



S3 Family 8-Bit Microcontrollers

S3F84B8

Product Specification

PS031102-0215

PRELIMINARY





Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

©2017 Zilog, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

S3 and Z8 are trademarks or registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.

Revision History

Each instance in this document's revision history reflects a change from its previous edition. For more details, refer to the corresponding page(s) or appropriate links furnished in the table below.

Date	Revision Level	Description	Page
Feb 2015	02	Updated the Third Parties for Development Tools section.	301
Aug 2013	01	Original Zilog issue.	All

1

OVERVIEW OF S3F84B8 MICROCONTROLLER

1.S3F8-SERIES MICROCONTROLLERS

Zilog'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. Owing to its address/data bus architecture and a large number of bit-configurable I/O ports, these microcontrollers provide a flexible programming environment for applications with varied memory and I/O requirements. To support real-time operations, timer/counters with selectable operating modes are included.

1.1.1 S3F84B8 MICROCONTROLLER

The S3F84B8 single-chip CMOS microcontrollers are designed using a highly advanced CMOS process technology based on Zilog's latest CPU architecture.

S3F84B8 specifies a microcontroller with built-in 8K-byte full-flash ROM.

Using a proven modular design approach, Zilog S3F84B8 integrates the following peripheral modules with a powerful SAM8 RC core:

- 3 configurable I/O ports (18 pins)
- 17 interrupt sources with 17 vectors and 6 interrupt levels
- 1 watchdog timer function (Basic Timer)
- 1 basic timer (8-bit) for oscillation stabilization
- 3 timer/counters (8-bit) with Time Interval, PWM, and Capture modes (Timer C and Timer D can be used for 16-bit Timer 0)
- 1 timer/counter (16-bit) with 2 operating modes: Interval timer mode and PWM mode (If Timer C and Timer D are used for Timer 0, S3F84B8 has 1 Timer0 (16-bit))
- Analog to digital converter with 8 input channels and 10-bit resolution
- 1 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, and so on. They are currently available in 20-pin SOP/DIP package.

1.1.2 KEY FEATURES OF S3F84B8

The key features of S3F84B8 include:

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

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

Interrupts

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

General I/O

- 3 I/O ports
- Bit programmable ports

10-bit IH PWM

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

Comparators

- 4 integrated comparators

A/D Converter

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

OP Amplifier

- 1 integrated OP Amplifier

Timer/Counters

- 1 basic timer (8-bit) for watchdog function
- 1 timer (8-bit) TimerA
 - Interval mode
 - Capture mode
 - 8-bit PWM mode
- 1 timer/counter (16-bit) Timer0
 - Configurable to 2 timer/counters (8-bit)
 - Interval mode
 - CMP0 event counter mode
 - 6-/7-/8-bit PWM mode

BUZ

- 1 programmable Buzzer

Oscillation Frequency

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

Built-in RESET Circuit (LVR)

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

Operating Temperature Range

- $-40^{\circ}C$ to $+85^{\circ}C$

Operating Voltage Range

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

Package Types

- S3F84B8: 20-SOP/20-DIP

1.1.3 BLOCK DIAGRAM OF S3F84B8

Figure 1-1 shows the block diagram of S3F84B8.

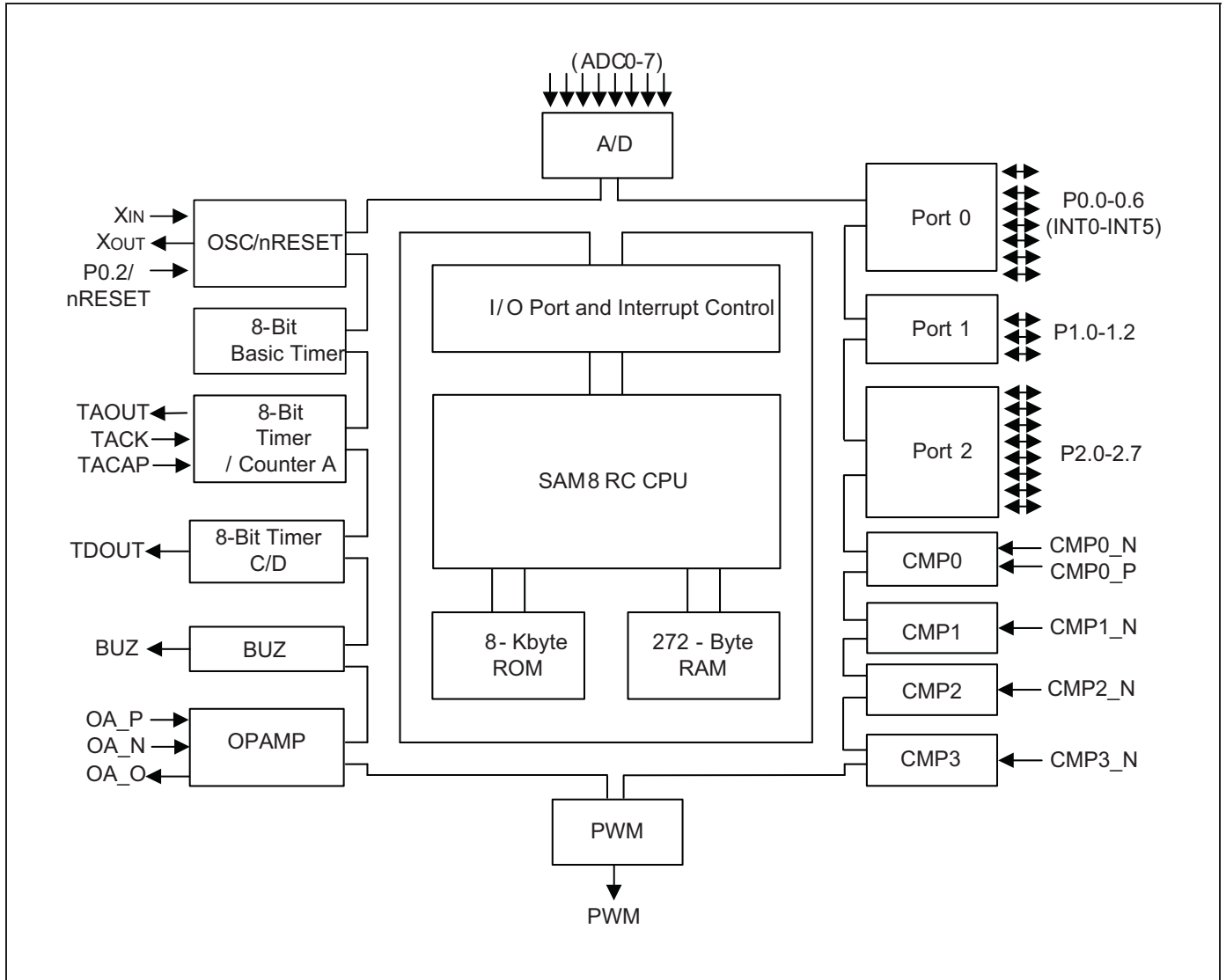


Figure 1-1 S3F84B8 Block Diagram

1.1.4 PIN ASSIGNMENTS

Figure 1-2 shows the pin assignments (20-DIP, 20-SOP) in S3F84B8.

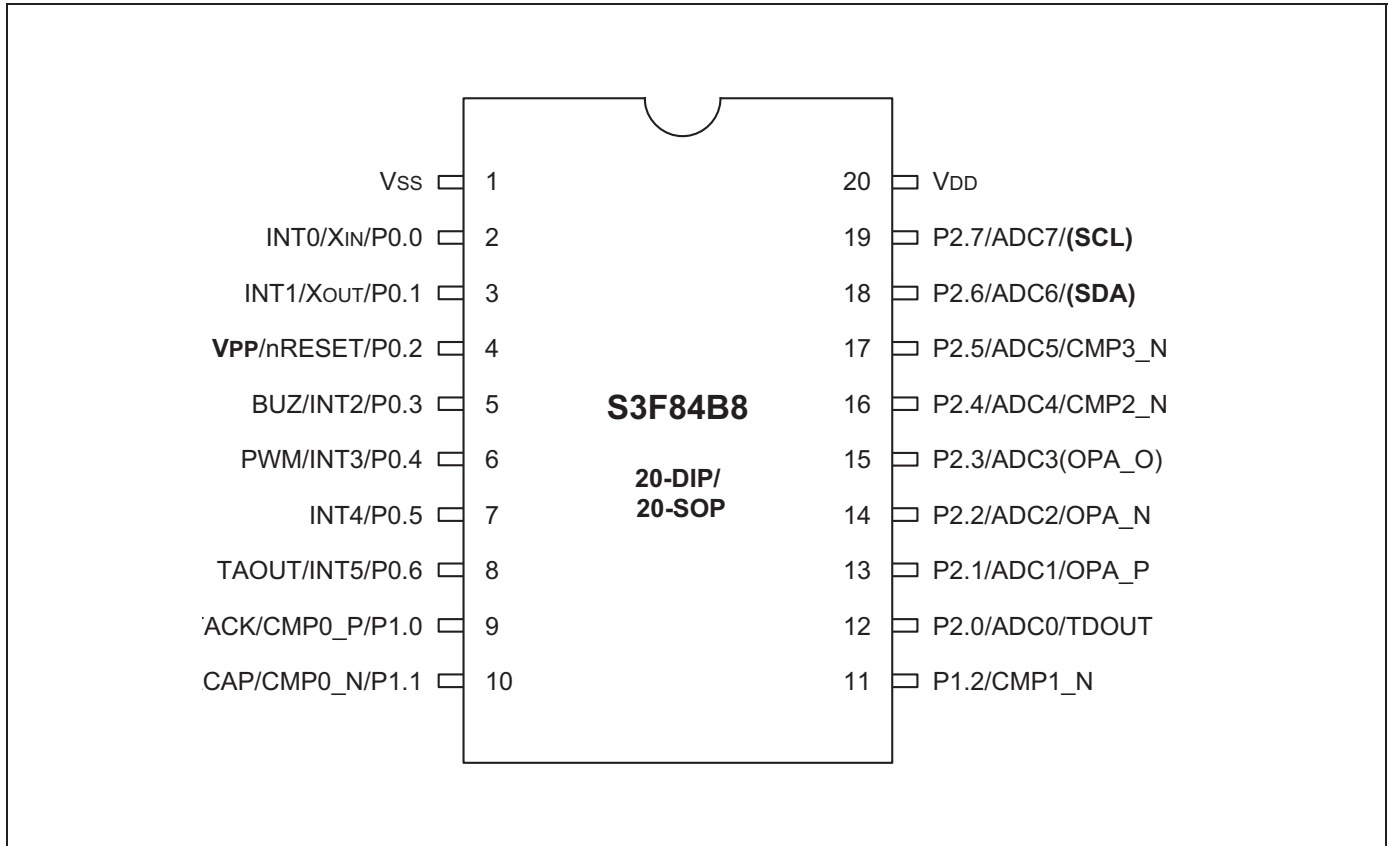


Figure 1-2 S3F84B8 Pin Assignment (20-DIP, 20-SOP)

1.1.5 PIN DESCRIPTIONS

[Table 1-1](#) shows the pin descriptions (20-DIP/20-SOP) in S3F84B8.

