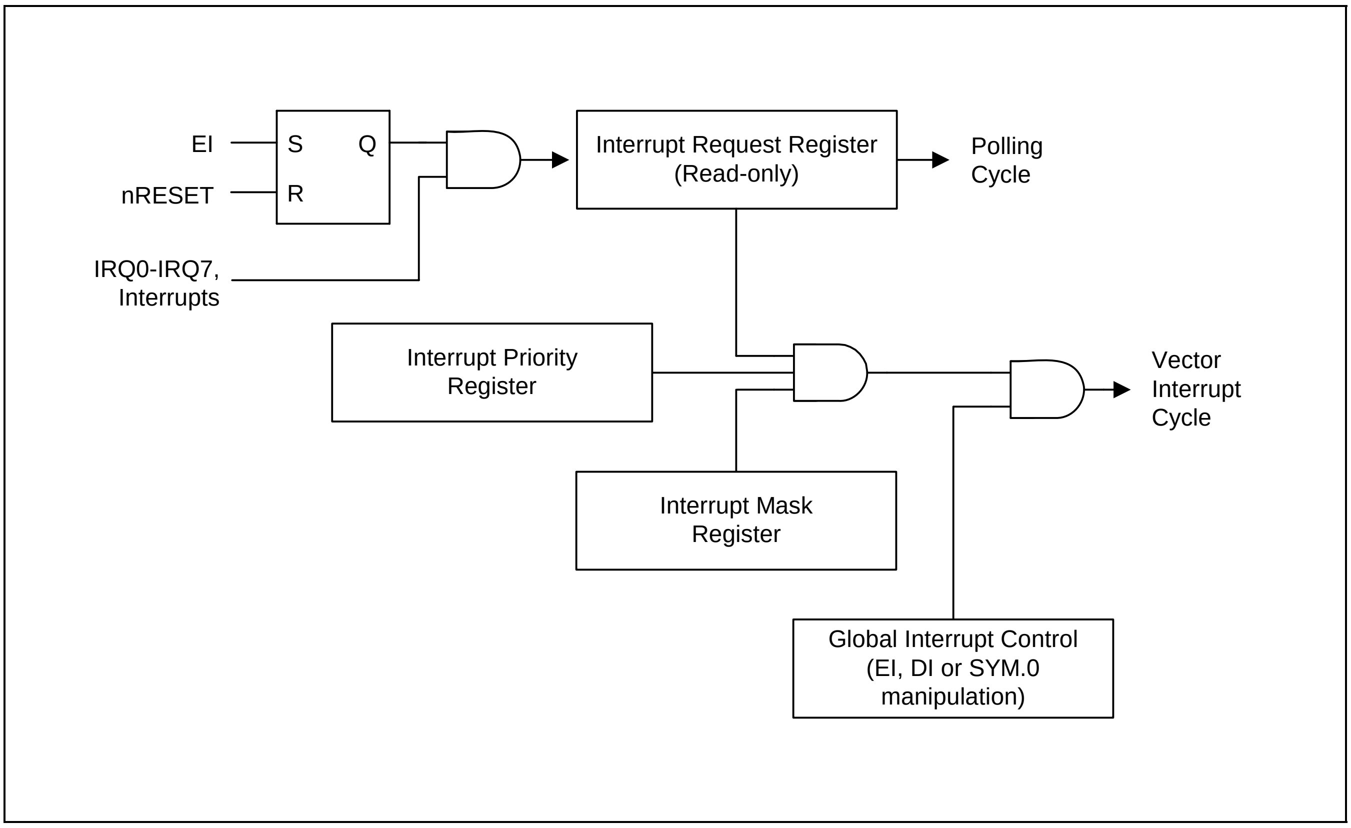


Figure 5-4. Interrupt Function Diagram

PERIPHERAL INTERRUPT CONTROL REGISTERS

For each interrupt source there is one or more corresponding peripheral control registers that let you control the interrupt generated by the related peripheral (see Table 5-2).

Table 5-2. Interrupt Source Control and Data Registers

Interrupt Source	Interrupt Level	Register(s)	Location(s)
Timer A overflow Timer A match/capture	IRQ0	TACON TAPS TADATA TACNT	E1H, BANK1 E2H, BANK1 E3H, BANK1 E4H, BANK1
CMP3 Interrupt CMP2 Interrupt CMP1 Interrupt CMP0 Interrupt	IRQ1	CMP3CON CMP2CON CMP1CON CMP0CON CMPINT	EDH, BANK0 ECH, BANK0 EBH, BANK0 FAH, BANK0 EEH, BANK0
Timer D overflow Timer D match Timer C match	IRQ2	TDCON TDPS TDDATA TDCNT	E9H, BANK1 EAH, BANK1 EBH, BANK1 ECH, BANK1
PWM overflow interrupt	IRQ3	PWMCON PWMCCON PWMDL PWMPDATAH/L PWMDATAH/L AMTDATA	EFH, BANK0 F0H, BANK0 F1H, BANK0 F2H/F3H, BANK0 F4H/F5H, BANK0 F6H, BANK0
P0.0 external interrupt P0.1 external interrupt P0.3 external interrupt P0.4 external interrupt P0.5 external interrupt P0.6 external interrupt	IRQ5	P0INT P0CONH/L P0PND	E3H, BANK0 E4H/E5H, BANK0 E6H, BANK0
ADC Interrupt	IRQ6	ADCDATAH/L ADCON	F8H/F9H, BANK0 FAH, BANK0

NOTE: If an interrupt is un-masked(Enable interrupt level) in the IMR register, a DI instruction should be executed before clearing the pending bit or changing the enable bit of the corresponding interrupt.

SYSTEM MODE REGISTER (SYM)

The system mode register, SYM (DEH, Set1), is used to globally enable and disable interrupt processing and to control fast interrupt processing (see Figure 5-5).

A reset clears SYM.1, and SYM.0 to "0". The 3-bit value for fast interrupt level selection, SYM.4–SYM.2, is 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.

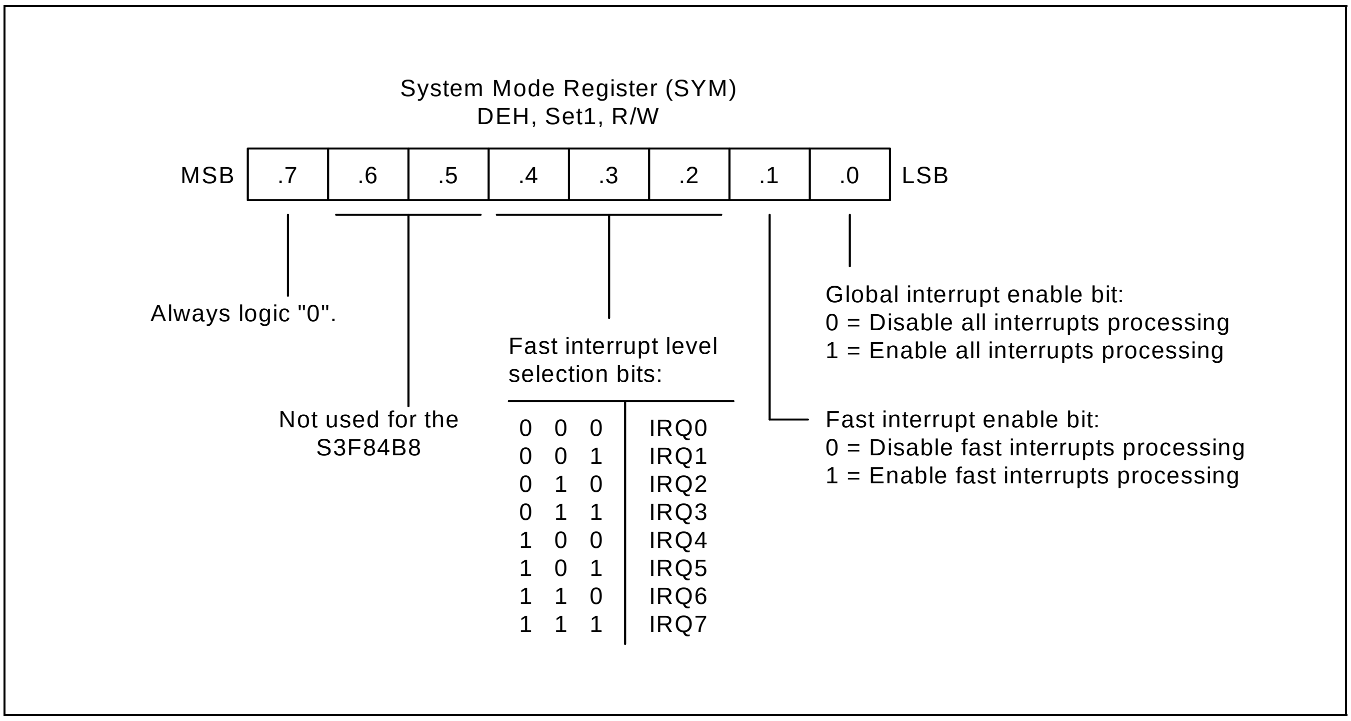

System Mode Register (SYM)
DEH, Set1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Always logic "0".
Not used for the S3F84B8
Fast interrupt level selection bits:
0 0 0 IRQ0
0 0 1 IRQ1
0 1 0 IRQ2
0 1 1 IRQ3
1 0 0 IRQ4
1 0 1 IRQ5
1 1 0 IRQ6
1 1 1 IRQ7
Global interrupt enable bit:
0 = Disable all interrupts processing
1 = Enable all interrupts processing
Fast interrupt enable bit:
0 = Disable fast interrupts processing
1 = Enable fast interrupts processing

Figure 5-5. System Mode Register (SYM)

INTERRUPT MASK REGISTER (IMR)

The interrupt mask register, IMR (DDH, Set1) 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, Set1. Bit values can be read and written by instructions using the Register addressing mode.

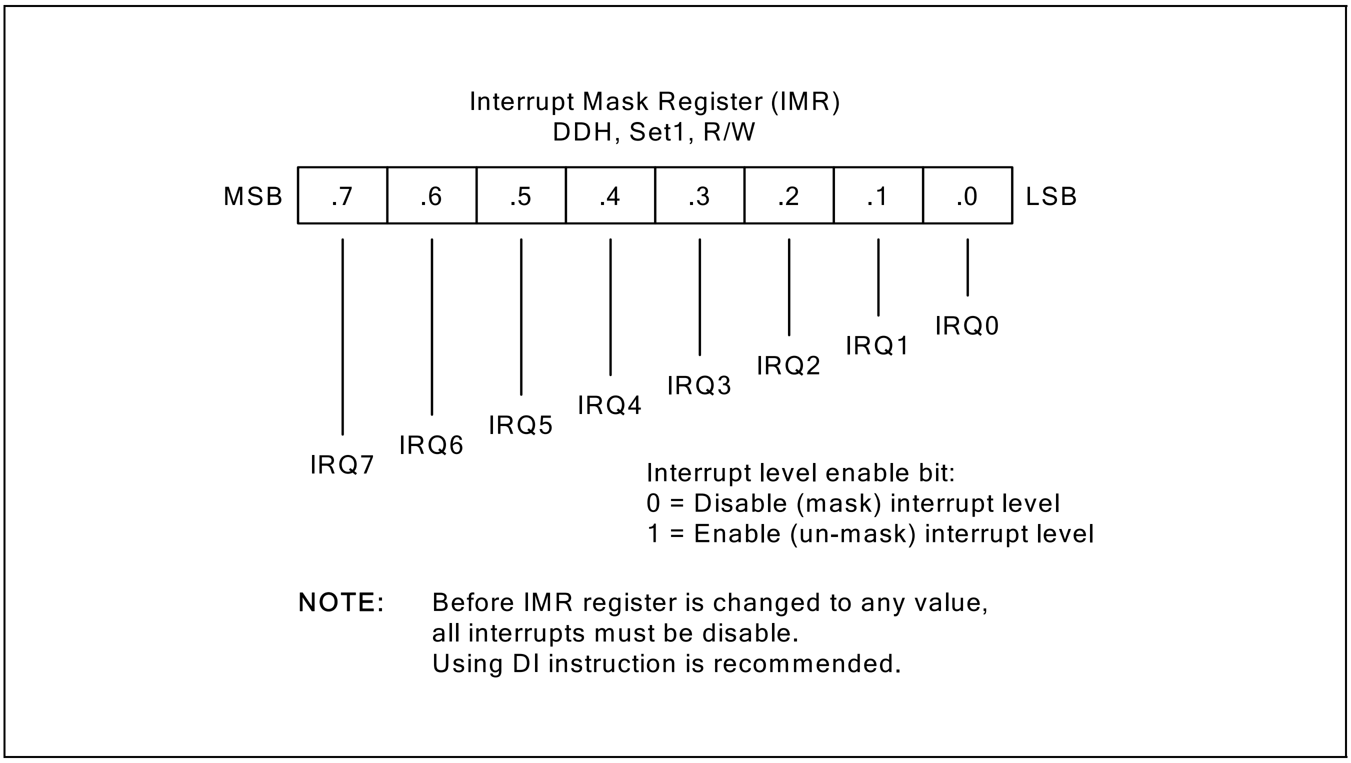

Interrupt Mask Register (IMR)
DDH, Set1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
IRQ0
IRQ1
IRQ2
IRQ3
IRQ4
IRQ5
IRQ6
IRQ7
Interrupt level enable bit:
0 = Disable (mask) interrupt level
1 = Enable (un-mask) interrupt level
NOTE: Before IMR register is changed to any value, all interrupts must be disable. Using DI instruction is recommended.

Figure 5-6. Interrupt Mask Register (IMR)

INTERRUPT PRIORITY REGISTER (IPR)

The interrupt priority register, IPR (FFH, Set1, Bank0), 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 level is serviced first. If two sources belong to the same interrupt level, the source with the lower vector address usually has the 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

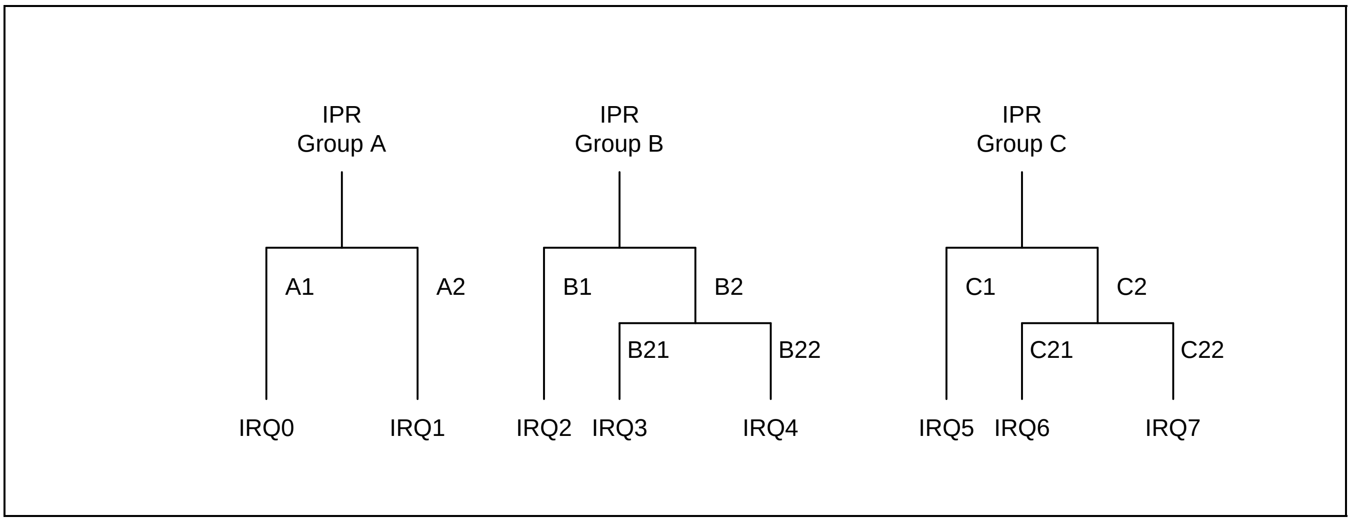


Figure 5-7. Interrupt Request Priority Groups

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:

— IPR.5 controls the relative priorities of group C interrupts.

— Interrupt group C includes a subgroup that has an additional priority relationship among the interrupt levels 5, 6, and 7. IPR.6 defines the subgroup C relationship. IPR.5 controls the interrupt group C.

— IPR.0 controls the relative priority setting of IRQ0 and IRQ1 interrupts.

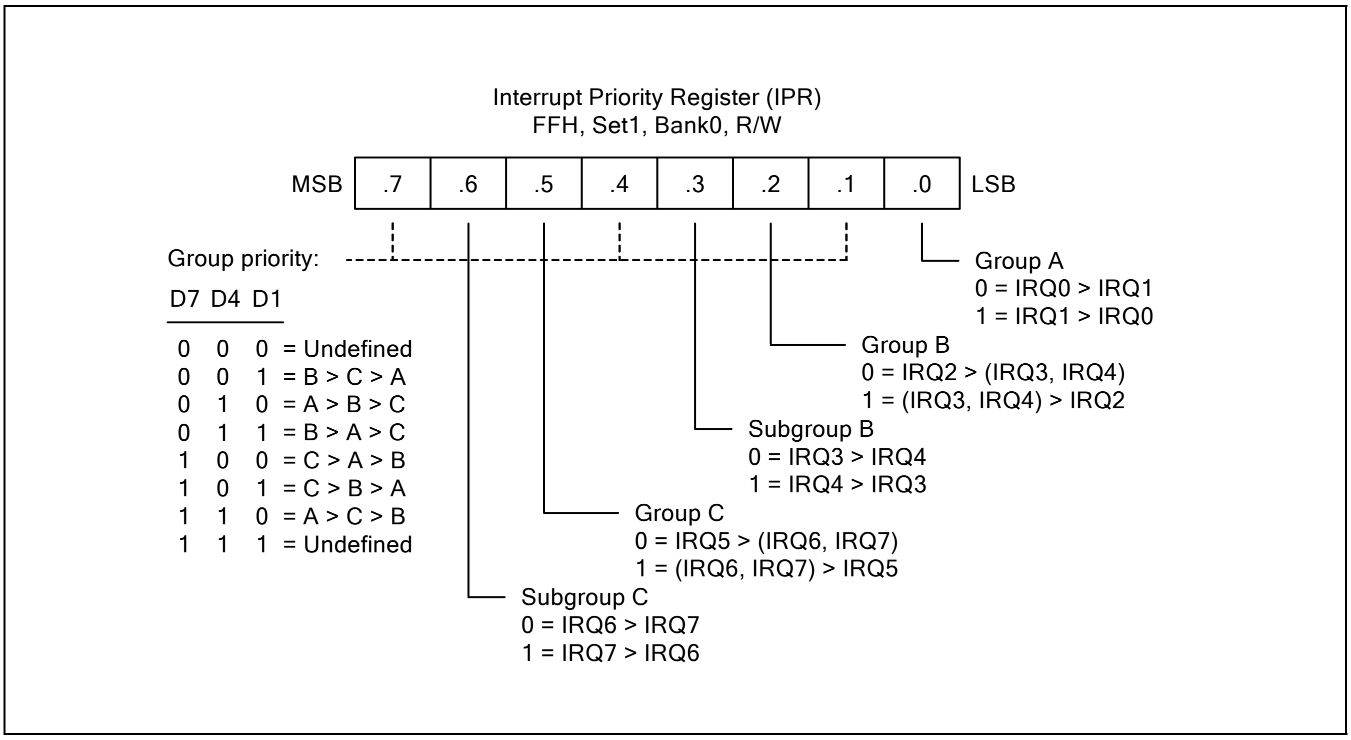


Figure 5-8. Interrupt Priority Register (IPR)

INTERRUPT REQUEST REGISTER (IRQ)

You can poll bit values in the interrupt request register, IRQ (DCH, Set1), 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 read-only addressable 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.

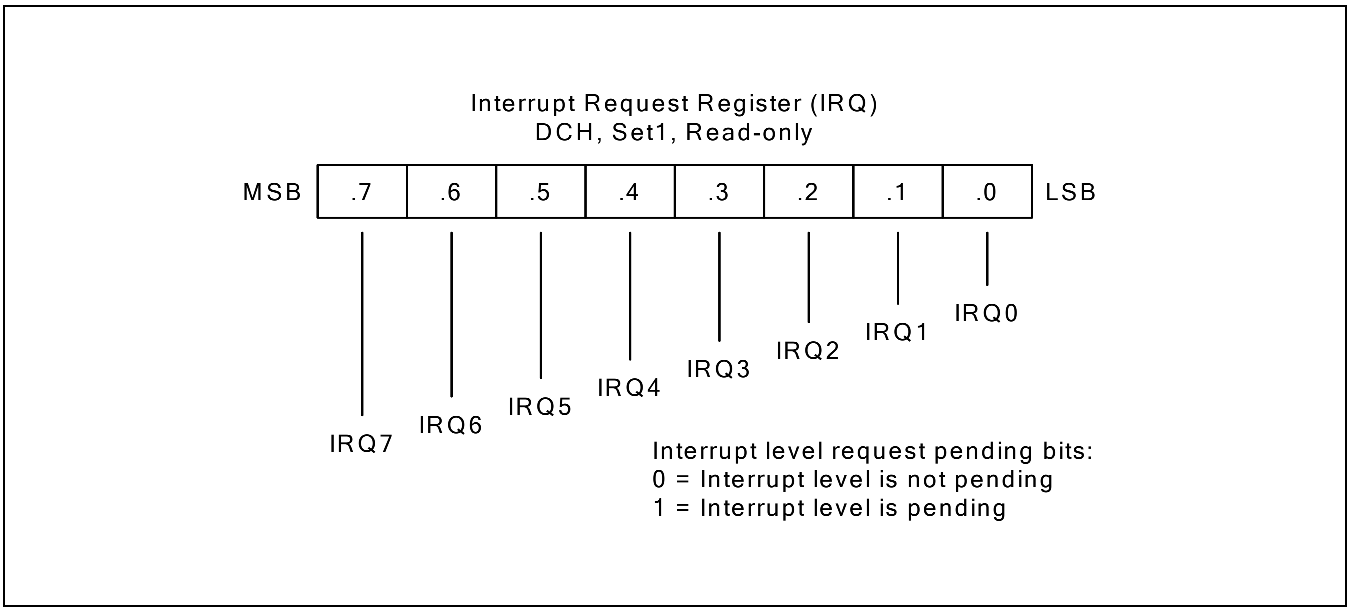

Interrupt Request Register (IRQ)
DCH, Set1, Read-only
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
IRQ0
IRQ1
IRQ2
IRQ3
IRQ4
IRQ5
IRQ6
IRQ7
Interrupt level request pending bits:
0 = Interrupt level is not pending
1 = Interrupt level is pending

Figure 5-9. Interrupt Request Register (IRQ)

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 in the 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 by sending an IACK, 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 S3F84B8 interrupt structure, TimerA, TimerD and PWM counter overflow interrupts belong to this category of interrupts in which pending bits can be 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.

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 level is currently requesting service
— The interrupt must be enabled at the interrupt's source (peripheral control register)

When all 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, execute DI, 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 procedure above to some extent.

INSTRUCTION POINTER (IP)

The instruction pointer (IP) is adopted by all the S3C8/S3F8-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 of IP 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 6 clock cycles rather than the usual 16 clock cycles. To select a specific interrupt level for fast interrupt processing, you write the appropriate 3-bit value to SYM.4–SYM.2. Then, to enable fast interrupt processing for the selected level, you set SYM.1 to "1".

FAST INTERRUPT PROCESSING (Continued)

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 are stored in an unmapped, dedicated register called FLAGS' ("FLAGS prime").

NOTE

For the S3F84B8 microcontroller, the service routine for any one of the eight interrupt levels: IRQ0–IRQ7, can be selected for fast interrupt processing.

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 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. 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. **If you use fast interrupts, remember to load the IP with a new start address when the fast interrupt service routine ends. Please refer to IRET instruction in chapter 6.**

NOTES

6 INSTRUCTION SET

OVERVIEW

The SAM8RC instruction set is specifically designed to support the large register files that are typical of most SAM8 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 SAM8 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 Chapter 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 Chapter 3, "Addressing Modes."

Table 6-1. Instruction Group Summary

Mnemonic	Operands	Instruction
Load Instructions		
CLR	dst	Clear
LD	dst,src	Load
LDB	dst,src	Load bit
LDE	dst,src	Load external data memory
LDC	dst,src	Load program memory
LDED	dst,src	Load external data memory and decrement
LDCD	dst,src	Load program memory and decrement
LDEI	dst,src	Load external data memory and increment
LDCI	dst,src	Load program memory and increment
LDEPD	dst,src	Load external data memory with pre-decrement
LDCPD	dst,src	Load program memory with pre-decrement
LDEPI	dst,src	Load external data memory with pre-increment
LDCPI	dst,src	Load program memory with pre-increment
LDW	dst,src	Load word
POP	dst	Pop from stack
POPUD	dst,src	Pop user stack (decrementing)
POPUI	dst,src	Pop user stack (incrementing)
PUSH	src	Push to stack
PUSHUD	dst,src	Push user stack (decrementing)
PUSHUI	dst,src	Push user stack (incrementing)

Table 6-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction
Arithmetic Instructions		
ADC	dst,src	Add with carry
ADD	dst,src	Add
CP	dst,src	Compare
DA	dst	Decimal adjust
DEC	dst	Decrement
DECW	dst	Decrement word
DIV	dst,src	Divide
INC	dst	Increment
INCW	dst	Increment word
MULT	dst,src	Multiply
SBC	dst,src	Subtract with carry
SUB	dst,src	Subtract
Logic Instructions		
AND	dst,src	Logical AND
COM	dst	Complement
OR	dst,src	Logical OR
XOR	dst,src	Logical exclusive OR

Table 6-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction
Program Control Instructions		
BTJRF	dst,src	Bit test and jump relative on false
BTJRT	dst,src	Bit test and jump relative on true
CALL	dst	Call procedure
CPIJE	dst,src	Compare, increment and jump on equal
CPIJNE	dst,src	Compare, increment and jump on non-equal
DJNZ	r,dst	Decrement register and jump on non-zero
ENTER		Enter
EXIT		Exit
IRET		Interrupt return
JP	cc,dst	Jump on condition code
JP	dst	Jump unconditional
JR	cc,dst	Jump relative on condition code
NEXT		Next
RET		Return
WFI		Wait for interrupt
Bit Manipulation Instructions		
BAND	dst,src	Bit AND
BCP	dst,src	Bit compare
BITC	dst	Bit complement
BITR	dst	Bit reset
BITS	dst	Bit set
BOR	dst,src	Bit OR
BXOR	dst,src	Bit XOR
TCM	dst,src	Test complement under mask
TM	dst,src	Test under mask

Table 6-1. Instruction Group Summary (Concluded)

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

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.

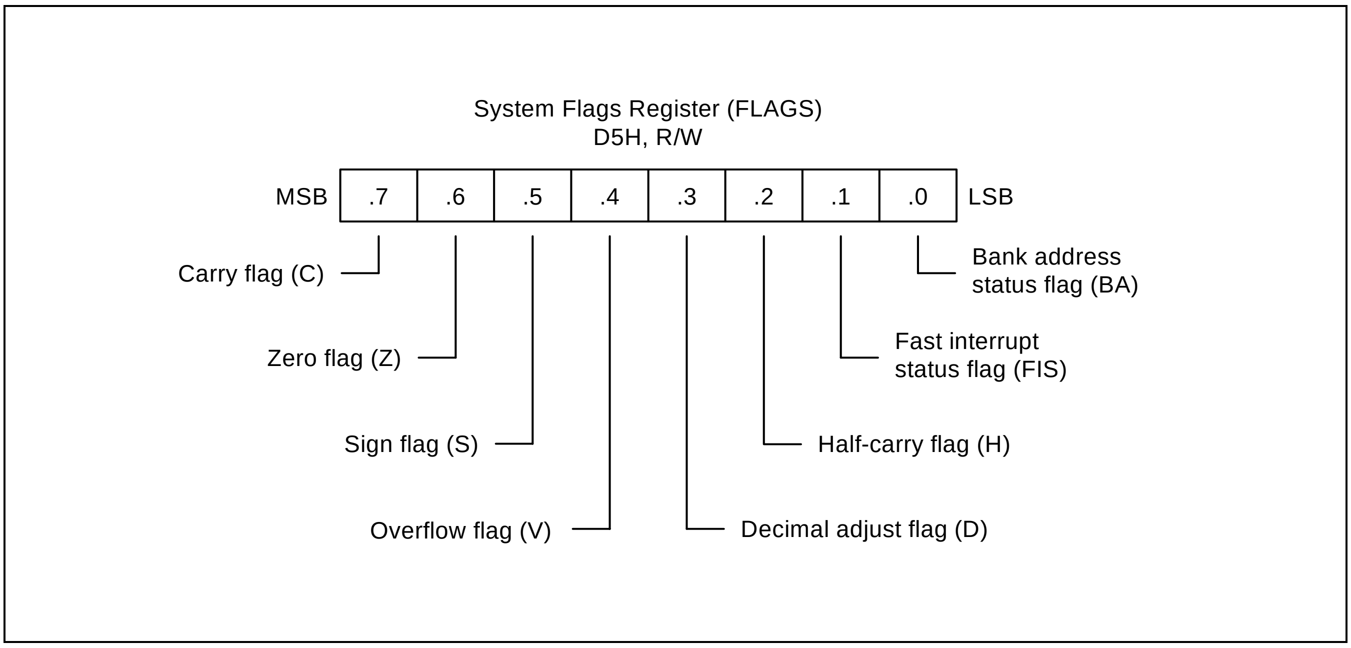


Figure 6-1. System Flags Register (FLAGS)

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

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

Table 6-3. Instruction Set Symbols

Symbol	Description
dst	Destination operand
src	Source operand
@	Indirect register address prefix
PC	Program counter
IP	Instruction pointer
FLAGS	Flags register (D5H)
RP	Register pointer
#	Immediate operand or register address prefix
H	Hexadecimal number suffix
D	Decimal number suffix
B	Binary number suffix
opc	Opcode

Table 6-4. Instruction Notation Conventions

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

Table 6-5. Opcode Quick Reference

OPCODE MAP									
LOWER NIBBLE (HEX)									
	–	0	1	2	3	4	5	6	7
U	0	DEC R1	DEC IR1	ADD r1,r2	ADD r1,Ir2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	BOR r0–Rb
P	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,Ir2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	BCP r1.b, R2
P	2	INC R1	INC IR1	SUB r1,r2	SUB r1,Ir2	SUB R2,R1	SUB IR2,R1	SUB R1,IM	BXOR r0–Rb
E	3	JP IRR1	SRP/0/1 IM	SBC r1,r2	SBC r1,Ir2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	BTJR r2.b, RA
R	4	DA R1	DA IR1	OR r1,r2	OR r1,Ir2	OR R2,R1	OR IR2,R1	OR R1,IM	LDB r0–Rb
	5	POP R1	POP IR1	AND r1,r2	AND r1,Ir2	AND R2,R1	AND IR2,R1	AND R1,IM	BITC r1.b
N	6	COM R1	COM IR1	TCM r1,r2	TCM r1,Ir2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	BAND r0–Rb
I	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,Ir2	TM R2,R1	TM IR2,R1	TM R1,IM	BIT r1.b
B	8	DECW RR1	DECW IR1	PUSHUD IR1,R2	PUSHUI IR1,R2	MULT R2,RR1	MULT IR2,RR1	MULT IM,RR1	LD r1, x, r2
B	9	RL R1	RL IR1	POPUD IR2,R1	POPUI IR2,R1	DIV R2,RR1	DIV IR2,RR1	DIV IM,RR1	LD r2, x, r1
L	A	INCW RR1	INCW IR1	CP r1,r2	CP r1,Ir2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, Irr2, xL
E	B	CLR R1	CLR IR1	XOR r1,r2	XOR r1,Ir2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, Irr2, xL
	C	RRC R1	RRC IR1	CPIJE Ir,r2,RA	LDC r1,Irr2	LDW RR2,RR1	LDW IR2,RR1	LDW RR1,IML	LD r1, Ir2
H	D	SRA R1	SRA IR1	CPIJNE Irr,r2,RA	LDC r2,Irr1	CALL IA1		LD IR1,IM	LD Ir1, r2
E	E	RR R1	RR IR1	LDCD r1,Irr2	LDCI r1,Irr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, Irr2, xs
X	F	SWAP R1	SWAP IR1	LDCPD r2,Irr1	LDCPI r2,Irr1	CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, Irr1, xs

Table 6-5. Opcode Quick Reference (Continued)

OPCODE MAP									
LOWER NIBBLE (HEX)									
	–	8	9	A	B	C	D	E	F
U	0	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NEXT
P	1	↓	↓	↓	↓	↓	↓	↓	ENTER
P	2								EXIT
E	3								WFI
R	4								SB0
	5								SB1
N	6								IDLE
I	7	↓	↓	↓	↓	↓	↓	↓	STOP
B	8								DI
B	9								EI
L	A								RET
E	B								IRET
	C								RCF
H	D	↓	↓	↓	↓	↓	↓	↓	SCF
E	E								CCF
X	F	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NOP

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

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

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 SAM8 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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**src**
opc | dst | src	2	4	12	r	r
		6	13	r	lr
opc | src | dst	3	6	14	R	R
		6	15	R	IR
opc | dst | src	3	6	16	R	IM

Examples: 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 dst,src

Operation: dst ← dst + 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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc | dst | src	2	4	02	r	r
		6	03	r	lr
opc | src | dst	3	6	04	R	R
		6	05	R	IR
opc | dst | src	3	6	06	R	IM

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

ADD	R1,R2	→	R1 = 15H, R2 = 03H
ADD	R1,@R2	→	R1 = 1CH, R2 = 03H
ADD	01H,02H	→	Register 01H = 24H, register 02H = 03H
ADD	01H,@02H	→	Register 01H = 2BH, register 02H = 03H
ADD	01H,#25H	→	Register 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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc | dst | src	2	4	52	r	r
		6	53	r	lr
opc | src | dst	3	6	54	R	R
		6	55	R	IR
opc | dst | src	3	6	56	R	IM

Examples: 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

BAND dst,src.b

BAND dst.b,src

Operation: dst(0) ← dst(0) AND src(b)

or

dst(b) ← dst(b) AND 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:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	dst | b | 0	src	3	6	67	r0	Rb
opc	src | b | 1	dst	3	6	67	Rb	r0

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.

Examples: 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

BCP dst,src.b

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:

			Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**src**
opc	dst | b | 0	src	3	6	17	r0	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 and register 01H = 01H:

BCP R1,01H.1 → R1 = 07H, 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:

		Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**
opc	dst | b | 0	2	4	57	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

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) $\leftarrow$ 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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst | b | 0	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 $\rightarrow$ 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 dst.b

Operation: 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:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc	dst | b | 1	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:

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	dst,src.b
BOR	dst.b,src

Operation: dst(0) ← dst(0) OR src(b)

or

dst(b) ← dst(b) OR src(0)

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:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | b | 0	src	3	6	07	r0	Rb
opc	src | b | 1	dst	3	6	07	Rb	r0

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.

Examples: 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:

	(Note 1)		Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	src | b | 0	dst	3	10	37	RA	rb

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:

(Note 1)	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | src | b | 1 | dst	3	10	37	RA	rb

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
BXOR	dst.b,src

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:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | b | 0 | src	3	6	27	r0	Rb
opc | src | b | 1 | dst	3	6	27	Rb	r0

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.

Examples: 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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode <u>dst</u>**
opc | dst	3	14	F6	DA
opc | dst	2	12	F4	IRR
opc | dst	2	14	D4	IA

Examples: 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

CCF

Operation: C ← NOT C

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

Flags: **C:** Complemented.

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	EF

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc | dst	2	4	B0	R
		4	B1	IR

Examples: 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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**
opc | dst	2	4	60	R
		4	61	IR

Examples: 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

CP dst,src

Operation: dst – src

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:

			Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**src**
opc	dst | src		2	4	A2	r	r
				6	A3	r	Ir
opc	src	dst	3	6	A4	R	R
				6	A5	R	IR
opc	dst	src	3	6	A6	R	IM

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | src | dst | RA	3	12	C2	r	Ir

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 02H (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:

				Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**src**
opc	src	dst	RA	3	12	D2	r	Ir

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:

CPIJNER1,@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):

Instruction	Carry Before DA	Bits 4–7 Value (Hex)	H Flag Before DA	Bits 0–3 Value (Hex)	Number Added to Byte	Carry After DA
	0	0–9	0	0–9	00	0
	0	0–8	0	A–F	06	0
	0	0–9	1	0–3	06	0
ADD	0	A–F	0	0–9	60	1
ADC	0	9–F	0	A–F	66	1
	0	A–F	1	0–3	66	1
	1	0–2	0	0–9	60	1
	1	0–2	0	A–F	66	1
	1	0–3	1	0–3	66	1
	0	0–9	0	0–9	00 = – 00	0
SUB	0	0–8	1	6–F	FA = – 06	0
SBC	1	7–F	0	0–9	A0 = – 60	1
	1	6–F	1	6–F	9A = – 66	1

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:

opc	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst
2	4	40	R
	4	41	IR

DA — Decimal Adjust

DA (Continued)

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

```
ADD    R1,R0   ;        C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = C, R1 ← 3CH
DA     R1      ;        R1 ← 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:

```
     0 0 0 1   0 1 0 1         15
   + 0 0 1 0   0 1 1 1         27
     0 0 1 1   1 1 0 0    =    3CH
```

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

```
     0 0 1 1   1 1 0 0
   + 0 0 0 0   0 1 1 0
     0 1 0 0   0 0 1 0    =    42
```

Assuming the same values given above, the statements

```
SUB    27H,R0 ;         C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = 1
DA     @R1    ;         @R1 ← 31–0
```

leave the value 31 (BCD) in address 27H (@R1).

DEC — Decrement

DEC dst

Operation: dst ← 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:

		Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode <u>dst</u>**
opc	dst	2	4	00	R
			4	01	IR

Examples: 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

DECW dst

Operation: dst ← 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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**
opc | dst	2	8	80	RR
		8	81	IR

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

```
DECW  RR0    →    R0  =  12H, R1  =  33H
DECW  @R2    →    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

DI

Operation: SYM (0) $\leftarrow$ 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:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	8F

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | src | dst	3	26/10	94	RR	R
		26/10	95	RR	IR
		26/10	96	RR	IM

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

DJNZ r,dst

Operation: r ← r – 1

If r ≠ 0, PC ← 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.

NOTE: In case of using DJNZ instruction, the working register being used as a counter should be set at the one of location 0C0H to 0CFH with SRP, SRP0, or SRP1 instruction.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
r | opc | dst	2	8 (jump taken)	rA	RA
		8 (no jump)	r = 0 to F	

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) $\leftarrow$ 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:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	4	9F

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

ENTER

Operation:

SP	←	SP – 2
@SP	←	IP
IP	←	PC
PC	←	@IP
IP	←	IP + 2

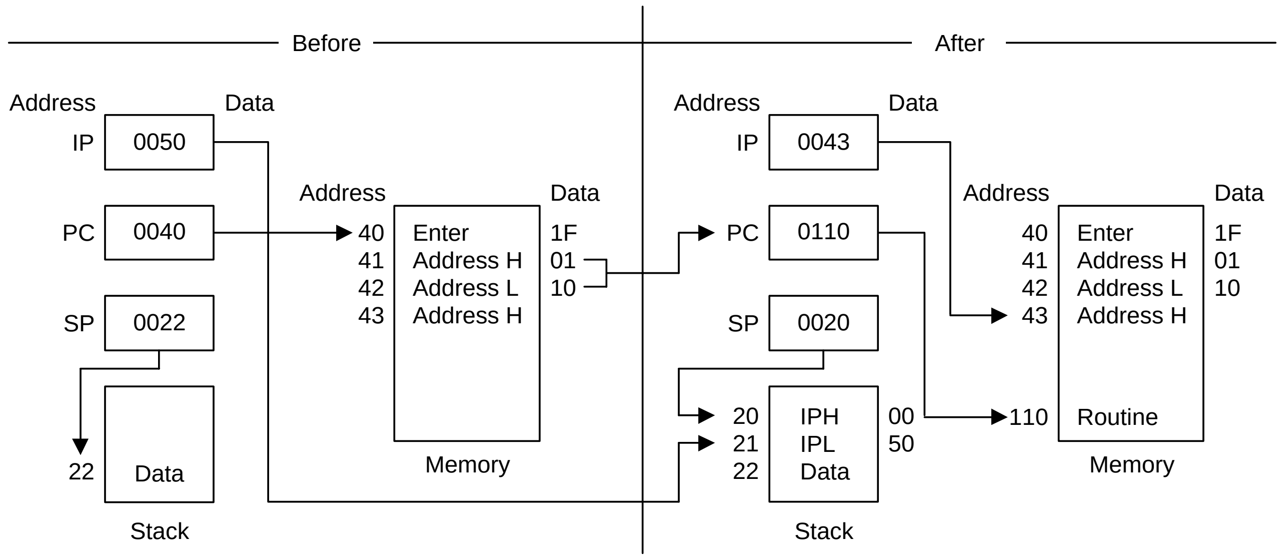
This instruction is useful when implementing threaded-code languages. The contents of the instruction pointer are pushed to the stack. The program counter (PC) value is then written to the instruction pointer. The program memory word that is pointed to by the instruction pointer is loaded into the PC,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

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	14	1F

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



EXIT — Exit

EXIT

Operation:

IP	←	@SP
SP	←	SP + 2
PC	←	@IP
IP	←	IP + 2

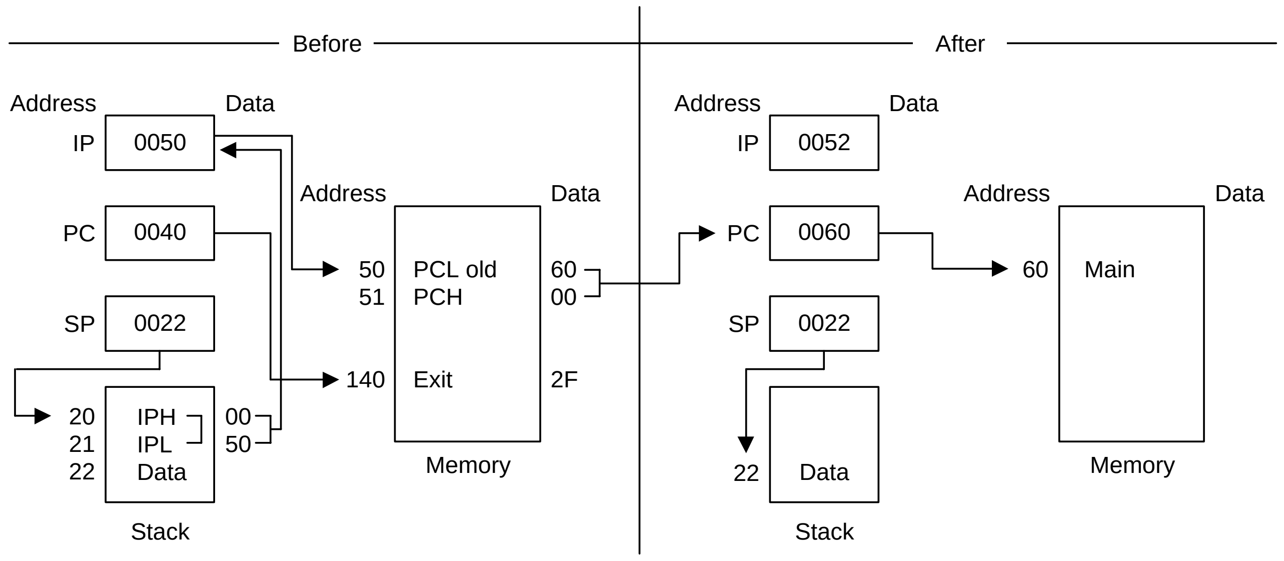
This instruction is useful when implementing threaded-code languages. The stack value is popped and loaded into the instruction pointer. The program memory word that is pointed to by the instruction pointer is then loaded into the program counter, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	14 (internal stack)	2F
		16 (internal stack)	

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.
In application programs, a IDLE instruction must be immediately followed by at least three NOP instructions. This ensures an adeguate time interval for the clock to stabilize before the next instruction is executed. If three or more NOP instructons are not used after IDLE instruction, leakage current could be flown because of the floating state in the internal bus.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	1	4	6F	–	–

Example: The instruction

```
IDLE          ; stops the CPU clock but not the system clock
NOP
NOP
NOP
```

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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode** <u>**dst**</u>
dst | opc	1	4	rE	r
			r = 0 to F	
opc | dst	2	4	20	R
		4	21	IR

Examples: 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 ← 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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	8	A0	RR
		8	A1	IR