Table 1-1 Pin Descriptions (20-DIP/20-SOP) in S3F84B8

Pin Name	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](#) shows the pin descriptions used to Read/Write the Flash ROM in S3F84B8.

Table 1-2 Pin Descriptions 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	VPP	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 the writing mode and when 5 V is applied, MTP is in the reading mode. (Optional)
V _{DD} /V _{SS}	V _{DD} /V _{SS}	20/1	I	Logic power supply pin.

1.1.6 PIN CIRCUITS

Figure 1-3 shows the pin circuit type 1 in S3F84B8.

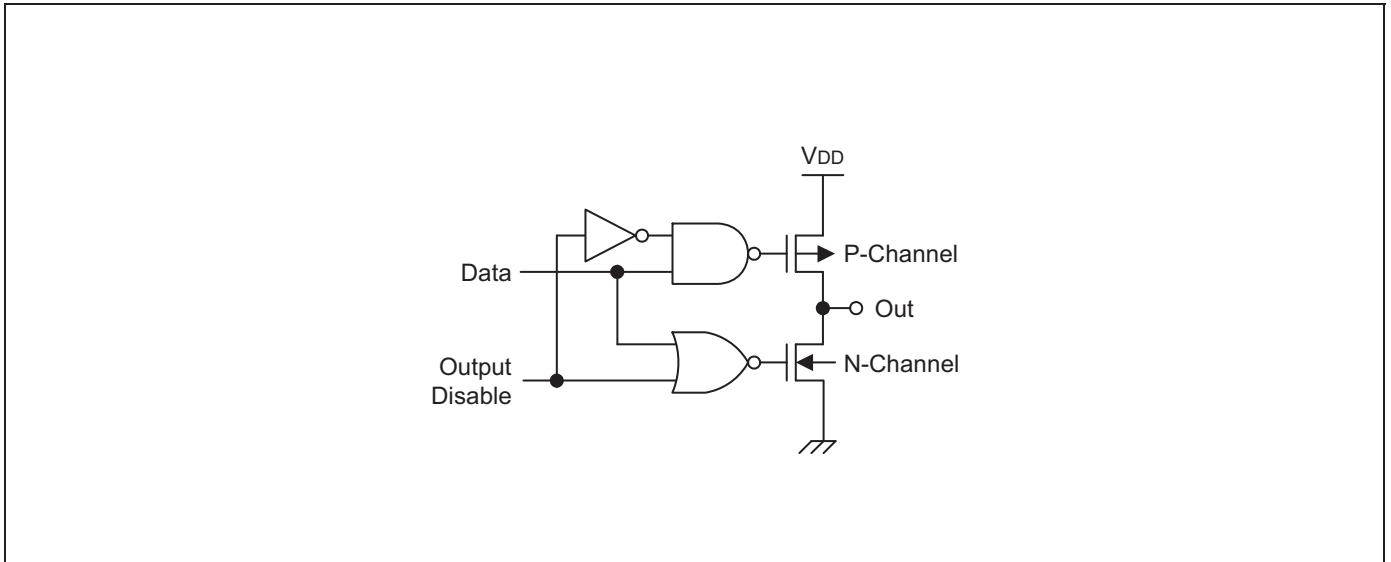


Figure 1-3 Pin Circuit Type 1

Figure 1-4 shows the pin circuit type 2 in S3F84B8.

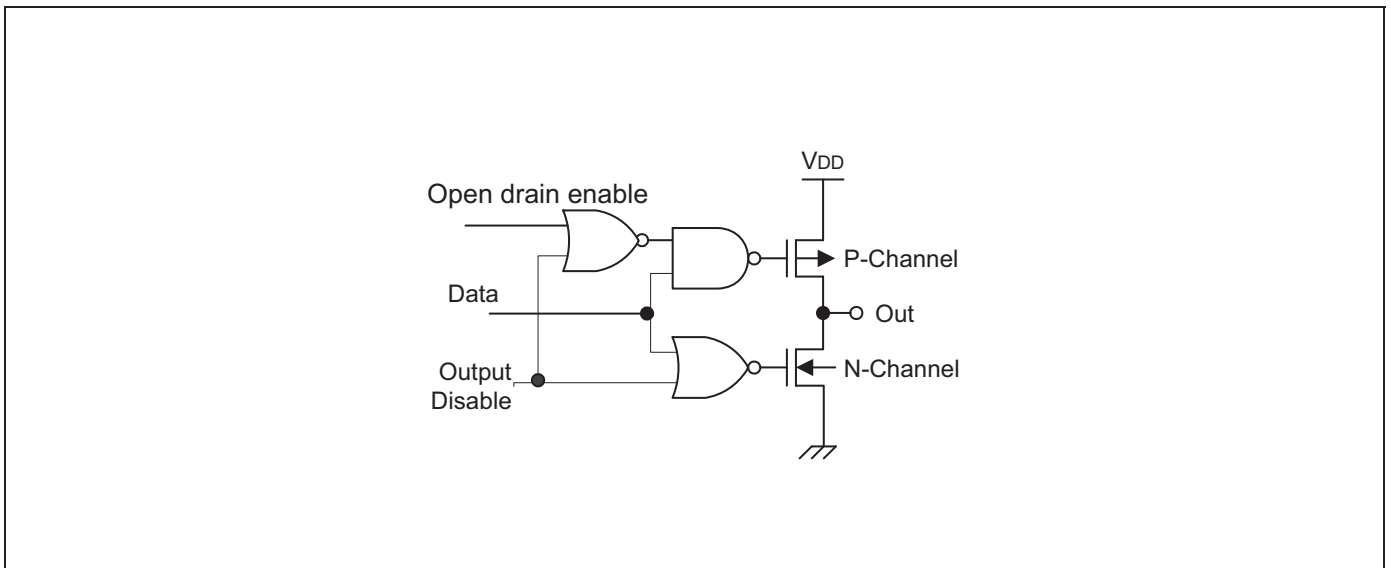


Figure 1-4 Pin Circuit Type 2

Figure 1-5 shows the pin circuit type 1-1 (P1.0-1.2, P2.0-2.2, P2.4-2.7) in S3F84B8.

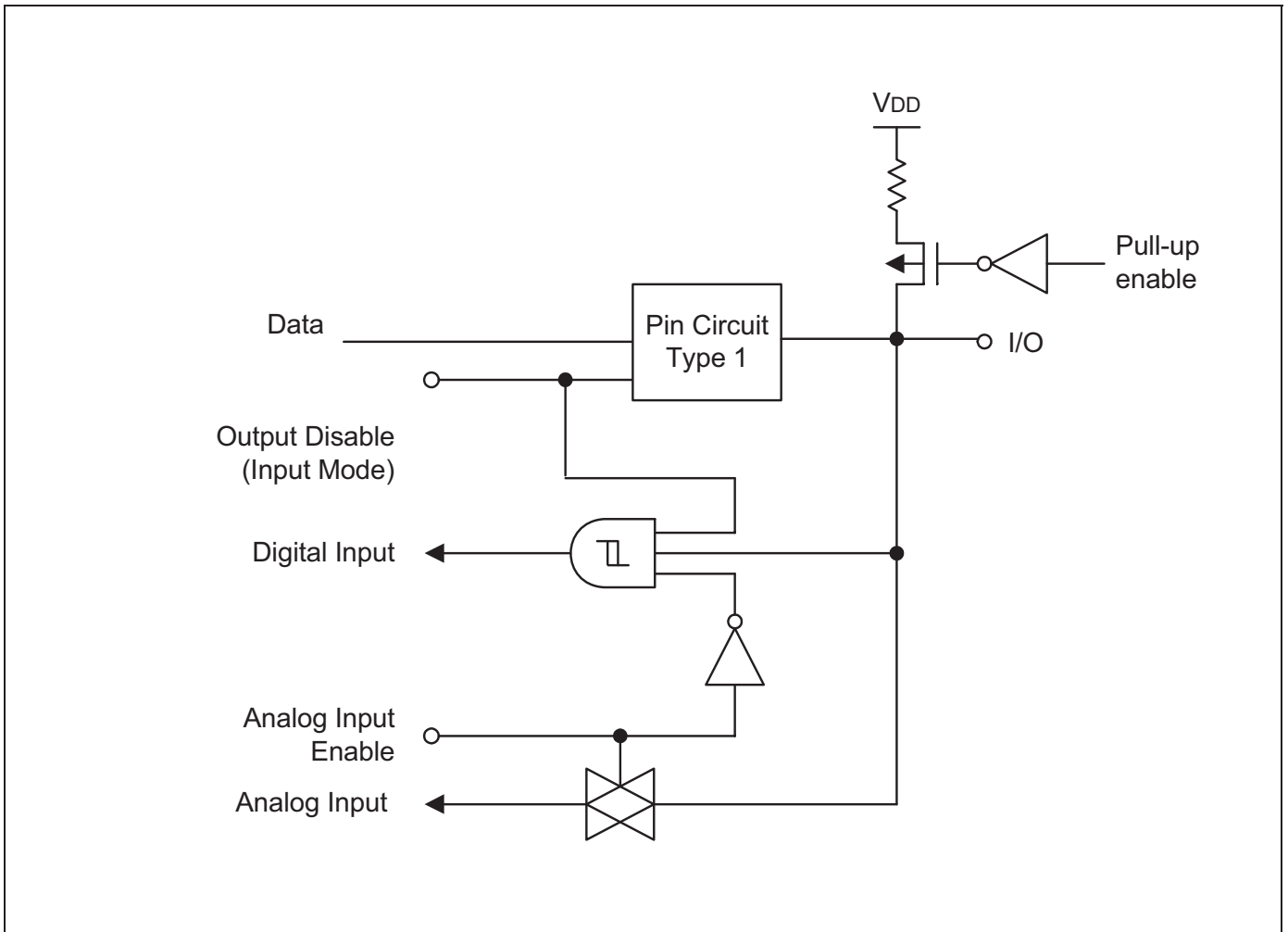


Figure 1-5 Pin Circuit Type 1-1 (P1.0-1.2, P2.0-2.2, P2.4-2.7)

Figure 1-6 shows the Pin Circuit Type 1-2 (P2.3) in S3F84B8.

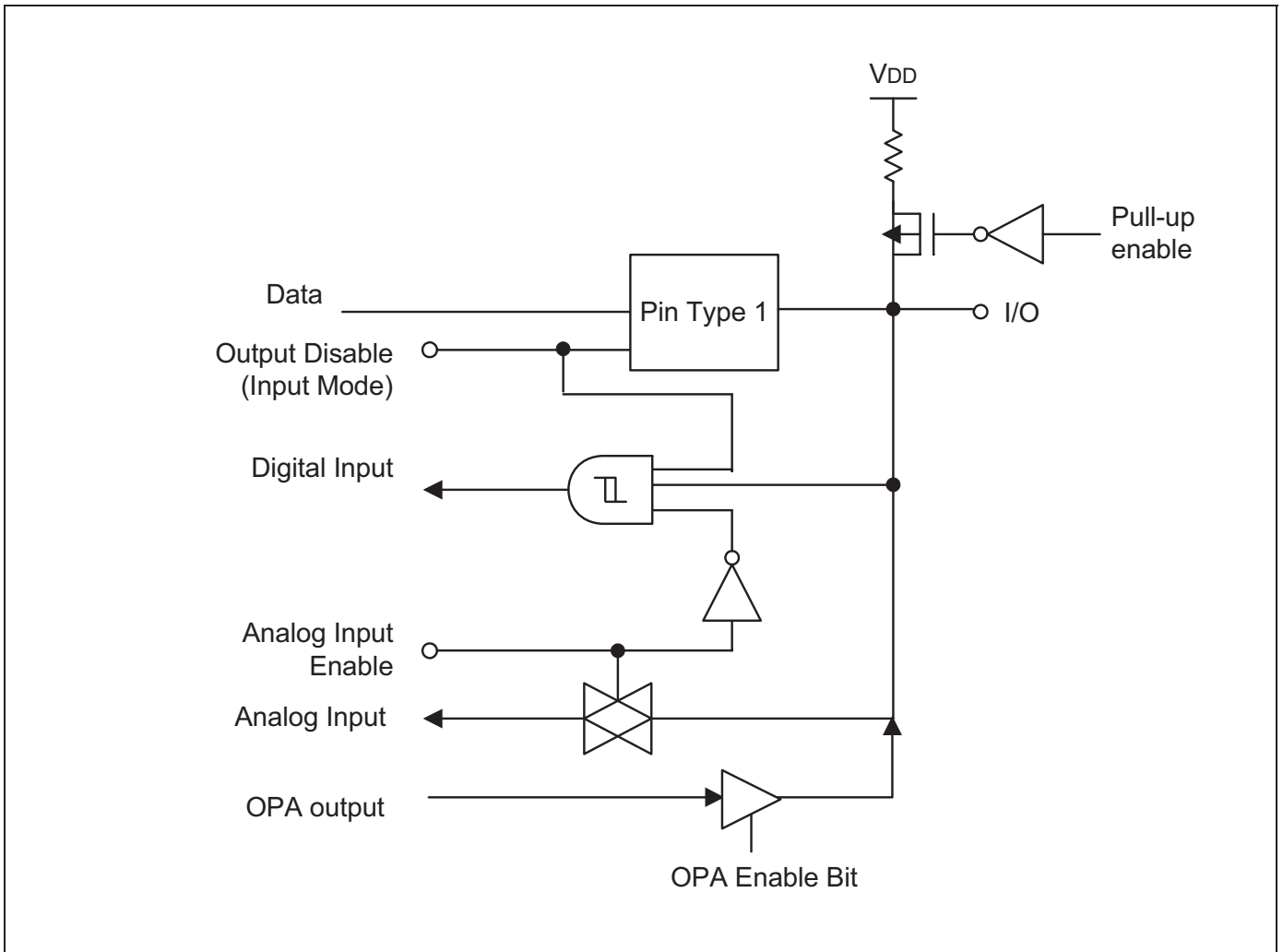


Figure 1-6 Pin Circuit Type 1-2 (P2.3)

Figure 1-7 shows the Pin Circuit Type 1-3 (P0.3, P0.4, P0.6) in S3F84B8.

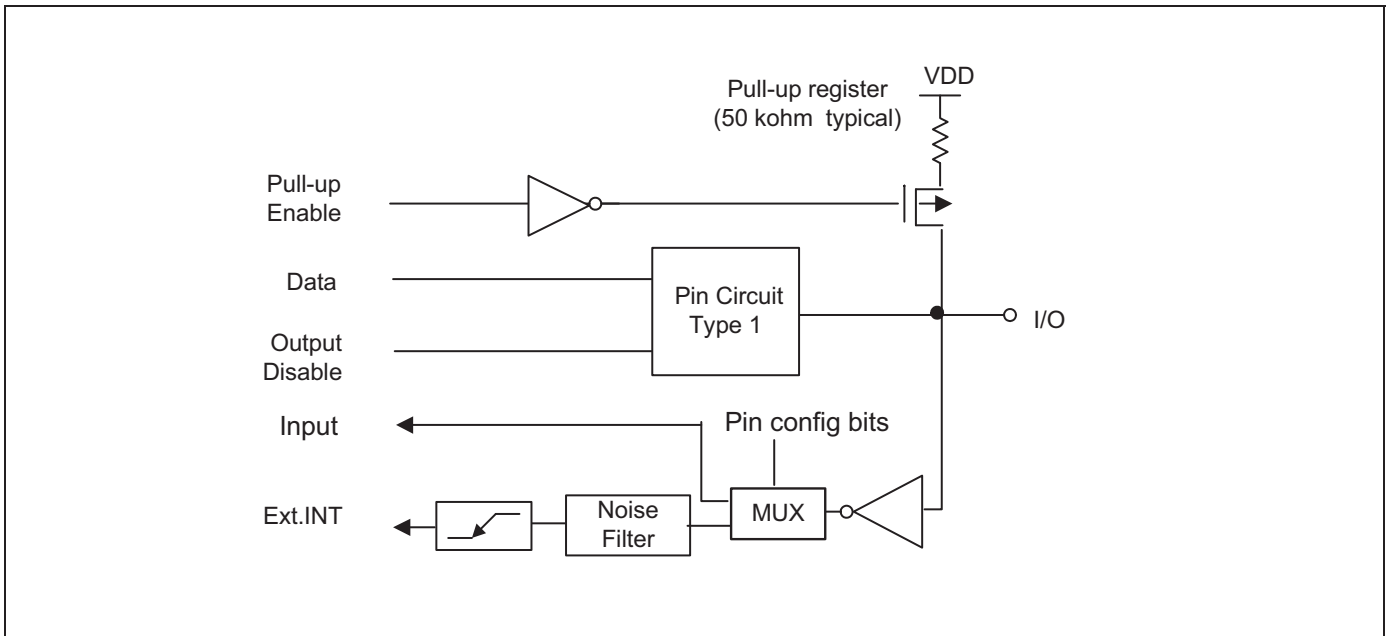


Figure 1-7 Pin Circuit Type 1-3 (P0.3, P0.4, P0.6)

Figure 1-8 shows the Pin Circuit Type 2-1 (P0.5) in S3F84B8.