Examples: Given: R0 = 1AH, R1 = 02H, register 02H = 0FH, and register 03H = 0FFH:

```
INCW   RR0   →   R0 = 1AH, R1 = 03H
INCW   @R1   →   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	IRET (Normal)	IRET (Fast)
Operation:	FLAGS ← @SP	PC ↔ IP
	SP ← SP + 1	FLAGS ← FLAGS'
	PC ← @SP	FIS ← 0
	SP ← SP + 2	
	SYM(0) ← 1	

This instruction 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. A "normal IRET" is executed only if the fast interrupt status bit (FIS, bit one of the FLAGS register, 0D5H) is cleared (= "0"). If a fast interrupt occurred, IRET clears the FIS bit that was set at the beginning of the service routine.

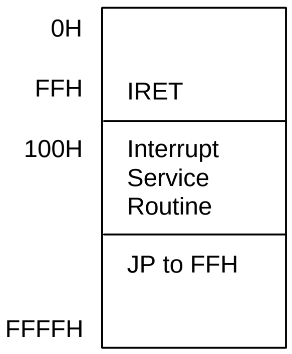
Flags: All flags are restored to their original settings (that is, the settings before the interrupt occurred).

Format:

IRET (Normal)	**Bytes**	**Cycles**	**Opcode (Hex)**
opc	1	10 (internal stack)	BF
		12 (internal stack)	

IRET (Fast)	**Bytes**	**Cycles**	**Opcode (Hex)**
opc	1	6	BF

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.



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)

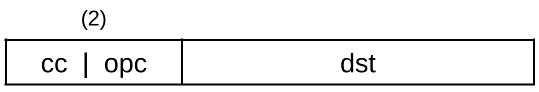
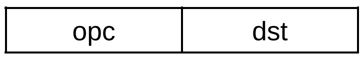
JP dst (Unconditional)

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: (1)

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
	3	8	ccD	DA
			cc = 0 to F	
	2	8	30	IRR

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.

Examples: 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:

(1)		**Bytes**	**Cycles**	**Opcode (Hex)**	**Addr Mode <u>dst</u>**
cc | opc	dst	2	6	ccB	RA
				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 dst,src

Operation: dst ← src

The contents of the source are loaded into the destination. The source's contents are unaffected.

Flags: No flags are affected.

Format:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
dst | opc ‖ src	2	4	rC	r	IM
		4	r8	r	R
src | opc ‖ dst	2	4	r9	R	r
			r = 0 to F		
opc ‖ dst | src	2	4	C7	r	Ir
		4	D7	Ir	r
opc ‖ src ‖ dst	3	6	E4	R	R
		6	E5	R	IR
opc ‖ dst ‖ src	3	6	E6	R	IM
		6	D6	IR	IM
opc ‖ src ‖ dst	3	6	F5	IR	R
opc ‖ dst | src ‖ x	3	6	87	r	x [r]
opc ‖ src | dst ‖ x	3	6	97	x [r]	r

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:

			Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc	dst | b | 0	src	3	6	47	r0	Rb
opc	src | b | 1	dst	3	6	47	Rb	r0

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:

					Bytes	Cycles	Opcode	Addr Mode	
							(Hex)	dst	src
1.	opc	dst | src			2	10	C3	r	Irr
2.	opc	src | dst			2	10	D3	Irr	r
3.	opc	dst | src	XS		3	12	E7	r	XS [rr]
4.	opc	src | dst	XS		3	12	F7	XS [rr]	r
5.	opc	dst | src	XL_L	XL_H	4	14	A7	r	XL [rr]
6.	opc	src | dst	XL_L	XL_H	4	14	B7	XL [rr]	r
7.	opc	dst | 0000	DA_L	DA_H	4	14	A7	r	DA
8.	opc	src | 0000	DA_L	DA_H	4	14	B7	DA	r
9.	opc	dst | 0001	DA_L	DA_H	4	14	A7	r	DA
10.	opc	src | 0001	DA_L	DA_H	4	14	B7	DA	r

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: 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 decremented. The contents of the source are unaffected.

LDCD references program memory and LDED references 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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	10	E2	r	Irr

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
                   ;  R8 = 0CDH, R6 = 10H, R7 = 32H (RR6 ← RR6 – 1)

LDED    R8,@RR6    ;  0DDH (contents of data memory location 1033H) is loaded
                   ;  into R8 and RR6 is decremented by one (RR6 ← RR6 – 1)
                   ;  R8 = 0DDH, R6 = 10H, 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 'Irr' even for program memory and odd for data memory.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	10	E3	r	Irr

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:	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 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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc | src | dst	2	14	F2	Irr	r

Examples: Given: R0 = 77H, R6 = 30H, and R7 = 00H:

```
LDCPD @RR6,R0      ; (RR6 ← 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 ← 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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc | src | dst	2	14	F3	Irr	r

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:

			Bytes	**Cycles**	**Opcode**	**Addr Mode**	
					(Hex)	**dst**	**src**
opc	src	dst	3	8	C4	RR	RR
				8	C5	RR	IR
opc	dst	src	4	8	C6	RR	IML

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 $\leftarrow$ dst $\times$ 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:

opc	src	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
3	22	84	RR	R
	22	85	RR	IR
	22	86	RR	IM

Examples: Given: Register 00H = 20H, register 01H = 03H, register 02H = 09H, register 03H = 06H:

MULT	00H, 02H	$\rightarrow$	Register 00H = 01H, register 01H = 20H, register 02H = 09H
MULT	00H, @01H	$\rightarrow$	Register 00H = 00H, register 01H = 0C0H
MULT	00H, #30H	$\rightarrow$	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

NEXT

Operation: PC ← @ IP

IP ← IP + 2

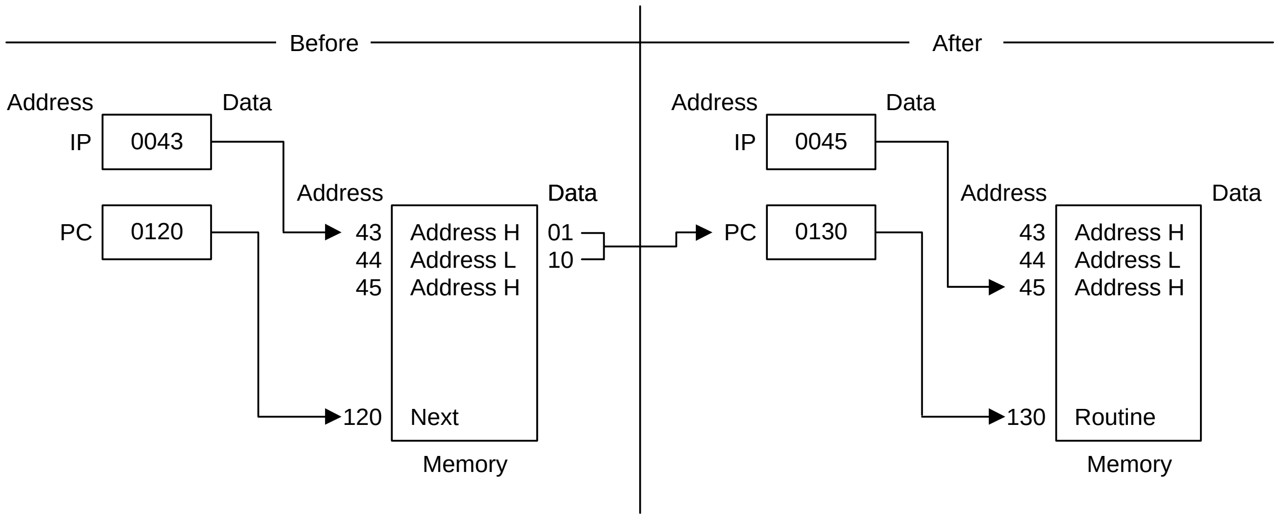
The NEXT instruction is useful when implementing threaded-code languages. The program memory word that is pointed to by the instruction pointer is loaded into the program counter. The instruction pointer is then incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	10	0F

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:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	FF

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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
$\boxed{\text{opc}}\boxed{\text{dst} \mid \text{src}}$	2	4	42	r	r
		6	43	r	lr
$\boxed{\text{opc}}\boxed{\text{src}}\boxed{\text{dst}}$	3	6	44	R	R
		6	45	R	IR
$\boxed{\text{opc}}\boxed{\text{dst}}\boxed{\text{src}}$	3	6	46	R	IM

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: dst ← @SP

SP ← 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:

Format		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst	2	8	50	R
			8	51	IR

Examples: Given: Register 00H = 01H, register 01H = 1BH, SPH (0D8H) = 00H, SPL (0D9H) = 0FBH, and stack register 0FBH = 55H:

POP	00H	→	Register 00H = 55H, SP = 00FCH
POP	@00H	→	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 dst,src

Operation: dst ← src

IR ← 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:

			Bytes	Cycles	Opcode (Hex)	Addr Mode **dst**	**src**
opc	src	dst	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:

			Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc	src	dst	3	8	93	R	IR

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc | src	2	8 (internal clock) 8 (external clock)	70	R
		8 (internal clock) 8 (external clock)	71	IR

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 0FFFFH. It then loads the contents of register 40H into location 0FFFFH 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:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst	src	3	8	82	IR	R

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	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

RCF	RCF
Operation:	$C \leftarrow 0$ The carry flag is cleared to logic zero, regardless of its previous value.
Flags:	**C:** Cleared to "0". No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
$\boxed{\text{opc}}$	1	4	CF

Example: Given: C = "1" or "0":

The instruction RCF clears the carry flag (C) to logic zero.

RET — Return

RET

Operation: PC ← @SP

SP ← SP + 2

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:

	Bytes	Cycles	Opcode (Hex)
opc	1	8 (internal stack)	AF
		10 (internal stack)	

Example: Given: SP = 00FCH, (SP) = 101AH, and PC = 1234:

RET → PC = 101AH, SP = 00FEH

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

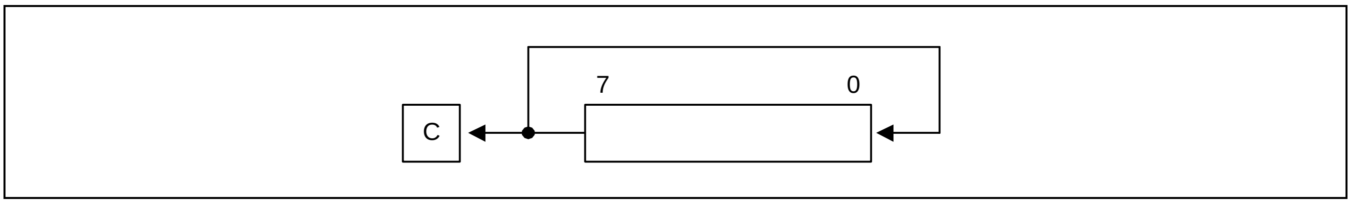
RL dst

Operation: C ← dst (7)

dst (0) ← dst (7)

dst (n + 1) ← 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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc | dst	2	4	90	R
		4	91	IR

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

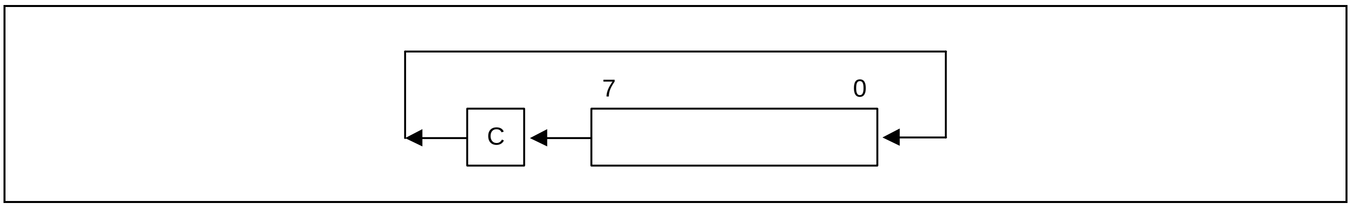
RLC dst

Operation: dst (0) ← C

C ← dst (7)

dst (n + 1) ← 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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc | dst	2	4	10	R
		4	11	IR

Examples: Given: Register 00H = 0AAH, register 01H = 02H, and register 02H = 17H, C = "0":

RLC 00H → Register 00H = 54H, C = "1"

RLC @01H → 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

RR dst

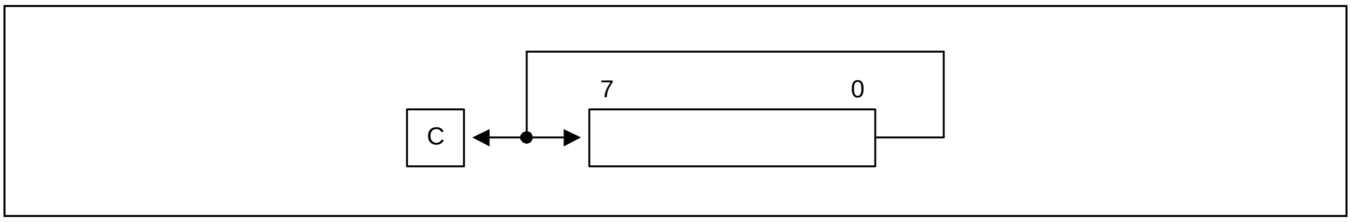
Operation:

C $\leftarrow$ dst (0)

dst (7) $\leftarrow$ dst (0)

dst (n) $\leftarrow$ 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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc | dst	2	4	E0	R
		4	E1	IR

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

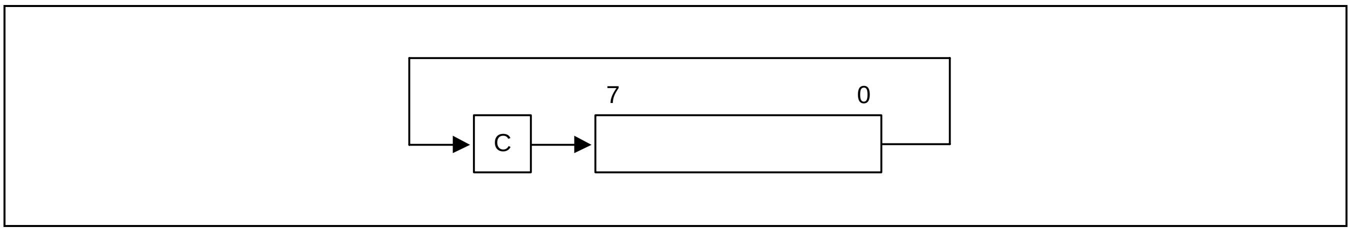
RRC dst

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	4	C0	R
		4	C1	IR

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:

	Bytes	**Cycles**	**Opcode (Hex)**
$\boxed{\text{opc}}$	1	4	4F

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 S3C8-series microcontrollers.)

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	4	5F

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	4	32	r	r
		6	33	r	Ir
opc | src | dst	3	6	34	R	R
		6	35	R	IR
opc | dst | src	3	6	36	R	IM

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

SCF

Operation: C ← 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:

	Bytes	Cycles	Opcode (Hex)
$\boxed{\text{opc}}$	1	4	DF

Example: The statement

SCF

sets the carry flag to logic one.

SRA — Shift Right Arithmetic

SRA dst

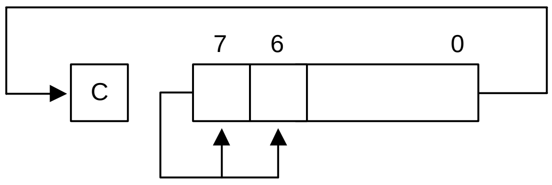
Operation:

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.



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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**
opc | dst	2	4	D0	R
		4	D1	IR

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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode <u>src</u>**
opc | src	2	4	31	IM

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

STOP

Operation:

The STOP instruction stops the 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 nRESET pin must be held to Low level until the required oscillation stabilization interval has elapsed.
In application programs, a STOP instruction must be immediately followed by at least three NOP instructions. This ensures an adeguate time interval for the clock to stabilize before the next instruction is executed. If three or more NOP instructons are not used after STOP instruction, leakage current could be flown because of the floating state in the internal bus.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	1	4	7F	–	–

Example: The statement

```
STOP                    ;  halts all microcontroller operations
NOP
NOP
NOP
```

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 of 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:

			Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc	dst | src		2	4	22	r	r
				6	23	r	Ir
opc	src	dst	3	6	24	R	R
				6	25	R	IR
opc	dst	src	3	6	26	R	IM

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"

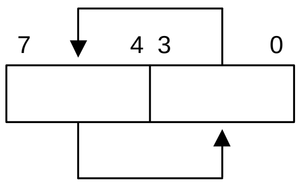
In the first example, if working register R1 contains the value 12H and if register R2 contains the value 03H, the statement "SUB R1,R2" subtracts the source value (03H) from the destination value (12H) and stores the result (0FH) in destination register R1.

SWAP — Swap Nibbles

SWAP dst

Operation: dst (0 – 3) ↔ dst (4 – 7)

The contents of the lower four bits and upper four bits of the destination operand are swapped.



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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
| opc | dst |	2	4	F0	R
		4	F1	IR

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 (11100011B).

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	4	62	r	r
		6	63	r	lr
opc | src | dst	3	6	64	R	R
		6	65	R	IR
opc | dst | src	3	6	66	R	IM

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

TM dst,src

Operation: dst AND src

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc | dst | src	2	4	72	r	r
		6	73	r	Ir
opc | src | dst	3	6	74	R	R
		6	75	R	IR
opc | dst | src	3	6	76	R	IM

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:

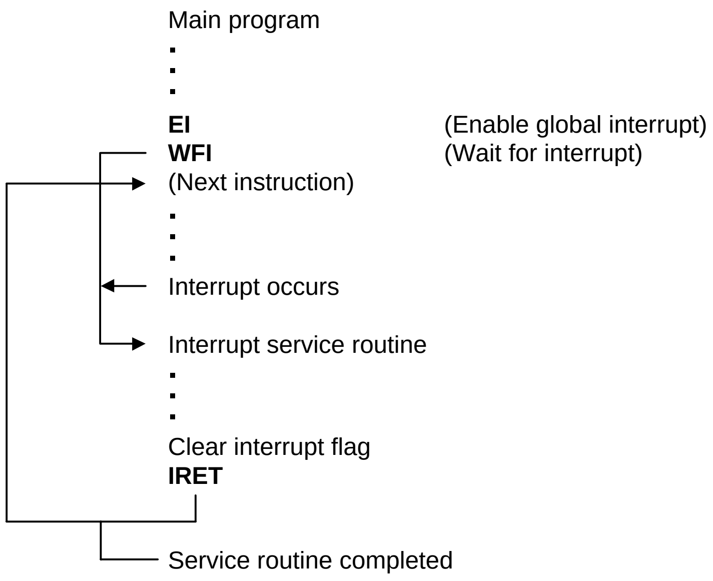
The CPU is effectively halted until an interrupt occurs, except that DMA transfers can still take place during this wait state. The WFI status can be released by an internal interrupt, including a fast interrupt .

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4n	3F
		(n = 1, 2, 3, …)	

Example: The following sample program structure shows the sequence of operations that follow a "WFI" statement:



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:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc | dst | src	2	4	B2	r	r
		6	B3	r	lr
opc | src | dst	3	6	B4	R	R
		6	B5	R	IR
opc | dst | src	3	6	B6	R	IM

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

7 CLOCK CIRCUIT

OVERVIEW

By smart option (3FH.1 – .0 in ROM), user can select internal RC oscillator, external RC oscillator, or external oscillator. In using internal oscillator, XIN (P0.0), XOUT (P0.1) can be used by normal I/O pins. An internal RC oscillator source provides a typical 8 MHz or 0.5 MHz depending on smart option.

An external RC oscillation source provides a typical 8MHz clock for S3F84B8. An internal capacitor supports the RC oscillator circuit. An external crystal or ceramic oscillation source provides a maximum 10 MHz clock. The XIN and XOUT pins connect the oscillation source to the on-chip clock circuit. Simplified external RC oscillator and crystal/ceramic oscillator circuits are shown in Figures 7-1 and 7-2. When you use external oscillator, P0.0, P0.1 must be set to output port to prevent current consumption.

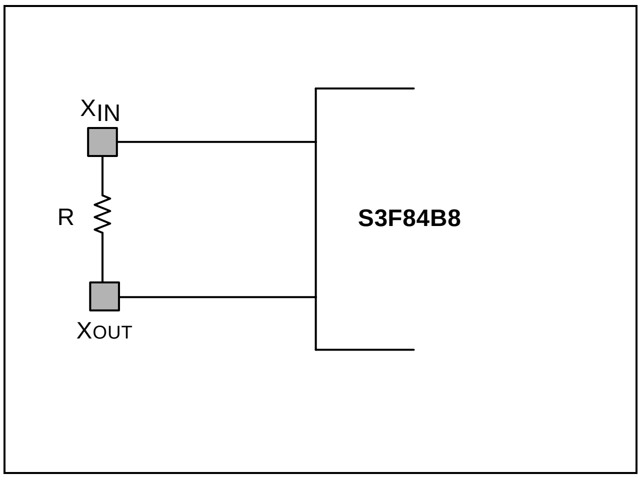


Figure 7-1. Main Oscillator Circuit (RC Oscillator with Internal Capacitor)

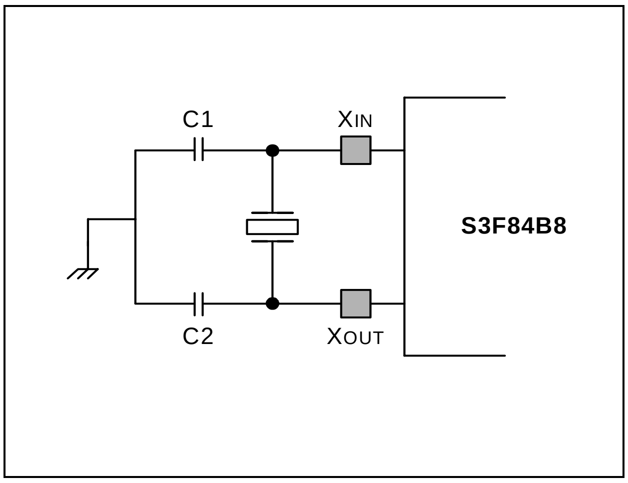


Figure 7-2. Main Oscillator Circuit (Crystal/Ceramic Oscillator)

CLOCK STATUS DURING POWER-DOWN MODES

The two power-down modes, Stop mode and Idle mode, affect clock oscillation as follows:

— In Stop mode, the main oscillator "freezes", halting the CPU and peripherals. The contents of the register file and current system register values are retained. Stop mode is released, and the oscillator started, by a reset operation or by an external interrupt with RC-delay noise filter (for S3F84B8, INT0–INT5).

— In Idle mode, the internal clock signal is gated off to the CPU, but not to interrupt control and the timer. The current CPU status is preserved, including stack pointer, program counter, and flags. Data in the register file is retained. Idle mode is released by a reset or by an interrupt (external or internally-generated).

SYSTEM CLOCK CONTROL REGISTER (CLKCON)

The system clock control register, CLKCON, is located in location D4H. It is read/write addressable and has the following functions:

— Oscillator IRQ wake-up function enable/disable (CLKCON.7)

— Oscillator frequency divide-by value: non-divided, 2, 8, or 16 (CLKCON.4 and CLKCON.3)

The CLKCON register controls whether or not an external interrupt can be used to trigger a Stop 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, and the fOSC/16 (the slowest clock speed) is selected as the CPU clock. If necessary, you can then increase the CPU clock speed to fOSC, fOSC/2 or fOSC/8.

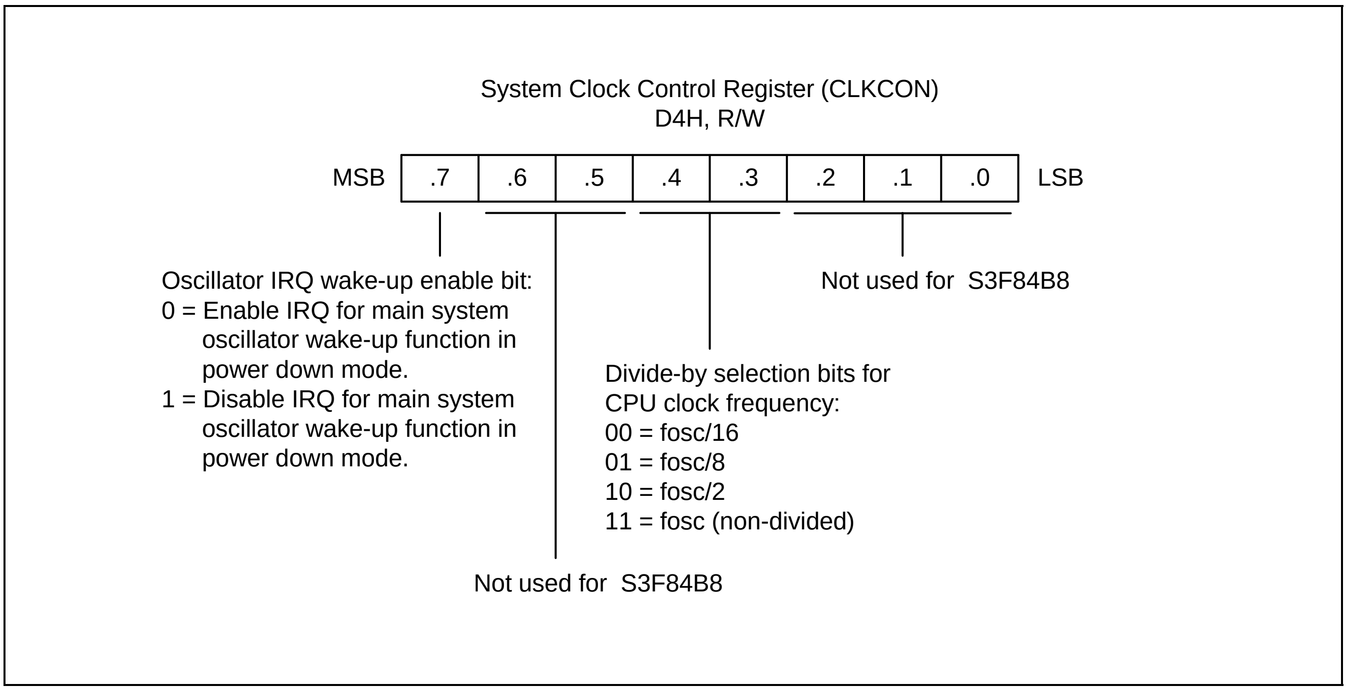
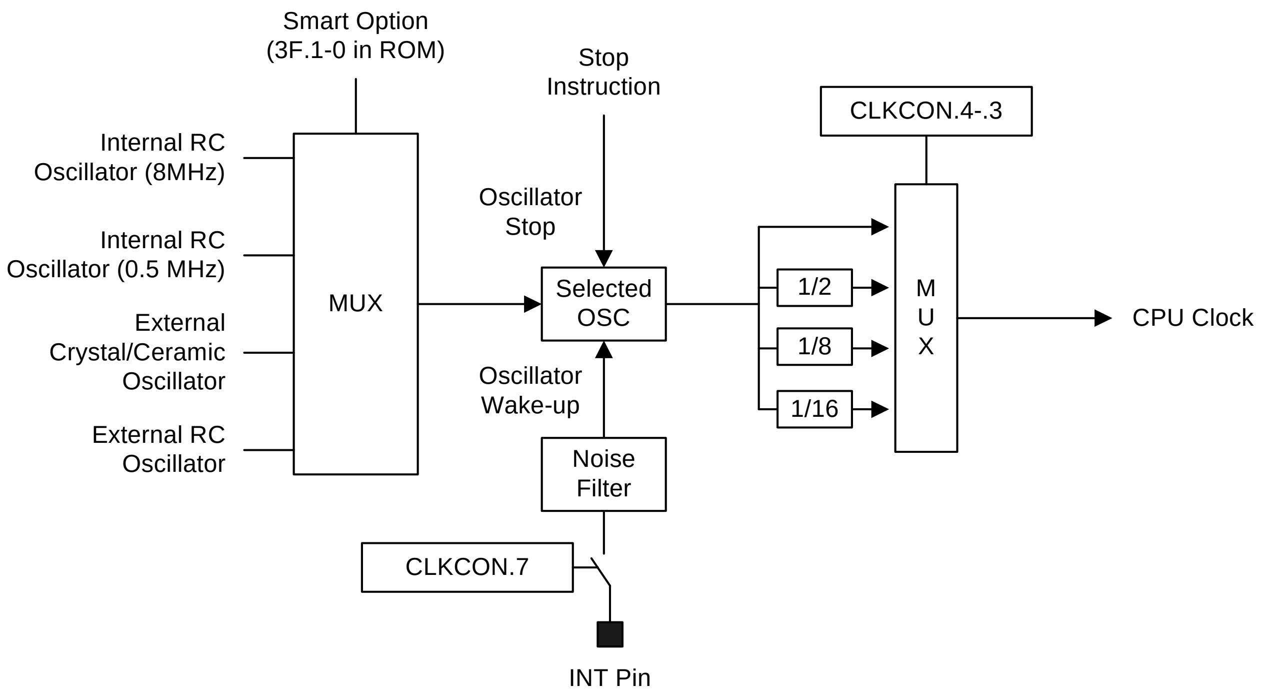

System Clock Control Register (CLKCON)
D4H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Oscillator IRQ wake-up enable bit:
0 = Enable IRQ for main system oscillator wake-up function in power down mode.
1 = Disable IRQ for main system oscillator wake-up function in power down mode.
Not used for S3F84B8
Divide-by selection bits for CPU clock frequency:
00 = fosc/16
01 = fosc/8
10 = fosc/2
11 = fosc (non-divided)
Not used for S3F84B8

Figure 7-3. System Clock Control Register (CLKCON)



NOTE: An external interrupt (with RC-delay noise filter) can be used to release stop mode and "wake-up" the main oscillator.
In the S3F84B8, the INT0-INT5 external interrupts are of this type.

Figure 7-4. System Clock Circuit Diagram

NOTES

8 RESET and POWER-DOWN

SYSTEM RESET

OVERVIEW

By smart option (3FH.7 in ROM), user can select internal RESET (LVR) or external RESET.

The S3F84B8 can be RESET in four ways:

— by external power-on-reset

— by the external nRESET input pin pulled low

— by digital watchdog peripheral timing out

— by Low Voltage Reset (LVR)

During an external power-on reset, the voltage at V_{DD} is High level and the nRESET pin is forced to Low level. The nRESET signal is an input through a Schmitt trigger circuit where it is then synchronized to CPU clock. This brings the S3F84B8 into a known operating status. To ensure correct start-up, the user should take care that nRESET signal is not released before the V_{DD} level is sufficient to allow MCU operation at the chosen frequency.

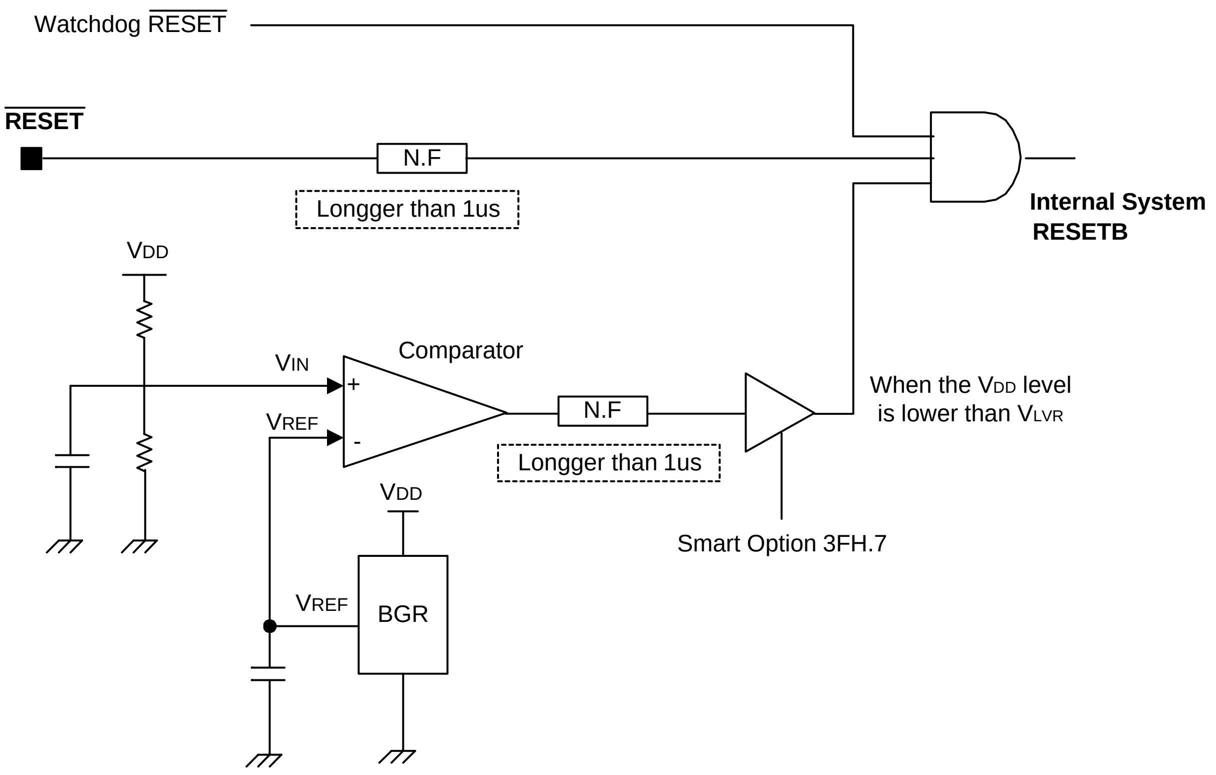
The nRESET pin must be held as Low level for a minimum time interval after the power supply comes within tolerance in order to allow time for internal CPU clock oscillation to stabilize.

When a reset occurs during normal operation (with both V_{DD} and nRESET at High level), the signal at the nRESET pin is forced Low and the Reset operation starts. All system and peripheral control registers are then set to their default hardware Reset values (see Table 8-1).

The MCU provides a watchdog timer function in order to ensure graceful recovery from software malfunction. If watchdog timer is not refreshed before an end-of-counter condition (overflow) is reached, the internal reset will be activated.

The on-chip Low Voltage, Reset features static Reset when supply voltage is below a reference value (Typ. 1.9, 2.3, 3.0, 3.6, 3.9V). Thanks to this feature, external reset circuit can be removed while keeping the application safety. As long as the supply voltage is below the reference value, there is an internal and static RESET. The MCU can start only when the supply voltage rises over the reference value.

When you calculate power consumption, please remember that a static current of LVR circuit should be added a CPU operating current in any operating modes such as Stop, Idle, and normal RUN mode.



NOTES:

1. The target of voltage detection level is the one you selected at smart option 3FH.
2. BGR is Band Gap voltage Reference

Figure 8-1. Low Voltage Reset Circuit

NOTE

To program the duration of the oscillation stabilization interval, you must make the appropriate settings 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.

MCU Initialization Sequence

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 reset to their initial values (see Table 8-1).
— The program counter is loaded with the ROM reset address, 0100H or other values set by smart option..
— When the programmed oscillation stabilization time interval has elapsed, the address stored in the first and second bytes of RESET address in ROM is fetched and executed.

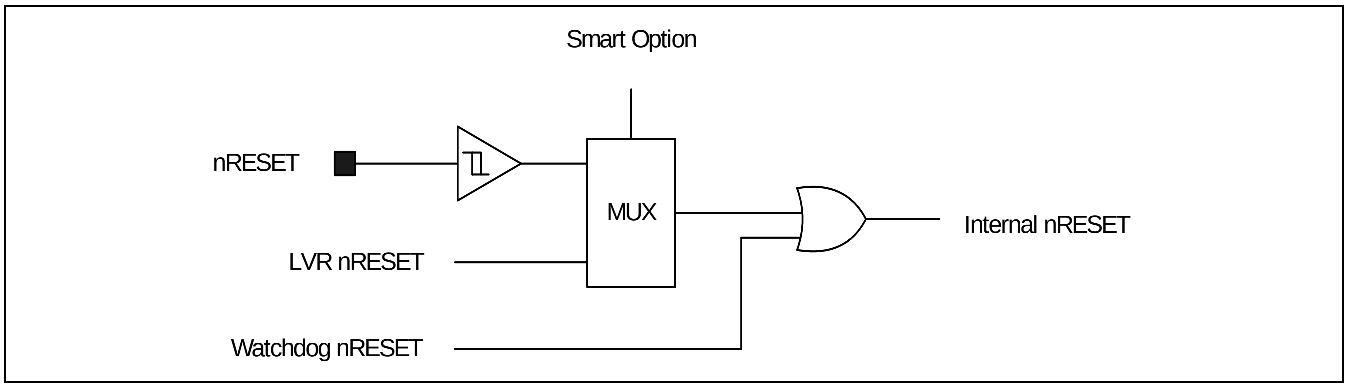


Figure 8-2. Reset Block Diagram

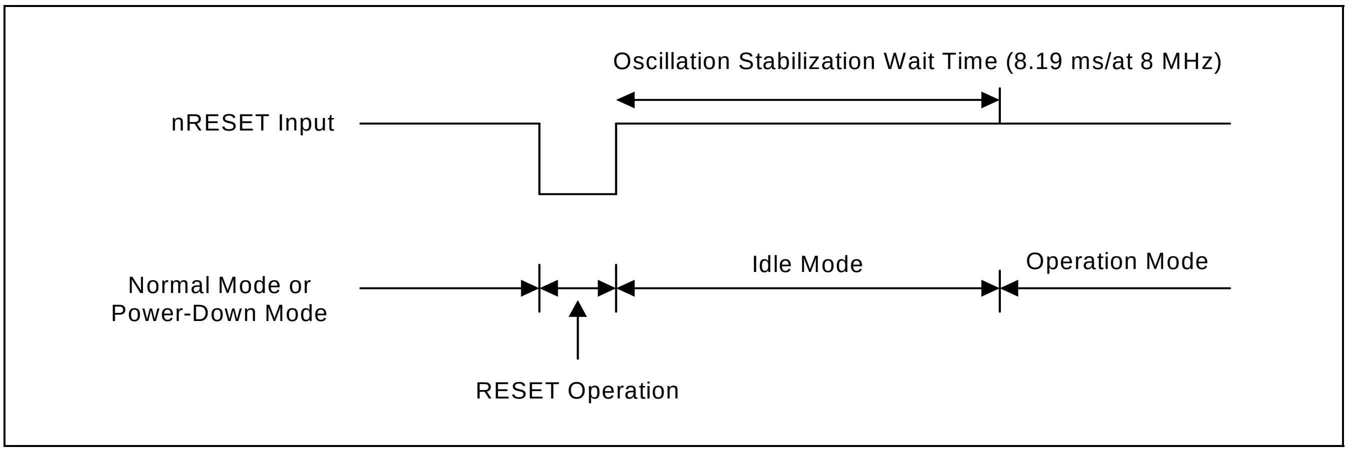

Oscillation Stabilization Wait Time (8.19 ms/at 8 MHz)
nRESET Input
Normal Mode or Power-Down Mode
Idle Mode
Operation Mode
RESET Operation

Figure 8-3. Timing for S3F84B8 after RESET

POWER-DOWN MODES

STOP MODE

Stop mode is invoked by the instruction STOP (opcode 7FH). 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 2μA when LVR (Low Voltage Reset) is disabled. All system functions are halted 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 an nRESET signal or by an external interrupt.

Note: Before executing the STOP instruction, STPCON register must be set to "10100101B".

Using RESET to Release Stop Mode

Stop mode is released when the nRESET signal is released and returns to High level. All system and peripheral control registers are then reset to their default values and the contents of all data registers are retained. A Reset operation automatically selects a slow clock (f_x/16) because CLKCON.3 and CLKCON.4 are cleared to "00B". After the oscillation stabilization interval has elapsed, the CPU executes the system initialization routine by fetching the 16-bit address stored in the first and second bytes of RESET address in ROM.

Using an External Interrupt to Release Stop Mode

External interrupts with an RC-delay noise filter circuit can be used to release Stop mode (Clock-related external interrupts cannot be used). External interrupts INT0-INT6 in the S3F84B8 interrupt structure meet this criterion.

Note that when Stop mode is released by an external interrupt, values in system and peripheral control registers are not changed. When you use an interrupt to release Stop mode, the CLKCON.3 and CLKCON.4 register values remain unchanged, and the currently selected clock value is used, thus you can also program the duration of the oscillation stabilization interval by putting the appropriate value to BTCON register *before* entering Stop mode.

The external interrupt is serviced when the Stop mode release occurs. Following the IRET from the service routine, the instruction immediately following 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 select peripherals remain active. During Idle mode, the internal clock signal is gated off to the CPU, but not to interrupt logic and timer/counters. 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 (f_{xx}/16) because CLKCON.3 and CLKCON.4 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.3 and CLKCON.4 register values remain unchanged, and the currently selected clock value is used. The interrupt is then serviced. Following the IRET from the service routine, the instruction immediately following the one that initiated idle mode is executed.

NOTES

1. Only external interrupts that are not clock-related can be used to release stop mode. To release Idle mode, however, any type of interrupt (that is, internal or external) can be used.
2. Before enter the STOP or IDLE mode, ADC must be disabled. Otherwise, the STOP or IDLE current

will be increased significantly.

HARDWARE RESET VALUES

The reset values for CPU and system registers, peripheral control registers, and peripheral data registers following 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, but read 0 is the bit value.

Table 8-1. S3F84B8 Set1 Registers Values after RESET

Register name	Mnemonic	Address & Location		RESET value (Bit)							
		Address	R/W	7	6	5	4	3	2	1	0
Locations D0-D2H are not mapped											
Basic timer control register	BTCON	D3H	R/W	0	0	0	0	0	0	0	0
Clock control register	CLKCON	D4H	R/W	0	–	–	0	0	–	–	–
System flags register	FLAGS	D5H	R/W	x	x	x	x	x	x	0	0
Register Pointer 0	RP0	D6H	R/W	1	1	0	0	0	–	–	–
Register Pointer 1	RP1	D7H	R/W	1	1	0	0	1	–	–	–
Location D8H is not mapped											
Stack Pointer register	SPL	D9H	R/W	x	x	x	x	x	x	x	x
Instruction Pointer (High Byte)	IPH	DAH	R/W	x	x	x	x	x	x	x	x
Instruction Pointer (Low Byte)	IPL	DBH	R/W	x	x	x	x	x	x	x	x
Interrupt Request register	IRQ	DCH	R	0	0	0	0	0	0	0	0
Interrupt Mask Register	IMR	DDH	R/W	x	x	x	x	x	x	x	x
System Mode Register	SYM	DEH	R/W	0	–	–	x	x	x	0	0
Register Page Pointer	PP	DFH	R/W	0	0	0	0	0	0	0	0

NOTE: – : Not mapped or not used, x: Undefined

Table 8-1. System and Peripheral Control Registers Set1 Bank 0(Continued)

Register Name	Mnemonic	Address	R/W	Bit Values After RESET							
		Hex		7	6	5	4	3	2	1	0
Port 0 data register	P0	E0H	R/W	–	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	R/W	–	–	–	–	–	0	0	0
Port 2 data register	P2	E2H	R/W	0	0	0	0	0	0	0	0
Port 0 interrupt control register	P0INT	E3H	R/W	–	0	0	0	0	–	0	0
Port 0 control register (High byte)	P0CONH	E4H	R/W	–	–	0	0	0	0	0	0
Port 0 control register (Low byte)	P0CONL	E5H	R/W	0	0	–	–	0	0	0	0
Port 0 interrupt pending register	P0PND	E6H	R/W	–	0	0	0	0	–	0	0
Port 1 control register (Low byte)	P1CON	E7H	R/W	–	–	0	0	0	0	0	0
Port 2 control register (High byte)	P2CONH	E8H	R/W	0	0	0	0	0	0	0	0
Port 2 control register (Low byte)	P2CONL	E9H	R/W	0	0	0	0	0	0	0	0
Comparator 0 control register	CMP0CON	EAH	R/W	–	–	–	0	0	0	1	0
Comparator 1 control register	CMP1CON	EBH	R/W	0	0	0	0	0	0	1	0
Comparator 2 control register	CMP2CON	ECH	R/W	0	0	0	0	0	0	1	0
Comparator 3 control register	CMP3CON	EDH	R/W	0	0	0	0	0	0	1	0
Comparator interrupt control register	CMPINT	EEH	R/W	1	1	1	1	1	1	1	1
PWM control register	PWMCON	EFH	R/W	0	0	0	0	0	0	0	0
PWM CMP register	PWMCCON	F0H	R/W	0	0	0	0	0	0	0	0
PWM delay trigger register	PWMDL	F1H	R/W	0	0	0	0	0	0	0	0
PWM preset data register (High byte)	PWMPDATAH	F2H	R/W	0	0	0	0	0	0	0	0
PWM preset data register (Low byte)	PWMPDATAL	F3H	R/W	–	–	–	–	–	–	0	0
PWM data register (High byte)	PWMDATAH	F4H	R/W	0	0	0	0	0	0	0	0
PWM data register (Low byte)	PWMDATAL	F5H	R/W	–	–	–	–	–	–	0	0
Anti-mis-trigger register	AMTDATA	F6H	R/W	1	1	1	1	1	1	1	1
Buzzer control register	BUZCON	F7H	R/W	0	0	0	0	0	0	0	0
A/D converter data register (High byte)	ADDATAH	F8H	R	x	x	x	x	x	x	x	x
A/D converter data register (Low byte)	ADDDATAL	F9H	R	–	–	–	–	–	–	x	x
A/D control register	ADCON	FAH	R/W	0	0	0	0	0	0	0	0
Locations FB-FCH are not mapped											
Basic timer counter	BTCNT	FDH	R	0	0	0	0	0	0	0	0
Location FEH is not mapped											
Interrupt priority register	IPR	FFH	R/W	x	x	x	x	x	x	x	x

NOTES:1. – : Not mapped or not used, x: Undefined,

Table 8-1. System and Peripheral Control Registers Set1 Bank1

Register name	Mnemonic	Address & Location		RESET value (Bit)							
		Address	R/W	7	6	5	4	3	2	1	0
Operational Amplifier control register	OPACON	E0H	R/W	–	–	–	–	–	–	0	0
Timer A control register	TACON	E1H	R/W	0	0	0	0	0	0	0	0
Timer A clock pre-scalar	TAPS	E2H	R/W	0	–	–	–	0	0	0	0
Timer A data register	TADATA	E3H	R/W	1	1	1	1	1	1	1	1
Timer A counter register	TACNT	E4H	R	0	0	0	0	0	0	0	0
Timer C control register	TCCON	E5H	R/W	0	–	0	0	0	–	0	–
Timer C clock pre-scalar	TCPS	E6H	R/W	0	–	–	–	0	0	0	0
Timer C data register	TCDATA	E7H	R/W	1	1	1	1	1	1	1	1
Timer C counter	TCCNT	E8H	R	x	x	x	x	x	x	x	x
Timer D control register	TDCON	E9H	R/W	0	0	0	0	0	0	0	0
Timer D clock pre-scalar	TDPS	EAH	R/W	–	–	–	–	0	0	0	0
Timer D data register	TDDATA	EBH	R/W	1	1	1	1	1	1	1	1
Timer D counter	TDCNT	ECH	R	x	x	x	x	x	x	x	x
Locations EDH- F1H are not mapped											
Reset source indicating register	RESETID	F2H	R	Refer to the detail description							
Location F3H is not mapped											
STOP control register	STOPCON	F4H	R/W	0	0	0	0	0	0	0	0
Flash memory control register	FMCON	F5H	R/W	0	0	0	0	0	–	–	0
Flash memory user programming enable register	FMUSR	F6H	R/W	0	0	0	0	0	0	0	0
Flash memory sector address register (high byte)	FMSECH	F7H	R/W	0	0	0	0	0	0	0	0
Flash memory sector address register (low byte)	FMSECL	F8H	R/W	0	0	0	0	0	0	0	0
Locations F9H – FFH are not mapped											

NOTE: – : Not mapped or not used, read '0'; x: Undefined

NOTES

9 I/O PORTS

OVERVIEW

The S3F84B8 microcontroller has three bit-programmable I/O ports, P0-P2. This gives a total of 17 I/O 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 a general overview of the S3F84B8 I/O port functions.

Table 9-1. S3F84B8 Port Configuration Overview

Port	Configuration Options
0	I/O port with bit-programmable pins. Configurable to input or push-pull output mode. Pull-up resistors can be assigned by software. Pins can also be assigned individually as alternative function pins.
1	I/O port with bit-programmable pins. Configurable to input or push-pull output mode. Pull-up resistors can be assigned by software. Pins can also be assigned individually as alternative function pins.
2	I/O port with bit-programmable pins. Configurable to input mode, push-pull output mode. Pins can also be assigned individually as alternative function pins.

For better EFT performance, when P10, P11, P12, P24, and P25 (with alternative function as comparator input) are configured as input pins, it is recommended to add 102pF capacitor externally.

PORT DATA REGISTERS

Table 9-2 gives you an overview of the register locations of all three S3F84B8 I/O port data registers. Data registers for ports 0, 1, and 2 have the general format shown in Figure 9-1.

Table 9-2. Port Data Register Summary

Register Name	Mnemonic	Decimal	Hex	Location	R/W
Port 0 data register	P0	224	E0H	Set1, Bank0	R/W
Port 1 data register	P1	225	E1H	Set1, Bank0	R/W
Port 2 data register	P2	226	E2H	Set1, Bank0	R/W

PORT 0

Port 0 is a 6-bit I/O Port that you can use in two ways:

— General-purpose I/O
— Alternative function

Port 0 is accessed directly by writing or reading the port 0 data register, P0 at location E0H, Set1 Bank0.

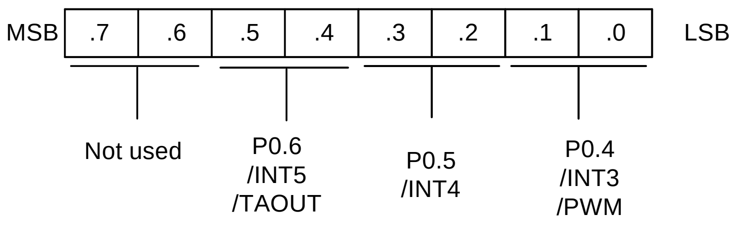
Port 0 Control Register (P0CONH, P0CONL)

Port 0 pins are configured individually by bit-pair settings in two control registers located:
P0CONH(high byte, E4H, Set1 Bank0) and P0CONL(low byte, E3H, Set1 Bank0).

When you select output mode, a push-pull or an open-drain circuit is configured. Many different selections are available:

— Input mode.
— Output mode(Push-pull or Open-drain)
— Alternative function: External Interrupt – INT0, INT1,INT2, INT3, INT4, INT5
— Alternative function: BUZ output- BUZ
— Alternative function: PWM output- PWM
— Alternative function: Timer A output – TAOUT
— Alternative function: RESETB (by smart option)

Port 0 High Control Register (P0CONH)
E3H, Set1, Bank0, R/W, Reset value:00H



.7 -.6 bit

XX	Not used for S3F84B8

.5 .4 bit/P0.6/INT5/TAOUT

00	Input mode/INT5 falling edge interrupt
01	Input mode with pull-up/INT5 falling edge interrupt
10	Push-pull output
11	Alternative function: TAOUT

.3 .2 bit/P0.5/INT4

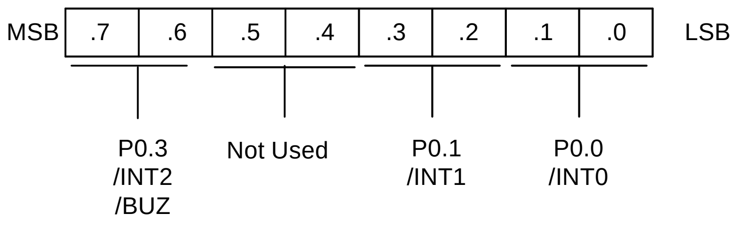
00	Input mode/INT4 falling edge interrupt
01	Input mode with pull-up/INT4 falling edge interrupt
10	Push-pull output
11	Open-drain output

.1 .0 bit/P0.4/INT3/PWM

00	Input mode/INT3 falling edge interrupt
01	Input mode with pull-up; INT3 falling edge interrupt
10	Push-pull output
11	Alternative function: PWM output

Figure 9-1. Port 0 Control Register High byte (P0CONH)

Port 0 Low Control Register (P0CONL)
E4H, Set1, Bank0, R/W, Reset value:00H



.7 -.6 bit/P0.3/INT2/BUZ

00	Input mode/INT2 falling edge interrupt
01	Input mode with pull up/INT2 falling edge interrupt
10	Push-pull output
11	Alternative function: BUZ output

.5 .4 bit Not used for S3F84B8

.3 .2 bit/P0.1/INT1

00	Input mode/INT1 falling edge interrupt
01	Input mode with pull-up; INT1 falling edge interrupt
10	Push-pull output
11	Open-drain output

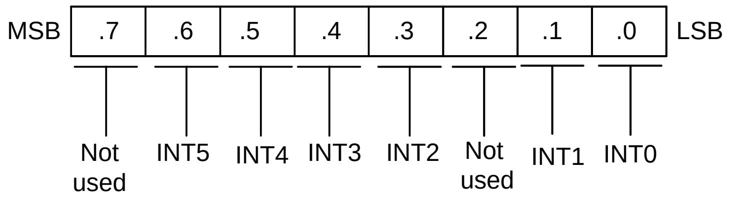
.1 .0 bit/P0.0/INT0

00	Input mode/INT0 falling edge interrupt
01	Input mode with pull-up; INT0 falling edge interrupt
10	Push-pull output
11	Open-drain output

Note: P1.2 could be used as either nRESET pin or normal input pin.

Figure 9-2. Port 0 Control Register Low byte (P0CONL)

Port 0 External Interrupt Register (P0INT)
E3H, Set1, Bank0, R/W, Reset value:00H



.7 bit Not used for S3F84B8

.6 bit INT5 Interrupt Enable/Disable Selection

0	Interrupt disable
1	Interrupt enable

.5 bit INT4 Interrupt Enable/Disable Selection

0	Interrupt disable
1	Interrupt enable

.4 bit INT3 Interrupt Enable/Disable Selection

0	Interrupt disable
1	Interrupt enable

.3 bit INT2 Interrupt Enable/Disable Selection

0	Interrupt disable
1	Interrupt enable

.2 bits Not used for S3F84B8

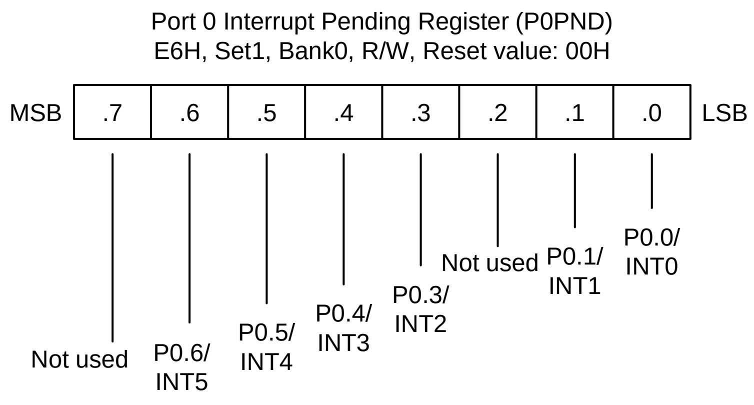
.1 bit INT1 Interrupt Enable/Disable Selection

0	Interrupt disable
1	Interrupt enable

.0 bit INT0 Interrupt Enable/Disable Selection

0	Interrupt disable
1	Interrupt enable

Figure 9-3. Port 0 Interrupt Control Register (P0INT)



P0.n bit configuration settings:

0	No interrupt pending (when read)
0	Pending bit clear (when write)
1	Interrupt is pending (when read)
1	No effect (when write)

NOTE: "n" is 0, 1, 3, 4, 5 or 6

Figure 9-4. Port 0 Interrupt Pending Register (P0PND)

PORT 1

Port 1 is an3-bit I/O port that you can use in two ways:

— General-purpose I/O
— Alternative function

Port 1 is accessed directly by writing or reading the port 1 data register, P1 at location E1H, Set1 Bank0.

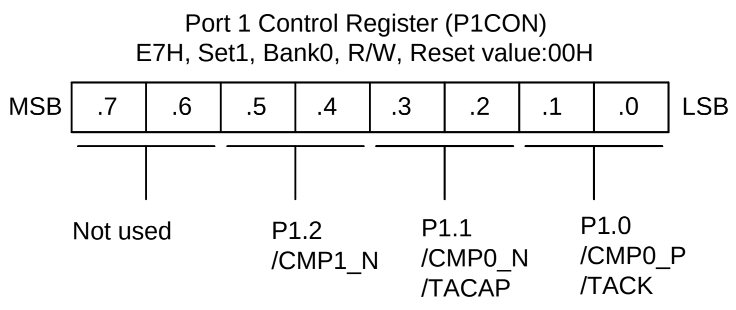
Port 1 Control Register (P1CON)

Port 1 pins are configured by settings in one control registers located:

P1CON (E7H, Set1 Bank0)

When you select output mode, push-pull circuit could be configured. In input mode, pull-up resister could be configured on or off. For alternative functions, many different selections are available:

— Input mode.
— Output mode(Push-pull)
— Alternative function: Timer A- TACK, TACAP
— Alternative function: Comparator-CMP0_N, CMP0_P, CMP1_N



.5 .4 bit/P1.2/CMP1_N

00	Input mode
01	Input mode with pull up resistor
10	Push-pull output
11	Alternative function: CMP1 negative input

.3 .2 bit/P1.1/CMP0_N_TACAP

00	Input mode/TACAP input
01	Input mode with pull-up/TACAP input
10	Push-pull output
11	Alternative function: CMP0 negative input

.1 .0 bit/P1.0/CMP0_P

00	Input mode/TACK input
01	Input mode with pull-up/TACK input
10	Push-pull output
11	Alternative function: CMP0 positive input

Figure 9-5. Port 1 Control Register (P1CON)

Port 2

Port 2 is an 8-bit I/O port that you can use in two ways:

- General-purpose I/O
- Alternative function

Port 2 is accessed directly by writing or reading the port 2 data register, P2 at location E2H, Set1 Bank0.

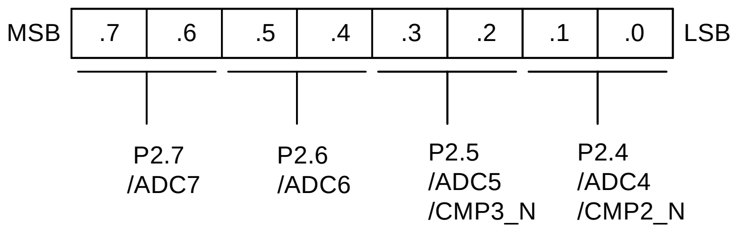
Port 2 Control Register (P2CONH, P2CONL)

Port 2 pins are configured individually by bit-pair settings in two control registers located:
P2CONL (low byte, E9H, Set1 Bank0) and P2CONH (high byte, E8H, Set1 Bank0).

When you select output mode, a push-pull circuit is configured. In input mode, pull-up resistor could be configured on or off. Many other different selections are available:

- Input mode.
- Output mode(Push-pull, Open-drain)
- Alternative function: ADC – ADC0-ADC7 analog input
- Alternative function: CMP2 – CMP2_N
- Alternative function: CMP3 – CMP3_N

Port 2 Control Register, High Byte (P2CONH)
E8H, Set1, Bank0, R/W, Reset value:00H



.7 .6 bit/P2.7/ADC7

00	Input mode
01	Input mode with pull-up
10	Push-pull output
11	Alternative function: ADC7 input

.5 .4 bit/P2.6/ADC6

00	Input mode
01	Input mode with pull-up
10	Push-pull output
11	Alternative function: ADC6 input

.3 .2 bit/P2.5/ADC5/CMP3_N

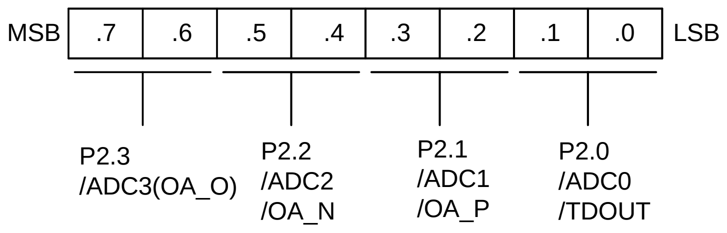
00	Input mode
01	Alternative function: CMP 3 negative input
10	Push-pull output
11	Alternative function: ADC5 input

.1 .0 bit/P2.4/ADC4/CMP2_N

00	Input mode
01	Alternative function: CMP2 negative input
10	Push-pull output
11	Alternative function: ADC4 input

Figure 9-6. Port 2 High-Byte Control Register (P2CONH)

Port 2 Control Register, Low Byte (P2CONL)
E8H, Set1, Bank0, R/W, Reset value:00H



.7 .6 bit/P2.3/ADC3(OA_O)

00	Input mode
01	Input mode with pull-up
10	Push-pull output
11	Alternative function: ADC3 input

.5 .4 bit/P2.2/ADC2/OA_N

00	Input mode
01	Alternative function: OPAMP negative input
10	Push-pull output
11	Alternative function: ADC2 input

.3 .2 bit/P2.1/ADC1/OA_P

00	Input mode
01	Alternative function: OPAMP positive input
10	Push-pull output
11	Alternative function: ADC1 input

.1 .0 bit/P2.0/ADC0/TDOUT

00	Input mode
01	Alternative function: TDOUT
10	Push-pull output
11	Alternative function: ADC0 input

Note: when OP AMP is enabled, P2CON.3 must be configured as ADC input no matter you want to use the internal ADC or not.

Figure 9-7. Port 2 Low-Byte Control Register (P2CONL)

NOTES

10 BASIC TIMER

OVERVIEW

Basic Timer (BT)

You can use the basic timer (BT) in two different ways:

— As a watchdog timer to provide an automatic reset mechanism in the event of a system malfunction.

— To signal 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 (FDH, Set1 Bank0, read-only)

— Basic timer control register, BTCON (D3H, Set1, read/write)

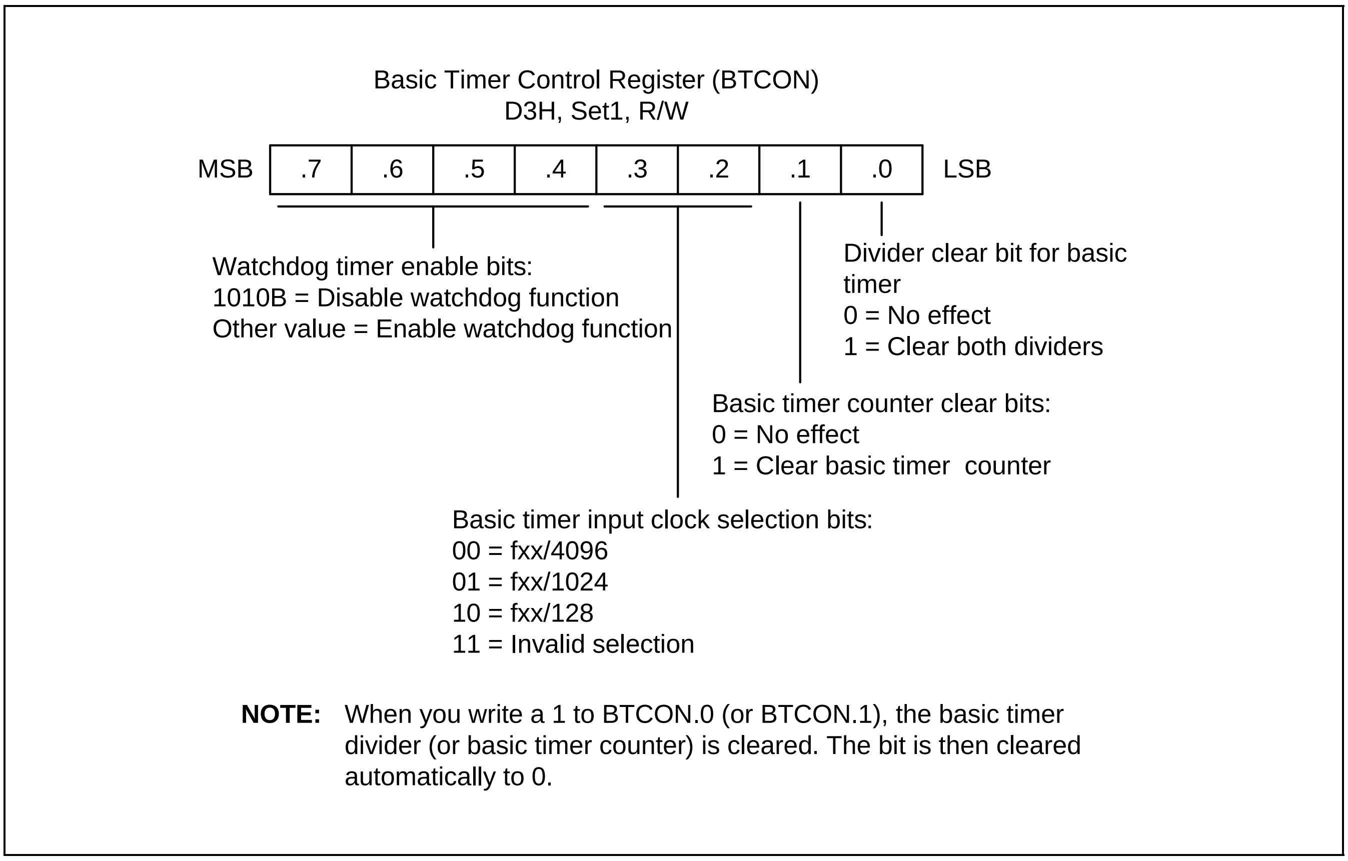
BASIC TIMER (BT)

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.

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, can be cleared during normal operation by writing a "1" to BTCON.1. To clear the frequency dividers for the basic timer input clock, you write a "1" to BTCON.0.


Basic Timer Control Register (BTCON)
D3H, Set1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Watchdog timer enable bits:
1010B = Disable watchdog function
Other value = Enable watchdog function
Divider clear bit for basic timer
0 = No effect
1 = Clear both dividers
Basic timer counter clear bits:
0 = No effect
1 = Clear basic timer counter
Basic timer input clock selection bits:
00 = fxx/4096
01 = fxx/1024
10 = fxx/128
11 = Invalid selection

NOTE: When you write a 1 to BTCON.0 (or BTCON.1), the basic timer divider (or basic timer counter) is cleared. The bit is then cleared automatically to 0.

Figure 10-1. Basic Timer Control Register (BTCON)

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 oscillator clock divided by 4096 as the BT clock.

A reset whenever a basic timer counter overflow occurs. During 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.

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 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

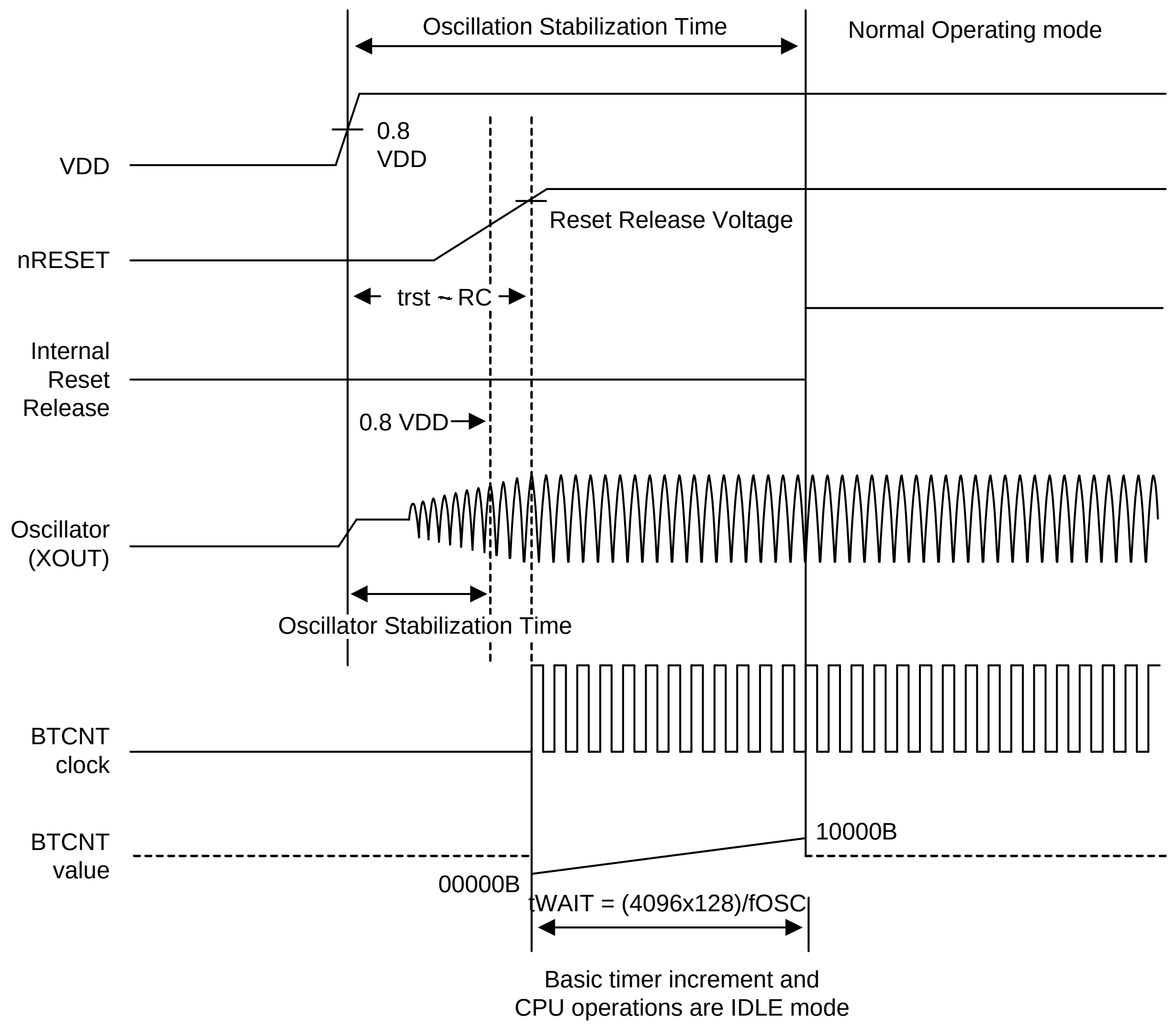
You can also use the basic timer to program a specific oscillation stabilization interval following a reset or when Stop mode has been released by an external interrupt.

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.7 is set, 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 normal operation.

In summary, the following events occur when Stop mode is released:

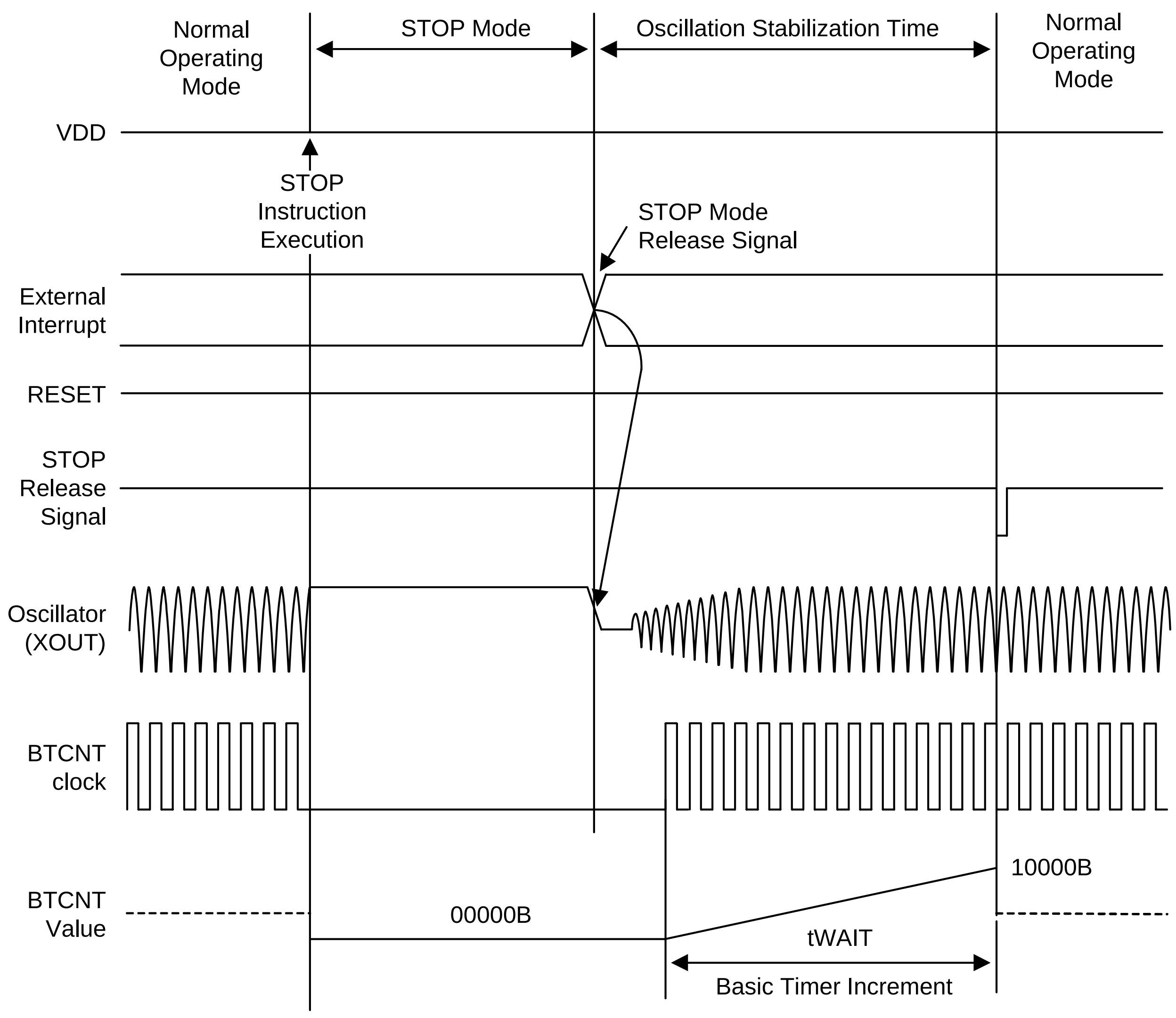
1. During Stop mode, an external power-on reset or an external interrupt occurs to trigger the Stop mode release and oscillation starts.
2. If an external power-on reset occurred, the basic timer counter will increase at the rate of $f_{OSC}/4096$. If an external interrupt is used to release Stop mode, 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 is set.
4. When a BTCNT.7 is set, normal CPU operation resumes.

Figure 10-2 and 10-3 shows the oscillation stabilization time on RESET and STOP mode release



NOTE: Duration of the oscillator stabilization wait time, tWAIT, when it is released by a Power-on-reset is 4096 x 128/fOSC.
tRST ≈RC (R and C are value of external power on reset)

Figure 10-2. Oscillation Stabilization Time on RESET



NOTE: Duration of the oscillator stabilzation wait time, tWAIT, it is released by an interrupt is determined by the setting in basic timer control register, BTCON.

BTCON.3	BTCON.2	tWAIT	tWAIT (When fOSC is 8 MHz)
0	0	(4096 x 128)/fosc	65.536 ms
0	1	(1024 x 128)/fosc	16.384 ms
1	0	(128 x 128)/fosc	2.048 ms
1	1	Invalid setting	—

Figure 10-3. Oscillation Stabilization Time on STOP Mode Release

PROGRAMMING TIP — Configuring the Basic Timer

This example shows how to configure the basic timer to sample specification.

```
            ORG         0000H

;--------------<< Smart Option >>

            ORG         003CH
            DB          0FFH                    ;   003CH, must be initialized to 0FF
            DB          0FFH                    ;   003DH, must be initialized to 0FF
            DB          0FFH                    ;   003EH, must be initialized to 0FF
            DB          0FFH                    ;   003FH, LVR enable, nRESET pin enable

;--------------<< Initialize System and Peripherals >>

            ORG         0100H

RESET:      DI                                  ;   Disable interrupt
            LD          CLKCON, #00011000B      ;   Select non-divided CPU clock
            LD          SPL, #0FFH              ;   Stack pointer must be set
            •
            •
            LD          BTCON, #02H             ;   Enable watchdog function
                                                ;   Basic timer clock: fOSC/4096
                                                ;   Basic counter (BTCNT) clear
            •
            •
            •
            EI                                  ;   Enable interrupt

;--------------<< Main loop >>

MAIN:       •
            LD          BTCON, #02H             ;   Enable watchdog function
                                                ;   Basic counter (BTCNT) clear
            •
            •
            •
            JR          T, MAIN                 ;
```

NOTES

11 8-BIT TIMER A

OVERVIEW

The 8-bit timer A is an 8-bit general-purpose timer/counter. Timer A has three operating modes. You can select one of them using the appropriate TACON setting:

— Interval timer mode (Toggle output at TAOUT pin)

— Capture input mode with a rising or falling edge trigger at the TACAP pin

— PWM mode (TAOUT)

Timer A has the following functional components:

— Precalar for clock frequency programmable from fx to fx/4096

— External clock input pin (TACK)

— 8-bit counter (TACNT), 8-bit comparator, and 8-bit reference data register (TADATA)

— I/O pins for capture input (TACAP) or PWM or match output (TAOUT)

— Timer A overflow interrupt and match/capture interrupt generation

— Timer A control register, TACON (E1H, Set1 Bank1, read/write)

FUNCTION DESCRIPTION

Timer A Interrupts

The timer A module can generate two interrupts: the timer A overflow interrupt (TAOVF), and the timer A match/ capture interrupt (TAINT).

Timer A overflow interrupt can be cleared by both software and hardware, and match/capture interrupt pending conditions are cleared by software when it has been serviced.

Interval Timer Function

The timer A module can generate an interrupt: the timer A match interrupt (TAINT).

When timer A interrupt occurs and is serviced by the CPU, the pending condition should be cleared by software.

In interval timer mode, a match signal is generated and TAOUT is toggled when the counter value is identical to the value written to the Timer A reference data register, TADATA. The match signal generates a timer A match interrupt and clears the counter.

If, for example, you write the value 10H to TADATA and 0BH to TACON, the counter will increment until it reaches 10H. At this point the TA interrupt request is generated; the counter value is reset, and counting resumes.

Pulse Width Modulation Mode

Pulse width modulation (PWM) mode lets you program the width (duration) of the pulse that is output at the TAOUT pin. As in interval timer mode, a match signal is generated when the counter value is identical to the value written to the timer A data register. In PWM mode, however, the match signal does not clear the counter. Instead, it runs continuously, overflowing at FFH, and then continues incrementing from 00H.

Although you can use the match signal to generate a timer A overflow interrupt, interrupts are not typically used in PWM-type applications. Instead, the pulse at the TAOUT pin is held to Low level as long as the reference data value is *less than or equal to* ($\leq$) the counter value and then the pulse is held to High level for as long as the data value is *greater than* (>) the counter value. One pulse width is equal to $t_{CLK} \bullet 256$.

Capture Mode

In capture mode, a signal edge that is detected at the TACAP pin opens a gate and loads the current counter value into the Timer A data register. You can select rising or falling edges to trigger this operation.

Timer A also gives you capture input source: the signal edge at the TACAP pin. You select the capture input by setting the value of the timer A capture input selection bit in P1CON, (E7H, Set1 Bank0).
When P1CON.3.2 is 00 and 01, the TACAP input or normal input is selected. When P1CON.2.2 is set to 10 and 11, output is selected.

Both kinds of timer A interrupts can be used in capture mode: the timer A overflow interrupt is generated whenever a counter overflow occurs; the timer A match/capture interrupt is generated when the counter value is loaded into the Timer A data register.

By reading the captured data value in TADATA, and assuming a specific value for the timer A clock frequency, you can calculate the pulse width (duration) of the signal that is being input at the TACAP pin.

TIMER A CONTROL REGISTER (TACON)

You use the timer A control register, TACON

- Select the timer A operating mode (interval timer, capture mode and PWM mode)
- Clear the timer A counter, TACNT
- Enable the timer A overflow interrupt or timer A match/capture interrupt
- Timer A start/stop
- Clear Timer A match/capture interrupt pending conditions

You can use timer A prescaler register, TAPS to

- Select clock source. Internal or external clock source
- Program clock prescaler

TACON is located at address E1H, Set1 Bank1, and is read/write addressable using Register addressing mode.

A reset clears TACON to '00H'. This sets timer A to normal interval timer mode, and disables all Timer A Interrupts. You can clear the timer A counter at any time during normal operation by writing a "1" to TACON.5. You can start Timer A counter by writing a "1" to TACON.2.

The timer A overflow interrupt (TAOVF) has the vector address D0H. When a timer A overflow interrupt occurs and is serviced by the CPU, and the pending condition can be cleared by both software and hardware.

To enable timer A match/capture interrupt, you must write TACON.3 to "1". To generate the exact time interval, you should write TACON.5 and TACON.1, which clears counter and interrupt pending bit. When interrupt service routine is served, the pending condition must be cleared by software by writing a '0' to the interrupt pending bit.

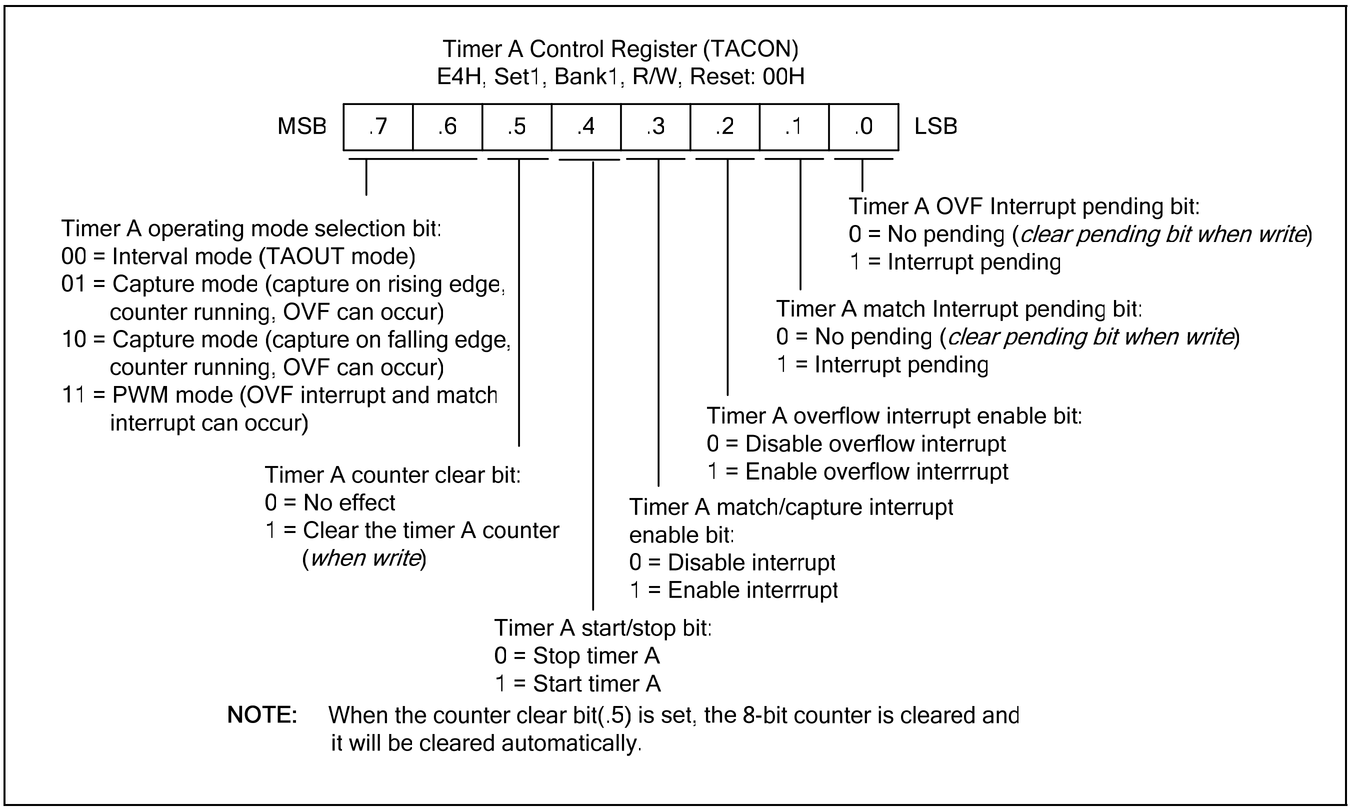

Timer A Control Register (TACON)
E4H, Set1, Bank1, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Timer A operating mode selection bit:
00 = Interval mode (TAOUT mode)
01 = Capture mode (capture on rising edge, counter running, OVF can occur)
10 = Capture mode (capture on falling edge, counter running, OVF can occur)
11 = PWM mode (OVF interrupt and match interrupt can occur)
Timer A counter clear bit:
0 = No effect
1 = Clear the timer A counter (*when write*)
Timer A start/stop bit:
0 = Stop timer A
1 = Start timer A
Timer A match/capture interrupt enable bit:
0 = Disable interrupt
1 = Enable interrrupt
Timer A overflow interrupt enable bit:
0 = Disable overflow interrupt
1 = Enable overflow interrrupt
Timer A match Interrupt pending bit:
0 = No pending (*clear pending bit when write*)
1 = Interrupt pending
Timer A OVF Interrupt pending bit:
0 = No pending (*clear pending bit when write*)
1 = Interrupt pending
NOTE: When the counter clear bit(.5) is set, the 8-bit counter is cleared and it will be cleared automatically.

Figure 11-1. Timer A Control Register (TACON)

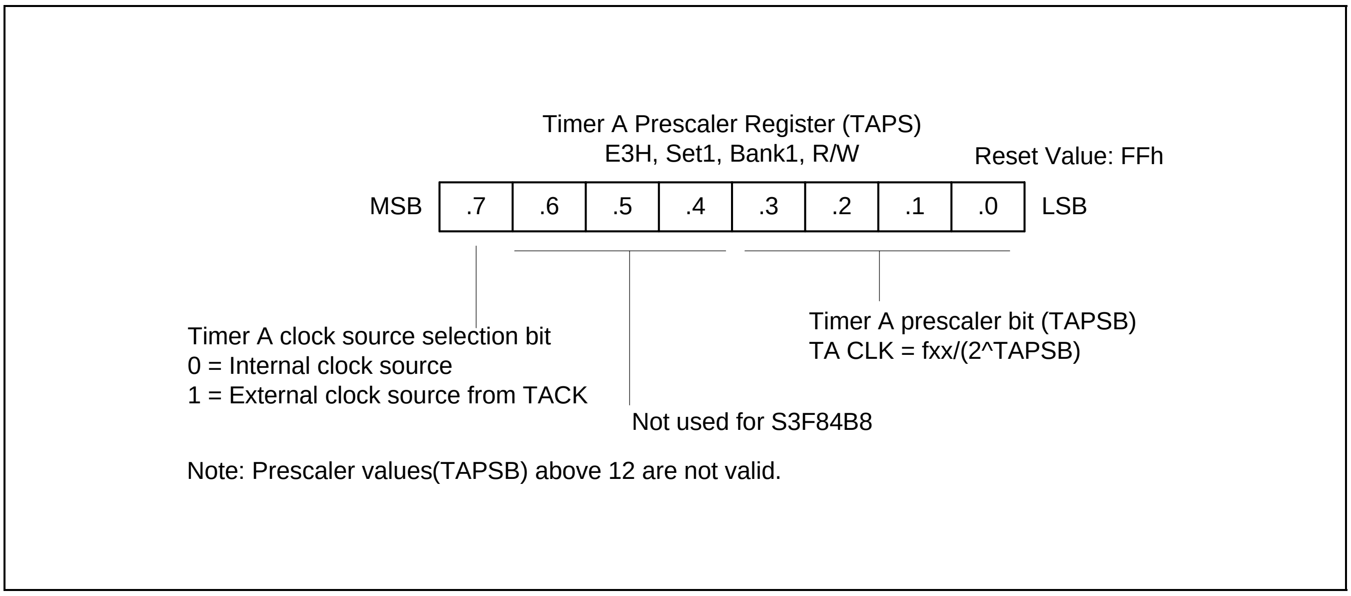

Timer A Prescaler Register (TAPS)
E3H, Set1, Bank1, R/W
Reset Value: FFh
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Timer A clock source selection bit
0 = Internal clock source
1 = External clock source from TACK
Not used for S3F84B8
Timer A prescaler bit (TAPSB)
TA CLK = fxx/(2^TAPSB)
Note: Prescaler values(TAPSB) above 12 are not valid.

Figure 11-3. Timer A Prescaler Register (TAPS)

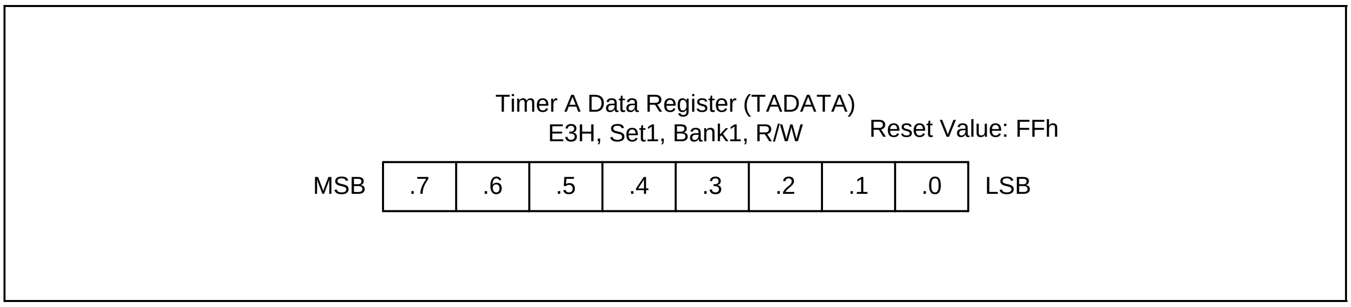


Figure 11-4. Timer A DATA Register (TADATA)

BLOCK DIAGRAM

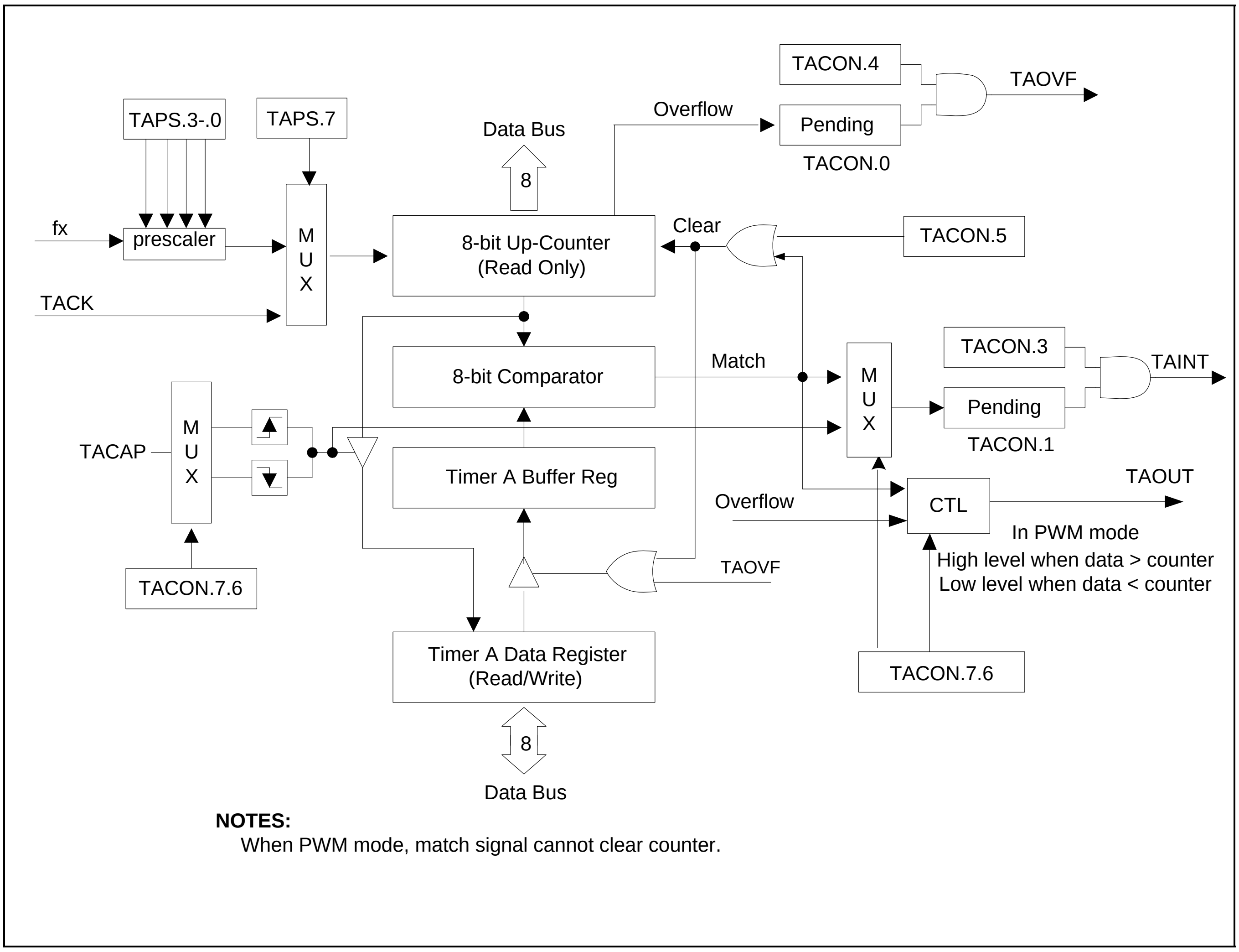


Figure 11-4. Simplified timer A Functional Block Diagram

NOTES

12 TIMER 0

ONE 16-BIT TIMER MODE (TIMER 0)

The 16-bit timer 0 is used in one 16-bit timer or two 8-bit timers mode. If TCCON.7 is set to "1", Timer 0 is used as a 16-bit timer. If TCCON.7 is set to "0", timer 0 is used as two 8-bit timers.