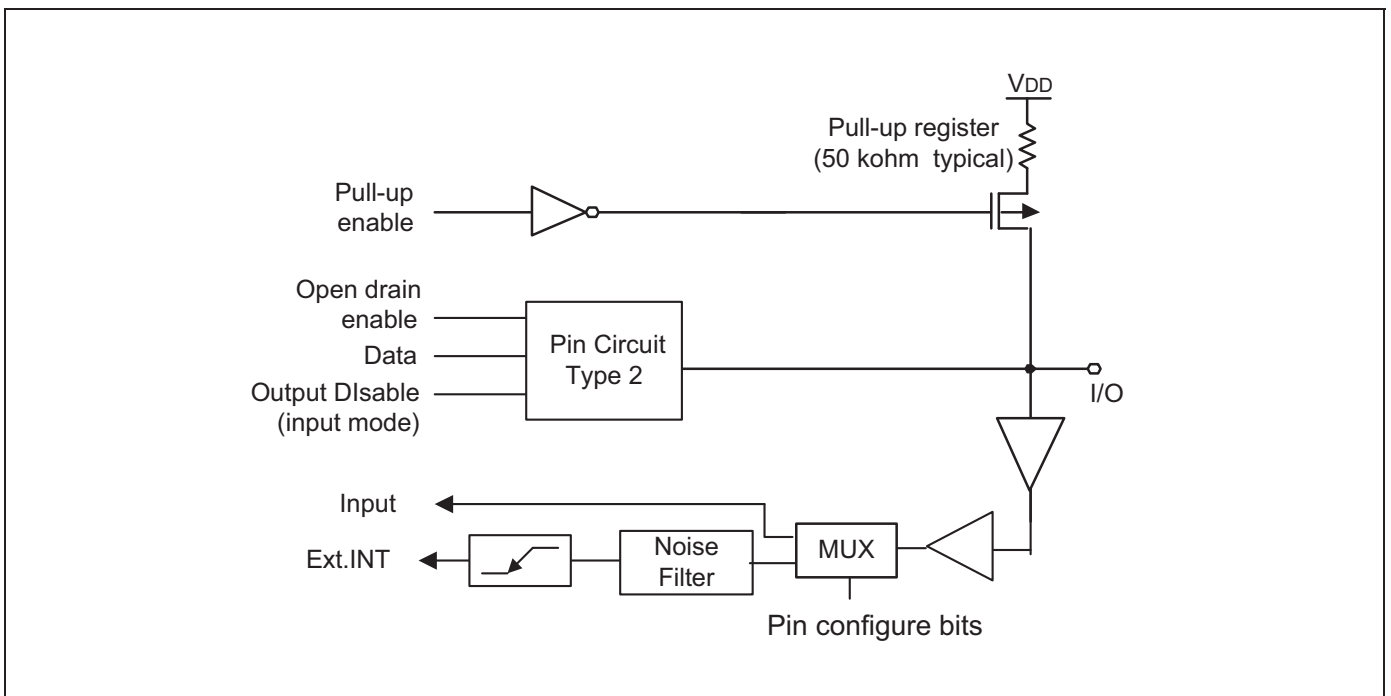


Figure 1-8 Pin Circuit Type 2-1 (P0.5)

Figure 1-9 shows the Pin Circuit Type 3 (P0.2) in S3F84B8.

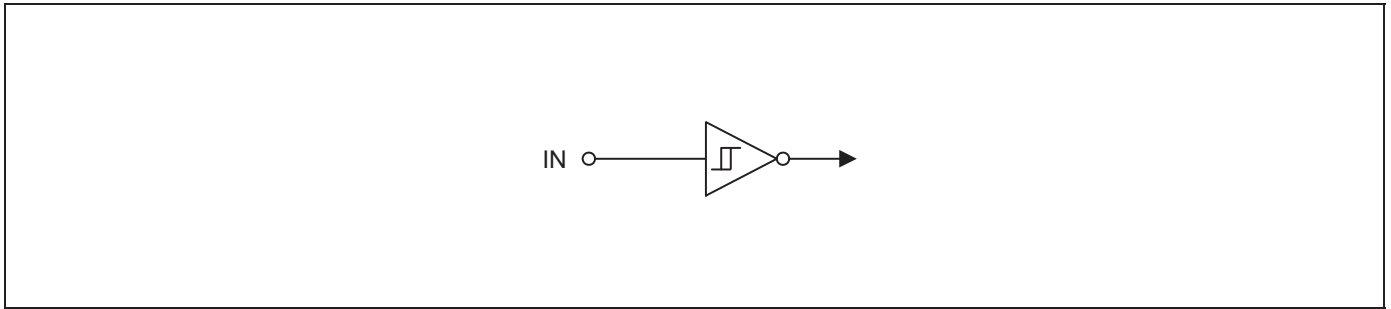


Figure 1-9 Pin Circuit Type 3 (P0.2)

Figure 1-10 shows the Pin Circuit Type 2-2 (P0.0, P0.1) in S3F84B8.

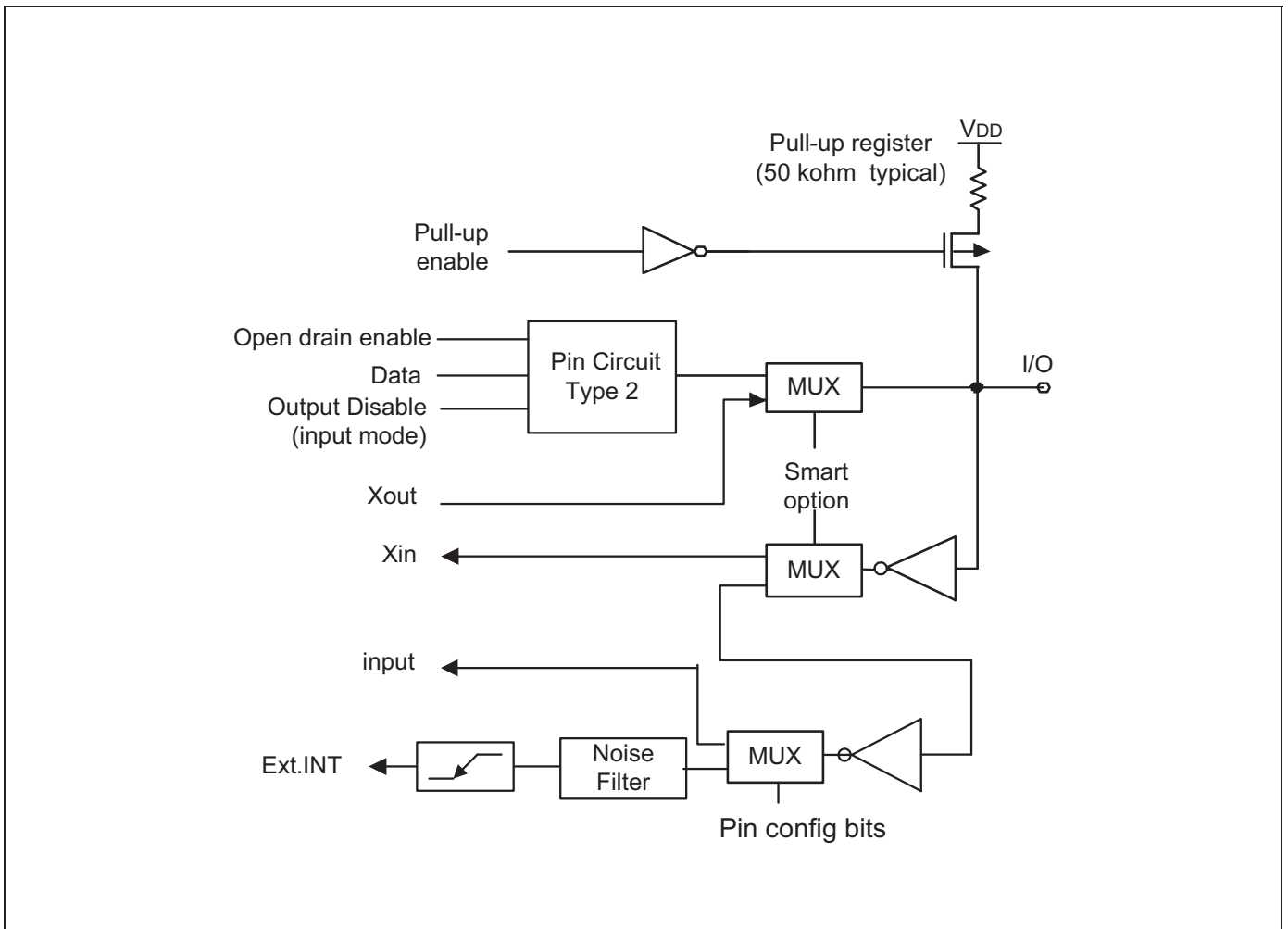


Figure 1-10 Pin Circuit Type 2-2 (P0.0, P0.1)

2 ADDRESS SPACES

2.1 OVERVIEW OF ADDRESS SPACES

The S3F84B8 microcontroller consists of two kinds of address spaces, namely:

- Internal program memory (ROM)
- Internal register file

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

The S3F84B8 microcontroller contains 8Kbytes of on-chip program memory configured as Internal ROM. It also contains 272 general-purpose registers in the internal register file, where 59 bytes are mapped for system and peripheral control functions.

2.2 INTERNAL PROGRAM MEMORY (ROM)

The internal program memory (ROM) stores program codes or table data. The S3F84B8 microcontroller contains 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, do not overwrite the vector addresses stored in these locations.

003CH, 003DH, 003EH, and 003FH are used as smart option ROM cells.