— One 16-bit timer mode (Timer 0)

— Two 8-bit timers mode (Timer C and D)

OVERVIEW

The 16-bit timer 0 is a 16-bit general-purpose timer. Timer 0 has the interval timer mode by using the appropriate TCCON setting.

Timer 0 has the following functional components:

— Precalar for clock frequency programmable from fx to fx/4096

— 16-bit comparator, and 16-bit reference data register (TCDATA, TDDATA)

— Timer 0 match interrupt generation (Interrupt vector address: E0H)

— Timer 0 control register, TCCON (E5H, Set1 Bank1, read/write)

FUNCTION DESCRIPTION

Interval Timer Function

The timer 0 module can generate timer 0 match interrupt (TCINT). TCINT pending bit will be set whenever the match condition is met, in spite of global interrupt and peripheral interrupt enable status. The TCINT pending condition should be cleared by software when it has been serviced.

In interval timer mode, a match signal is generated when the counter value is identical to the values written to the timer 0 reference data registers, TCDATA and TDDATA. The match signal generates a timer 0 match interrupt and clears the counter.

If, for example, you write value 32H and 10H to TCDATA and TDDATA, respectively, and B8H to TCCON, the counter will increment until it reaches 3210H. At this point the timer 0 interrupt request is generated; the counter value is reset, and counting resumes thereafter.

Timer 0 Control Register (TCCON)

You can use timer 0 control register, TCCON, to

— Enable the timer 0 operating (interval timer)

— Clear the timer 0 counter

— Enable the timer 0 interrupt

— Clear timer 0 interrupt pending conditions

You can use timer0 prescaler register, TCPS to

— Select clock source. Comparator 0's output could be configured as the clock source of Timer0.

— Program clock prescaler

TCCON is located at address E5H, Set1 Bank1, and is read/write addressable using register addressing mode.

A reset clears TCCON to "00H". This sets timer 0 to work in a 16bit timer mode, and disables timer 0 interrupt. You can clear the timer 0 counter at any time during normal operation by writing a "1" to TCCON.5.

To enable the timer 0 interrupt, you must write '1' to TCCON.3.
To generate the exact time interval, you should write TCCON.5 and TCCON.1 to clear the counter and interrupt pending bit. When Timer 0 is disabled, interrupt pending bit can still be set when meets the interrupt condition. Application program can poll for the pending bit, TCCON.1. When a "1" is detected, a timer 0 interrupt is pending. The pending condition must be cleared by software by writing a "0" to the timer 0 interrupt pending bit, TCCON.1.

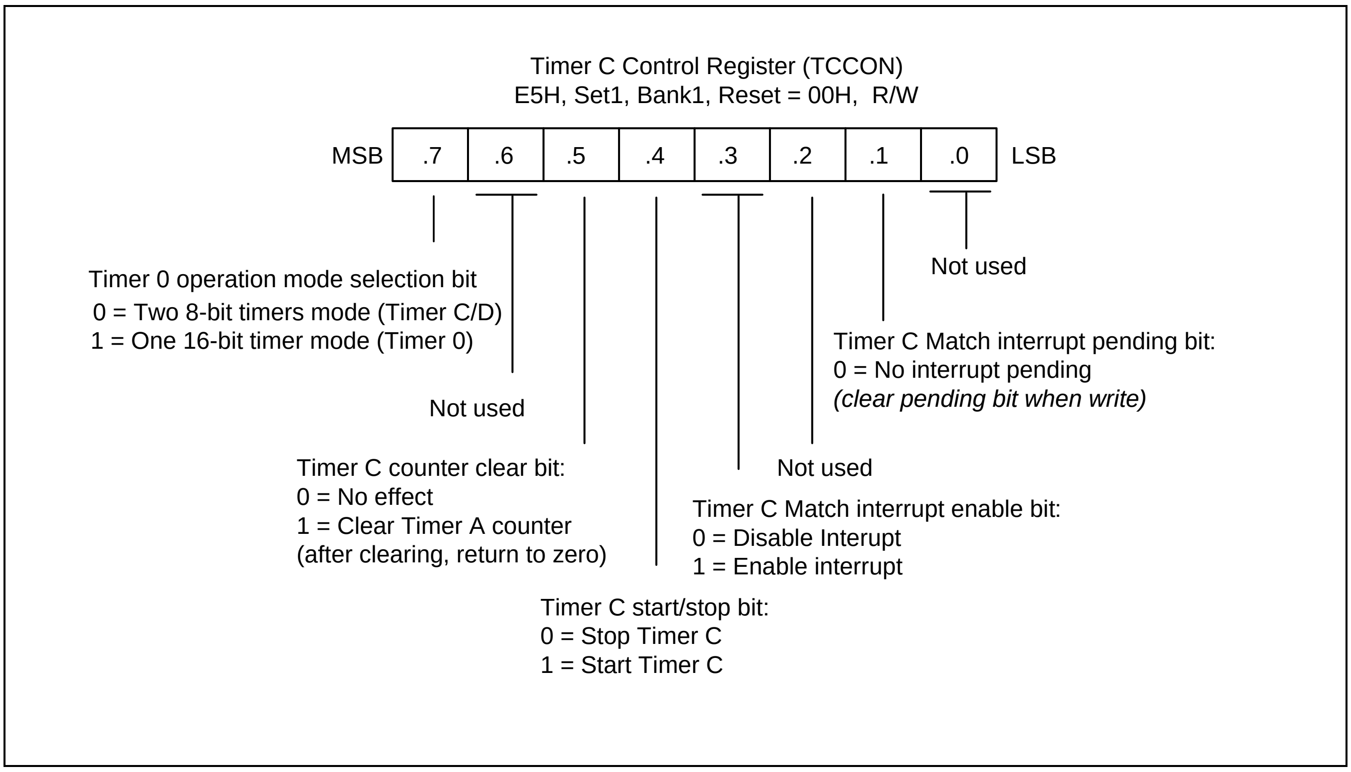

Timer C Control Register (TCCON)
E5H, Set1, Bank1, Reset = 00H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
Timer 0 operation mode selection bit
0 = Two 8-bit timers mode (Timer C/D)
1 = One 16-bit timer mode (Timer 0)
Timer C Match interrupt pending bit:
0 = No interrupt pending
(clear pending bit when write)
Not used
Timer C counter clear bit:
0 = No effect
1 = Clear Timer A counter
(after clearing, return to zero)
Not used
Timer C Match interrupt enable bit:
0 = Disable Interupt
1 = Enable interrupt
Timer C start/stop bit:
0 = Stop Timer C
1 = Start Timer C

Figure 12-1. Timer 0 Control Register (TCCON)

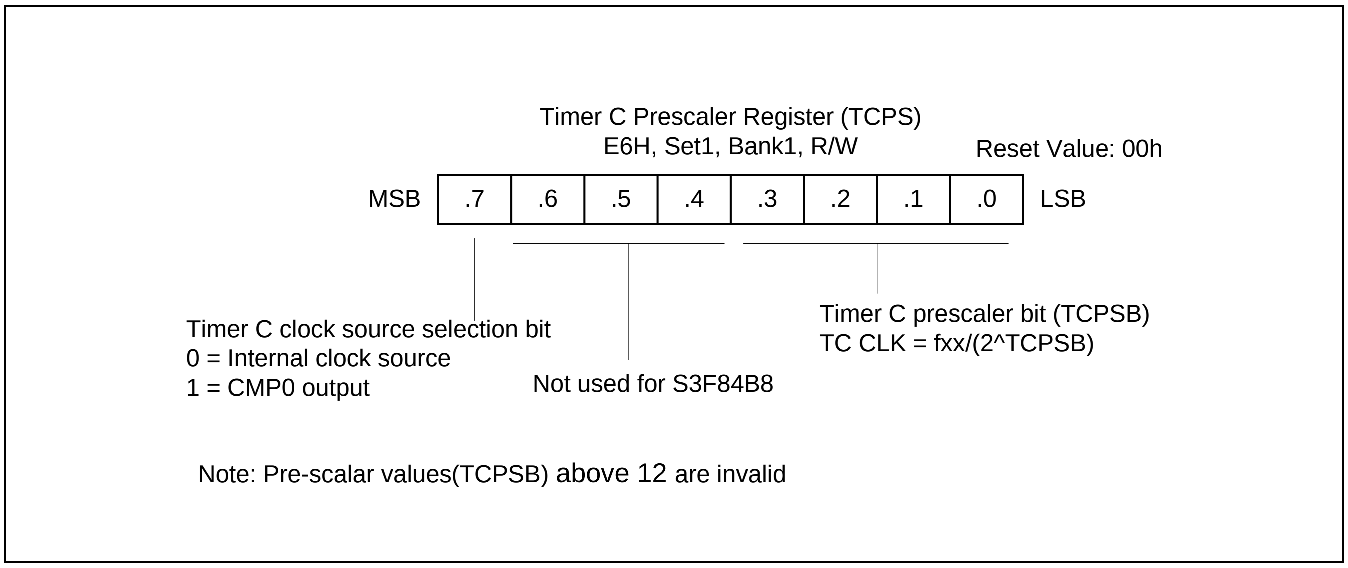
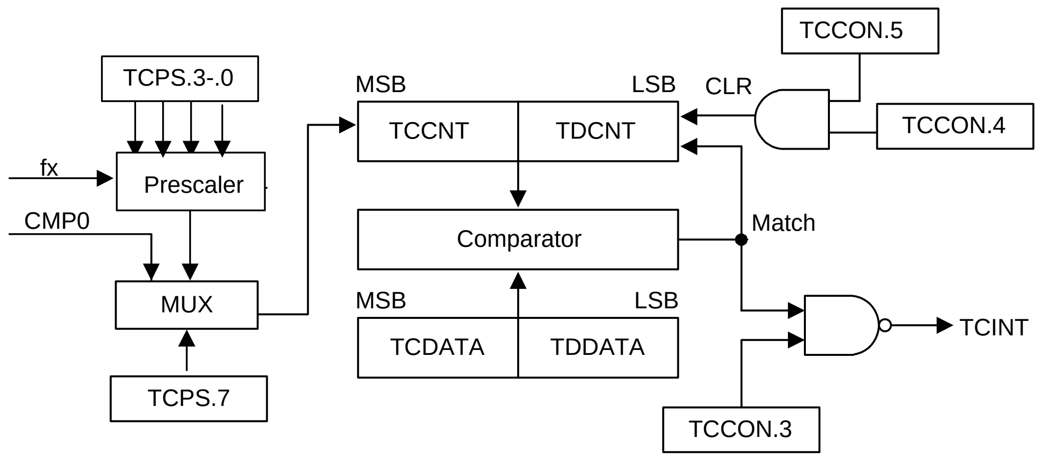

Timer C Prescaler Register (TCPS)
E6H, Set1, Bank1, R/W
Reset Value: 00h
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Timer C clock source selection bit
0 = Internal clock source
1 = CMP0 output
Not used for S3F84B8
Timer C prescaler bit (TCPSB)
TC CLK = fxx/(2^TCPSB)
Note: Pre-scalar values(TCPSB) above 12 are invalid

Figure 12-2. Timer 0 Prescaler Register (TCPS)

BLOCK DIAGRAM



NOTE: When TCCON.7 is '1', one 16-bit Timer 0.

Figure 12-3. Timer 0 Functional Block Diagram

TWO 8-BIT TIMERS MODE (TIMER C and D)

OVERVIEW

The 8-bit Timer C and D are the 8-bit general-purpose timers. Timer C have the interval timer mode, and the Timer D have the interval timer mode and PWM mode by using the appropriate TCCON and TDCON setting, respectively.

Timer C and D have the following functional components:

— Precalar for Timer C clock frequency programmable from fx to fx/4096
  Precalar for Timer D clock frequency programmable from fx to fx/4096
— 8-bit counter (TCCNT, TDCNT), 8-bit comparator, and 8-bit reference data register (TCDATA, TDDATA))
— Timer C match interrupt generation
— Timer C control register, TCCON (E5H, bank1, read/write)
— Timer D have I/O pin for match and PWM output (P2.0, TDOUT)
— Timer D overflow interrupt generation
— Timer D match interrupt generation
— Timer D control register, TDCON (E9H, bank1, read/write)

Timer C and D Control Register (TCCON, TDCON)

You can use the Timer C and D control register, TCCON and TDCON to

— Enable the Timer C (interval timer mode) and D operating (interval timer mode and PWM mode)
— Select the Timer C clock source
— Clear Timer C and D counter, TCCNT and TDCNT
— Enable the Timer C and D interrupt
— Clear Timer C and D interrupt pending conditions

You can use timerC prescaler register, TCPS to

— Select clock source. Comparator 0's output could be configured to the clock source of TimerC.
— Select clock prescaler

You can use timerD prescaler register, TDPS to

— Program clock prescaler

TCCON and TDCON are located in address E5H and E9H, Set1 Bank1, and are read/write addressable using register addressing mode.

A reset clears TCCON to "00H". This disables Timer C interrupt. You can clear Timer C counter at any time during normal operation by writing a "1" to TCCON.5.

A reset clears TDCON to "00H". This sets Timer D to work in interval timer mode, and disables Timer D interrupt. You can clear the Timer D counter at any time during normal operation by writing a "1" to TDCON.5.

To enable the Timer C interrupt (TCINT) and Timer D interrupt (TDINT) you must write TCCON.7 to "0", TCCON.3 (TDCON.3) to "1". To generate the exact time interval, you should write TCCON.5 (TDCON.5) and TCCON.1 (TDCON.1), which cleared counter and interrupt pending bit. To detect an interrupt pending condition when TCINT and TDINT are disabled the application program can poll for the pending bit, TCCON.1 and TDCON.1. When a "1" is detected, a Timer C interrupt (TCINT) or Timer D interrupt (TDINT) is pending. When the TCINT and TDINT sub-routine has been serviced, the pending condition must be cleared by software by writing a "0" to the Timer C and D interrupt pending bit, TCCON.1 andTDCON.1.

Also, to enable Timer D overflow interrupt (TDOVF), you must write TCCON.7 to "0", TDCON.2 to "1".

To generate the exact time interval, you should write TDCON.5 and TDCON.1, which cleared counter and interrupt pending bit.

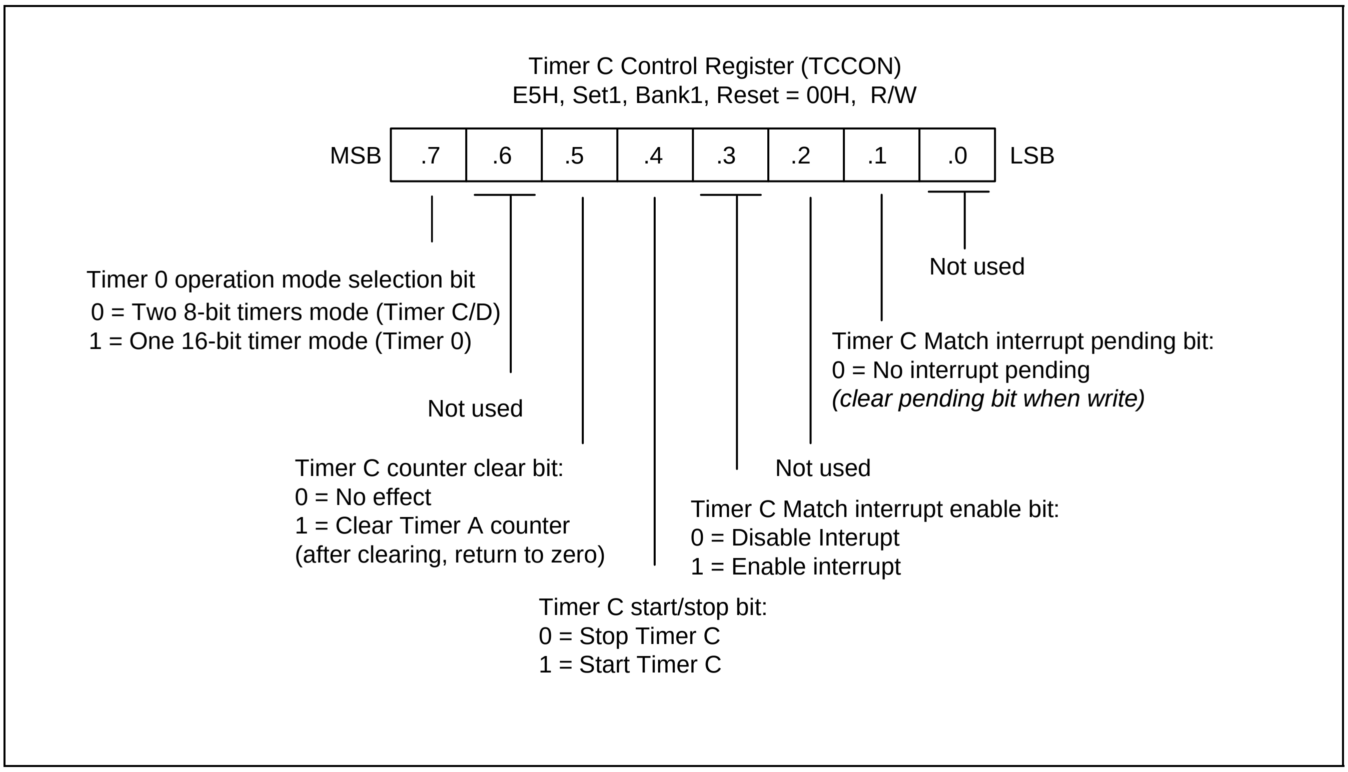

Timer C Control Register (TCCON)
E5H, Set1, Bank1, Reset = 00H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Timer 0 operation mode selection bit
0 = Two 8-bit timers mode (Timer C/D)
1 = One 16-bit timer mode (Timer 0)
Not used
Timer C counter clear bit:
0 = No effect
1 = Clear Timer A counter
(after clearing, return to zero)
Timer C start/stop bit:
0 = Stop Timer C
1 = Start Timer C
Timer C Match interrupt enable bit:
0 = Disable Interupt
1 = Enable interrupt
Not used
Timer C Match interrupt pending bit:
0 = No interrupt pending
(clear pending bit when write)
Not used

Figure 12-4. Timer C Control Register (TCCON)

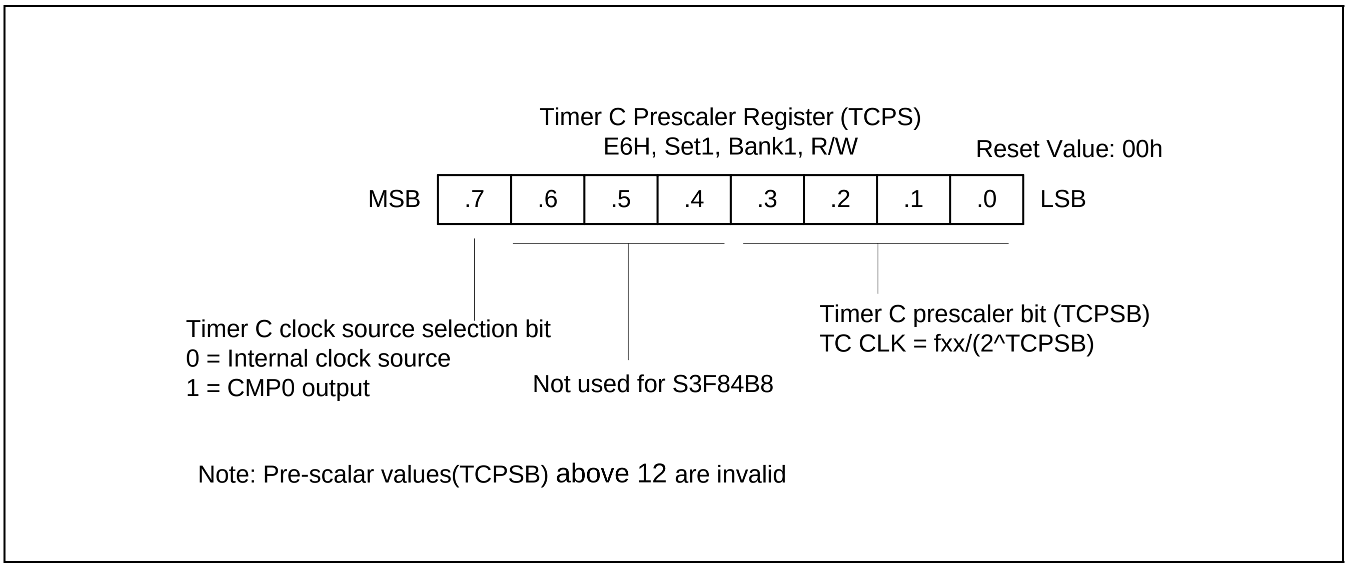

Timer C Prescaler Register (TCPS)
E6H, Set1, Bank1, R/W
Reset Value: 00h
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Timer C clock source selection bit
0 = Internal clock source
1 = CMP0 output
Not used for S3F84B8
Timer C prescaler bit (TCPSB)
TC CLK = fxx/(2^TCPSB)
Note: Pre-scalar values(TCPSB) above 12 are invalid

Figure 12-5. Timer C Prescaler Register (TCPS)

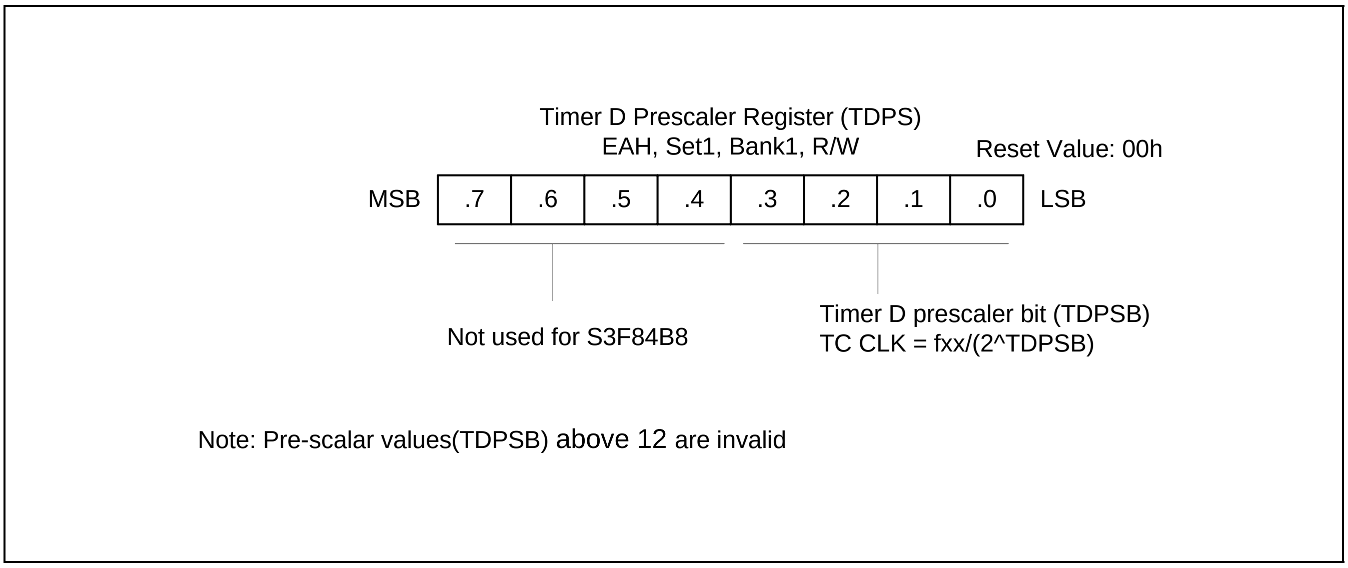

Timer D Prescaler Register (TDPS)
EAH, Set1, Bank1, R/W
Reset Value: 00h
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used for S3F84B8
Timer D prescaler bit (TDPSB)
TC CLK = fxx/(2^TDPSB)
Note: Pre-scalar values(TDPSB) above 12 are invalid

Figure 12-6. Timer D Prescaler Register (TDPS)

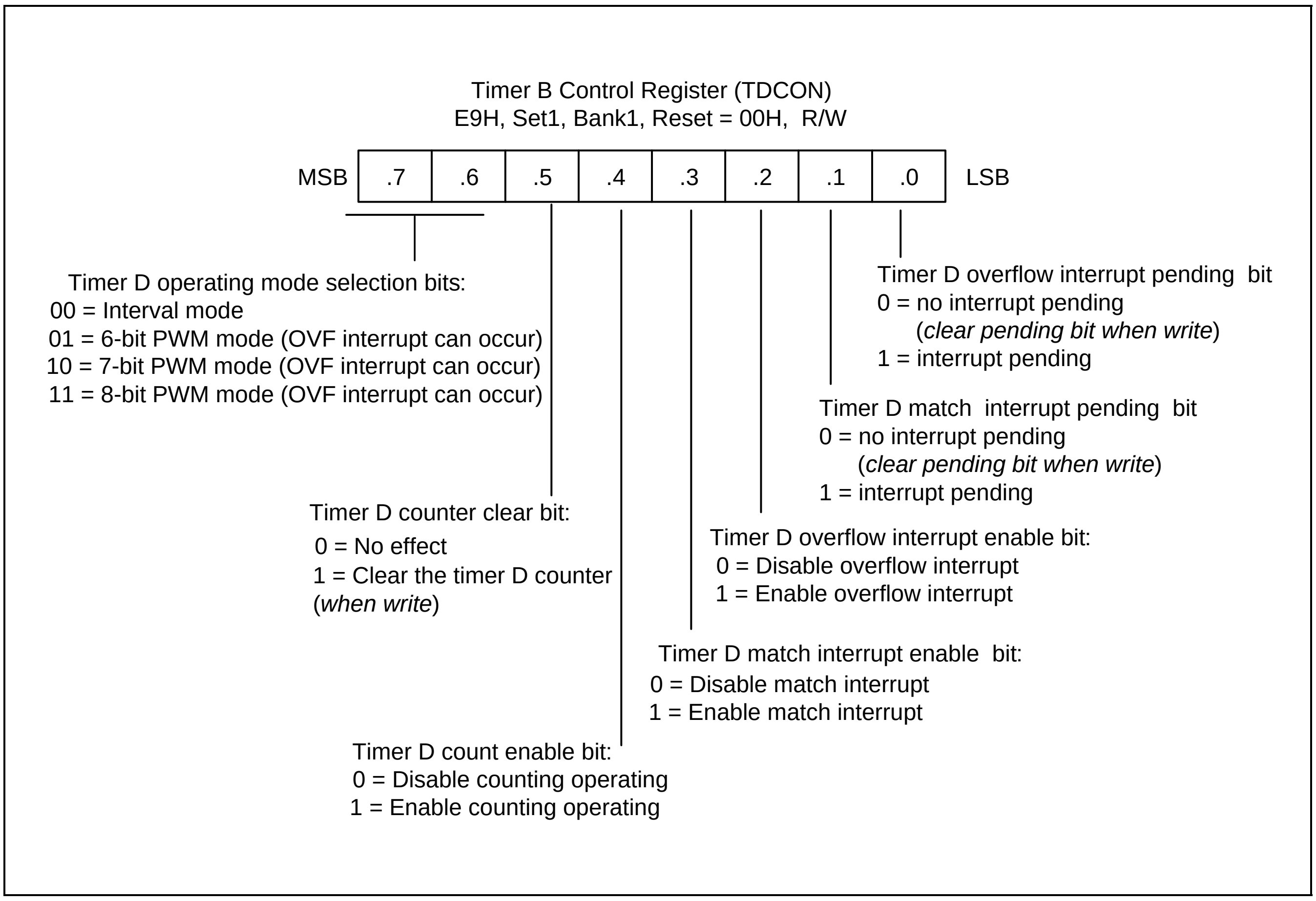

Timer B Control Register (TDCON)
E9H, Set1, Bank1, Reset = 00H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Timer D operating mode selection bits:
00 = Interval mode
01 = 6-bit PWM mode (OVF interrupt can occur)
10 = 7-bit PWM mode (OVF interrupt can occur)
11 = 8-bit PWM mode (OVF interrupt can occur)
Timer D counter clear bit:
0 = No effect
1 = Clear the timer D counter (*when write*)
Timer D count enable bit:
0 = Disable counting operating
1 = Enable counting operating
Timer D match interrupt enable bit:
0 = Disable match interrupt
1 = Enable match interrupt
Timer D overflow interrupt enable bit:
0 = Disable overflow interrupt
1 = Enable overflow interrupt
Timer D match interrupt pending bit
0 = no interrupt pending (*clear pending bit when write*)
1 = interrupt pending
Timer D overflow interrupt pending bit
0 = no interrupt pending (*clear pending bit when write*)
1 = interrupt pending

Figure 12-7. Timer D Control Register (TDCON)

FUNCTION DESCRIPTION

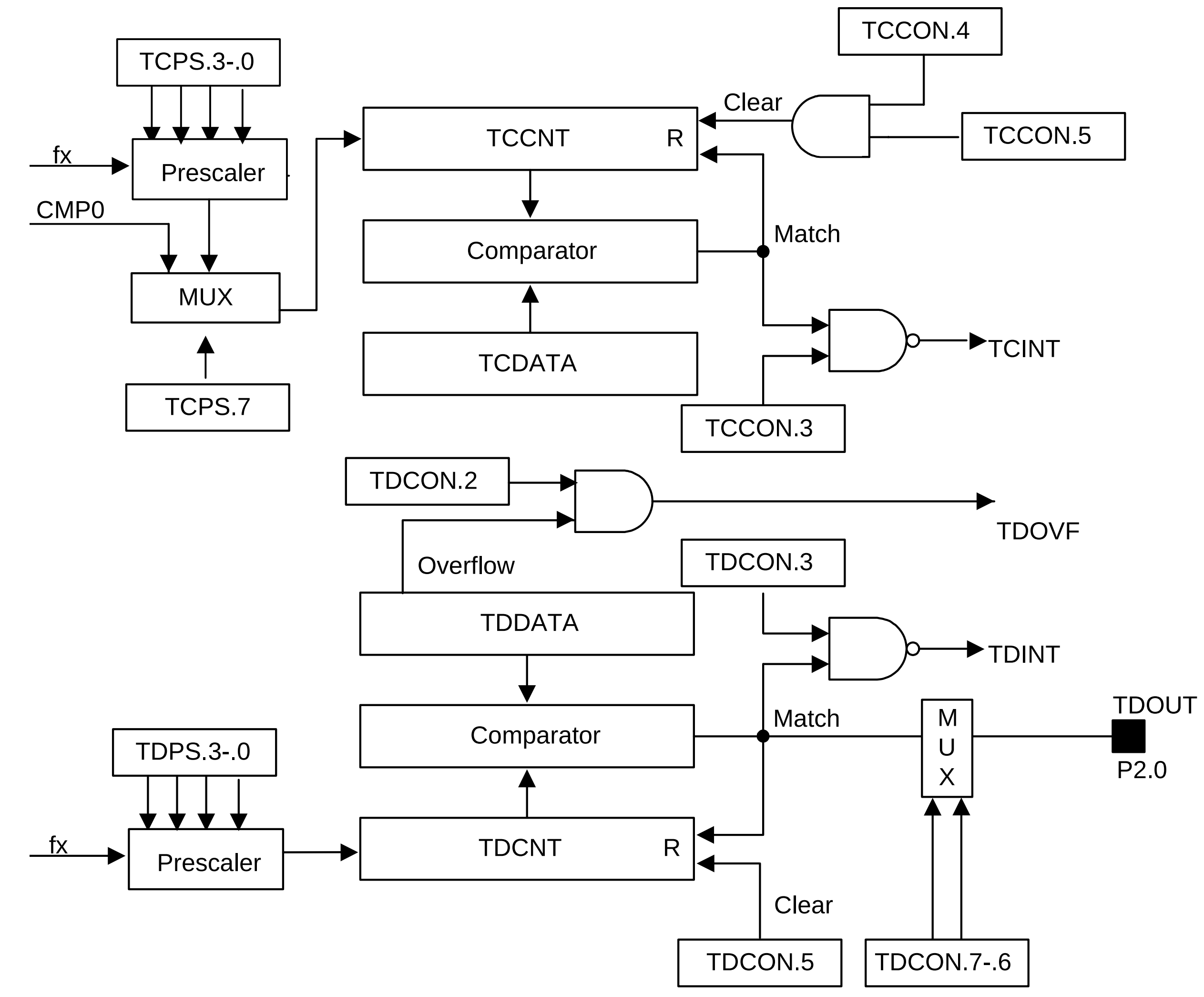
Interval Timer Function (Timer C and Timer D)

The Timer C and D module can generate an interrupt: the Timer C match interrupt (TCINT) and the Timer D match interrupt (TDINT). The Timer C match interrupt pending condition (TCCON.1) and the Timer D match interrupt pending condition (TDCON.1) must be cleared by software in the application's interrupt service by means of writing a "0" to the TCCON.1 and TDCON.1interrupt pending bit.

When the global interrupt is enabled, even though TCINT and TDINT are disabled, the application's service routine can detect a pending condition of TCINT and TDINT by the software and jump to execute the corresponding sub-routine.

In interval timer mode, a match signal is generated when the counter value is identical to the values written to the Timer C or Timer D reference data registers, TCDATA or TDDATA. The match signal generates corresponding match interrupt (TCINT, TDINT) and clears the counter.

If, for example, you write the value 20H to TCDATA and 38H to TCCON, the counter will increment until it reaches 20H. At this point, TD interrupt request is generated and the counter value is cleared and counting resumes.



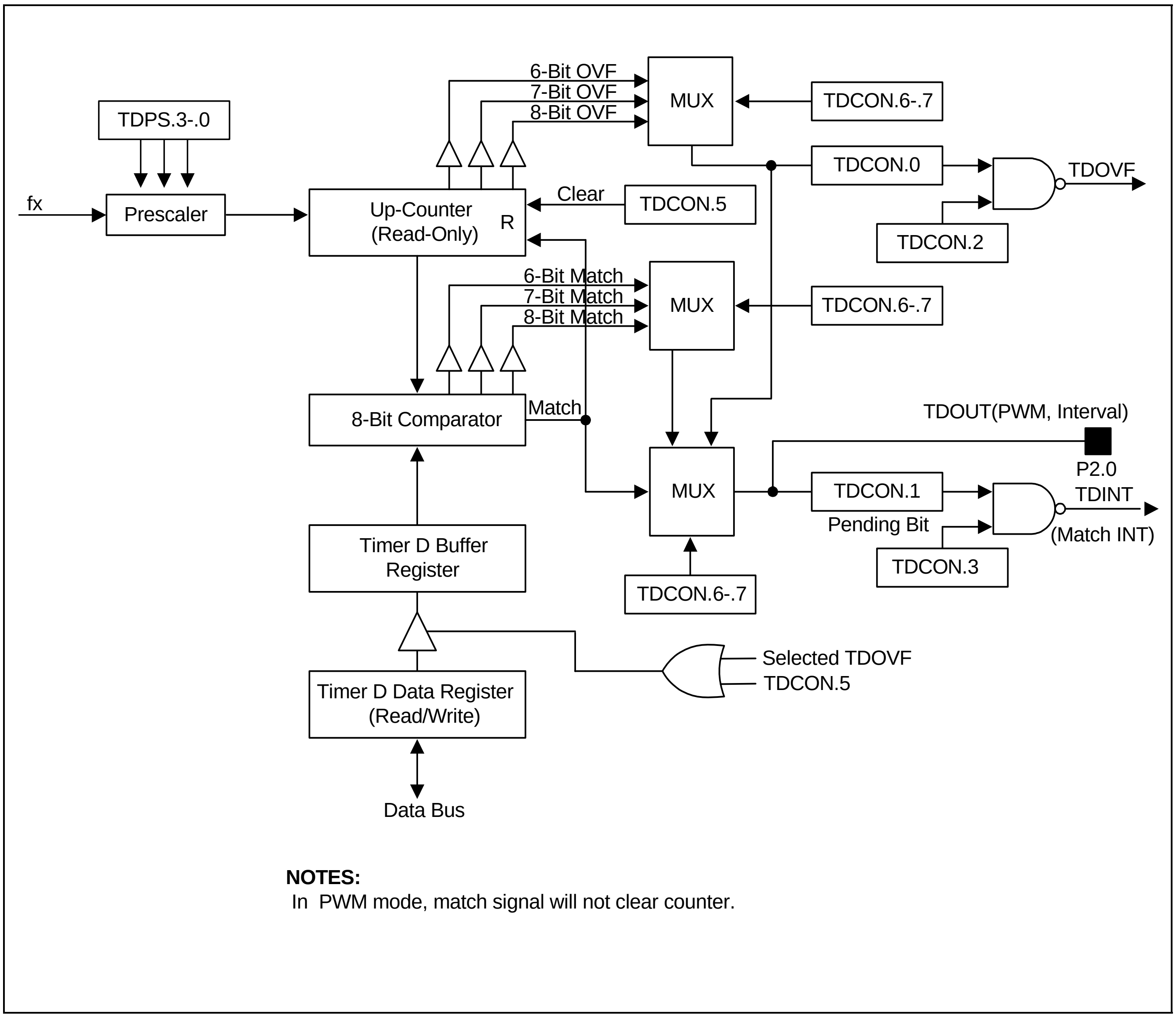
NOTE: When TCCON.7 is '0', two 8-bit timer C/D (Interval mode).

Figure 12-8. Timer C and D Function Block Diagram

Pulse Width Modulation Mode (Timer D)

Pulse width modulation (PWM) mode lets you program the width (duration) of the pulse that is output at the TDOUT (P2.0) pin. As in interval timer mode, a match signal is generated when the counter value is identical to the value written to the Timer D data register. In PWM mode, however, the match signal does not clear the counter. Instead, it runs continuously, overflowing at "FFH" in case of 8-bit PWM mode, and then continues incrementing from "00H".

Although you can use the match signal to generate a Timer D overflow interrupt, interrupts are not typically used in PWM-type applications. Instead, the pulse at the TDOUT pin is held to Low level as long as the reference data value is less than or equal to (≤) the counter value and then the pulse is held to High level for as long as the data value is greater than (>) the counter value. One pulse width is equal to $t_{CLK} \times 256$ in case of 8-bit PWM mode is selected (see Figure 12-6).



NOTES:
In PWM mode, match signal will not clear counter.

Figure 12- 9. Timer D PWM Function Block Diagram

13 A/D CONVERTER

OVERVIEW

The **10**-bit A/D converter (ADC) module uses successive approximation logic to convert analog levels entering at one of the eight input channels to equivalent 10-bit digital values. The analog input level must lie between the V_{DD} and V_{SS} values. The A/D converter has the following components:

— Analog comparator with successive approximation logic

— D/A convert logic

— ADC control register (ADCON)

— Eight multiplexed analog data input pins (ADC0–ADC7)

— 10-bit A/D conversion data output register (ADDATAH/L)

To initiate an analog-to-digital conversion procedure, you write the channel selection data in the A/D converter control register ADCON to select one of the eight analog input pins (ADCn, n = 0–7) and set the conversion start or enable bit, ADCON.0. The read-write ADCON register is located at address FAH.

During a normal conversion, ADC logic initially sets the successive approximation register to 200H (the approximate half-way point of a 10-bit register). This register is then updated automatically during each conversion step. The successive approximation block performs 10-bit conversions for one input channel at a time. You can dynamically select different channels by manipulating the channel selection bit value (ADCON.7–.5) in the ADCON register. To start the A/D conversion, you should set the enable bit, ADCON.0. When a conversion is completed, ACON.3, the end-of-conversion (EOC) bit is automatically set to 1; the result is dumped into the ADDATA register where it can be read and if ADC interrupt is enabled (ADCON.4 = 1), an interrupt request will be generated. 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

Because the ADC does not use sample-and-hold circuitry, it is important that any fluctuations in the analog level at the ADC0–ADC7 input pins during a conversion procedure be kept to an absolute minimum. Any change in the input level, perhaps due to circuit noise, will invalidate the result.

USING A/D PINS FOR STANDARD DIGITAL INPUT

The ADC module's input pins are alternatively used as digital input in port2.

A/D CONVERTER CONTROL REGISTER (ADCON)

The A/D converter control register, ADCON, is located at address FAH. ADCON has four functions:

- Bits 7-5 select an analog input pin (ADC0–ADC7)
- Bit 4 enables/disables ADC interrupt
- Bit 3 indicates the status of the A/D conversion.
- Bits 2-1 select a conversion speed.
- Bit 0 starts the A/D conversion.

Only one analog input channel can be selected at a time. You can dynamically select any one of the eight analog input pins (ADC0–ADC7) by manipulating ADCON.7–ADCON.5.

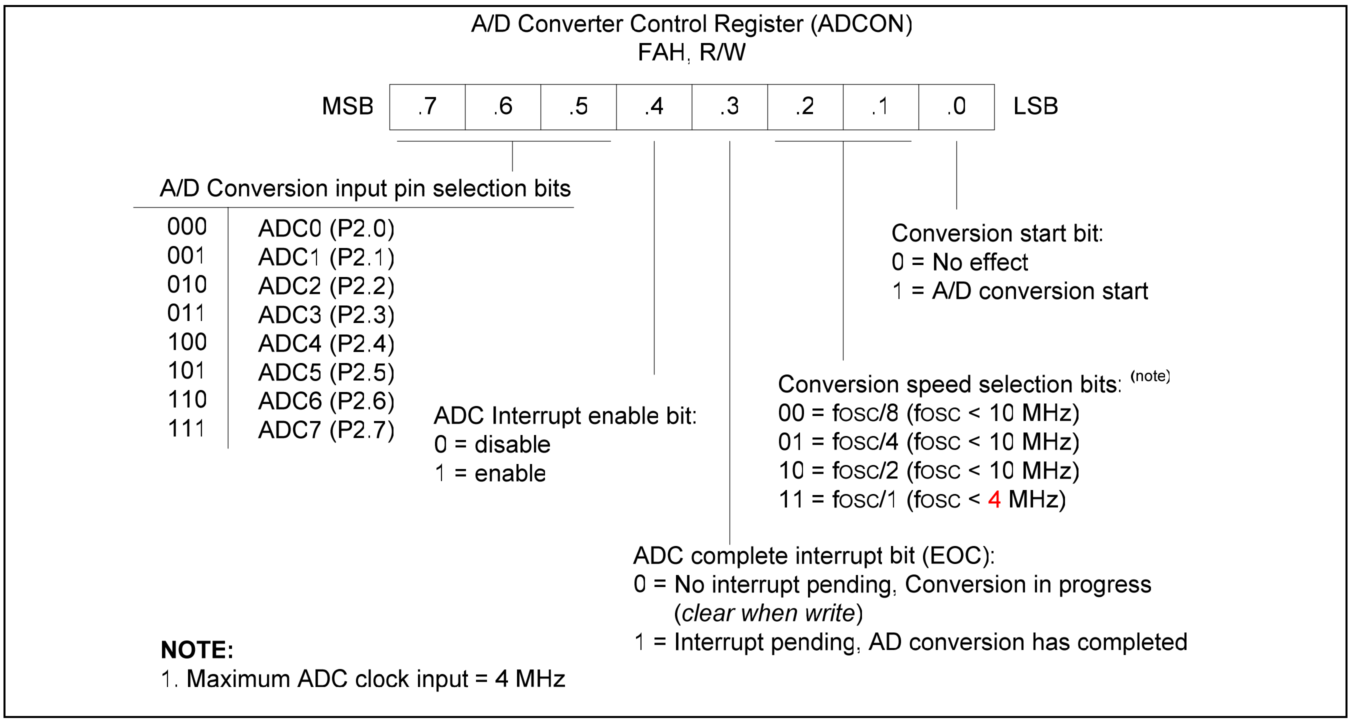

A/D Converter Control Register (ADCON)
FAH, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
A/D Conversion input pin selection bits
000 ADC0 (P2.0)
001 ADC1 (P2.1)
010 ADC2 (P2.2)
011 ADC3 (P2.3)
100 ADC4 (P2.4)
101 ADC5 (P2.5)
110 ADC6 (P2.6)
111 ADC7 (P2.7)
ADC Interrupt enable bit:
0 = disable
1 = enable
Conversion start bit:
0 = No effect
1 = A/D conversion start
Conversion speed selection bits: (note)
00 = fosc/8 (fosc < 10 MHz)
01 = fosc/4 (fosc < 10 MHz)
10 = fosc/2 (fosc < 10 MHz)
11 = fosc/1 (fosc < 4 MHz)
ADC complete interrupt bit (EOC):
0 = No interrupt pending, Conversion in progress (clear when write)
1 = Interrupt pending, AD conversion has completed
NOTE:
1. Maximum ADC clock input = 4 MHz

Figure 13-1. 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. In S3F84B8, the reference voltage is internally connected to VDD. Thus the analog input level must remain within the range V_{SS} to V_{DD}.

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 bit conversion is always 1/2 V_{DD}.