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

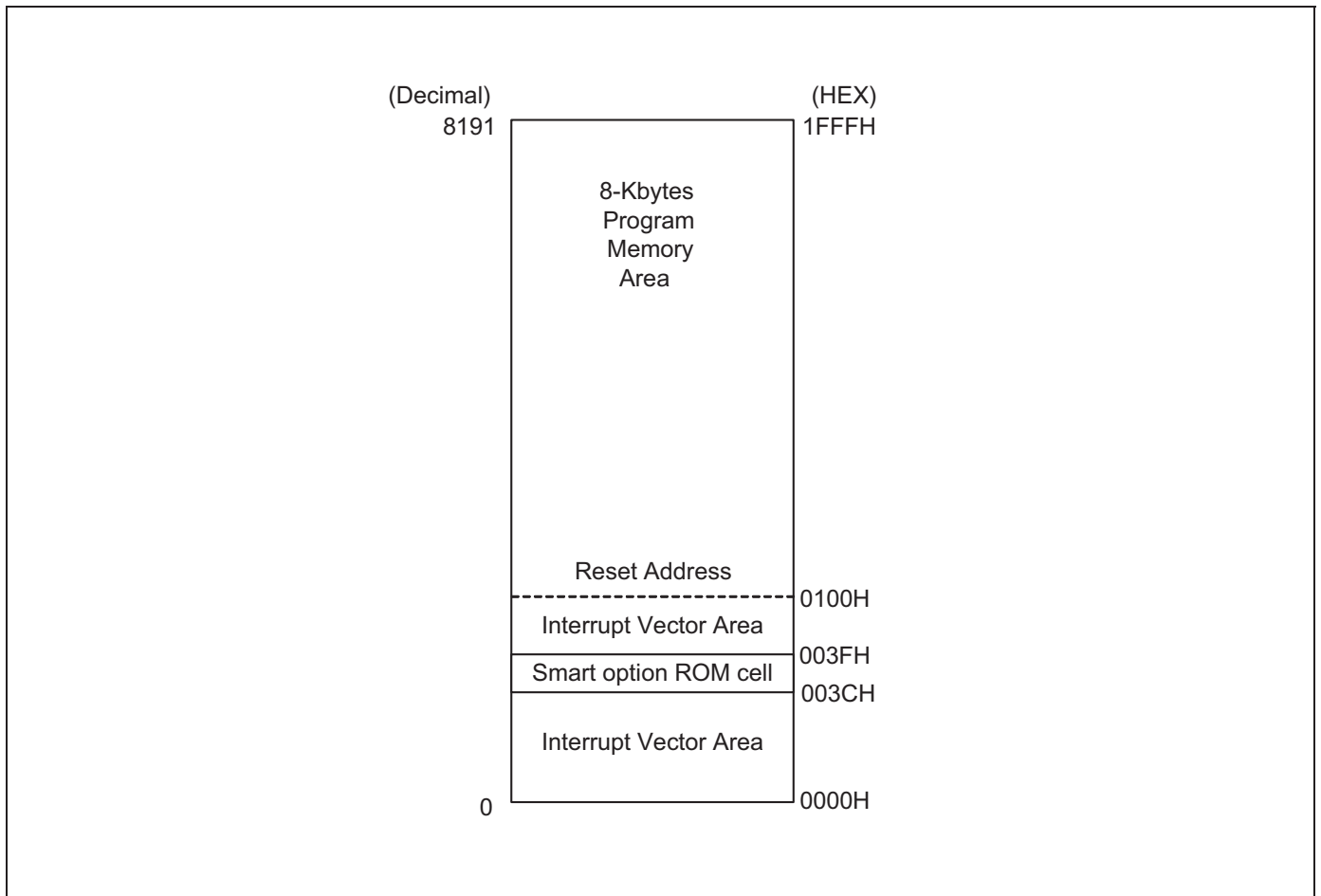


Figure 2-1 Program Memory Address Space

2.2.1 SMART OPTION

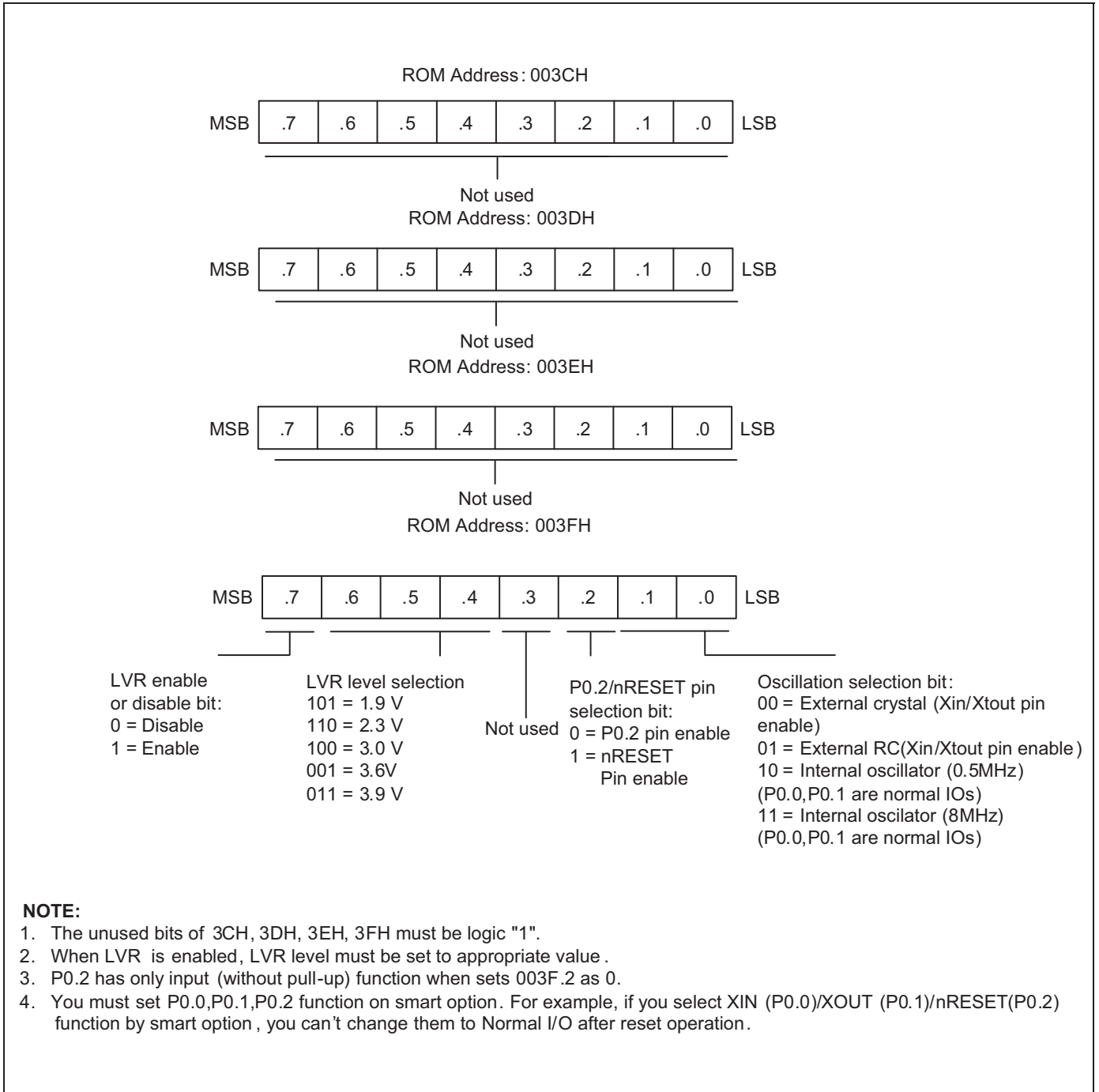


Figure 2-2 Smart Option

For Start condition of the chip, Smart option specifies the ROM option. The ROM address used by Smart option is from 003EH to 003FH. Note that 003CH and 003DH are not used in S3F84B8.

2.3 REGISTER ARCHITECTURE

In the S3F84B8 microcontroller implementation, the upper 64 byte area of register files is expanded into two 64 byte areas called set 1 and set 2. The upper 32 byte area of set 1 is further expanded into two 32 byte register banks called bank 0 and bank 1, while the lower 32 byte area specifies 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 meant for the CPU and system control registers, 49 bytes are meant for peripheral control and data registers, 16 bytes are meant for shared working registers. 272 registers are meant for general-purpose use, page 0 (For more information about page 0, refer to [Figure 2-3](#)).

Registers in Set 1 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, select bank instructions (SB0 and SB1), and register page pointer (PP).

[Table 2-1](#) shows the specific register types and area (in bytes) they occupy in the register file.

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

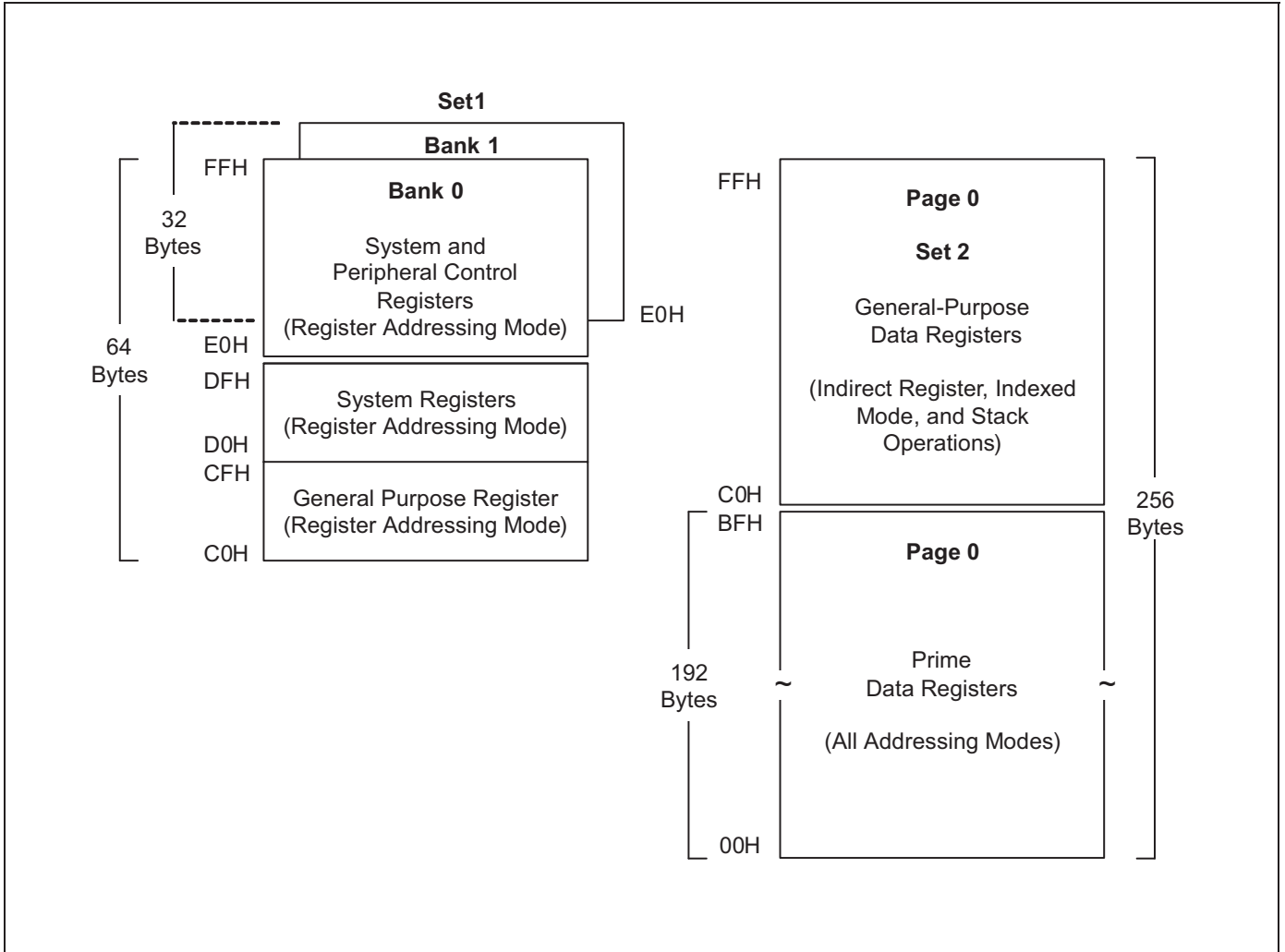


Figure 2-3 Internal Register File Organization in S3F84B8

2.3.1 REGISTER PAGE POINTER (PP)

The S3F8 series architecture supports the logical expansion of 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 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.