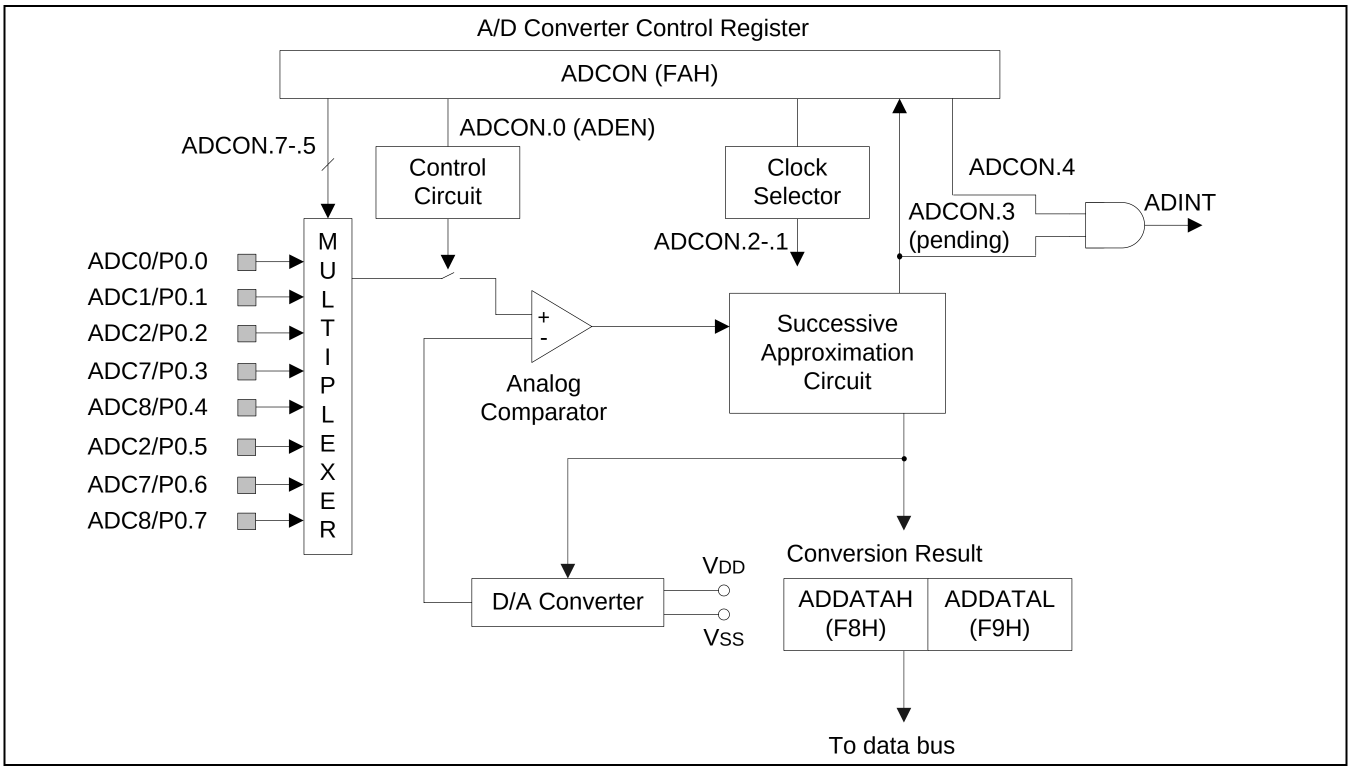
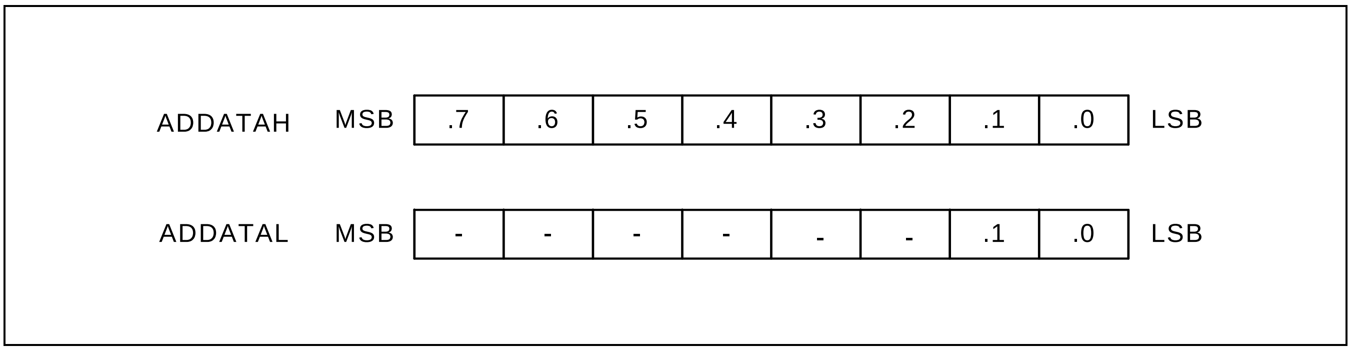


Figure 13-2. A/D Converter Circuit Diagram



ADDATAH	MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB
ADDATAL	MSB	-	-	-	-	-	-	.1	.0	LSB

Figure 13-3. A/D Converter Data Register (ADDATAH/L)

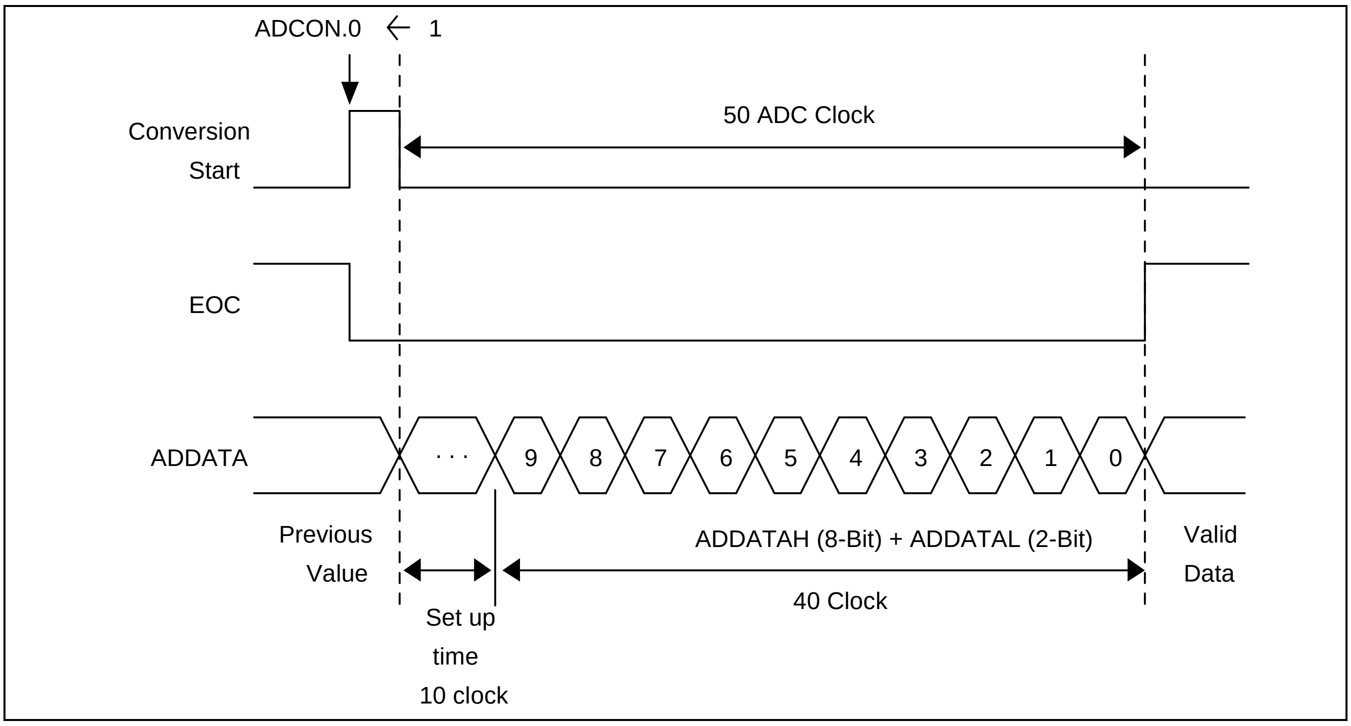


Figure 13-4. A/D Converter Timing Diagram

CONVERSION TIMING

The A/D conversion process requires 4 steps (4 clock edges) to convert each bit and 10 clocks to step-up A/D conversion. Therefore, total of 50 clocks is required to complete a 10-bit conversion: With a 8 MHz CPU clock frequency, one clock cycle is 500 ns (4/fxx). If each bit conversion requires 4 clocks, the conversion rate is calculated as follows:

4 clocks/bit x 10-bits + step-up time (10 clock) = 50 clocks
50 clock x 500 ns = 25 µs at 8 MHz, 1 clock time = 4/fxx (assuming ADCON.2–.1 = 01)

INTERNAL A/D CONVERSION PROCEDURE

1. Analog input must remain between the voltage range of V_{SS} and $V_{DD.}$
2. Configure the analog input pins to input mode by making the appropriate settings in P2CONH and P2CONL registers.
3. Before the conversion operation starts, you must first select one of the eight input pins (ADC0–ADC7) by writing the appropriate value to the ADCON register.
4. When conversion has been completed, (50 clocks have elapsed), the Interrupt pending bit (EOC flag) is set to "1". If ADC interrupt is enabled, a request will generate to CPU, or EOC check can be made to verify that the conversion was successful.
5. The converted digital value is loaded to the output register, ADDATAH (8-bit) and ADDATAL (2-bit), and then the ADC module enters an idle state.
6. The digital conversion result can now be read from the ADDATAH and ADDATAL register.

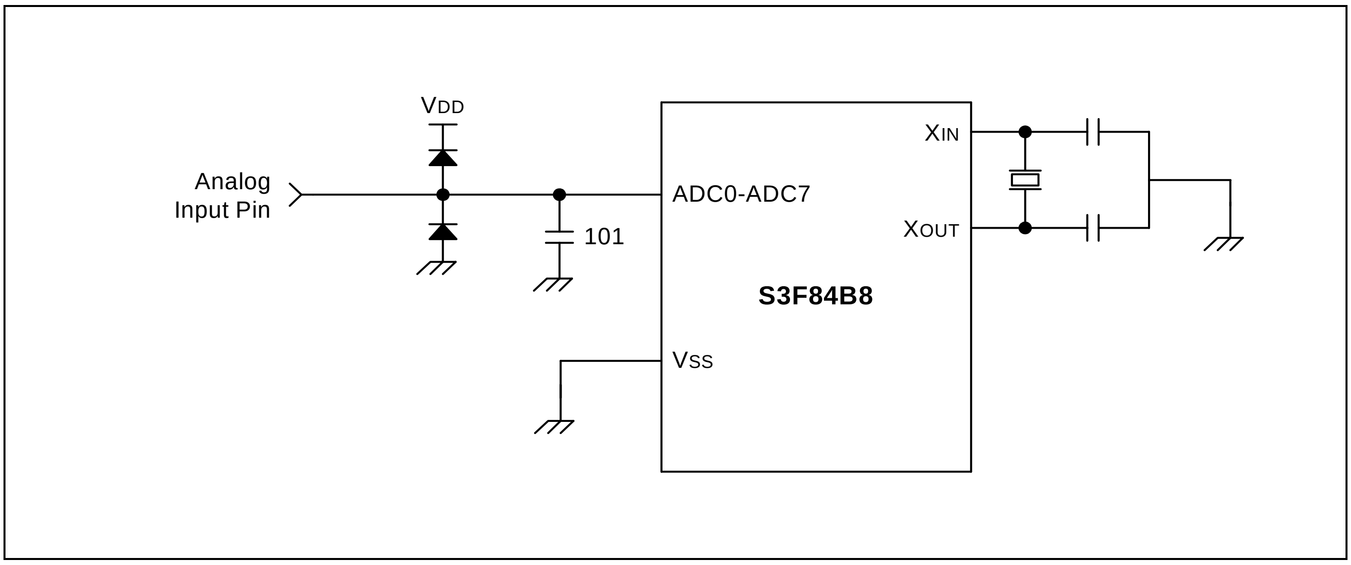


Figure 13-5. Recommended A/D Converter Circuit for Highest Absolute Accuracy

☞ PROGRAMMING TIP – Configuring A/D Converter

```
;-----------------<< Interrupt Vector Address >>

        VECTOR          F0H, INT_ADC            ;

;--------------<< Smart Option >>

                ORG     003CH
                DB      0FFH                    ;  003CH, must be initialized to 1
                DB      0FFH                    ;  003DH, must be initialized to 1
                DB      0FFH                    ;  003EH, must be initialized to 1
                DB      0FFH                    ;  003FH, disable LVR , internal RC oscillator

                ORG     0100H
RESET:          DI                              ;  disable interrupt
                LD      BTCON,#10100011B        ;  Watchdog disable
                •
                •
                •
                LD      P2CONH,#11111111B       ;  Configure P2.4–P2.7 AD input
                LD      P2CONL,#11111111B       ;  Configure P2.0–P2.3 AD input
                EI                              ;  Enable interrupt

;--------------<< Main loop >>

MAIN:           •
                •
                •
                •
                •
                JR     t, MAIN

AD_CONV:        LD      ADCON, #00110001B       ;  Select analog input channel → P2.1
                                                ;  Enable ADC interrupt

                                                ;  select conversion speed → fOSC/8
                                                ;  set conversion start bit

                NOP
                                                ;  If you select conversion speed to fOSC/8
                                                ;  at least one nop must be included
```

☞ PROGRAMMING TIP – Configuring A/D Converter (Continued)

```
INT_ADC:

            LD          R2, ADDATAH             ;
            LD          R3, ADDATAL             ;

            AND         ADCON, #11110111B       ; clear pending bit
            •                                   ;


            IRET                                ;
            •
            •
            END
```

NOTES

14 COMPARATOR

OVERVIEW

This microcontroller has 4 comparators. The operation of 4 comparators is individually controlled by four registers, CMP0CON, CMP1CON, CMP2CON and CMP3CON. The Interrupt control register CMPINT controls interrupt mode of four comparators.

FUNCTION DESCRIPTION

COMPARATOR0

Comparator 0 has both of its positive and negative inputs as chip pins.

The polarity of comparator 0 output can be set inverted or non-inverted. User could check the real input status by reading CMP0CON.1.

The output (falling edge) can be configured as trigger signal to start a new PWM cycle when the PWM-CMP0 linkage is enabled by writing '1' to PWMCCON.0. Meanwhile, the output can have a programmable delay to realize delay trigger by configuring AMTDATA register. It is useful when realizing timing adjusting.

COMPARATOR 0 CONTROL REGISTER (CMP0CON)

You use comparator 0 control registers to

— Enable comparator 0

— Enable comparator 0 interrupt

— Set comparator 0 output polarity

— Check comparator 0 input status

— Clear interrupt pending bit

CMP0CON is located at address EAH, Set1 Bank0, and is read/write addressable (except CMP0CON.1) using Register addressing mode.

To enable comparator0, you must write '1' to CMP0CON.3. The output polarity is programmable by configuring CMP0CON.4. Meanwhile, CMP0CON.1 represents the real status of 2 inputs, reading as '0' when CMP0_N>CMP0_P or '1' when CMP0_N<CMP0_P.

Comparator 0 can generate interrupt to indicate the alternation of 2 input pins. Interrupt trigger mode (rising/falling/rising and falling edge) can be configured in CMPINT register. Write '1' in CMP0CON.2 to enable the interrupt, and '0' to CMP0CON.0 to clear the interrupt pending bit. It must be cleared by software.

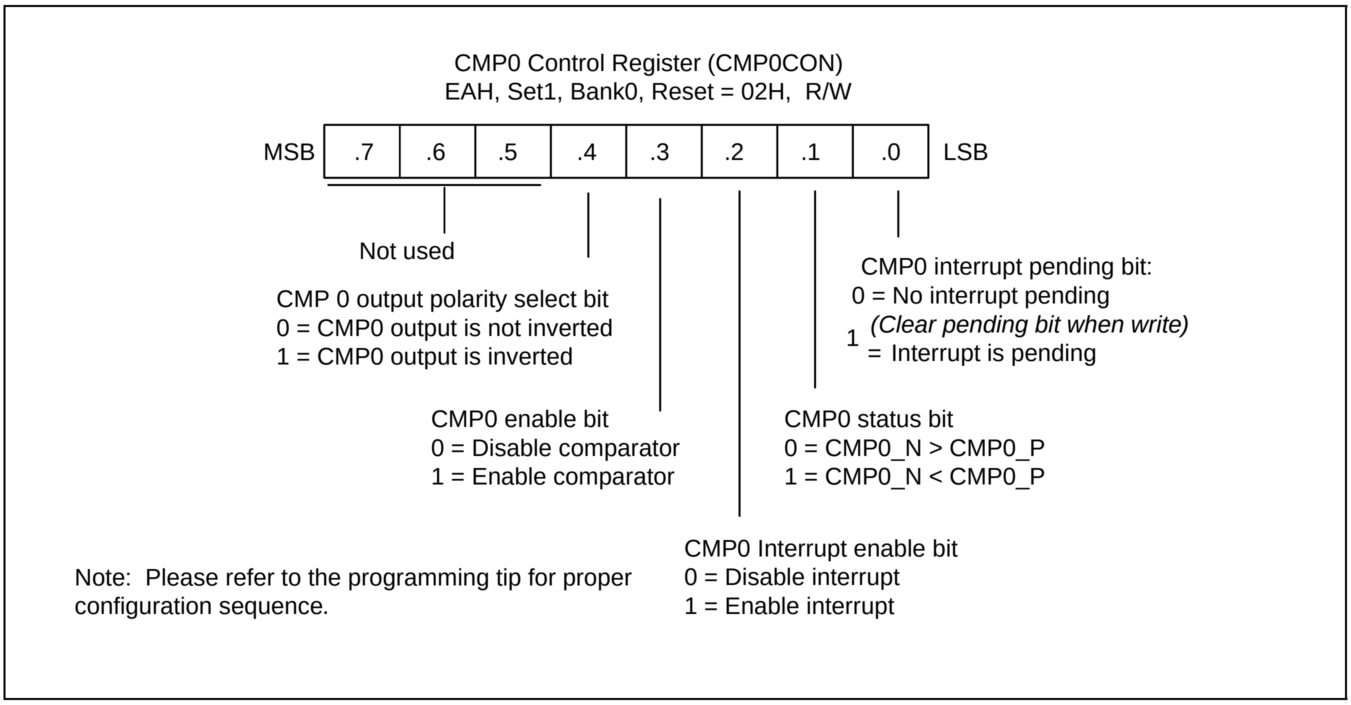

CMP0 Control Register (CMP0CON)
EAH, Set1, Bank0, Reset = 02H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
CMP 0 output polarity select bit
0 = CMP0 output is not inverted
1 = CMP0 output is inverted
CMP0 enable bit
0 = Disable comparator
1 = Enable comparator
CMP0 Interrupt enable bit
0 = Disable interrupt
1 = Enable interrupt
CMP0 status bit
0 = CMP0_N > CMP0_P
1 = CMP0_N < CMP0_P
CMP0 interrupt pending bit:
0 = No interrupt pending
(Clear pending bit when write)
1 = Interrupt is pending
Note: Please refer to the programming tip for proper configuration sequence.

Figure 14-1. CMP0 Control Register (CMP0CON)

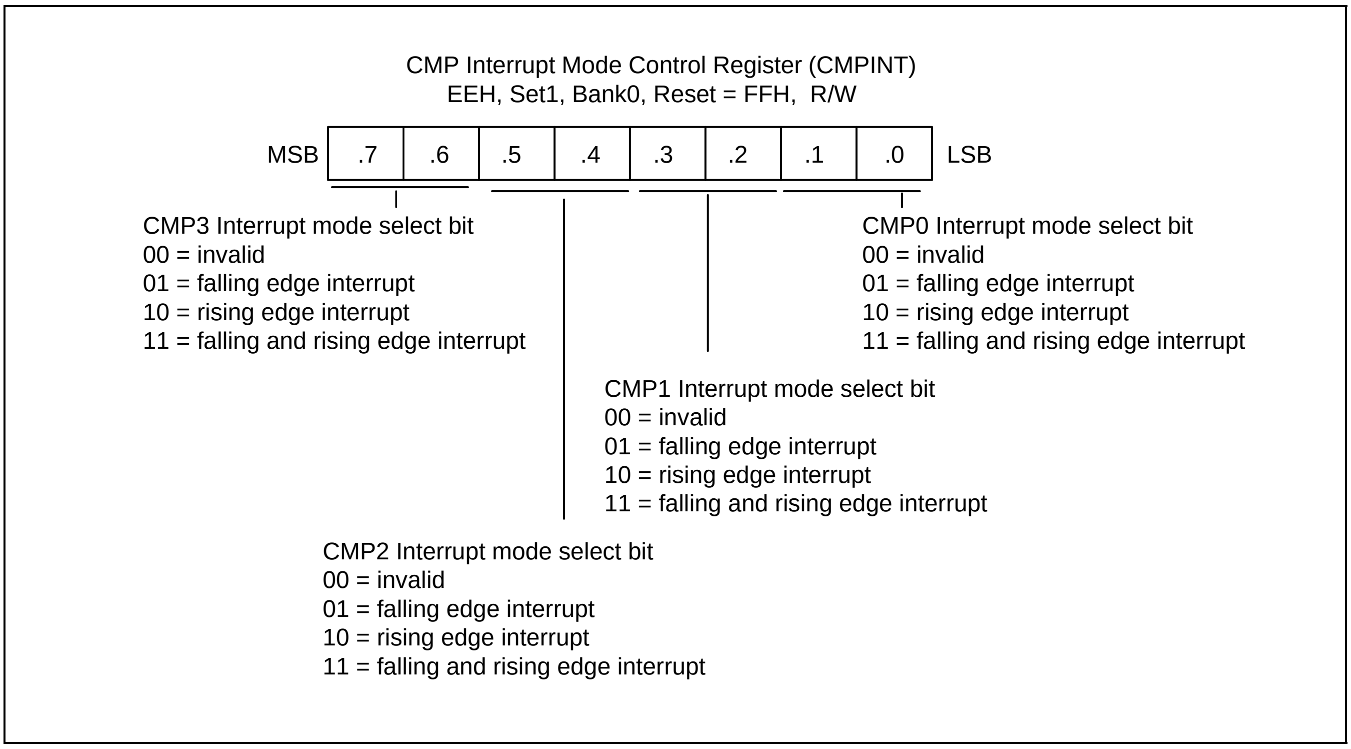
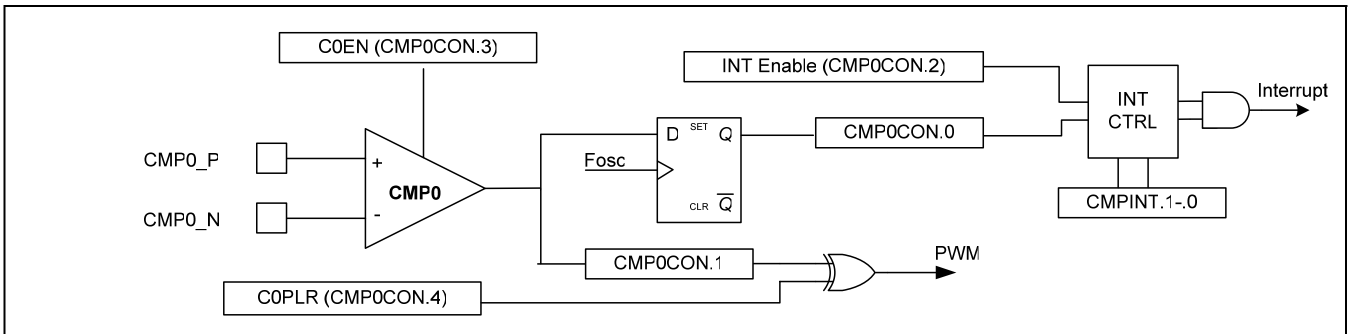

CMP Interrupt Mode Control Register (CMPINT)
EEH, Set1, Bank0, Reset = FFH, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
CMP3 Interrupt mode select bit
00 = invalid
01 = falling edge interrupt
10 = rising edge interrupt
11 = falling and rising edge interrupt
CMP0 Interrupt mode select bit
00 = invalid
01 = falling edge interrupt
10 = rising edge interrupt
11 = falling and rising edge interrupt
CMP1 Interrupt mode select bit
00 = invalid
01 = falling edge interrupt
10 = rising edge interrupt
11 = falling and rising edge interrupt
CMP2 Interrupt mode select bit
00 = invalid
01 = falling edge interrupt
10 = rising edge interrupt
11 = falling and rising edge interrupt

Figure 14-2. CMP Interrupt mode Control Register (CMPINT)

BLOCK DIAGRAM



Note:
1 Polarity selection bit (CMP0CON.4) will not affect interrupt generation logic.
2. PWM trigger signal is falling edge active only.

Figure 14-3 Comparator 0 diagram

COMPARATOR1/2/3

Comparator 1, 2 and 3 has the same structure. Their positive input is internally connected with reference voltage programmable from 0.45VDD to 0.8VDD with the step length of 0.05VDD.

The output (falling edge) of comparator 1, 2 and 3 can be configured to generate PWM hard lock (PWMCCON.1-.2/.3-0.4/.5/.6 = 11) or soft lock trigger signal (PWMCCON.1-.2/.3-0.4/.5/.6 = 01).

When hard lock happens, PWM output will stop immediately (stop voltage level is determined by PWM output polarity bit: when PWMCON.5 = 0, PWM output '0'', when PWMCON. = 1, PWM output '1'). To unlock the hard lock, write '1' to PWMCON.3.

When soft lock happens, PWM output will stop immediately (stop voltage level is determined by PWM output polarity bit: when PWMCON.5 = 0, PWM output '0'', when PWMCON.5 = 1, PWM output '1') and reload PWMDATA with PWMPDATA.

Soft lock will be automatically unlocked at next PWM cycle.

COMPARATOR CONTROL REGISTER (CMP1CON, COM2CON, CMP3CON)

You use comparator control registers to

— Select comparator reference voltage
— Enable comparator
— Enable comparator interrupt
— Set comparator output polarity
— Check comparator status
— Clear interrupt pending bit

CMP1CON, CMP2CON and CMP3CON are located at address EBH, ECH and EDH, Set1 Bank0, and are read/write addressable (except CMP1/2/3CON.1) using Register addressing mode.

To enable comparator1/2/3, you must write '1' to CMP1/2/3CON.3. Their positive input is internally connected with reference voltage programmable from 0.45VDD to 0.8VDD with the step length of 0.05VDD.

The output polarity is programmable by configuring CMP1/2/3CON.4. Meanwhile, CMP1/2/3CON.1 represents the real status of 2 inputs, reading as '0' when CMP1/2/3_N> reference voltage or '1' when CMP1/2/3_N< reference voltage.

Comparator 1/2/3 can generate interrupt to indicate the alternation of 2 input pins. You can choose falling edge, rising edge or falling and rising edge to trigger comparator interrupt by configuring CMPINT register. Write '1' in CMP1/2/3CON.2 to enable the interrupt, and '0' to CMP1/2/3CON.0 to clear the interrupt pending bit. It must be cleared by software.

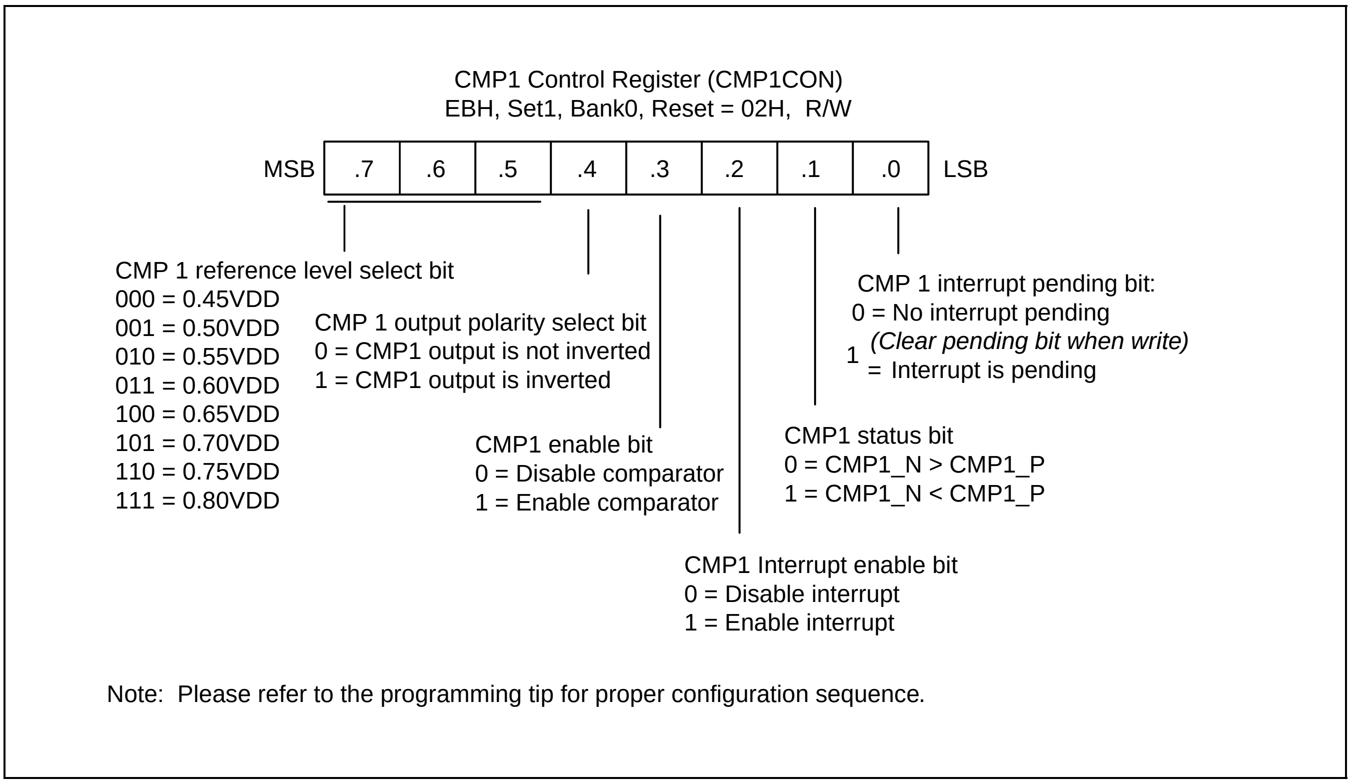

CMP1 Control Register (CMP1CON)
EBH, Set1, Bank0, Reset = 02H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
CMP 1 reference level select bit
000 = 0.45VDD
001 = 0.50VDD
010 = 0.55VDD
011 = 0.60VDD
100 = 0.65VDD
101 = 0.70VDD
110 = 0.75VDD
111 = 0.80VDD
CMP 1 output polarity select bit
0 = CMP1 output is not inverted
1 = CMP1 output is inverted
CMP1 enable bit
0 = Disable comparator
1 = Enable comparator
CMP1 Interrupt enable bit
0 = Disable interrupt
1 = Enable interrupt
CMP1 status bit
0 = CMP1_N > CMP1_P
1 = CMP1_N < CMP1_P
CMP 1 interrupt pending bit:
0 = No interrupt pending
(Clear pending bit when write)
1 = Interrupt is pending
Note: Please refer to the programming tip for proper configuration sequence.

Figure 14-4. CMP1 Control Register (CMP1CON)

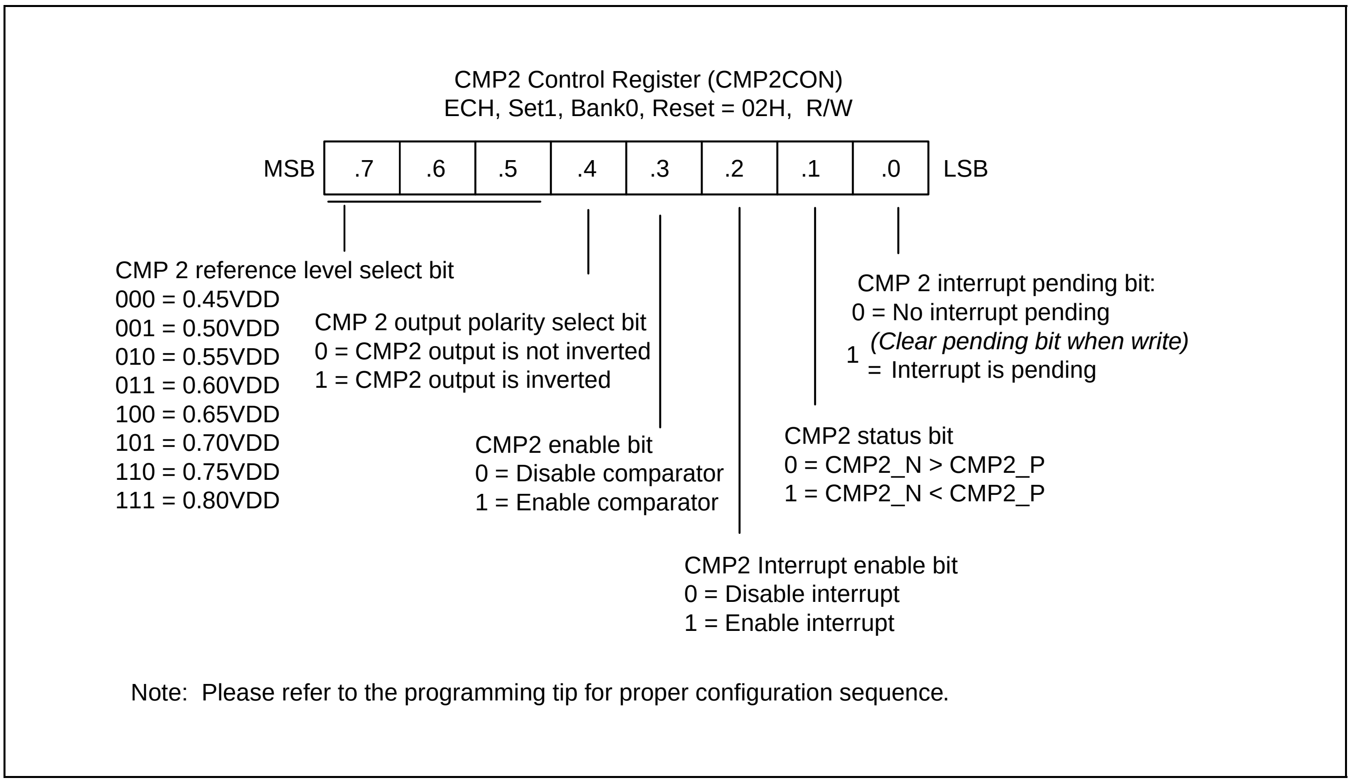

CMP2 Control Register (CMP2CON)
ECH, Set1, Bank0, Reset = 02H, R/W
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
CMP 2 reference level select bit
000 = 0.45VDD
001 = 0.50VDD
010 = 0.55VDD
011 = 0.60VDD
100 = 0.65VDD
101 = 0.70VDD
110 = 0.75VDD
111 = 0.80VDD
CMP 2 output polarity select bit
0 = CMP2 output is not inverted
1 = CMP2 output is inverted
CMP2 enable bit
0 = Disable comparator
1 = Enable comparator
CMP2 Interrupt enable bit
0 = Disable interrupt
1 = Enable interrupt
CMP2 status bit
0 = CMP2_N > CMP2_P
1 = CMP2_N < CMP2_P
CMP 2 interrupt pending bit:
0 = No interrupt pending
(Clear pending bit when write)
1 = Interrupt is pending
Note: Please refer to the programming tip for proper configuration sequence.

Figure 14-5. CMP2 Control Register (CMP2CON)

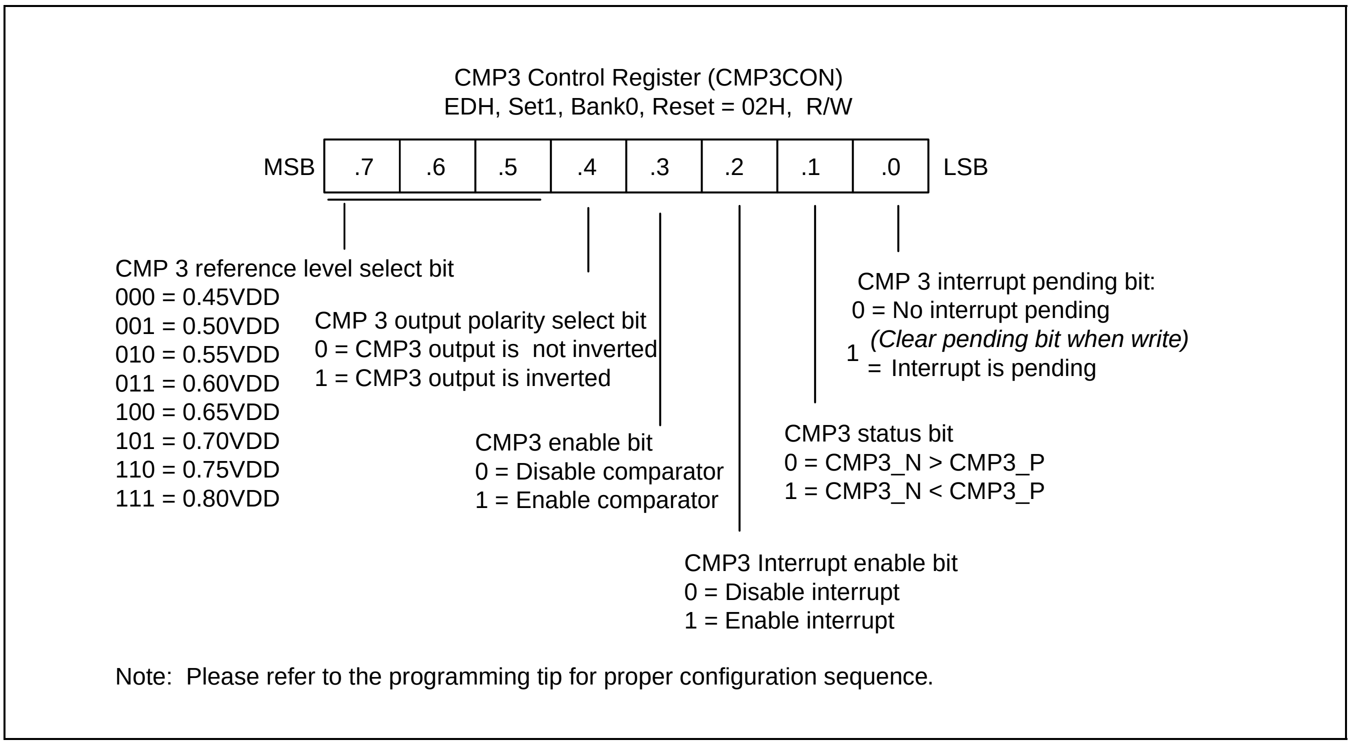

CMP3 Control Register (CMP3CON)
EDH, Set1, Bank0, Reset = 02H, R/W
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
CMP 3 reference level select bit
000 = 0.45VDD
001 = 0.50VDD
010 = 0.55VDD
011 = 0.60VDD
100 = 0.65VDD
101 = 0.70VDD
110 = 0.75VDD
111 = 0.80VDD
CMP 3 output polarity select bit
0 = CMP3 output is not inverted
1 = CMP3 output is inverted
CMP3 enable bit
0 = Disable comparator
1 = Enable comparator
CMP3 Interrupt enable bit
0 = Disable interrupt
1 = Enable interrupt
CMP3 status bit
0 = CMP3_N > CMP3_P
1 = CMP3_N < CMP3_P
CMP 3 interrupt pending bit:
0 = No interrupt pending
(Clear pending bit when write)
1 = Interrupt is pending
Note: Please refer to the programming tip for proper configuration sequence.

Figure 14-6. CMP3 Control Register (CMP3CON)

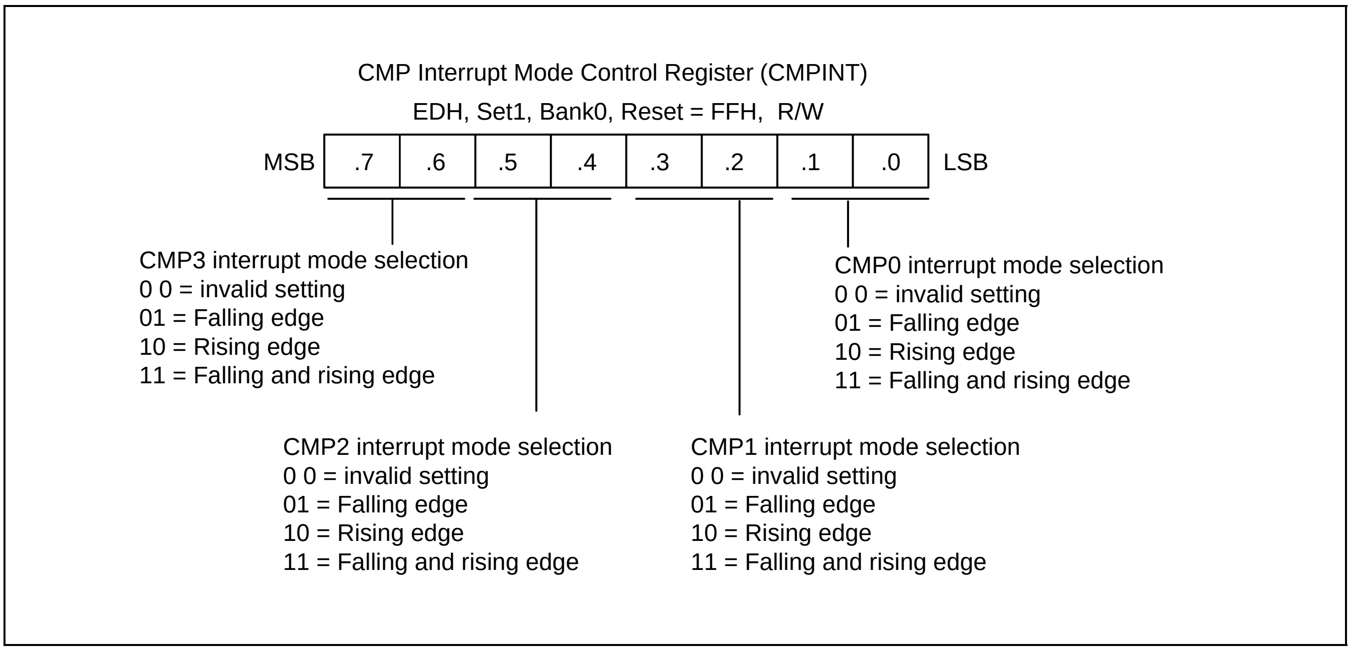
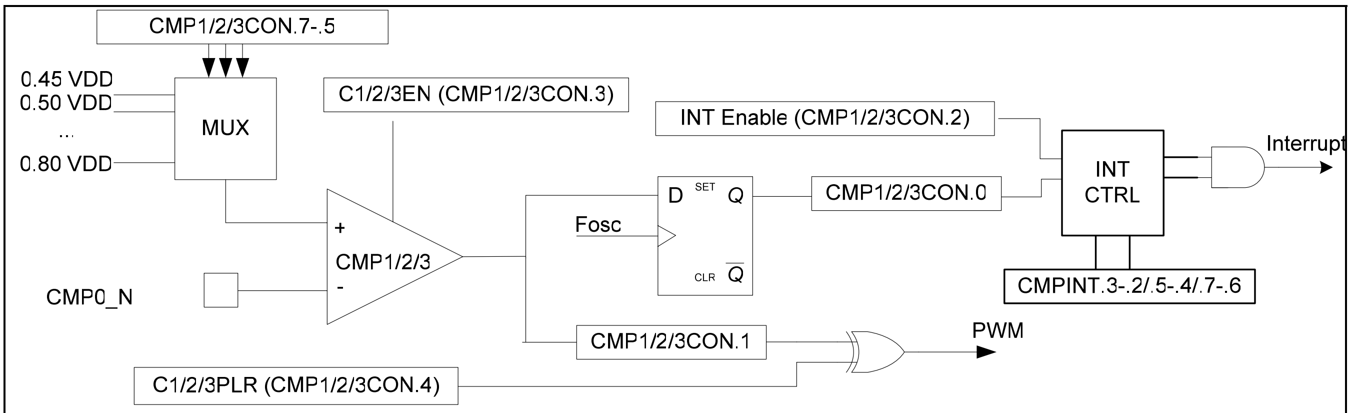

CMP Interrupt Mode Control Register (CMPINT)
EDH, Set1, Bank0, Reset = FFH, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
CMP3 interrupt mode selection
0 0 = invalid setting
01 = Falling edge
10 = Rising edge
11 = Falling and rising edge
CMP0 interrupt mode selection
0 0 = invalid setting
01 = Falling edge
10 = Rising edge
11 = Falling and rising edge
CMP2 interrupt mode selection
0 0 = invalid setting
01 = Falling edge
10 = Rising edge
11 = Falling and rising edge
CMP1 interrupt mode selection
0 0 = invalid setting
01 = Falling edge
10 = Rising edge
11 = Falling and rising edge

Figure 14-7. CMP Interrupt Mode Control Register (CMPINT)

CMP BLOCK DIAGRAM



Note:
1. Polarity selection bit (CMP1/2/3CON.4) will not affect interrupt generation logic.
2. PWM lock signal is falling edge active only.

Figure 14-8 Comparator 1/2/3 diagram

☞ PROGRAMMING TIP —comparator configuration

```
•
•
DI
LD          CMPINT, #055H                ; Falling edge interrupt
AND         CMP0/1/2/3CON, #0FEH          ; Must clear the pending bit before enabling CMP
LD          CMP0/1/2/3CON, #0CH           ; Enable CMP, enable interrupt
EI
•
•
```

NOTES

15 OPERATIONAL AMPLIFIER

OVERVIEW

This microcontroller has an Operational Amplifier. The operation of the OP AMP is controlled by OPACON.

FUNCTION DESCRIPTION

OP AMP has two operation modes, on chip mode and off chip mode.

- On chip mode. Positive input is internally connected to ground.
  OP AMP can only work as an inverting amplifier.
- Off chip mode. All the input and output pins should be externally connected.
  OP AMP could work either as an inverting or non-inverting amplifier.

OPAMP CONTROL REGISTER (OPACON)

You use OPAMP control register, OPACON

— Enable OPAMP

— Select operating mode

OPACON is located at address E0H, Set1 Bank1, and is read/write addressable using Register addressing mode. OP AMP is enabled when OPACON.0=1 and disabled when OPACON.0=0. When the OP AMP is enabled, the output of OP AMP will be the analog input signal of ADC3.

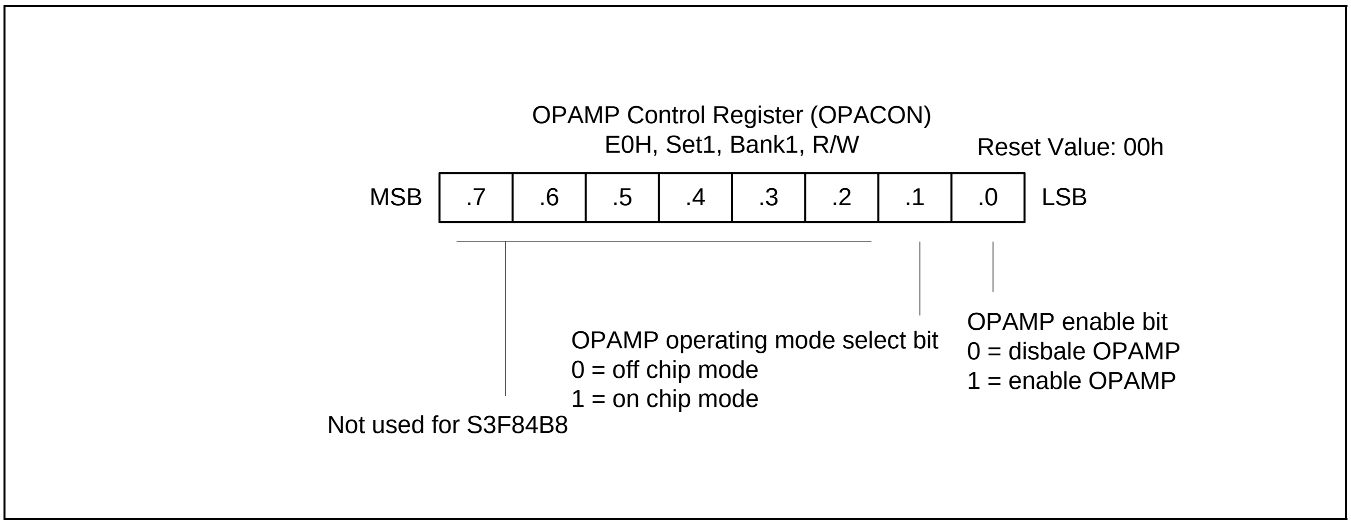


Figure 15-1. OPAMP Control Register (OPACON)

OP AMP BLOCK DIAGRAM

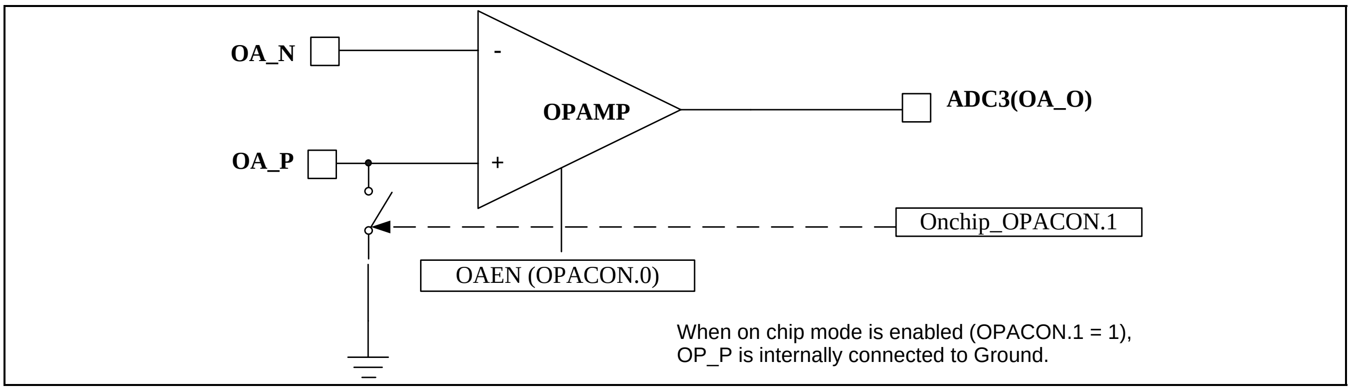

OA_N
-
OPAMP
ADC3(OA_O)
OA_P
+
Onchip_OPACON.1
OAEN (OPACON.0)
When on chip mode is enabled (OPACON.1 = 1),
OP_P is internally connected to Ground.

Figure 15-2. OPAMP Block Diagram

REFERENCE CIRCUIT

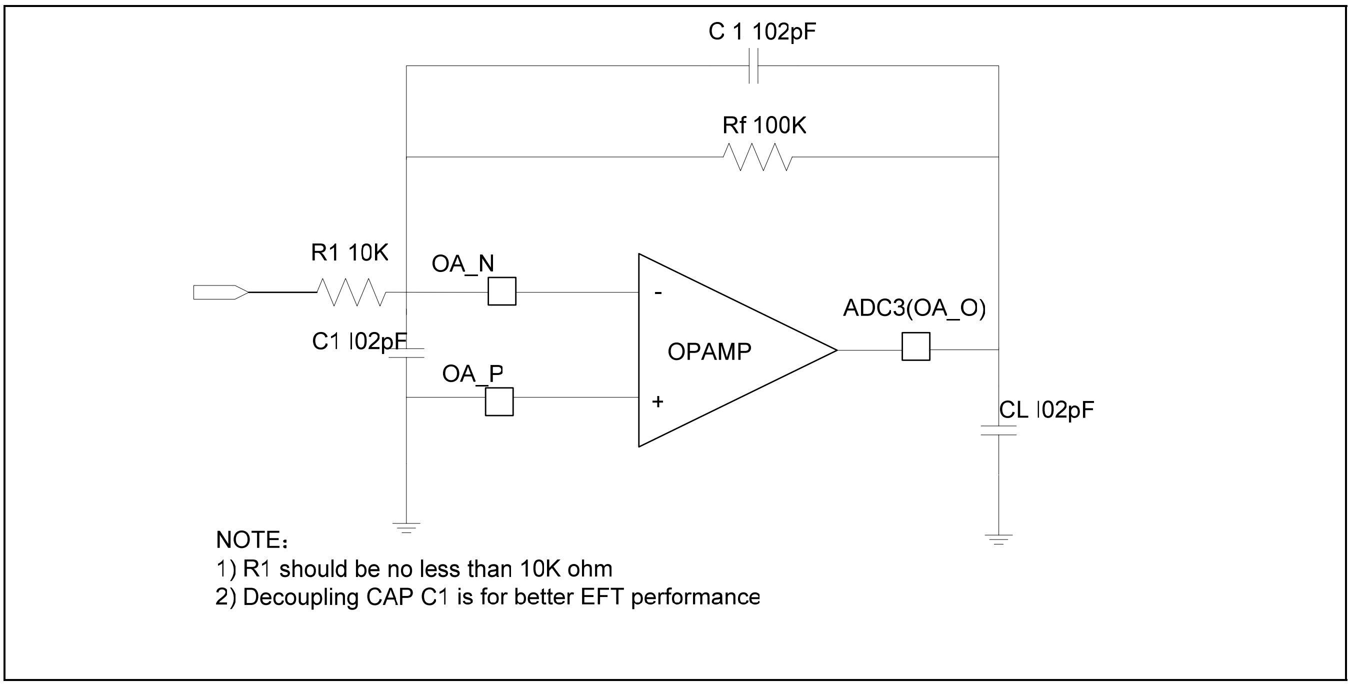

C 1 102pF
Rf 100K
R1 10K
OA_N
-
OPAMP
ADC3(OA_O)
C1 102pF
OA_P
+
CL 102pF
NOTE:
1) R1 should be no less than 10K ohm
2) Decoupling CAP C1 is for better EFT performance

Figure 15-3. OPAMP Application reference circuit @ gain=10

NOTES

16 10-BIT IH-PWM

OVERVIEW

This microcontroller has a 10-bit IH-PWM circuit that can cooperate with comparators. It is dedicatedly designed for IH cooker application.

The operation of all PWM circuits is controlled by control register, PWMCON. The linkage of comparators and PWM is controlled by PWMCCON.

PWM can work in:

— Normal 10 bit PWM mode （When all the linkages with comparators are disabled）
— Comparator-cooperation mode

In comparator-cooperation mode, the PWM circuit can realize:
— Delay trigger function
— Anti-mis-trigger function
— Hard/soft lock function

FUNCTION DESCRIPTION

PWM

The 10-bit PWM circuits have the following components:

— 10-bit comparator circuit
— 10-bit counter
— 10-bit reference data registers (PWMDATAH/L)
— 10-bit preset PWM data registers (PWMPDATAH/L)
— PWM output pins (P0.3/PWM)

PWM clock rate

The timing characteristic of PWM output is based on the f_{OSC} clock frequency. The PWM counter clock value is determined by the setting of PWMCON.6–.7.

Table 16-1. PWM Control and Data Registers

Register Name	Mnemonic	Addres s	Location	Function
PWM data registers	PWMDATAH	F4H	Set1, Bank0	PWMDATA high byte
	PWMDATAL	F5H	Set1, Bank0	PWMDATA low byte
PWM preset data registers	PWMPDATAH	F2H	Set1, Bank0	For soft lock operation
	PWMPDATAL	F3H	Set1, Bank0	For soft lock operation
PWM control register	PWMCON	EFH	Set1, Bank0	PWM counter stop/start (resume), clock settings, anti-mis-trigger function enable, .etc.
PWM CMP register	PWMCCON	F0H	Set1, Bank0	PWM CMP linkage settings

PWM function Description

By disabling the linkage of CMPs and PWM (setting PWMCCON to '00H'), PWM module can work in normal 10 bit mode, PWM output will toggle either on PWM counter match or overflow. The output level can be set as inverted (PWMCON.5=1) or non-inverted (PWMCON.5=0)

In comparator-cooperation mode, when the linkage is enabled (PWMCCON.6/4/2/0= 1), PWM will work according to the outputs of the four comparators, but if all the comparators do not generate valid trigger signal, the PWM will word as a normal 10bit PWM.

For comparator0, the output **falling edge** will clear the PWM counter and restart one PWM cycle immediately (max delay = 4/f_{PWM}) or after some programmable delay period which is called delay trigger function (enabled when PWMCCON.0 = 1). The delay period is programmable through PWMDL register.

Anti-mis-trigger function could be used to prevent PWM from being triggered by unwanted noise. There is an internal timer used to realize PWM anti-mis-trigger function. When the PWM starts a new cycle, the internal timer will reset and starts to up count at PWM clock. Before match happens, signals from Comparator 0 will be neglected, thus will not trigger PWM to start another new cycle.

For comparator1, 2 and 3, the output **falling edge** will either directly stop PWM (hard lock), or stop the current PWM cycle and restart PWM when next cycle begins with a preset PWM data PWMPDATA (soft lock).

To avoid invalid trigger or lock, register PWMCCON must be set to appropriate value before enabling PWM module.

You can select clock for the PWM counter by set PWMCON.6-.7. Clocks which you can select are f_{OSC} /64, f_{OSC} /8, f_{OSC} /2, f_{OSC} /1.

PWM CONTROL REGISTER (PWMCON)

The control register for the PWM module, PWMCON, is located at register address EFH, Set 1, Bank 0.

Bit settings in the PWMCON register control the following functions:

— PWM counter clock selection
— PWM output polarity selection
— PWM counter clear
— PWM counter disable/enable(or resume) operation
— Anti-Mis-Trigger function selection
— PWM counter overflow interrupt control

A reset clears all PWMCON bits to logic zero, disabling the entire PWM module.

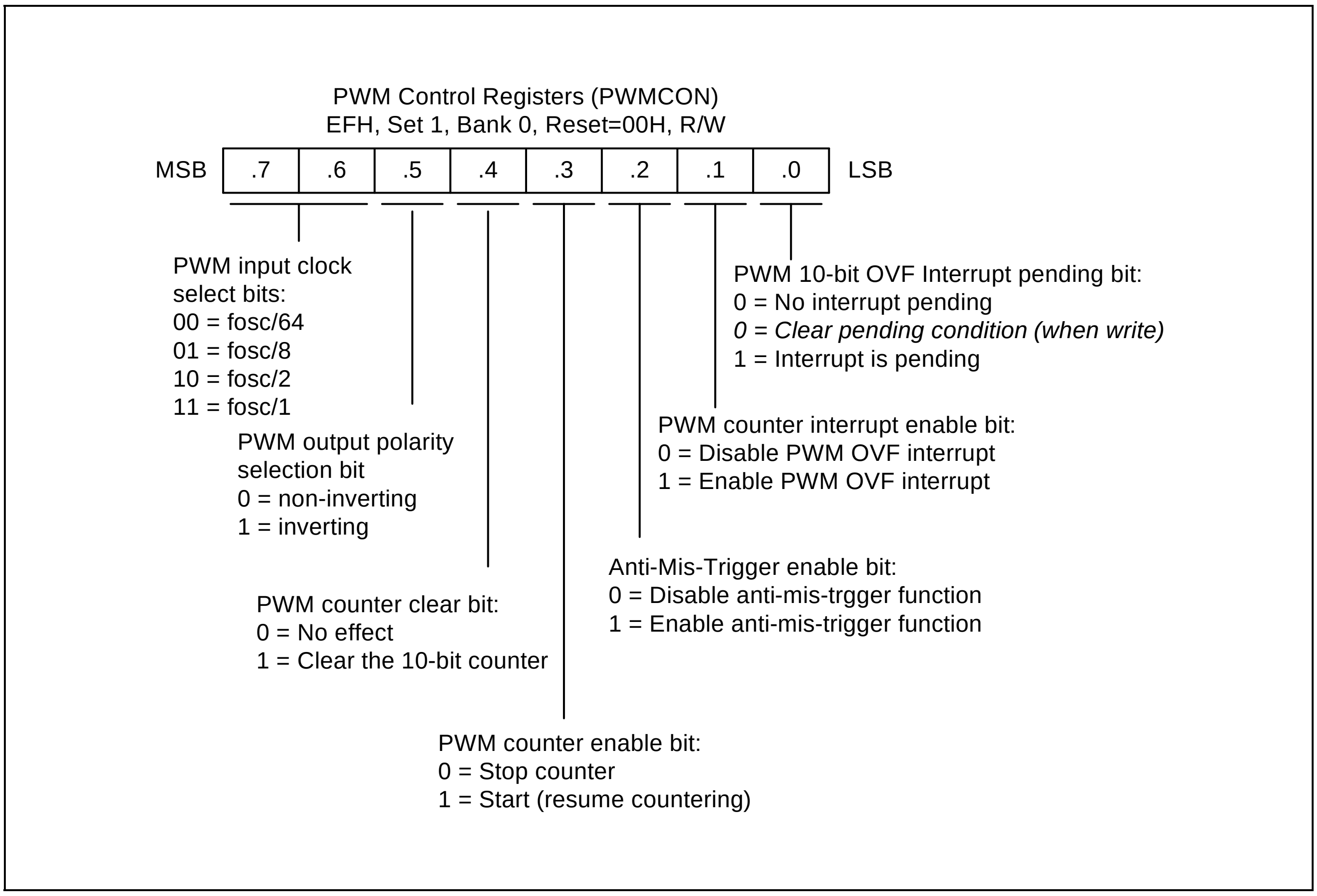

PWM Control Registers (PWMCON)
EFH, Set 1, Bank 0, Reset=00H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
PWM input clock select bits:
00 = fosc/64
01 = fosc/8
10 = fosc/2
11 = fosc/1
PWM output polarity selection bit
0 = non-inverting
1 = inverting
PWM counter clear bit:
0 = No effect
1 = Clear the 10-bit counter
PWM counter enable bit:
0 = Stop counter
1 = Start (resume countering)
Anti-Mis-Trigger enable bit:
0 = Disable anti-mis-trgger function
1 = Enable anti-mis-trigger function
PWM counter interrupt enable bit:
0 = Disable PWM OVF interrupt
1 = Enable PWM OVF interrupt
PWM 10-bit OVF Interrupt pending bit:
0 = No interrupt pending
0 = Clear pending condition (when write)
1 = Interrupt is pending

Figure 16-2. PWM Module Control Register (PWMCON)

PWM CMP LINKAGE CONTROL REGISTER (PWMCCON)

The control register for the linkage of CMP and PWM module, PWMCCON, is located at register address F0H, Set 1, Bank 0.

Bit settings in the PWMCCON register control the following functions:

— PWM CMP0 linkage configuration

— PWM CMP1 linkage configuration

— PWM CMP2 linkage configuration

— PWM CMP3 linkage configuration

A reset clears all PWMCCON bits to logic zero, disabling the entire linkage.

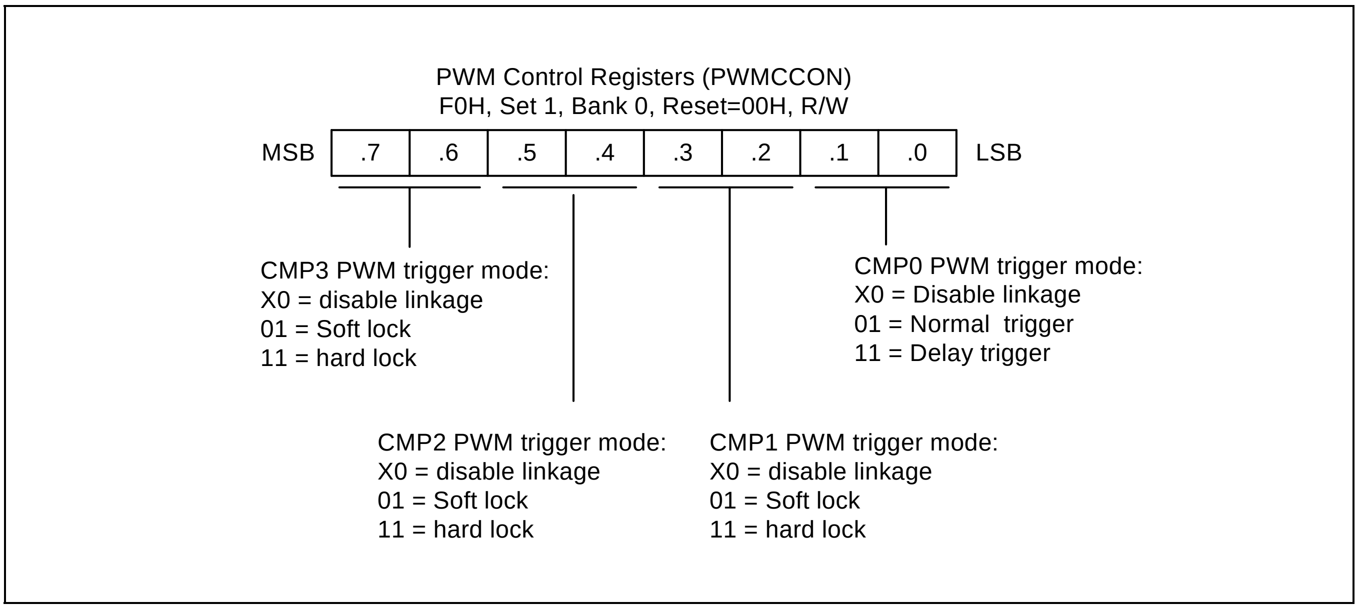


Figure 16-3. PWM CMP Linkage Control Register (PWMCCON)

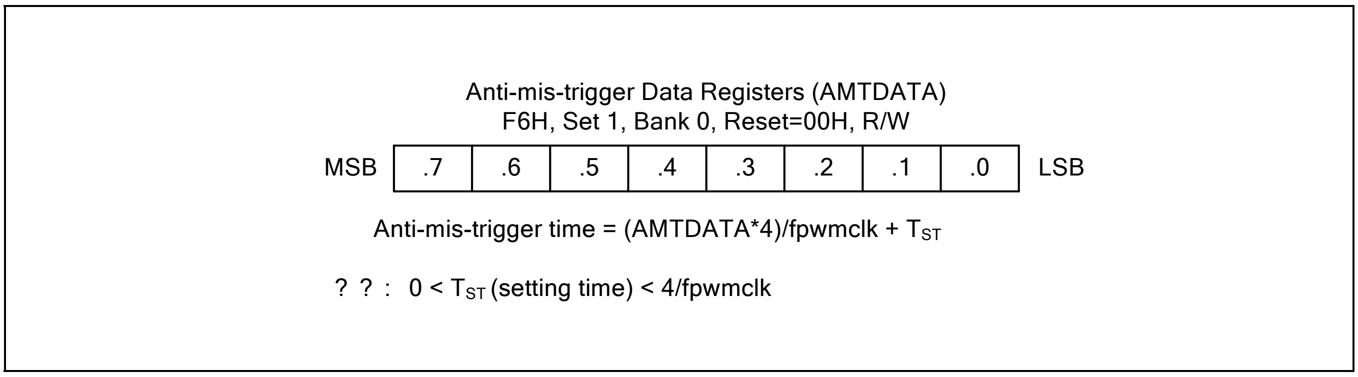

Anti-mis-trigger Data Registers (AMTDATA)
F6H, Set 1, Bank 0, Reset=00H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Anti-mis-trigger time = (AMTDATA*4)/fpwmclk + T_ST
? ? : 0 < T_ST (setting time) < 4/fpwmclk

Figure 16-4 Anti-mis-trigger Data Register (AMTDATA)

PWM Delay trigger Registers (PWMDL)
F5H, Set 1, Bank 0, Reset=00H, R/W

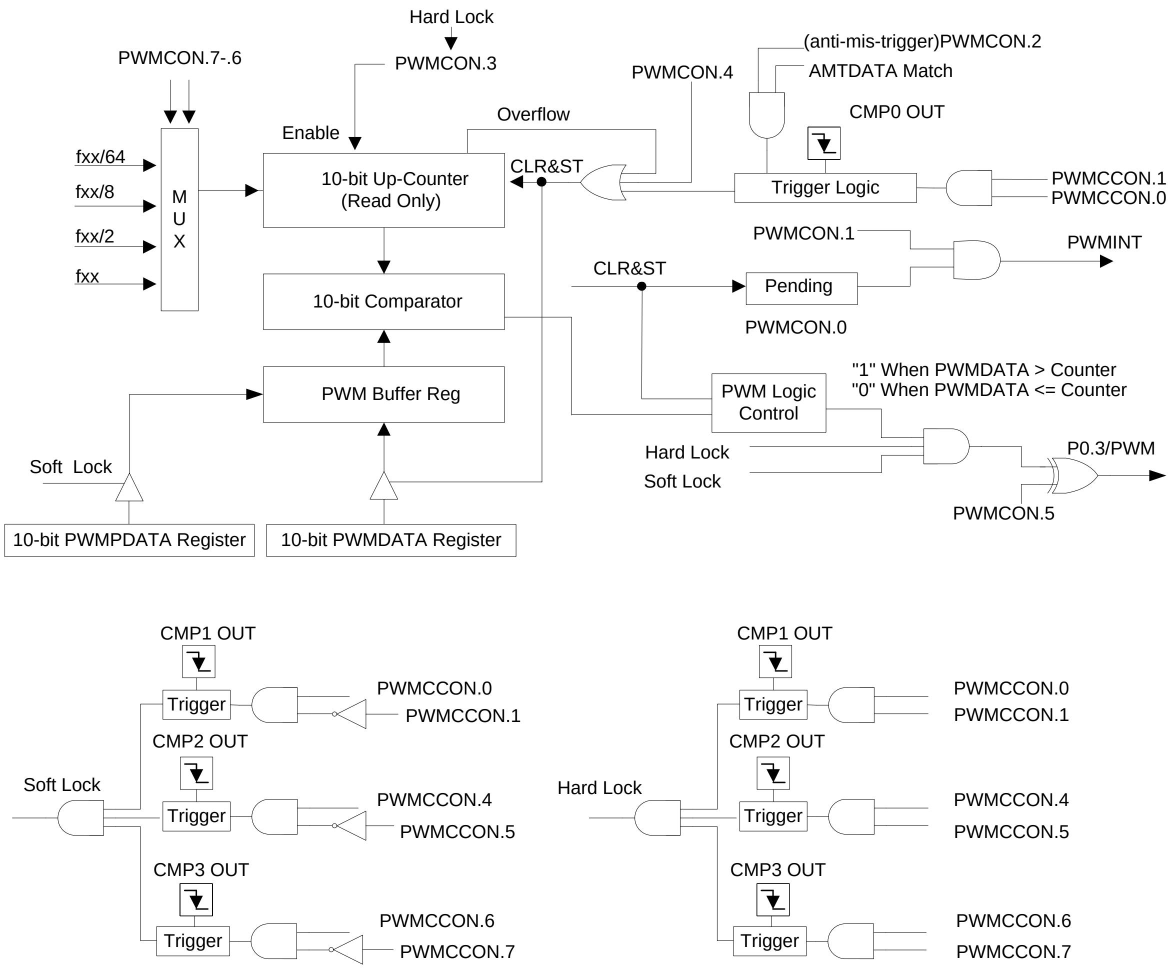
MSB	-	-	-	-	.3	.2	.1	.0	LSB

Delay Time = (PWMDL+1)*4/fpwmclk + T_{ST}

注释： 0 <T_{ST} (Setting time) < 4/fpwmclk

Figure 16-5. Delay trigger Data Register (PWMDL)

BLOCK DIAGRAM



NOTES:

1. CLR&ST (Active high) is valid all the time when PWM is operating. It will clear the counter and restart a new PWM cycle immediately.
2. CLR (Active high) is valid all the time when PWM is operating. It will force the current remaining PWM cycle to low level when PWMCON.5 = 0 or high level when PWMCON.5 = 1.
3. Hard lock (active low) stops the PWM until unlock operation; Soft lock (active low) stops the current PWM and restart PWM at PWMDATA = PWMPDATA

Figure 16-6. PWM Module Functional Block Diagram

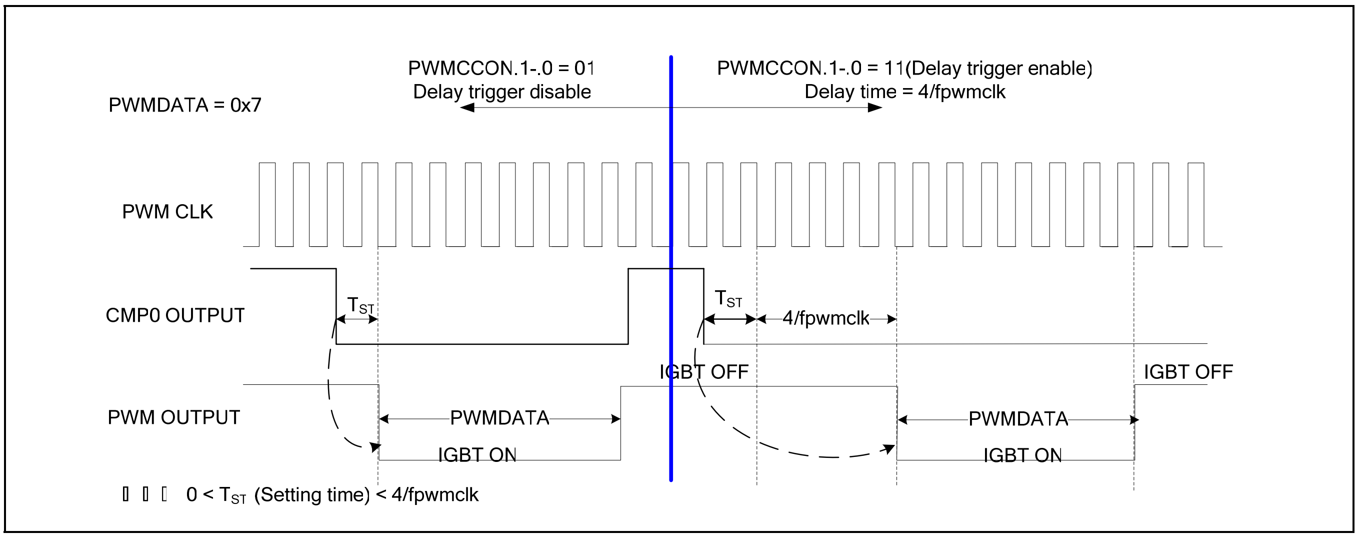


Figure 16-7. An example of the cooperation of PWM and Comparator 0_Delay Trigger

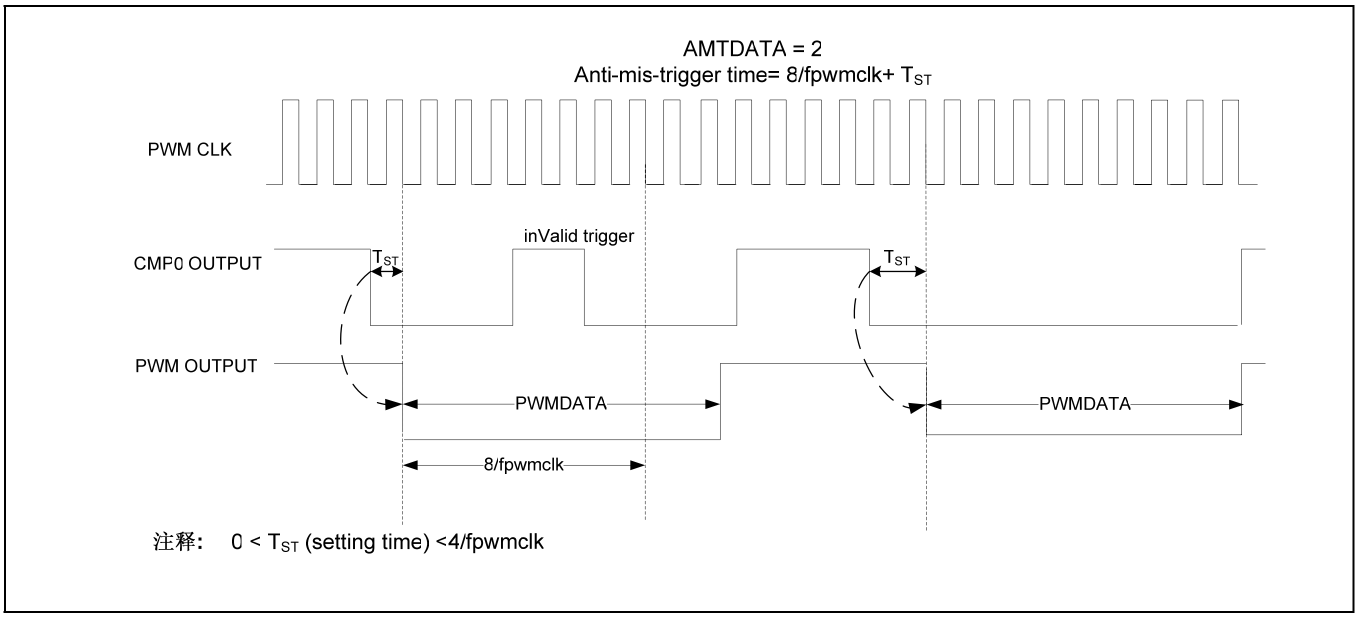
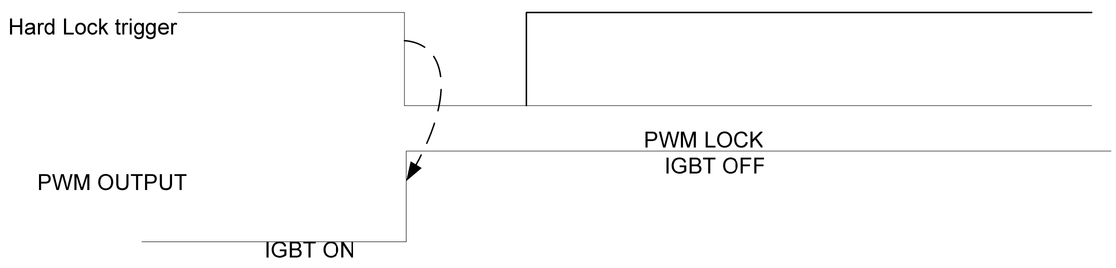
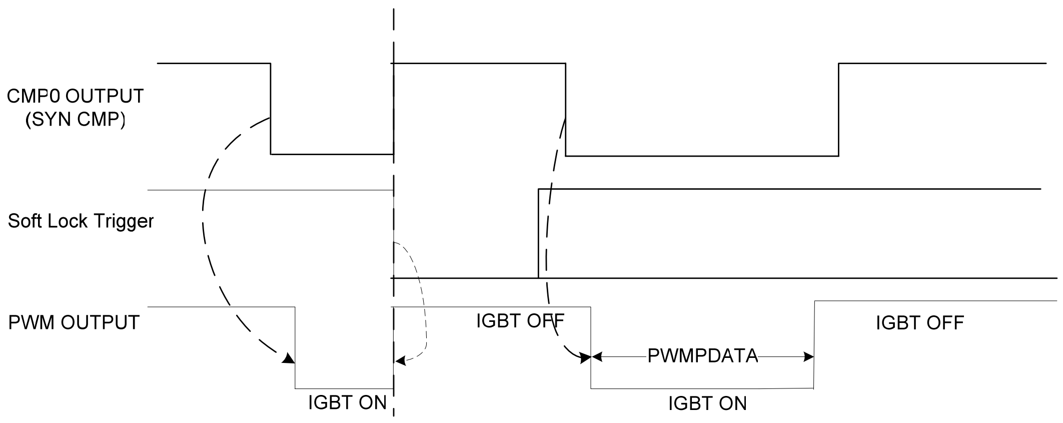


Figure 16-8. An example of the cooperation of PWM and Comparator 0_Anti-mis-Trigger



NOTE：Because CMP1/2/3 is asynchronous，lock action happens immediately without any setting time .

Figure 16-9. An example of the cooperation of PWM and Comparator 1/2/3 _ Hard Lock



NOTE：Because CMP1/2/3 is asynchronous，lock action happens immediately without any setting time .

Figure 16-10. An example of the cooperation of PWM and Comparator 1/2/3_Soft Lock

NOTES

17 PROGRAMABLE BUZZER

OVERVIEW

This microcontroller has integrated a programmable buzzer. The operation of Buzzer is controlled by a single control register, BUZCON.

Function description

The buzzer can output square wave with wide range frequency.

— 0.488KHz – 125KHz @ fOSC = 4MHz

BUZ CONTROL REGISTERS (BUZCON)

You use BUZ control register, BUZCON to

— Enable BUZ

— Select input clock frequency

— Program output frequency

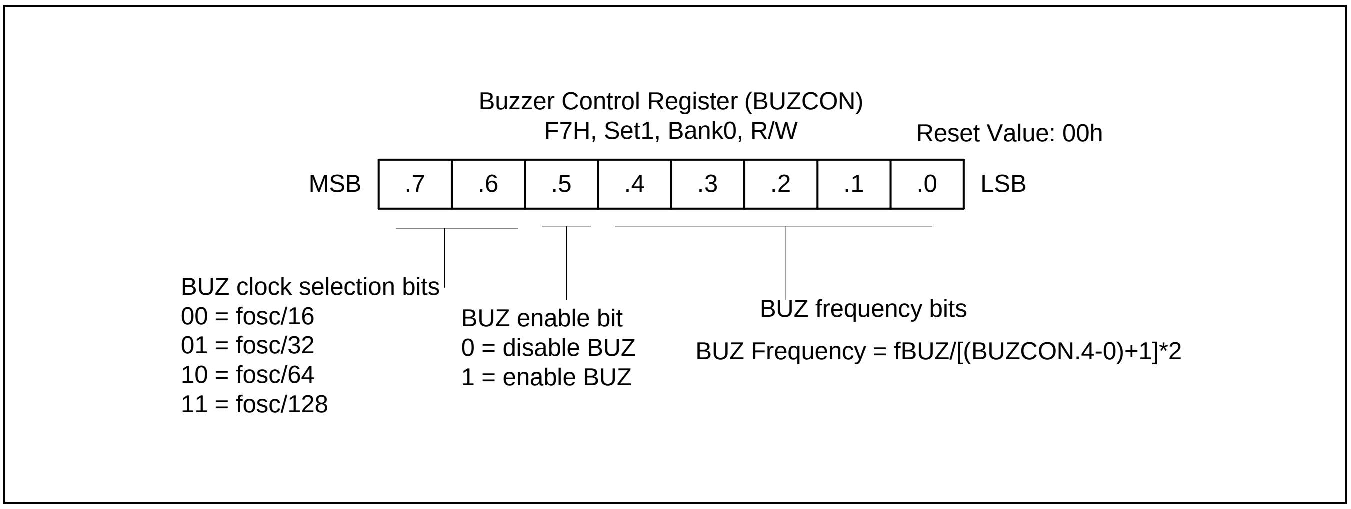


Figure 17-1. Buzzer Control Register (BUZCON)

BUZ FREQUENCY TABLE （@4MHZ）

BUZCON	Output frequency (KHz)				BUZCON	Output frequency (KHz)			
	f/16	f/32	f/64	f/128		f/16	f/32	f/64	f/128
31	3.906	1.953	0.977	0.488	15	7.813	3.906	1.953	0.977
30	4.032	2.016	1.008	0.504	14	8.333	4.167	2.083	1.042
29	4.167	2.083	1.042	0.521	13	8.929	4.464	2.232	1.116
28	4.310	2.155	1.078	0.539	12	9.615	4.808	2.404	1.202
27	4.464	2.232	1.116	0.558	11	10.417	5.208	2.604	1.302
26	4.630	2.315	1.157	0.579	10	11.364	5.682	2.841	1.420
25	4.808	2.404	1.202	0.601	9	12.500	6.250	3.125	1.563
24	5.000	2.500	1.250	0.625	8	13.889	6.944	3.472	1.736
23	5.208	2.604	1.302	0.651	7	15.625	7.813	3.906	1.953
22	5.435	2.717	1.359	0.679	6	17.857	8.929	4.464	2.232
21	5.682	2.841	1.420	0.710	5	20.833	10.417	5.208	2.604
20	5.952	2.976	1.488	0.744	4	25.000	12.5	6.25	3.125
19	6.250	3.125	1.563	0.781	3	31.250	15.625	7.813	3.906
18	6.579	3.289	1.645	0.822	2	41.667	20.833	10.417	5.208
17	6.944	3.472	1.736	0.868	1	62.500	31.250	15.625	7.813
16	7.353	3.676	1.838	0.919	0	125.000	62.500	31.250	15.625

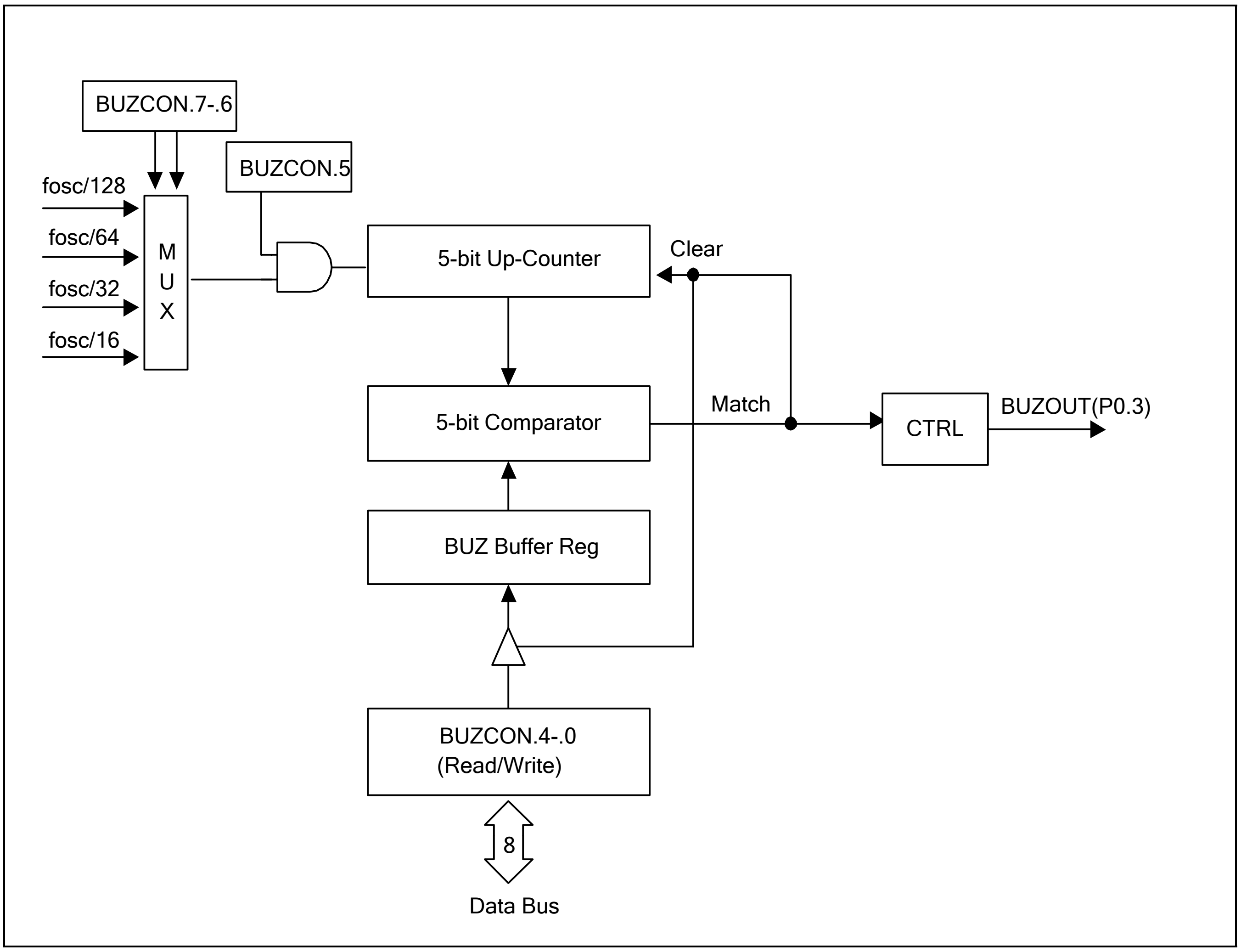


Figure 17-2. BUZ Functional Block Diagram

NOTES

18 S3F84B8 FLASH MCU

OVERVIEW

The S3F84B8 single-chip CMOS microcontroller has an on-chip Flash MCU ROM. The Flash ROM can be accessed by serial data format.

NOTE

This chapter is about the Tool Program Mode of Flash MCU. Please refer to chapter 19 Embedded Flash Memory Interface for detail about User Program Mode.

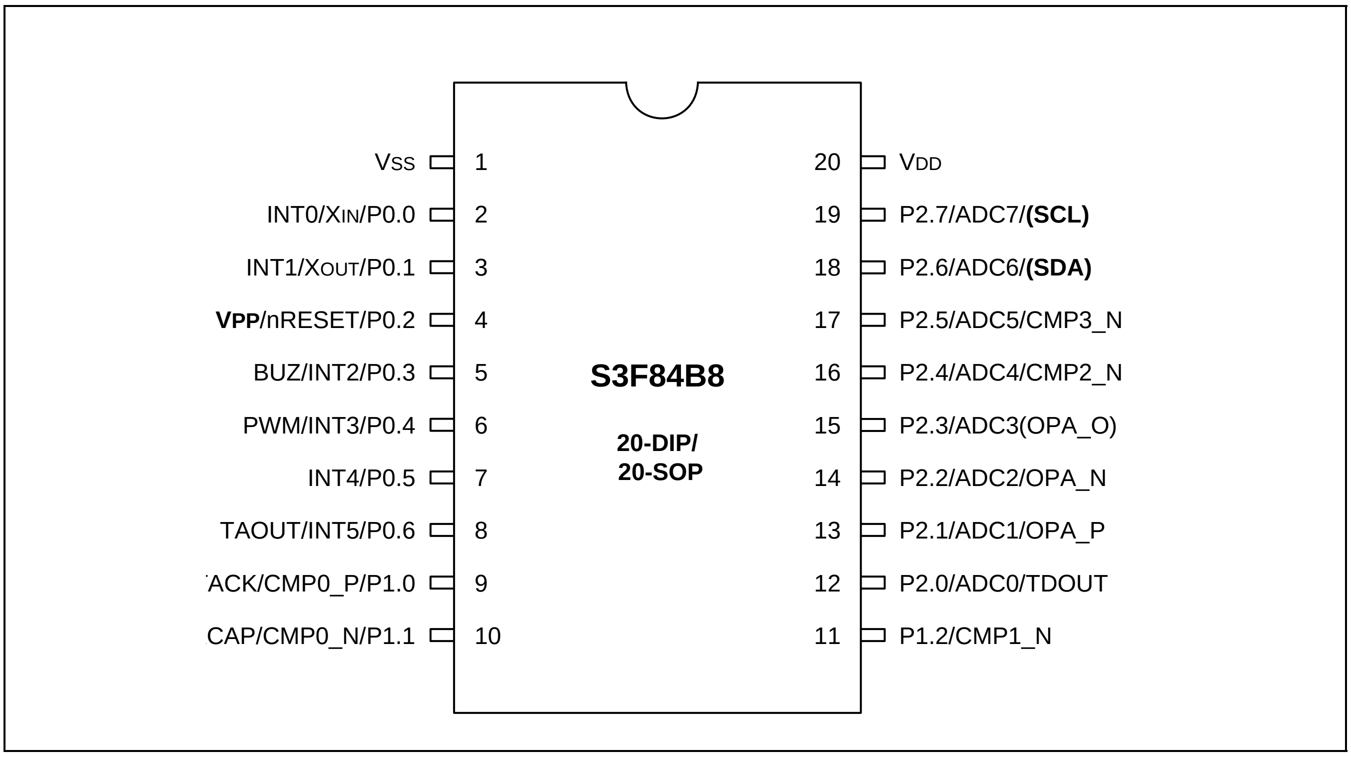


Figure 18-1. Pin Assignment Diagram (20-Pin SOP/DIP Package)

Table 18-1. Descriptions of Pins Used to Read/Write the Flash ROM

Main Chip	**During Programming**			
Pin Name	**Pin Name**	**Pin No.**	**I/O**	**Function**
P2.6	SDAT	18	I/O	Serial data pin. Output port when reading and input port when writing.
P2.7	SCLK	19	I	Serial clock pin.
RESET/P0.2	V_{PP}	4	I	Power supply pin for flash ROM cell writing (indicates that MTP enters into the writing mode). **When 11 V is applied, MTP is in Tool mode**
V_{DD}, V_{SS}	V_{DD}, V_{SS}	20, 1	–	Power supply pin for logic circuit. VDD should be tied to +5.0V during programming.

NOTE: ***Vpp Pin Voltage***
The Vpp pin on socket board for OTP/MTP writer should be 11V. So Vpp pin on socket board must not be connected Vpp(12.5V) which is generated from some OTP/MTP writer. Thus a specific adapter board for S3F84B8 must be used, when using these OTP/MTP writers.

19 EMBEDDED FLASH MEMORY INTERFACE

OVERVIEW

The S3F84B8 has an on-chip flash memory internally instead of masked ROM. The flash memory is accessed by instruction 'LDC'. This is a sector erasable and a byte programmable flash. User can program the data in a flash memory area any time you want. The S3F84B8's embedded 8K-byte memory has two operating features as below:

— User Program Mode

— Tool Program Mode: Refer to the chapter 18. S3F84B8 FLASH MCU

Flash ROM Configuration

The S3F84B8 flash memory consists of 64 sectors. Each sector consists of 128bytes. So, the total size of flash memory is 128x64 bytes (8KB). User can erase the flash memory by a sector unit at a time and write the data into the flash memory by a byte unit at a time.