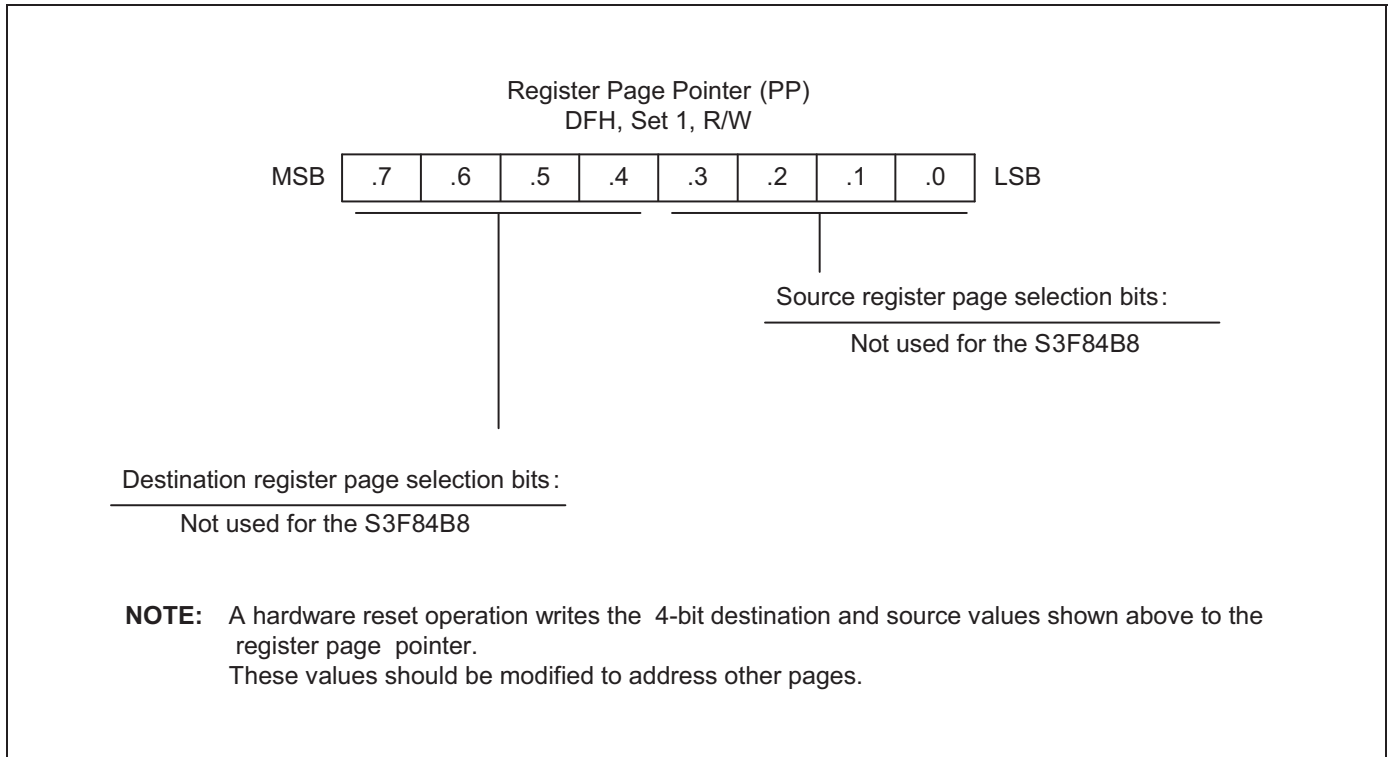


Figure 2-4 Register Page Pointer (PP)

2.3.1.1 Register Set 1

The term set 1 refers to the upper 64 bytes of register file in locations C0H–FFH.

The upper 32 byte area of this 64 byte space (E0H–FFH) is expanded into two 32 byte register banks, bank 0 and bank 1. The set register bank instructions, SB0 or SB1, are used to address one of the banks. 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.

Using the Register Addressing mode, the registers in set 1 location can be directly accessed at any time. The 16 byte working register area can only be accessed using working register addressing (For more information about working register addressing, refer to Chapter 3, “Addressing Modes”).

2.3.1.2 Register Set 2

The same 64 byte physical space that is used for set 1 location 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 only implemented 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.

2.3.1.3 Prime Register Space

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

The prime register area on page 0 the prime register area on page 0 can be addressed immediately after a reset. But to address prime registers on page 0 or 1, you must set the register page pointer (PP) to its appropriate source and destination values.

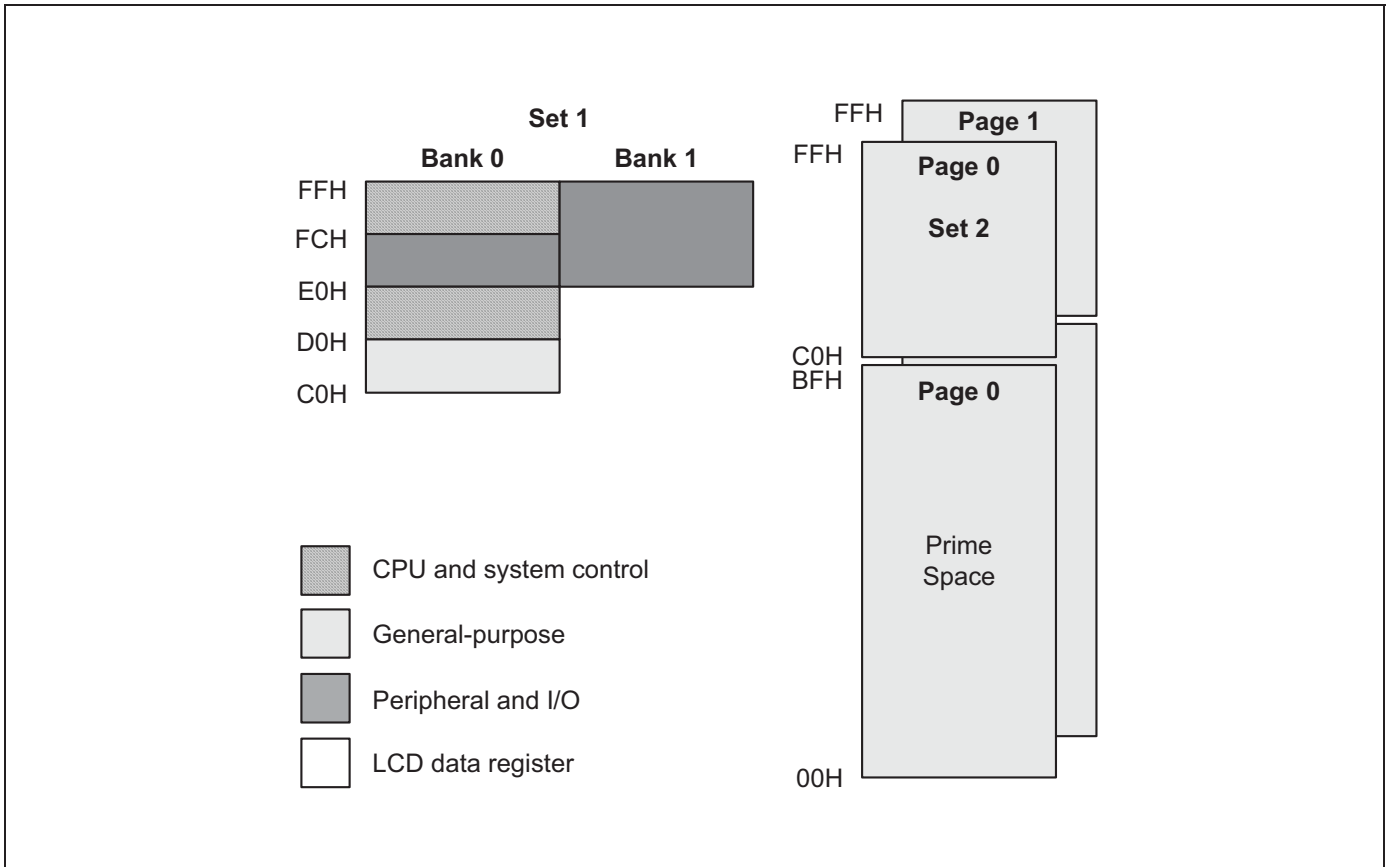


Figure 2-5 Set 1, Set 2, Prime Area Register Map

2.3.1.4 Working Registers

Instructions can access specific 8-bit registers or 16-bit register pairs using 4-bit or 8-bit address fields. When 4-bit working register addressing is used, consider the 256 byte register file as the 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), the two working register slices can be selected at 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, both RP0 and RP1 always point to the 16 byte common area in set 1 (C0H–CFH).