— 8Kbyte Internal flash memory

— Sector size: 128-Bytes

— 10years data retention

— Fast programming Time:
Sector Erase: 4ms (min)
Byte Program: 20us (min)

— Byte programmable

— User programmable by 'LDC' instruction

— Sector (128-Bytes) erase available

— External serial programming support

— Endurance: 10,000 Erase/Program cycles (min)

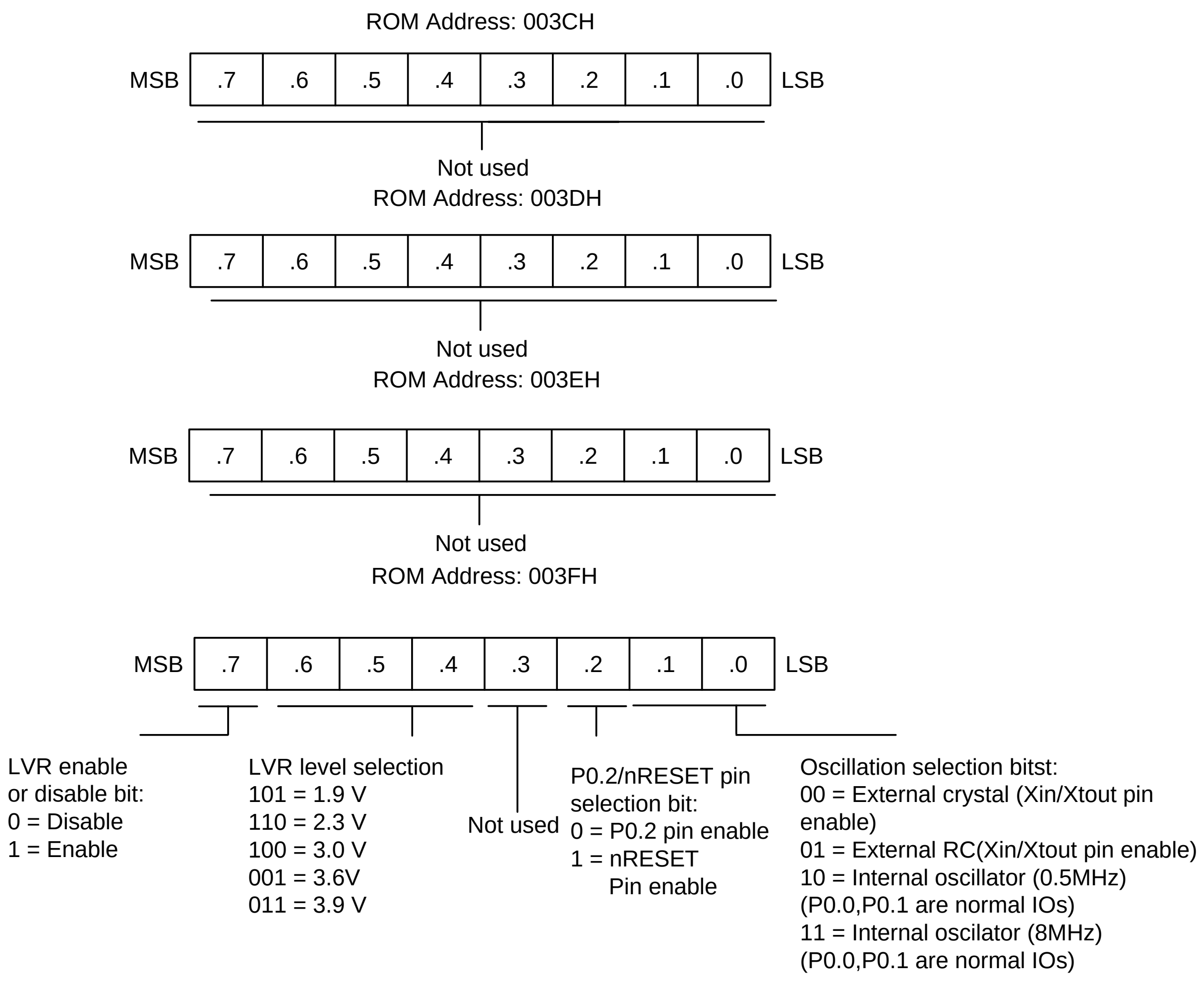
— Expandable OBPTM (On Board Program)

User Program Mode

This mode supports sector erase, byte programming, byte read and one protection mode (Hard Lock Protection). The S3F84B8 has the internal pumping circuit to generate high voltage. Therefore, 12.5V into Vpp (TEST) pin is not needed. To program a flash memory in this mode several control registers will be used.
There are four kind functions in user program mode – programming, reading, sector erase, and one protection mode (Hard lock protection).

SMART OPTION

Smart option is the program memory option for starting condition of the chip. The program memory addresses used by smart option are from 003CH to 003FH. The S3F84B8 only use 003FH. The default value of smart option bits in program memory is 0FFH. Before execution the program memory code, user can set the smart option bits according to the hardware option for user to want to select.


ROM Address: 003CH
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
Not used
ROM Address: 003DH
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
Not used
ROM Address: 003EH
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
Not used
ROM Address: 003FH
MSB .7 .6 .5 .4 .3 .2 .1 .0 LSB
LVR enable or disable bit: 0 = Disable 1 = Enable
LVR level selection 101 = 1.9 V 110 = 2.3 V 100 = 3.0 V 001 = 3.6V 011 = 3.9 V
Not used
P0.2/nRESET pin selection bit: 0 = P0.2 pin enable 1 = nRESET Pin enable
Oscillation selection bitst: 00 = External crystal (Xin/Xtout pin enable) 01 = External RC(Xin/Xtout pin enable) 10 = Internal oscillator (0.5MHz) (P0.0,P0.1 are normal IOs) 11 = Internal oscilator (8MHz) (P0.0,P0.1 are normal IOs)

NOTES:
1. The unused bits of 3CH, 3DH, 3EH, 3FH must be logic "1".
2. When LVR is enabled, LVR level must be set to appropriate value.
3. P0.2 has only input (without pull-up) function when sets 003F.2 as 0.
4. You must set P0.0,P0.1,P0.2 function on smart option. For example, if you select XIN (P0.0)/XOUT (P0.1)/nRESET(P0.2) function by smart option, you can't change them to Normal I/O after reset operation.

Figure 19-1. Smart Option

FLASH MEMORY CONTROL REGISTERS (USER PROGRAM MODE)

FLASH MEMORY CONTROL REGISTER (FMCON)

FMCON register is available only in user program mode to select the flash memory operation mode; sector erase, byte programming, and to make the flash memory into a hard lock protection.

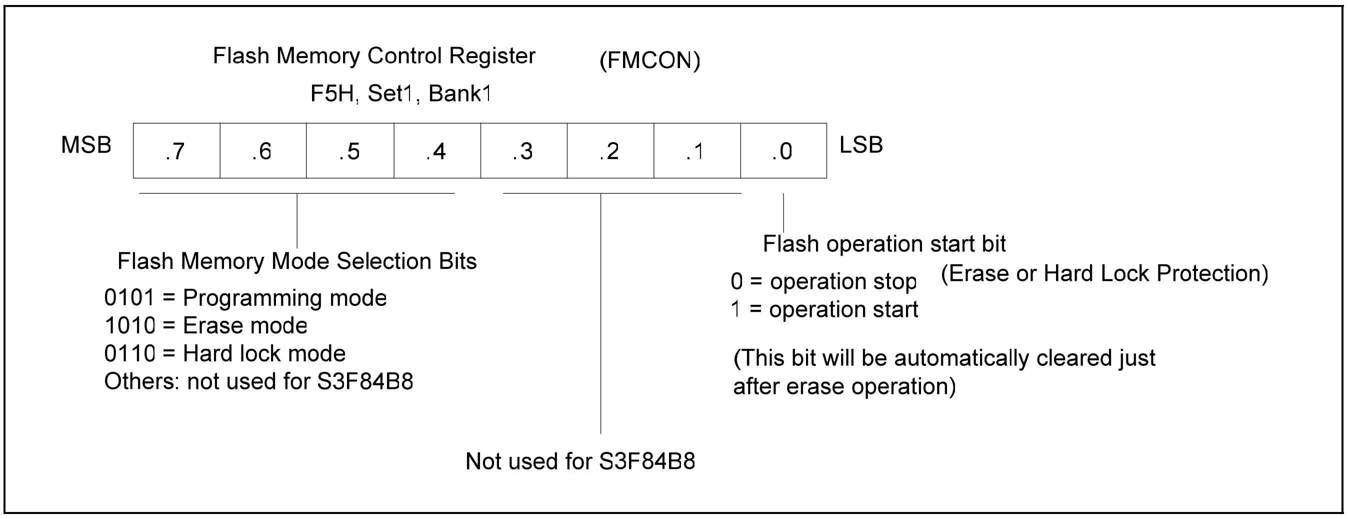

Flash Memory Control Register
(FMCON)
F5H, Set1, Bank1
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Flash Memory Mode Selection Bits
0101 = Programming mode
1010 = Erase mode
0110 = Hard lock mode
Others: not used for S3F84B8
Flash operation start bit
0 = operation stop
1 = operation start
(Erase or Hard Lock Protection)
(This bit will be automatically cleared just after erase operation)
Not used for S3F84B8

Figure 19-2. Flash Memory Control Register (FMCON)

The bit 0 of FMCON register (FMCON.0) is a bit for the operation start of Erase and Hard Lock Protection. Therefore, operation of Erase and Hard Lock Protection is activated when you set FMCON.0 to "1". If you write FMCON.0 to 1 for erasing, CPU is stopped automatically for erasing time (min.4ms). After erasing time, CPU is restarted automatically. When you read or program a byte data from or into flash memory, this bit does not need to manipulate.

FLASH MEMORY USER PROGRAMMING ENABLE REGISTER (FMUSR)

The FMUSR register is used for a safe operation of the flash memory. This register will protect undesired erase or program operation from malfunctioning of CPU caused by an electrical noise. After reset, the user-programming mode is disabled, because the value of FMUSR is "00000000B" by reset operation. If necessary to operate the flash memory, you can use the user programming mode by setting the value of FMUSR to "10100101B". The other value of "10100101B," user program mode is disabled.

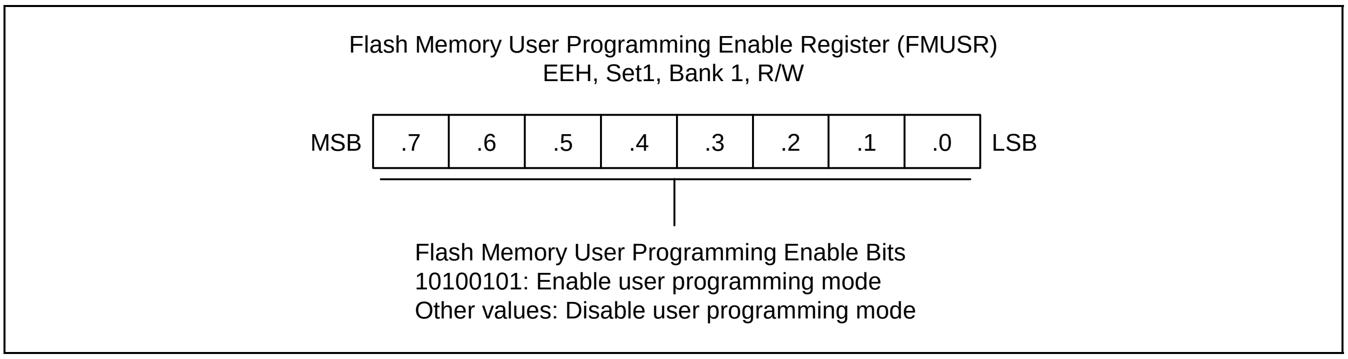

Flash Memory User Programming Enable Register (FMUSR)
EEH, Set1, Bank 1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Flash Memory User Programming Enable Bits
10100101: Enable user programming mode
Other values: Disable user programming mode

Figure 19-3. Flash Memory User Programming Enable Register (FMUSR)

FLASH MEMORY SECTOR ADDRESS REGISTERS

There are two sector address registers for the erase or programming flash memory. The FMSECL (Flash Memory Sector Address Register Low Byte) indicates the low byte of sector address and FMSECH (Flash Memory Address Sector Register High Byte) indicates the high byte of sector address.

One sector consists of 128-bytes. Each sector's address starts XX00H or XX80H, that is, a base address of sector is XX00H or XX80H. So bit.6-.0 of FMSECL is meaningless. When programming the flash memory, user should load sector base address before program. If the next operation is write one byte data too, user should check whether the next destination address is located in the same sector or not. In case of other sectors, user should reload sector address to FMSECH and FMSECL Register. (Refer to page 19-19 PROGRAMMING TIP — Programming)

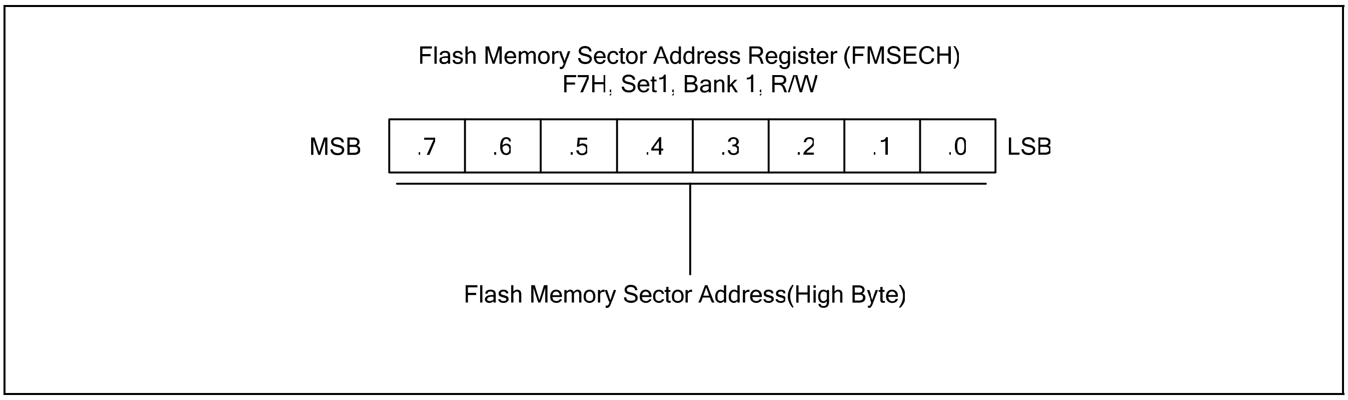

Flash Memory Sector Address Register (FMSECH)
F7H, Set1, Bank 1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Flash Memory Sector Address(High Byte)

Figure 19-4. Flash Memory Sector Address Register (FMSECH)

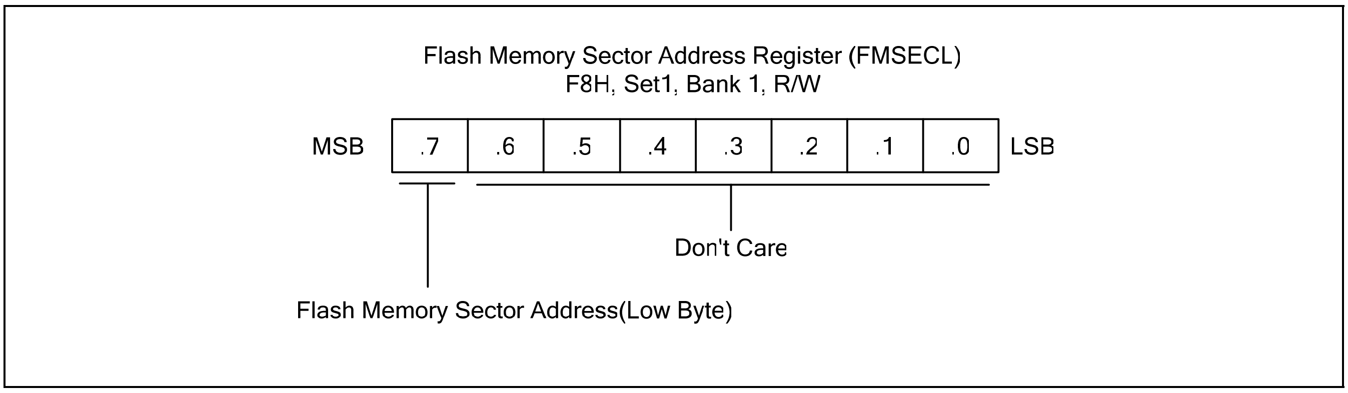

Flash Memory Sector Address Register (FMSECL)
F8H, Set1, Bank 1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Don't Care
Flash Memory Sector Address(Low Byte)

Figure 19-5. Flash Memory Sector Address Register (FMSECL)

SECTOR ERASE

User can erase a flash memory partially by using sector erase function only in user program mode. The only unit of flash memory to be erased in the user program mode is a sector.
The program memory of S3F84B8, 8Kbytes flash memory, is divided into 64 sectors. Every sector has 128-byte sizes. If you want to program a new data into flash memory, sector (128 bytes) erase is needed except the destination address has not been written yet after previous erase operation. Minimum 4ms' delay time for the erase is required after setting sector address and triggering erase start bit (FMCON.0). Sector erase is not supported in tool program modes (MDS mode tool or programming tool).

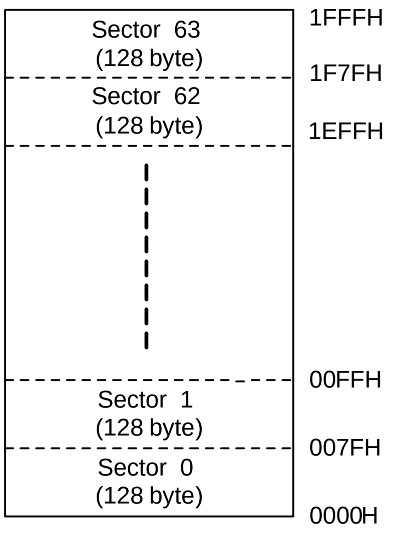
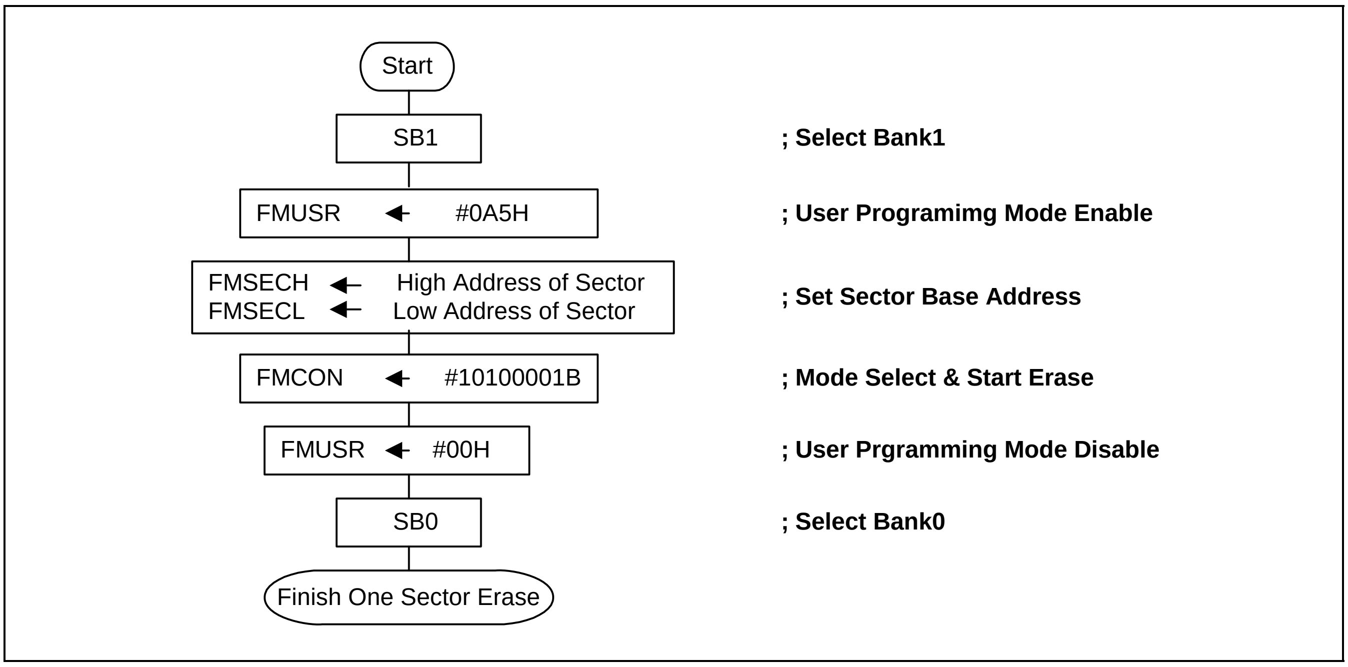


Figure 19-6. Sector Configurations in User Program Mode

The Sector Erase Procedure in User Program Mode

1. Set Flash Memory User Programming Enable Register (FMUSR) to "10100101B".
2. Set Flash Memory Sector Address Register (FMSECH and FMSECL).
3. Set Flash Memory Control Register (FMCON) to "10100001B".
4. Set Flash Memory User Programming Enable Register (FMUSR) to "00000000B".



Start
SB1
; Select Bank1
FMUSR ← #0A5H
; User Programimg Mode Enable
FMSECH ← High Address of Sector
FMSECL ← Low Address of Sector
; Set Sector Base Address
FMCON ← #10100001B
; Mode Select & Start Erase
FMUSR ← #00H
; User Prgramming Mode Disable
SB0
; Select Bank0
Finish One Sector Erase

Figure 19-7. Sector Erase Flowchart in User Program Mode

NOTES

1. If user erases a sector selected by Flash Memory Sector Address Register FMSECH and FMSECL, FMUSR should be enabled just before starting sector erase operation. And to erase a sector, Flash Operation Start Bit of FMCON register is written from operation stop '0' to operation start '1'. That bit will be cleared automatically just after the corresponding operation completed. In other words, when S3F94C8/F94C4 is in the condition that flash memory user programming enable bits is enabled and executes start operation of sector erase, it will get the result of erasing selected sector as user's a purpose and Flash Operation Start Bit of FMCON register is also clear automatically.
2. If user executes sector erase operation with FMUSR disabled, FMCON.0 bit, Flash Operation Start Bit, remains 'high', which means start operation, and is not cleared even though next instruction is executed. So user should be careful to set FMUSR when executing sector erase, for no effect on other flash sectors.

☞ PROGRAMMING TIP — Sector Erase

Case1. Erase one sector

```
                      •

                      •
ERASE_ONESECTOR:

                LD    FMUSR,#0A5H        ; User program mode enable
                LD    FMSECH,#04H        ; Set sector address 0400H, sector 8,
                LD    FMSECL,#00H        ; among sector 0~32
                LD    FMCON,#10100001B   ; Select erase mode enable & Start sector erase

ERASE_STOP:     LD    FMUSR,#00H         ; User program mode disable
```

PROGRAMMING

A flash memory is programmed in one-byte unit after sector erase. The write operation of programming starts by 'LDC' instruction.

The program procedure in user program mode

1. Must erase target sectors before programming.
2. Set Flash Memory User Programming Enable Register (FMUSR) to "10100101B".
3. Set Flash Memory Control Register (FMCON) to "0101000XB".
4. Set Flash Memory Sector Address Register (FMSECH and FMSECL) to the sector base address of destination address to write data.
5. Load a transmission data into a working register.
6. Load a flash memory upper address into upper register of pair working register.
7. Load a flash memory lower address into lower register of pair working register.
8. Load transmission data to flash memory location area on 'LDC' instruction by indirectly addressing mode
9. Set Flash Memory User Programming Enable Register (FMUSR) to "00000000B".

NOTE

In programming mode, FMCON.0 could either be '0' or '1'.

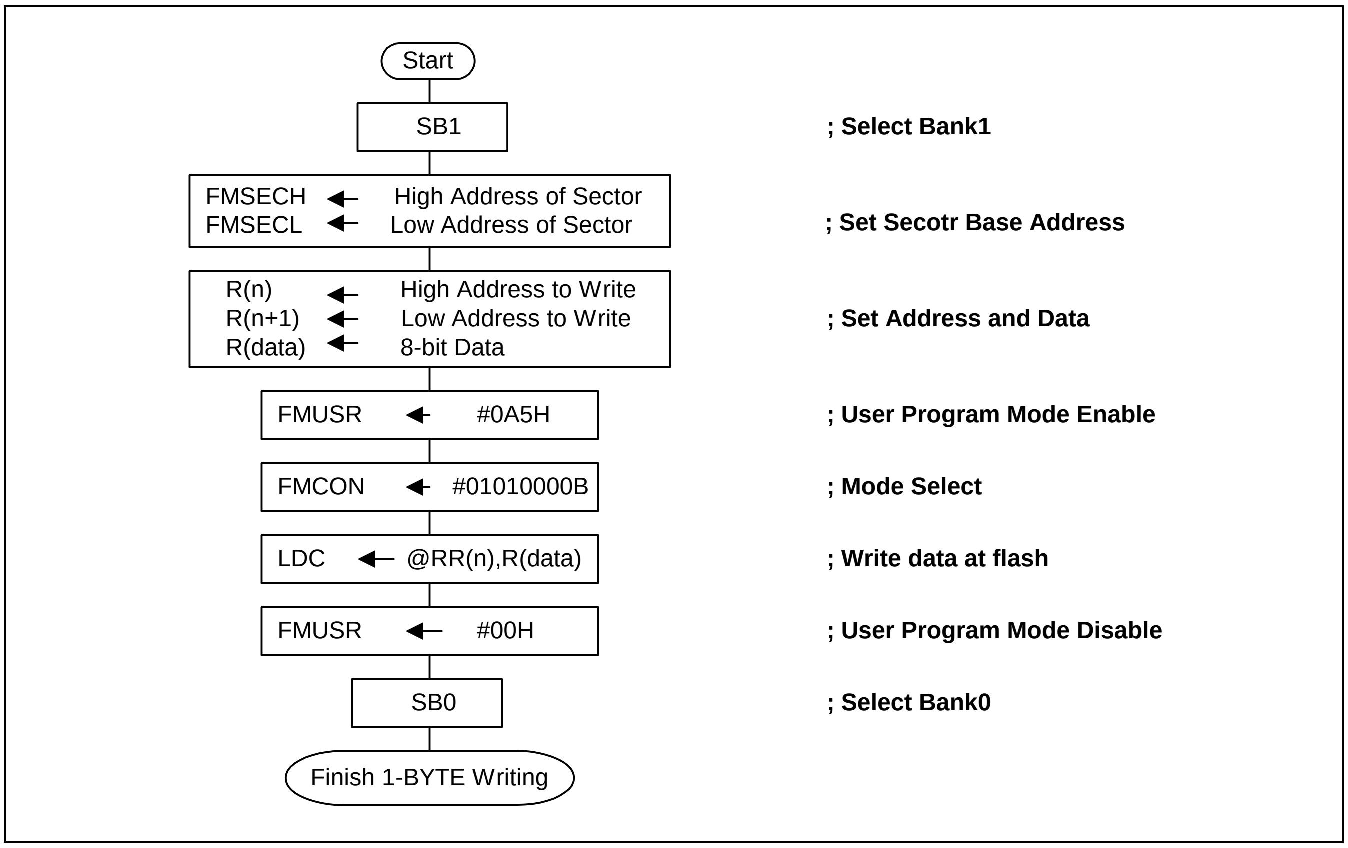


Figure 19-8. Byte Program Flowchart in a User Program Mode

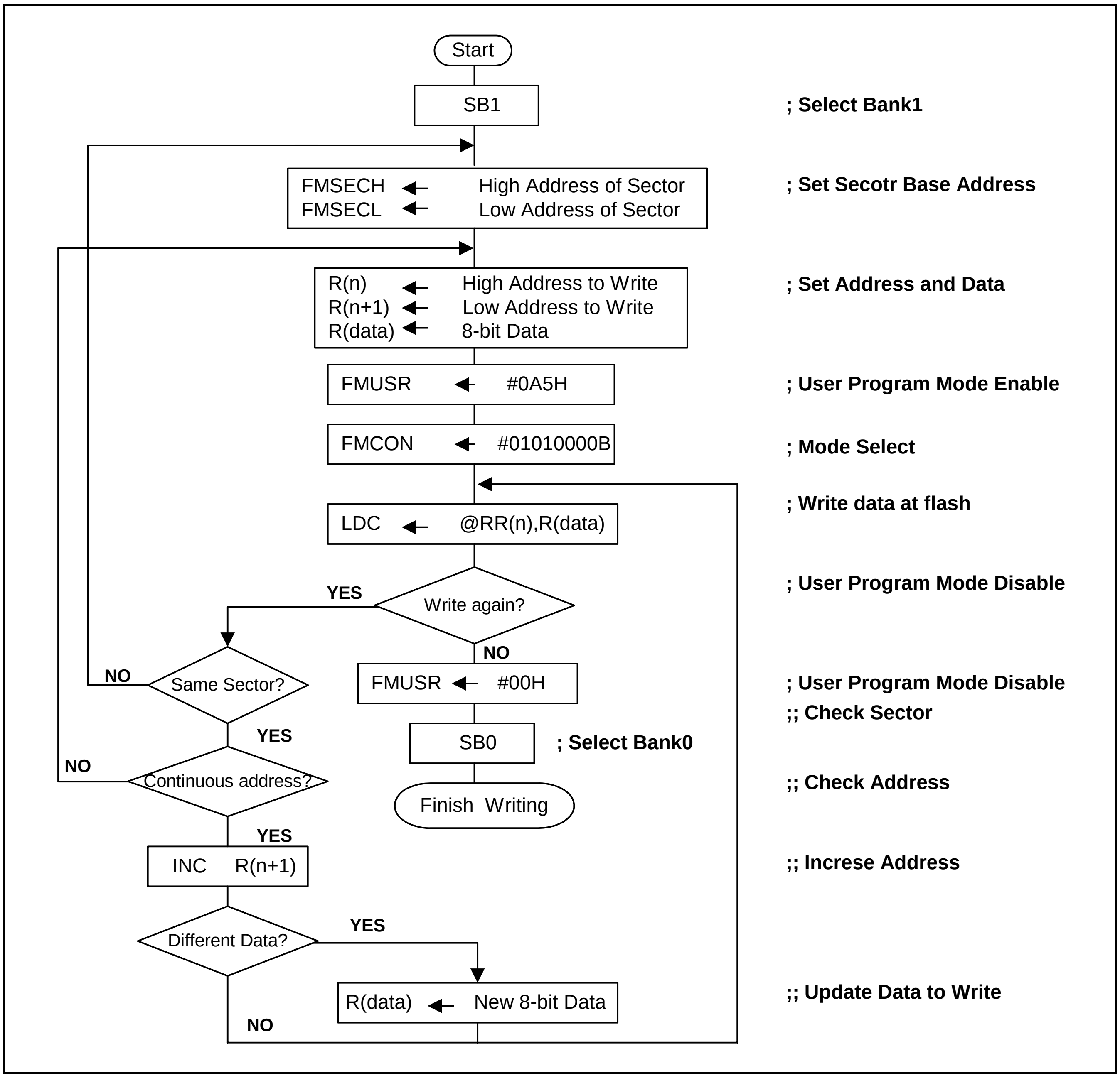


Figure 19-9. Program Flowchart in a User Program Mode

. PROGRAMMING TIP — Programming

Case1. 1-Byte Programming

```
      •
      •
WR_BYTE:                                ; Write data "AAH"  to destination address 0310H

      LD          FMUSR,#0A5H           ; User program mode enable
      LD          FMCON,#01010000B      ; Selection programming mode
      LD          FMSECH, #03H          ; Set the base address of sector (0300H)
      LD          FMSECL, #00H
      LD          R9,#0AAH              ; Load data "AA" to write
      LD          R10,#03H              ; Load flash memory upper address into upper register of pair working
                                        ; register
      LD          R11,#10H              ; Load flash memory lower address into lower register of pair working
                                        ; register
      LDC         @RR10,R9              ; Write data 'AAH' at flash memory location (0310H)

      LD          FMUSR,#00H             ; User program mode disable
```

Case2. Programming in the same sector

```
      •
      •
WR_INSECTOR:                            ; RR10-->Address copy  (R10 –high address,R11-low address)

      LD          R0,#40H


     LD           FMUSR,#0A5H           ; User program mode enable
     LD           FMCON,#01010000B      ; Selection programming mode and Start programming
     LD           FMSECH,#06H           ; Set the base address of sector located in target address to write data
     LD           FMSECL,#00H           ; The sector 12's base address is 0600H.
     LD           R9,#33H               ; Load data "33H" to write
     LD           R10,#06H              ; Load flash memory upper address into upper register of pair working
                                        ; register
      LD          R11,#00H              ; Load flash memory lower address into lower register of pair working
                                        ; register
WR_BYTE:
      LDC         @RR10,R9              ; Write data '33H' at flash memory location
      INC         R11                   ; Reset address in the same sector by INC instruction
     DEC          R0
    JP            NZ,   WR_BYTE          ; Check whether the end address for programming reach 0640H or not.

      LD          FMUSR,#00H            ; User Program mode disable
```

Case3. Programming to the flash memory space located in other sectors

```
    •
    •
WR_INSECTOR2:
    LD          R0,#40H
    LD          R1,#40H


    LD          FMUSR,#0A5H         ; User program mode enable
    LD          FMCON,#01010000B    ; Selection programming mode and Start programming
    LD          FMSECH,#01H         ; Set the base address of sector located in target address to write data
    LD          FMSECL,#00H         ; The sector 2's base address is 100H
    LD          R9,#0CCH            ; Load data "CCH" to write
    LD          R10,#01H            ; Load flash memory upper address into upper register of pair working
                                    ; register
    LD          R11,#40H            ; Load flash memory lower address into lower register of pair working
                                    ; register
    CALL        WR_BYTE

    LD          R0,#40H
WR_INSECTOR5:
    LD          FMSECH,#02H         ; Set the base address of sector located in target address to write data
    LD          FMSECL,#80H         ; The sector 5's base address is 0280H
    LD          R9,# 55H            ; Load data "55H" to write
    LD          R10,#02H            ; Load  flash memory upper address into upper register of pair working
                                    ; register
    LD          R11,#90H            ; Load flash memory lower address into lower register of pair working
                                    ; register
    CALL        WR_BYTE

WR_INSECTOR12:
    LD          FMSECH,#06H         ; Set the base address of sector located in target address to write data
    LD          FMSECL,#00H         ; The sector 12's base address is 0600H
    LD          R9,#0A3H            ; Load data "A3H" to write
    LD          R10,#06H            ; Load  flash memory upper address into upper register of pair working
                                    ; register
    LD          R11,#40H            ; Load flash memory lower address into lower register of pair working
                                    ; register
WR_BYTE1:
    LDC         @RR10,R9            ; Write data 'A3H' at flash memory location
    INC         R11
    DEC         R1
    JP          NZ, WR_BYTE1
    LD          FMUSR,#00H          ; User Program mode disable
    •
    •
WR_BYTE:
    LDC         @RR10,R9            ; Write data written by R9  at flash  memory location
    INC         R11
    DEC         R0
    JP          NZ, WR_BYTE
    RET
```

READING

The read operation starts by 'LDC' instruction.

The program procedure in user program mode

1. Load flash memory upper address into upper register of pair working register.
2. Load flash memory lower address into lower register of pair working register.
3. Load data from flash memory on 'LDC' instruction by indirectly addressing mode

PROGRAMMING TIP — Reading

```
          •
          •
          LD       R2,#03H     ; Load flash memory's upper address

                               ; to upper register of pair working register

          LD       R3,#00H     ; Load flash memory's lower address

                               ; to lower register of pair working register


LOOP:     LDC      R0,@RR2     ; Read data from flash memory location

                               ; (Between 300H and 3FFH)

          INC      R3

          CP       R3,#0FFH

          JP       NZ,LOOP

          •
          •
          •
          •
```

HARD LOCK PROTECTION

User can set Hard Lock Protection by writing '0110B' in FMCON7-4. This function prevents the changes of data in flash memory area. If this function is enabled, users cannot write or erase data in flash memory any more. This protection can be released by chip erase execution in tool program mode. In terms of user program mode, the procedure of setting Hard Lock Protection is as follows. In tool mode, please refer to the manual of serial program writer tool for hard lock protection usage.

The program procedure in user program mode

1. Set Flash Memory User Programming Enable Register (FMUSR) to "10100101B".
2. Set Flash Memory Control Register (FMCON) to "01100001B".
3. Set Flash Memory User Programming Enable Register (FMUSR) to "00000000B".

☞ **PROGRAMMING TIP — Hard Lock Protection**

```
•
•
SB1
LD      FMUSR,#0A5H             ; User program mode enable

LD      FMCON,#01100001B        ; Select Hard Lock Mode and Start protection

LD      FMUSR,#00H              ; User program mode disable
SB0

•

•
```

NOTES

20 LOW VOLTAGE RESET

OVERVIEW

By smart option (3FH.7 in ROM), user can select internal RESET (LVR) or external RESET.

The S3F84B8 can be reset in four ways:

— By external power-on-reset

— By the external reset input pin pulled low

— By the digital watchdog timing out

— By the Low Voltage reset circuit (LVR)

During an external power-on reset, the voltage V_{DD} is High level and the RESETB pin is forced Low level. The RESETB signal is input through a Schmitt trigger circuit where it is then synchronized with the CPU clock. This brings the S3F84B8 into a known operating status. To ensure correct start-up, the user should take that reset signal is not released before the V_{DD} level is sufficient to allow MCU operation at the chosen frequency.

The RESETB pin must be held to Low level for a minimum time interval after the power supply comes within tolerance in order to allow time for internal CPU clock oscillation to stabilize.

When a reset occurs during normal operation (with both V_{DD} and RESETB at High level), the signal at the RESETB pin is forced Low and the reset operation starts. All system and peripheral control registers are then set to their default hardware reset values (see Table 8-1).

The MCU provides a watchdog timer function in order to ensure graceful recovery from software malfunction. If watchdog timer is not refreshed before an end-of-counter condition (overflow) is reached, the internal reset will be activated.

The S3F84B8 has a built-in low voltage reset circuit that provides detection of power voltage drop of external V_{DD} input level to prevent MCU from malfunctioning in an unstable MCU power level. This voltage detector works for the reset operation of MCU. This Low Voltage reset includes an analog comparator and Vref circuit. The value of a detection voltage is set internally by hardware. The on-chip Low Voltage Reset, features static reset when supply voltage is below a reference voltage value (Typical 1.9/2.3/3.03.6/3.9 V). Thanks to this feature, external reset circuit can be removed while keeping the application safe. As long as the supply voltage is below the reference value, an internal static RESET will be triggered. The MCU can start only when the supply voltage rises over the reference voltage.

When you calculate power consumption, please remember that a static current of LVR circuit should be added a CPU operating current in any operating modes such as Stop, Idle, and normal RUN mode.

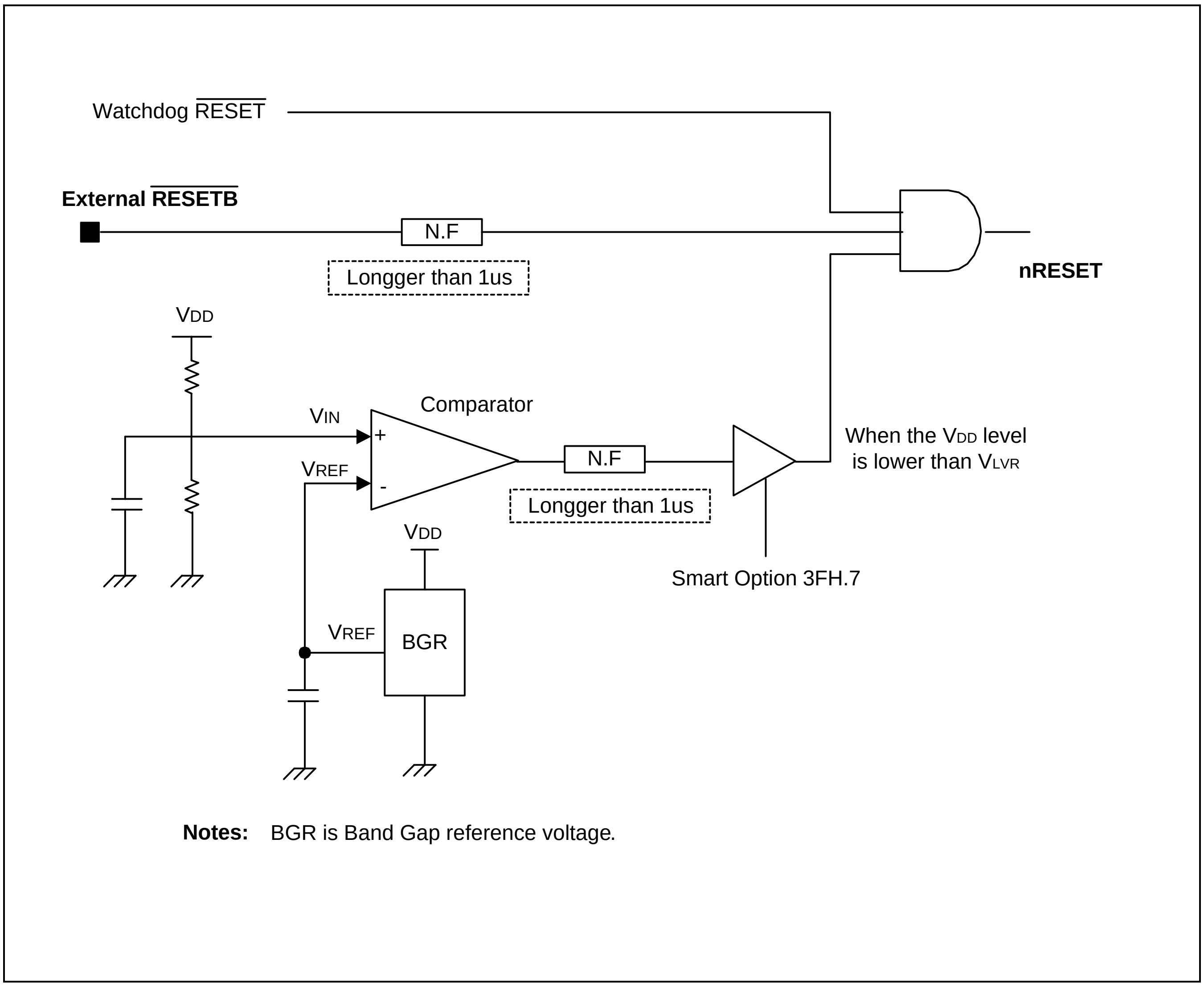

Watchdog $\overline{\text{RESET}}$
External $\overline{\text{RESETB}}$
N.F
Longger than 1us
nRESET
VDD
VIN
Comparator
+
VREF
-
N.F
Longger than 1us
When the VDD level is lower than VLVR
VDD
Smart Option 3FH.7
VREF
BGR
Notes: BGR is Band Gap reference voltage.

Figure 20-1. Low Voltage Reset Circuit

NOTE

To program the duration of the oscillation stabilization interval, you make the appropriate settings 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.

21 ELECTRICAL DATA

OVERVIEW

In this section, the following S3F84B8 electrical characteristics are presented in tables and graphs:

— Absolute maximum ratings
— D.C. electrical characteristics
— A.C. electrical characteristics
— Input timing measurement points
— Oscillator characteristics
— Oscillation stabilization time
— Operating voltage range
— Schmitt trigger input characteristics
— Data retention supply voltage in stop mode
— Stop mode release timing when initiated by a RESET
— A/D converter electrical characteristics
— OP Amp electrical characteristics
— Comparator electrical characteristics
— LVR circuit characteristics
— LVR reset timing
— Full-Flash memory characteristics
— ESD Characteristics

Table 21-1. Absolute Maximum Ratings

(T_A = 25 °C)

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{DD}	–	– 0.3 to + 6.5	V
Input voltage	V_I	All ports	– 0.3 to V_{DD} + 0.3	V
Output voltage	V_O	All output ports	– 0.3 to V_{DD} + 0.3	V
Output current high	I_{OH}	One I/O pin active	– 25	mA
		All I/O pins active	– 80	
Output current low	I_{OL}	One I/O pin active	+ 30	mA
		All I/O pins active	+ 100	
Operating temperature	T_A	–	– 40 to + 85	°C
Storage temperature	T_{STG}	–	– 65 to + 150	°C

Table 21-2. DC Electrical Characteristics

(T_A = – 40 °C to + 85 °C, V_{DD} = 1.8 V to 5.5 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
Operating Voltage	Vdd	f_{main}=0.4 – 2 MHz		1.8	–	5.5	V
		f_{main}=0.4 – 4 MHz		2.0	–	5.5	
		f_{main}=0.4 – 10 MHz		2.7	–	5.5	
Main crystal or ceramic frequency	f_{main}	VDD = 2.7 V to 5.0V		0.4	–	10	MHz
		VDD = 1.8 V to 2.7V		0.4	–	4	
Input high voltage	V_{IH1}	Ports 0,1, 2 and RESET	V_{DD}= 1.8 to 5.5 V	0.8 V_{DD}	–	V_{DD}	V
	V_{IH2}	X_{IN}		V_{DD}- 0.1			
Input low voltage	V_{IL1}	Ports 0, 1, 2 and RESET	V_{DD}= 1.8 to 5.5 V	–	–	0.2 V_{DD}	V
	V_{IL2}	X_{IN}				0.1	
Output high voltage	V_{OH}	I_{OH} = – 10 mA Ports 0,1,2	V_{DD}= 4.5 to 5.5 V	V_{DD}-1.5	V_{DD}- 0.4	–	V
Output low voltage	V_{OL}	I_{OL} = 25 mA Ports 0,1,2	V_{DD}= 4.5 to 5.5 V	–	0.4	2.0	V
Input high leakage current	I_{LIH1}	All input except P0.2 and I_{LIH2}	V_{IN} = V_{DD}	–	–	1	uA
	I_{LIH2}	X_{IN}	V_{IN} = V_{DD}			20	
Input low leakage current	I_{LIL1}	All input except P0.2 and I_{LIL2}	V_{IN} = 0 V	–	–	–1	uA
	I_{LIL2}	X_{IN}	V_{IN} = 0 V			–20	
Output high leakage current	I_{LOH}	All output pins	V_{OUT} = V_{DD}	–	–	2	uA
Output low leakage current	I_{LOL}	All output pins	V_{OUT} = 0 V	–	–	–2	uA
Pull-up resistors	R_{P1}	V_{IN} = 0 V, Ports 0, 1,2	V_{DD} = 5 V T_A=25°C	25	50	100	kΩ
Supply current	I_{DD1}	Run mode 10 MHz CPU clock	V_{DD} = 4.5 to 5.5 V	–	3	6	mA
	I_{DD2}	Idle mode 10 MHz CPU clock	V_{DD} = 4.5 to 5.5 V	–	2	4	
	I_{DD3}	Stop mode	V_{DD} = 4.5 to 5.5 V (LVR disable) T_A = – 40 °C~85 °C	–	0.6	4.0	uA
			V_{DD} = 4.5 to 5.5 V (LVR enable) T_A = – 40 °C~85 °C		40	100	

NOTE: Supply current does not include current drawn through internal pull-up resistors or external output current loads and ADC module.