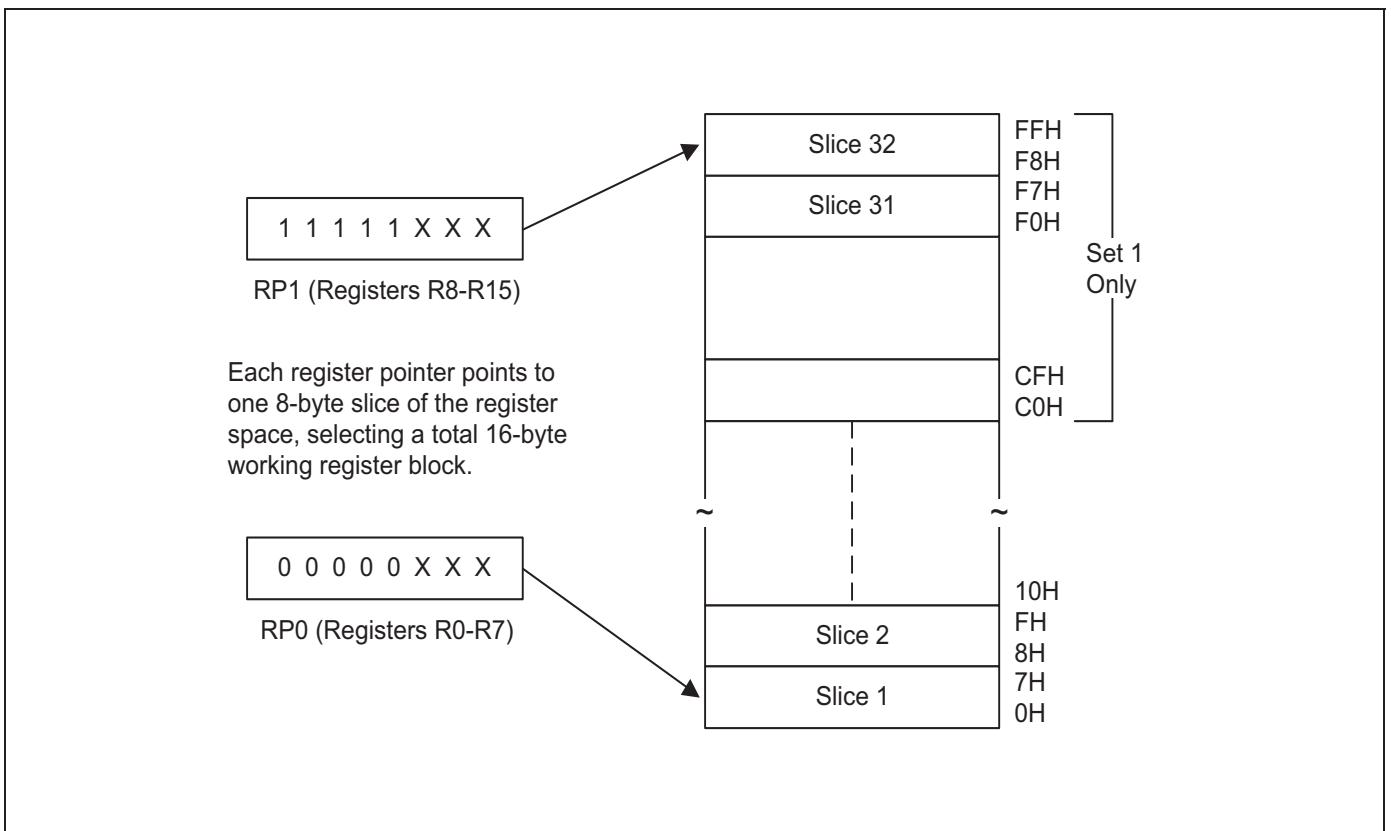


Figure 2-6 8 Byte Working Register Areas (Slices)

2.3.2 USING THE REGISTER POINTS (RP)

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 [Figure 2-7](#) and [Figure 2-8](#)).

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

Usually, the selected 16 byte working register block 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 2-7](#)). In some cases, it may be necessary to define working register areas in different (non-contiguous) areas of the register file. In [Figure 2-7](#), RP0 and RP1 point to the “upper” slice and “lower” slice respectively.

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

Example 2-1 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

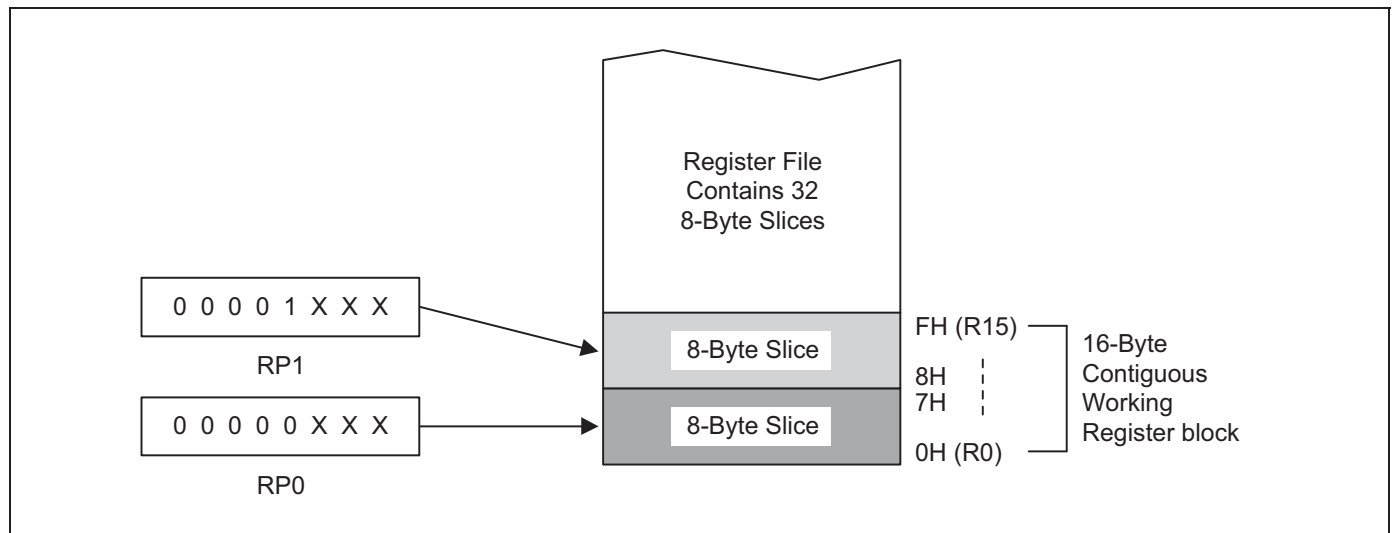


Figure 2-7 Contiguous 16 Byte Working Register Block

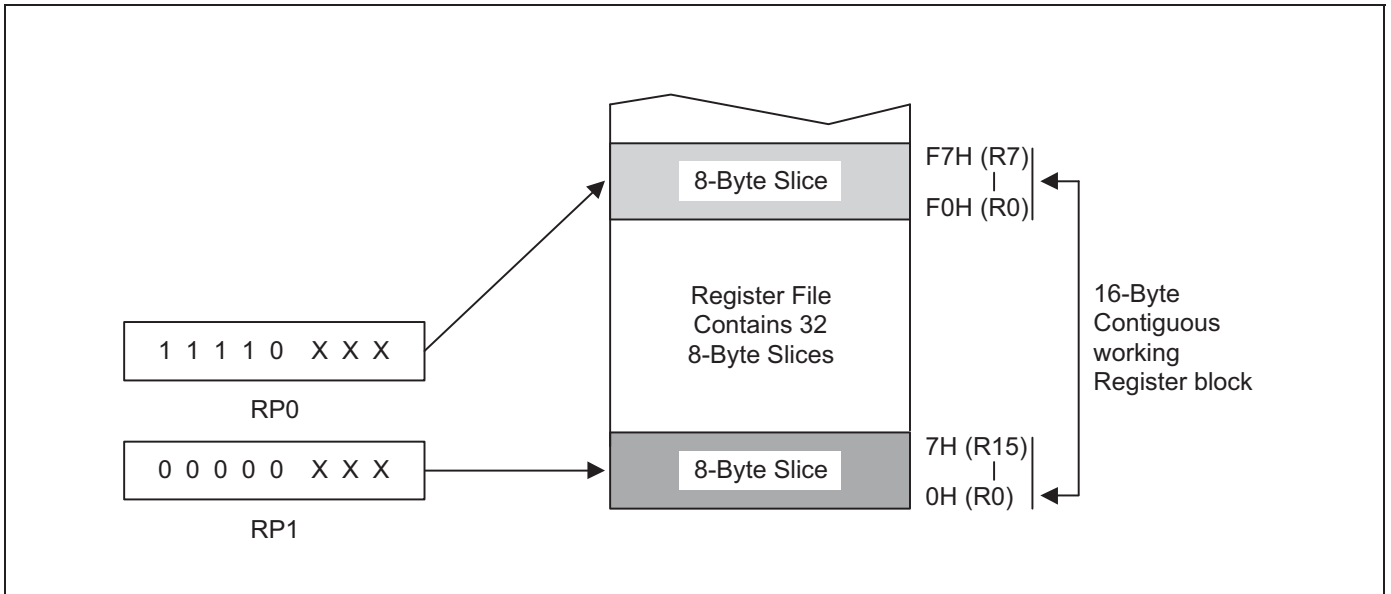


Figure 2-8 Non-Contiguous 16 Byte Working Register Block

Example 2-2 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 should 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 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.

2.4 REGISTER ADDRESSING

The S3F8 series register architecture provides an efficient method of working register addressing and takes full advantage of shorter instruction formats to reduce the execution time.