Table 21-3. AC Electrical Characteristics

(T_A = – 40 °C to + 85 °C, V_{DD} = 1.8 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	t_{INTH} t_{INTL}	INT0, INT1 V_{DD} = 5 V ± 10 %	–	200	–	ns
RESET input low width	t_{RSL}	Input V_{DD} = 5 V ± 10 %	10	–	–	us

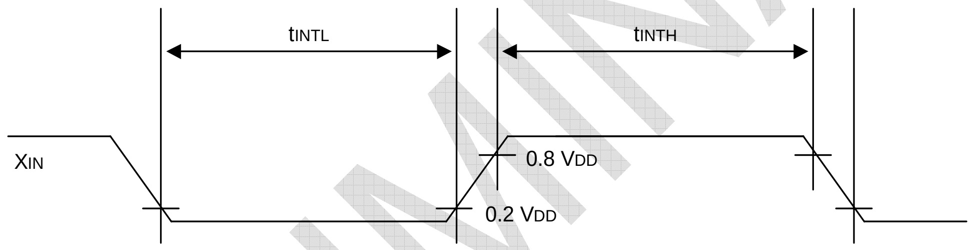
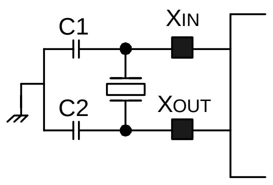
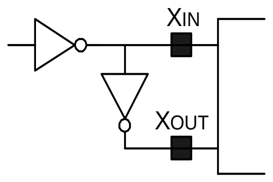


Figure 21-1. Input Timing Measurement Points

Table 21-4. Oscillator Characteristics

(T_A = – 40°C to + 85 °C)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Main crystal or ceramic		V_{DD} = 2.7 to 5.5 V	0.4	–	10	MHz
		$V_{DD}^{(1)}$ = 2.0 to 2.7 V	0.4	–	4	MHz
		$V_{DD}^{(1)}$ = 1.8 to 2.0 V	0.4	–	2	MHz
External clock (Main System)		V_{DD} = 2.7 to 5.5 V	0.4	–	10	MHz
		V_{DD} = 1.8 to 2.7 V	0.4	–	4	MHz
External RC oscillator	–	V_{DD} = 5.0 V	–	8	–	
Tolerance of Internal RC	–	Factory calibrated at 25°C, 5.0V	–	–	±3	%
	–	V_{DD} = 5.0V T_A = – 40°C to + 85 °C	–	–	±6	%
	–	V_{DD} =2.0 to 5.5V T_A = – 40°C to + 85 °C	–	–	±9	%

NOTE: 1. Please refer to the figure of Operating Voltage Range.

Table 21-5. Oscillation Stabilization Time

(T_A = - 40 °C to + 85 °C, V_{DD} = 1.8 V to 5.5 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Main crystal stabilization time	f_{OSC} > 1.0 MHz	–	–	20	ms
Main ceramic stabilization time	Oscillation stabilization occurs when V_{DD} is equal to the minimum oscillator voltage range.	–	–	10	ms
External clock (main system)	X_{IN} input high and low width (t_{XH}, t_{XL})	25	–	500	ns
Oscillator stabilization wait time	t_{WAIT} when released by a reset (1)	–	$2^{19}/f_{OSC}$	–	ms
	t_{WAIT} when released by an interrupt (2)	–	–	–	ms

NOTES:
1. f_{OSC} is the oscillator frequency.
2. The duration of the oscillator stabilization wait time, t_{WAIT}, when it is released by an interrupt is determined by the settings in the basic timer control register, BTCON.

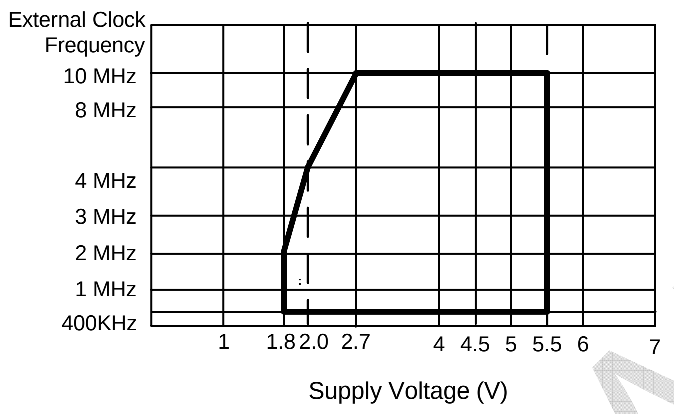


Figure 21-2. Operating Voltage Range @ External clock

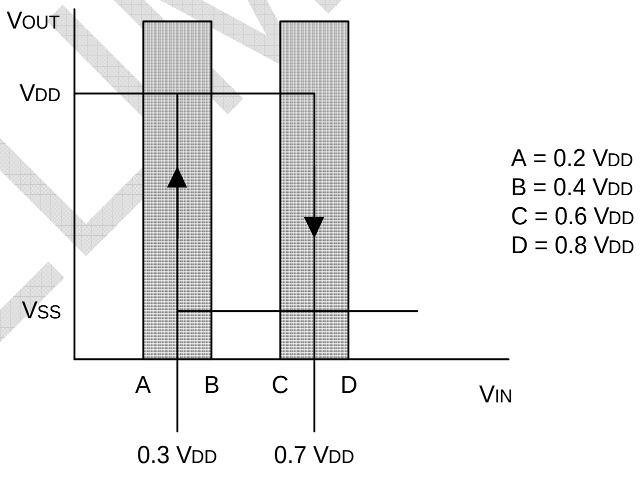


Figure 21-3. Schmitt Trigger Input Characteristics Diagram

Table 21-6. Data Retention Supply Voltage in Stop Mode

(T_A = – 40 °C to + 85 °C, V_{DD} = 1.8 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}	Stop mode	1.0	–	5.5	V
Data retention supply current	I_{DDDR}	Stop mode; V_{DDDR} = 1.8 V	–	–	1	uA

NOTE: Supply current does not include current drawn through internal pull-up resistors or external output current loads.

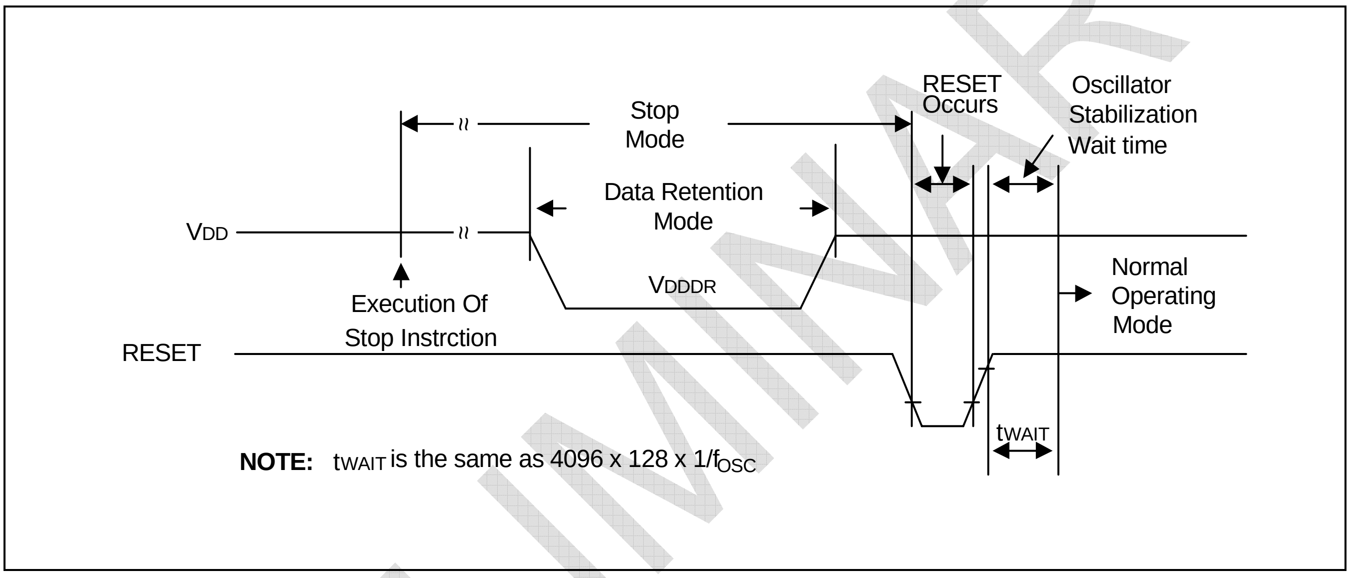

Stop Mode
RESET Occurs
Oscillator Stabilization Wait time
Data Retention Mode
VDD
VDDDR
Execution Of Stop Instrction
Normal Operating Mode
RESET
tWAIT
NOTE: tWAIT is the same as 4096 x 128 x 1/fOSC

Figure 21-4. Stop Mode Release Timing When Initiated by a RESET

Table 21-7. A/D Converter Electrical Characteristics

(T_A = – 40 °C to + 85 °C, V_{DD} = 1.8 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Unit
Resolution			–	10	–	bit
Total accuracy		V_{DD} = 5.12 V(1) CPU clock = 10 MHz V_{SS} = 0 V	–	–	±3	LSB
Integral linearity error	ILE		–	–	± 2	LSB
Differential linearity error	DLE	″	–	–	± 1	LSB
Offset error of top	EOT	″	–	± 1	± 3	LSB
Offset error of bottom	EOB	″	–	± 1	± 3	LSB
Conversion time (2)	t_{CON}	–	12.5	20		μs
Analog input voltage	V_{IAN}	–	V_{SS}	–	V_{DD}	V
Analog input impedance	R_{AN}	–	2	1000	–	MΩ
Analog input current	I_{ADIN}	V_{DD} = 5 V	–	–	10	μA
Analog block current (3)	I_{ADC}	V_{DD} = 5 V	–	0.5	1.5	mA
		V_{DD} = 5 V power down mode		100	500	nA

NOTES:
1. When VDD = 2.7V – 5.5V, the total accuracy is characterized to be 3LSB (max), but not tested.
2. "Conversion time" is the time required from the moment a conversion operation starts until it ends.
2. I_{ADC} is operating current during A/D conversion.

Table 21-1. OPAMP electrical characteristics

(T_A = - 40 C to + 85 C , V_{DD} = 2.0 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Input Offset Voltage	|Vio|	VDD=2.0V	–	10	30[1]	mV
		VDD=5.5V	–	10	30[1]	mV
Input Common-Mode Voltage Range [2]	Vcm		GND	–	VDD-0.1	V
Output Voltage	Vout		GND+0.1	–	VDD-0.1	V

NOTES:

1. Refer to application note for the hardware and software calibration methods.
2. The input signal voltage should not go below -0.3V

Table 21-1. Comparator electrical characteristics

(TA = - 40 C to + 85 C , VDD = 2.0 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
CMP0 Input Offset Voltage [(1)]	|Vio|	VDD = 2V	–	10	20	mV
		VDD = 5.5V	–	10	20	mV
CMP1/2/3 Input Offset Voltage [(1)] [(2)]	|Vio|	VDD = 2V	–	15	30	mV
		VDD = 5.5V	–	15	30	mV
CMP0 Input Common-Mode Voltage Range	Vcm		GND	–	VDD-0.1	V

NOTES:

1. These parameters are characterized only.
2. The parameter includes the tolerance of internal voltage reference.

Table 21-8. LVR Circuit Characteristics

(T_A = 25 °C, V_{DD} = 1.8 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Low voltage reset	V_{LVR}	–	1.8 2.1 2.8 3.4 3.7	1.9 2.3 3.0 3.6 3.9	2.0 2.5 3.2 3.8 4.1	V

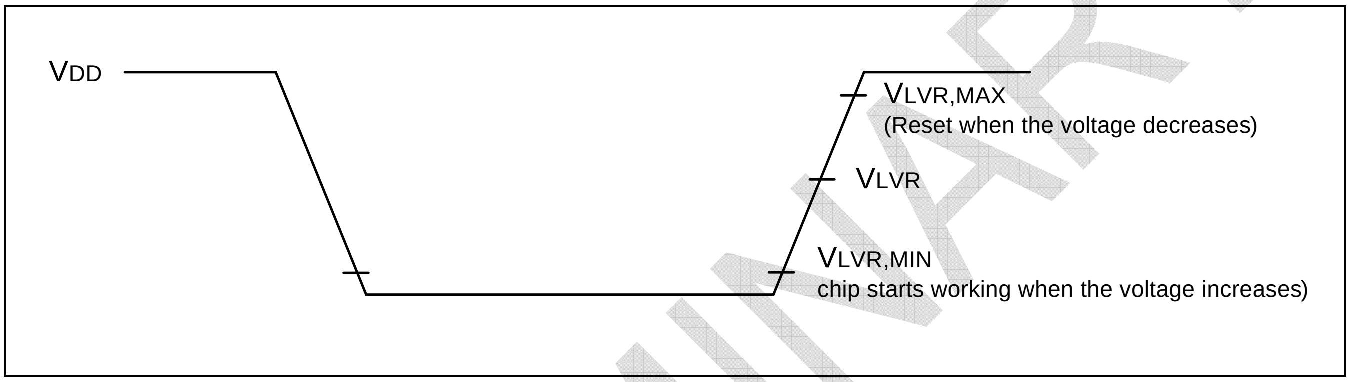

VDD
VLVR,MAX
(Reset when the voltage decreases)
VLVR
VLVR,MIN
chip starts working when the voltage increases)

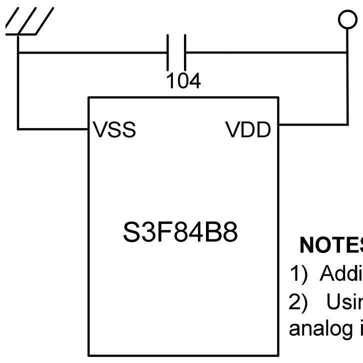
Figure 21-5. LVR Reset Timing

Table 21-9. FLASH MEMORY AC Electrical characteristics

(T_A = – 40 °C to + 85 °C at V_{DD} = 1.8 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Flash Erase/Write/Read Voltage	Fewrv	VDD	1.8	5.0	5.5	V
Programming time(1)	Ftp		20	–	30	uS
Chip Erasing time (2)	Ftp1		32	–	70	mS
Sector Erasing time (3)	Ftp2		4	–	12	mS
Data Access Time	Ft_{RS}	VDD = 2.0V	–	250	–	nS
Number of writing/erasing	FNwe	–	10,000	–	–	Times
Data Retention	Ftdr	–	10	–	–	Years

Notes:
1. The programming time is the time during which one byte (8-bit) is programmed.
2. The Chip erasing time is the time during which entire program memory is erased.
3. The Sector erasing time is the time during which all 128byte block is erased.
4. The chip erasing is available in Tool Program Mode only.



NOTES: To have better EFT performance, we recommend
1) Adding 104 capacitor at VDD pin.
2) Using 104, 102, or 101 capacitor at all input pins, especially analog input pins.

Figure 21-6. The Circuit Diagram to Improve EFT Characteristics

Table 21-9 ESD Characteristics

Parameter	Symbol	Conditions	Min	Typ.	Max	Unit
Electrostatic discharge	V_{ESD}	HBM	2000	~	~	V
		MM	200	~	~	V
		CDM	500	~	~	V

NOTES

22 DEVELOPMENT TOOLS

OVERVIEW

Samsung provide a powerful and ease-to-use development support system on a turnkey basis. The development support system is composed of a host system, debugging tools, and supporting software. For a host system, any standard computer that employs Win95/98/2000/XP as its operating system can be used. A sophisticated debugging tool is provided both in hardware and software: the powerful in-circuit emulator, OPENice-i500 and SK-1200, for the S3F7-, S3F9-and S3F8- microcontroller families. Samsung also offers supporting software that includes, debugger, an assembler, and a program for setting options.

TARGET BOARDS

Target boards are available for all the S3C8/S3F8-series microcontrollers. All the required target system cables and adapters are included on the device-specific target board. TB84B8 is a specific target board for the development of application systems using S3F84B8.

PROGRAMMING SOCKET ADAPTER

When you program S3F84B8's flash memory by using an emulator or OTP/MTP writer, you need a specific programming socket adapter for S3F84B8.

[Development System Configuration]

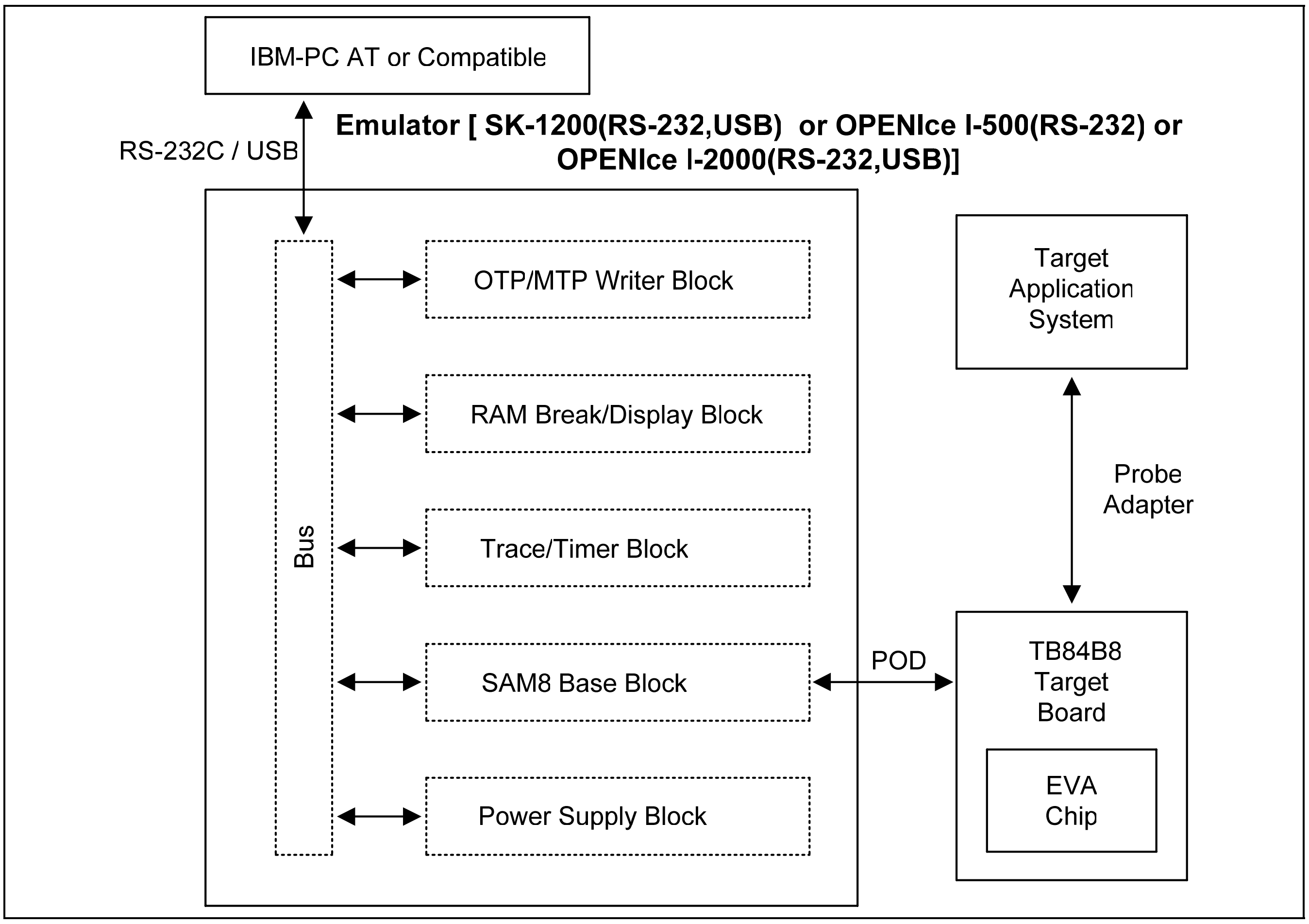


Figure 22-1. Development System Configuration

TB84B8 TARGET BOARD

The TB84B8 target board is used for the S3F84B8 microcontrollers. The TB84B8 target board is operated as target CPU with Emulator (OPENIce I-500/2000, SK-1200).

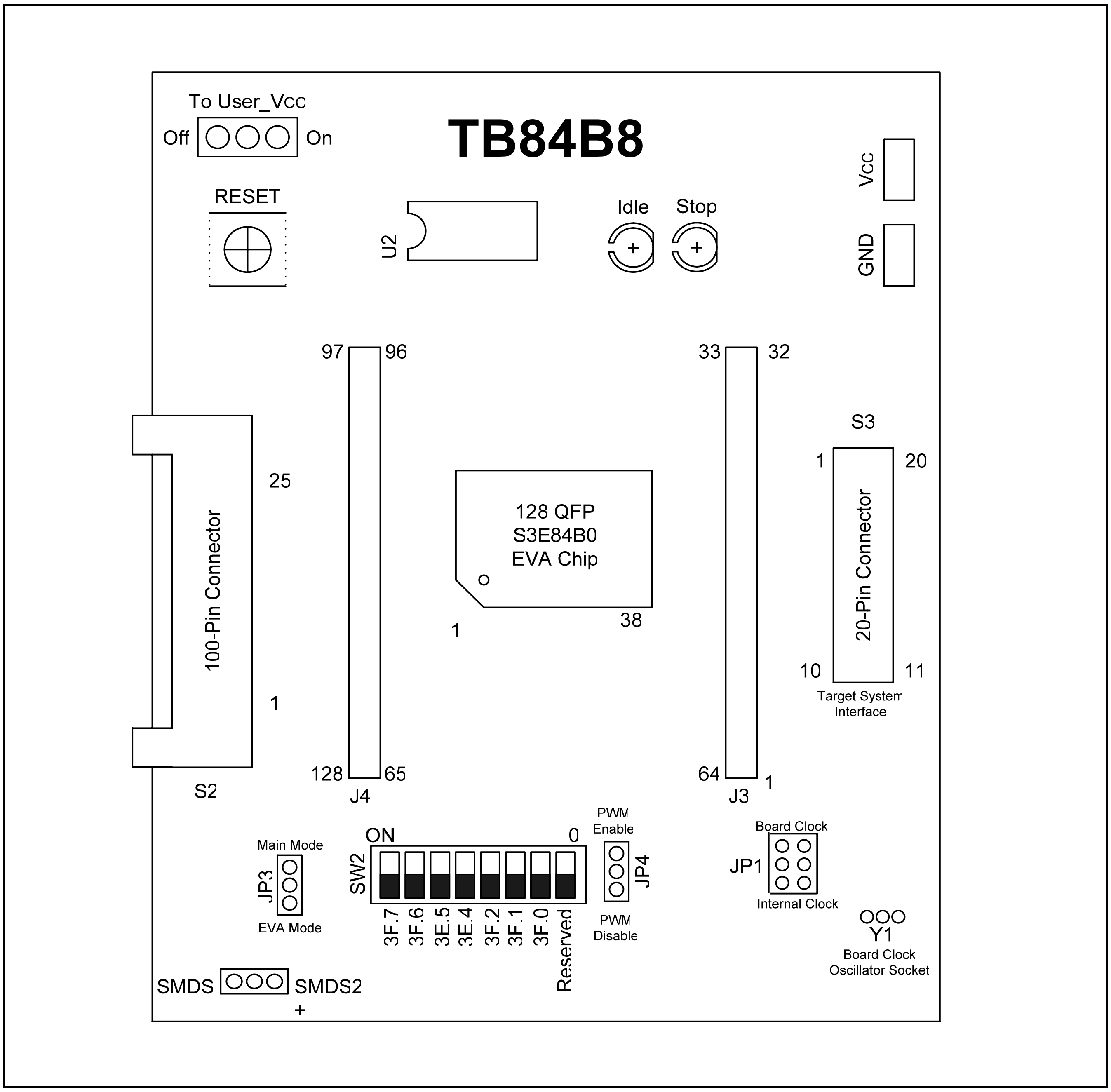
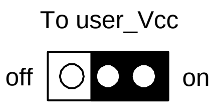
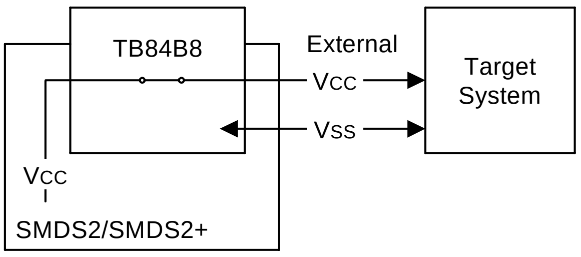
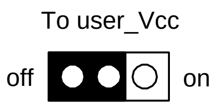
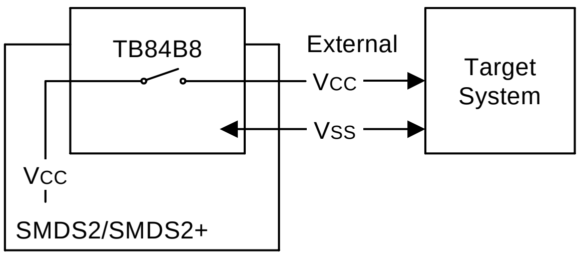


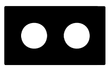
Figure 22-2. TB84B8 Target Board Configuration

NOTE: TB84B8 should be supplied 5V normally. So the power supply from Emulator should be set 5V for the target board operation.

Table 22-1. Power Selection Settings for TB84B8

"To User_Vcc" Settings	Operating Mode	Comments
	 TB84B8 External Vcc Vss Target System Vcc SMDS2/SMDS2+	The SMDS2/SMDS2+ main board supplies V_{CC} to the target board (evaluation chip) and the target system.
	 TB84B8 External Vcc Vss Target System Vcc SMDS2/SMDS2+	The SMDS2/SMDS2+ main board supplies V_{CC} only to the target board (evaluation chip). The target system must have its own power supply.

NOTE: The following symbol in the "To User_Vcc" Setting column indicates the electrical short (off) configuration:



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 22-2. The SMDS2+ Tool Selection Setting

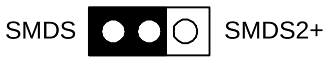
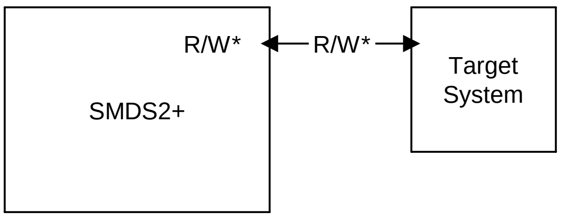
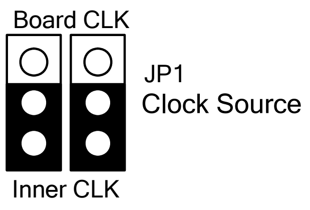
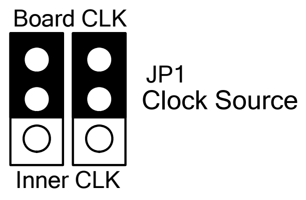
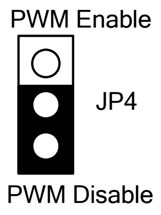
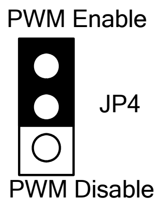
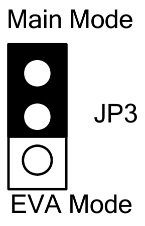
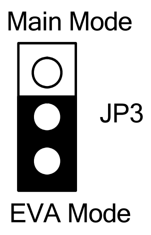
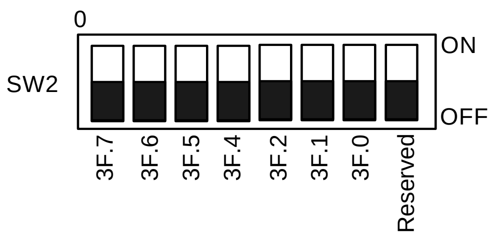
"SW1" Setting	Operating Mode
	

Table 22-3. Using Single Header Pins to Select Clock Source and Enable/Disable PWM

Target Board Part	Comments
	Use SMDS2/SMDS2+ internal clock source as the system clock. Default Setting
	Use external crystal or ceramic oscillator as the system clock.
	PWM function is DISABLED.
	PWM function is ENABLED. Default Setting
	The S3E94C0 run in main mode, just same as S3F84B8. The debug interface is not available.
	The S3E94C0 run in EVA mode, available. When debug program, please set the jumper in this mode. Default Setting



ON	Low
OFF	High (Default)

NOTE:
1. For EVA chip, smart option is determined by DIP switch not software.
2. Please keep the reserved bits as default value (high).

Figure 22-3. DIP Switch for Smart Option

- **IDLE LED**

  This is LED is ON when the evaluation chip (S3E84B0) is in idle mode.

- **STOP LED**

  This LED is ON when the evaluation chip (S3E84B0) is in stop mode.

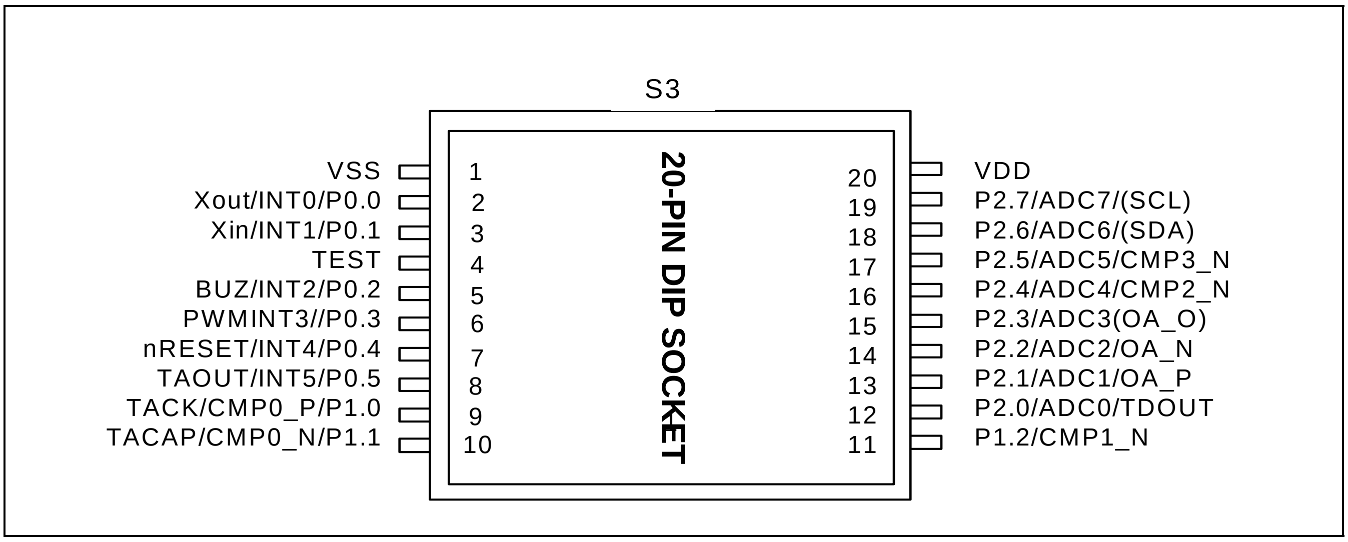


Figure 22-4. 40-Pin Connector for TB84B8

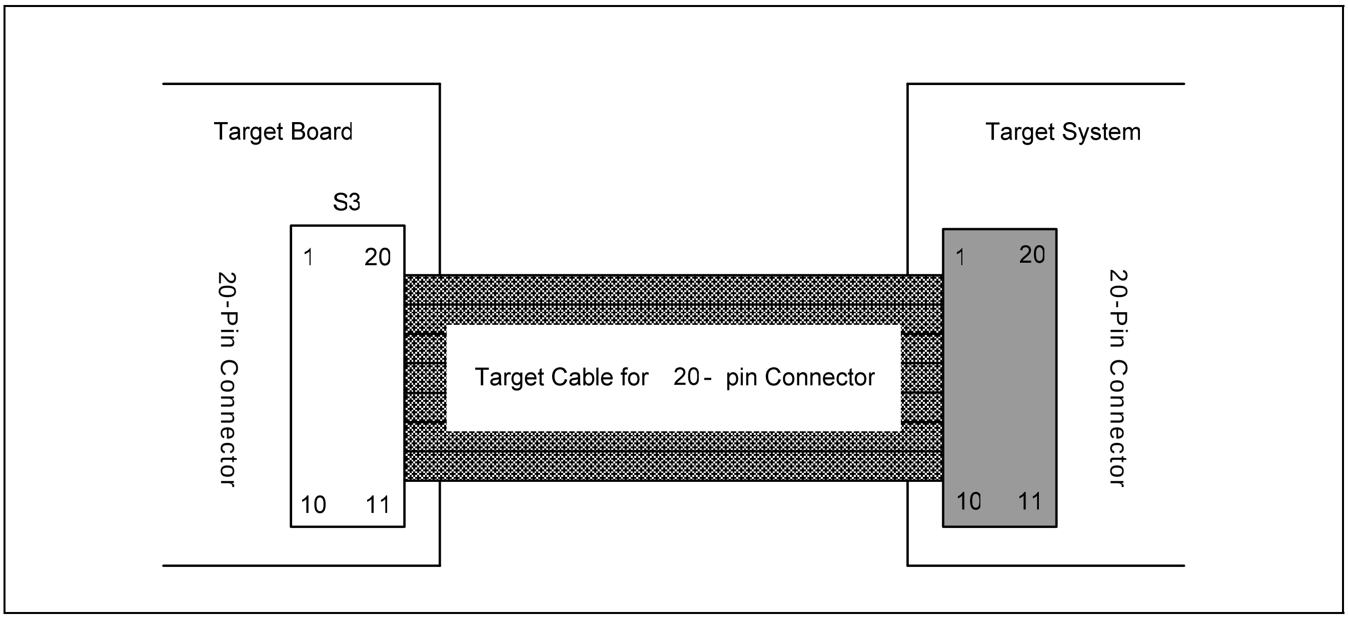


Figure 22-5. S3F84B8 Probe Adapter for 20-DIP Package

Third parties for Development Tools

SAMSUNG provides a complete line of development tools for SAMSUNG's microcontroller. With long experience in developing MCU systems, our third parties are leading companies in the tool's technology. SAMSUNG In-circuit emulator solution covers a wide range of capabilities and prices, from a low cost ICE to a complete system with an OTP/MTP programmer.

In-Circuit Emulator for SAM8 family

- OPENice-i500/2000
- SmartKit SK-1200

OTP/MTP Programmer

- SPW-uni
- AS-pro
- US-pro
- GW-PRO2 (8 - gang programmer)

Development Tools Suppliers

Please contact our local sales offices or the 3rd party tool suppliers directly as shown below for getting development tools.

8-bit In-Circuit Emulator

OPENice - i500	AIJI System • TEL: 82-31-223-6611 • FAX: 82-331-223-6613 • E-mail : openice@aijisystem.com • URL : http://www.aijisystem.com
SK-1200	Seminix • TEL: 82-2-539-7891 • FAX: 82-2-539-7819 • E-mail: sales@seminix.com • URL: http://www.seminix.com

OTP/MTP PROGRAMMER (WRITER)

	SPW-uni **Single OTP/ MTP/FLASH Programmer** • Download/Upload and data edit function • PC-based operation with USB port • Full function regarding OTP/MTP/FLASH MCU programmer (Read, Program, Verify, Blank, Protection..) • Fast programming speed (4Kbyte/sec) • Support all of SAMSUNG OTP/MTP/FLASH MCU devices • Low-cost • NOR Flash memory (SST,Samsung…) • NAND Flash memory (SLC) • New devices will be supported just by adding device files or upgrading the software.	**SEMINIX** • TEL: 82-2-539-7891 • FAX: 82-2-539-7819. • E-mail: sales@seminix.com • URL: http://www.seminix.com
	GW-uni **Gang Programmer for OTP/MTP/FLASH MCU** • 8 devices programming at one time • Fast programming speed :OTP(2Kbps) / MTP(10Kbps) • Maximum buffer memory:100Mbyte • Operation mode: PC base / Stand-alone(no PC) • Support full functions of OTP/MTP (Read, Program, Checksum, Verify, Erase, Read protection, Smart option) • Simple GUI(Graphical User Interface) • Device information setting by a device part no. • LCD display and touch key (Stand-alone mode operation) • System upgradable (Simple firmware upgrade by a user)	**SEMINIX** • TEL: 82-2-539-7891 • FAX: 82-2-539-7819. • E-mail: sales@seminix.com • URL: http://www.seminix.com

OTP/MTP PROGRAMMER (WRITER) (Continued)

	AS-pro **On-board programmer for Samsung Flash MCU** • Portable & Stand alone Samsung OTP/MTP/FLASH Programmer for After Service • Small size and Light for the portable use • Support all of SAMSUNG OTP/MTP/FLASH devices • HEX file download via USB port from PC • Very fast program and verify time (OTP:2Kbytes per second, MTP:10Kbytes per second) • Internal large buffer memory (118M Bytes) • Driver software run under various O/S (Windows 95/98/2000/XP) • Full function regarding OTP/MTP programmer (Read, Program, Verify, Blank, Protection..) • Two kind of Power Supplies (User system power or USB power adapter) • Support Firmware upgrade	**SEMINIX** • **TEL: 82-2-539-7891** • **FAX: 82-2-539-7819.** • **E-mail: sales@seminix.com** • **URL: http://www.seminix.com**
	US-pro **Portable Samsung OTP/MTP/FLASH Programmer** • Portable Samsung OTP/MTP/FLASH Programmer • Small size and Light for the portable use • Support all of SAMSUNG OTP/MTP/FLASH devices • Convenient USB connection to any IBM compatible PC or Laptop computers. • Operated by USB power of PC • PC-based menu-drive software for simple operation • Very fast program and verify time (OTP:2Kbytes per second, MTP:10Kbytes per second) • Support Samsung standard Hex or Intel Hex format • Driver software run under various O/S (Windows 95/98/2000/XP) • Full function regarding OTP/MTP programmer (Read, Program, Verify, Blank, Protection..) • Support Firmware upgrade	**SEMINIX** • TEL: 82-2-539-7891 • FAX: 82-2-539-7819. • E-mail: sales@seminix.com • URL: http://www.seminix.com

OTP/MTP PROGRAMMER (WRITER) (Continued)

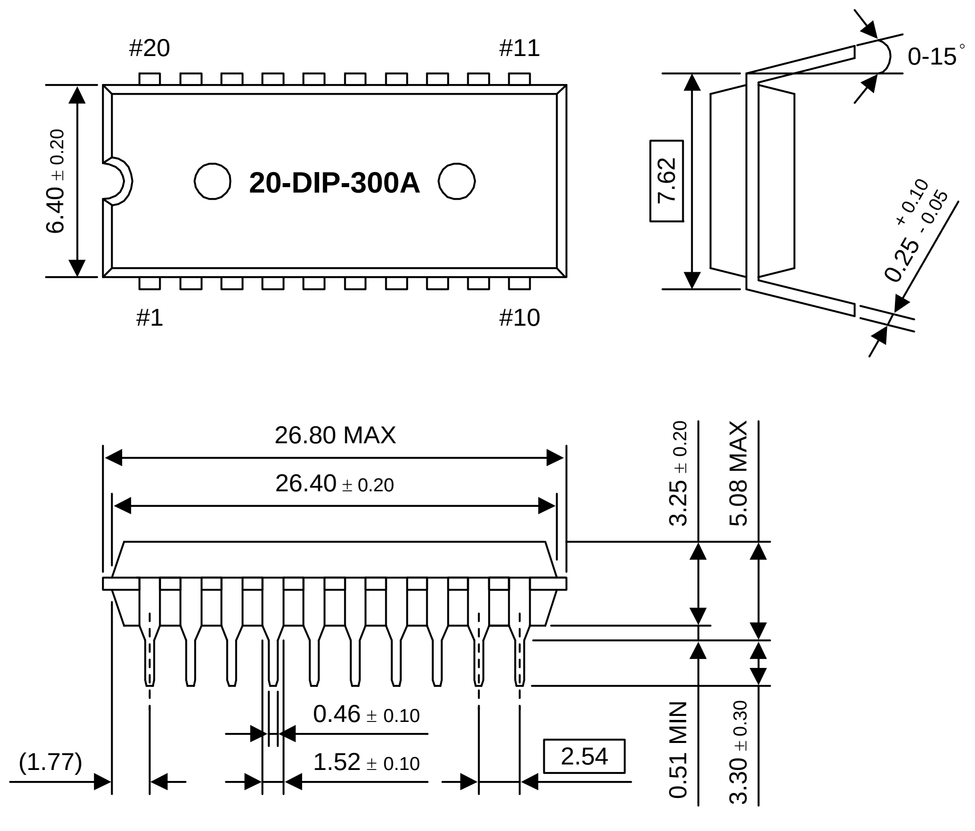
	US-pro **Portable Samsung OTP/MTP/FLASH Programmer** • Portable Samsung OTP/MTP/FLASH Programmer • Small size and Light for the portable use • Support all of SAMSUNG OTP/MTP/FLASH devices • Convenient USB connection to any IBM compatible PC or Laptop computers. • Operated by USB power of PC • PC-based menu-drive software for simple operation • Very fast program and verify time (OTP:2Kbytes per second, MTP:10Kbytes per second) • Support Samsung standard Hex or Intel Hex format • Driver software run under various O/S (Windows 95/98/2000/XP) • Full function regarding OTP/MTP programmer (Read, Program, Verify, Blank, Protection..) • Support Firmware upgrade	**SEMINIX** • TEL: 82-2-539-7891 • FAX: 82-2-539-7819. • E-mail: sales@seminix.com • URL: http://www.seminix.com
	GW-uni **Gang Programmer for OTP/MTP/FLASH MCU** • 8 devices programming at one time • Fast programming speed (1.2Kbyte/sec) • PC-based control operation mode or Stand-alone • Full Function regarding OTP/MTP program (Read, Program, Verify, Protection, Blank..) • Data back-up even at power break After setup in Design Lab, it can be moved to the factory site. • Key Lock protecting operator's mistake • Good/Fail quantity displayed and memorized • Buzzer sounds after programming • User friendly single-menu operation (PC) • Operation status displayed in LCD panel	**SEMINIX** • TEL: 82-2-539-7891 • FAX: 82-2-539-7819. • E-mail: sales@seminix.com • URL: http://www.seminix.com
	Flash writing adapter board • Special flash writing socket only for S3F84A5 - 28SOP	**C&A technology** • **TEL: 82-2-2612-9027** • **FAX: 82-2-2612-9044** • **E-mail: wisdom@cnatech.com** • **URL: http://www.cnatech.com**

NOTES

23 MECHANICAL DATA

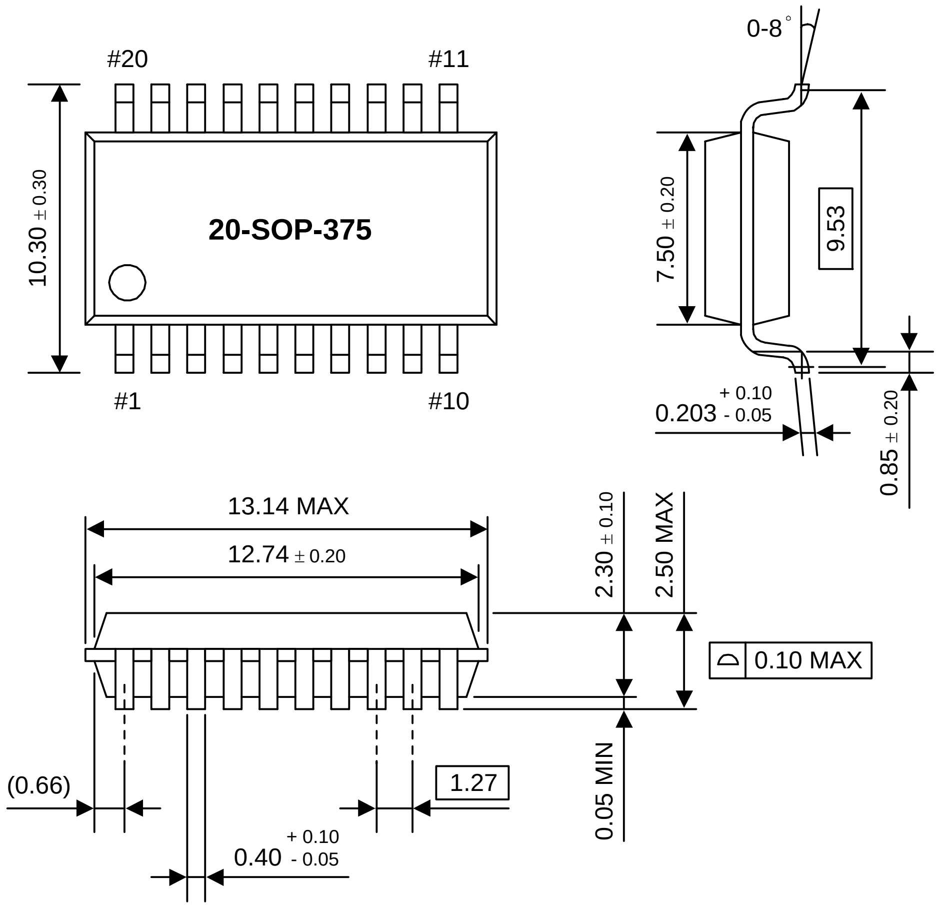
OVERVIEW

The S3F84B8 is available in a 20-pin DIP package (Samsung: 20-DIP-300A), and a 20-pin SOP package (Samsung: 20-SOP-375). Package dimensions are shown in Figure 23-1 and 23-2.



NOTE: Dimensions are in millimeters.

Figure 23-1. 20-DIP-300A Package Dimensions



NOTE: Dimensions are in millimeters.

Figure 23-2. 20-SOP-375 Package Dimensions