In Register (R) addressing mode, the operand value specifies the content of a specific register or register pair. Here, you can access any location in the register file, except for set 2. With working register addressing, you can 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 a paired 16-bit register space. In a 16-bit register pair, the address of first 8-bit register is always an even number and the address of next register is always an odd number. The most significant byte (MSB) of 16-bit data is always stored in the even-numbered register, and the least significant byte (LSB) is always stored in the next (+1) odd-numbered register.

Working register addressing differs from Register addressing, since 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.

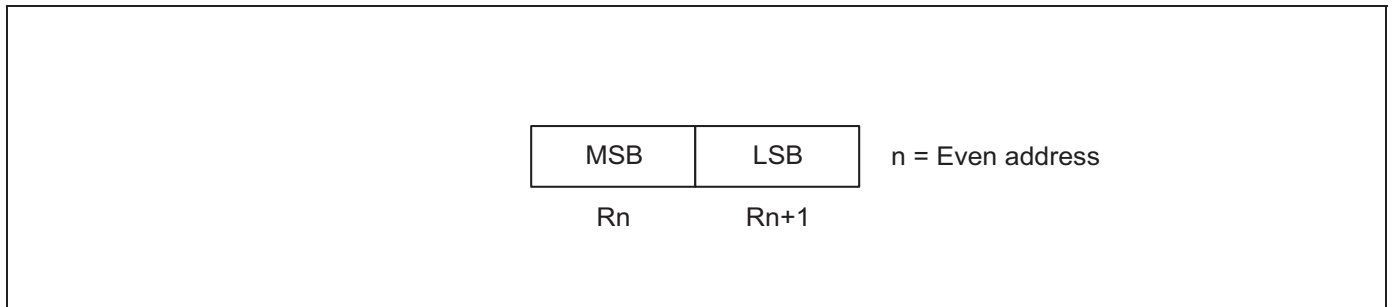


Figure 2-9 16-Bit Register Pair

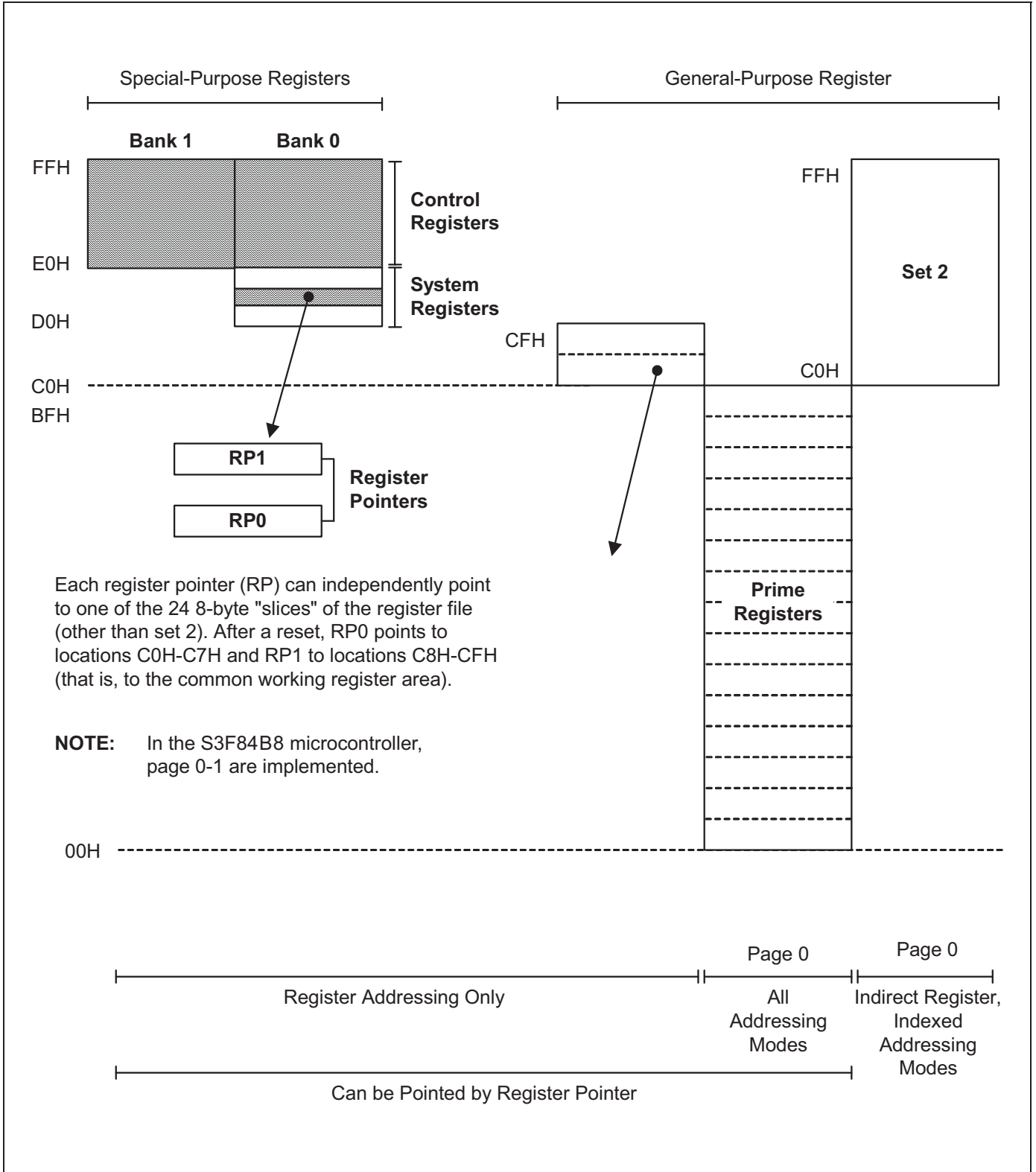


Figure 2-10 Register File Addressing

2.4.1 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 C0H–C7H
 RP1 C8H–CFH

This 16 byte address range is called common area. The locations in this area can be used as working registers for 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.

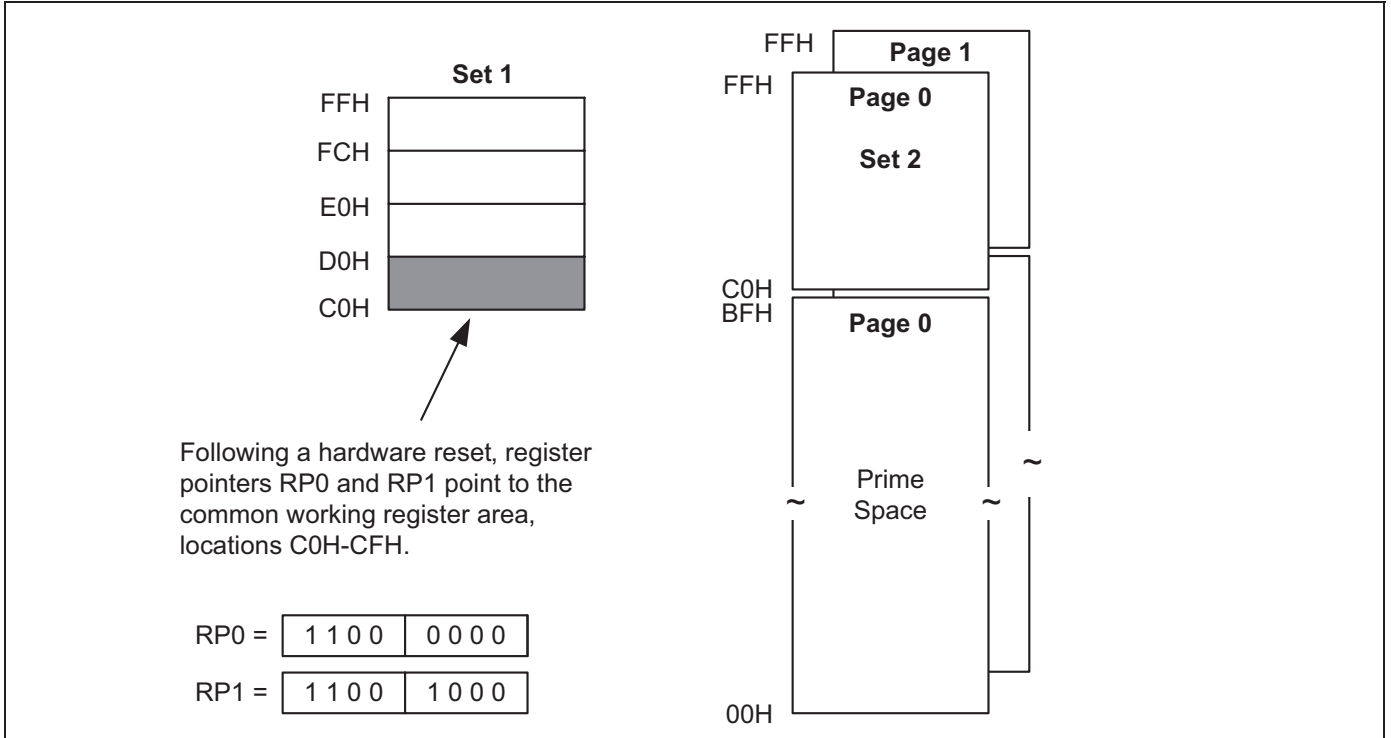


Figure 2-11 Common Working Register Area

Example 2-3 Addressing the Common Working Register Area

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

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

2.4.2 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 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 and “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 register pointer are concatenated with the three low-order bits from instruction address to form the complete address. If the address stored in register pointer remains unchanged, the three bits from the address will always point to an address in the same 8 byte register slice.

[Figure 2-13](#) shows a typical example of 4-bit working register addressing. The high-order bit of instruction “INC R6” is “0”, which selects the RP0. The five high-order bits stored in RP0 (01110B) are concatenated with three low-order bits of instruction’s 4-bit address (110B) to produce the register address 76H (01110110B).

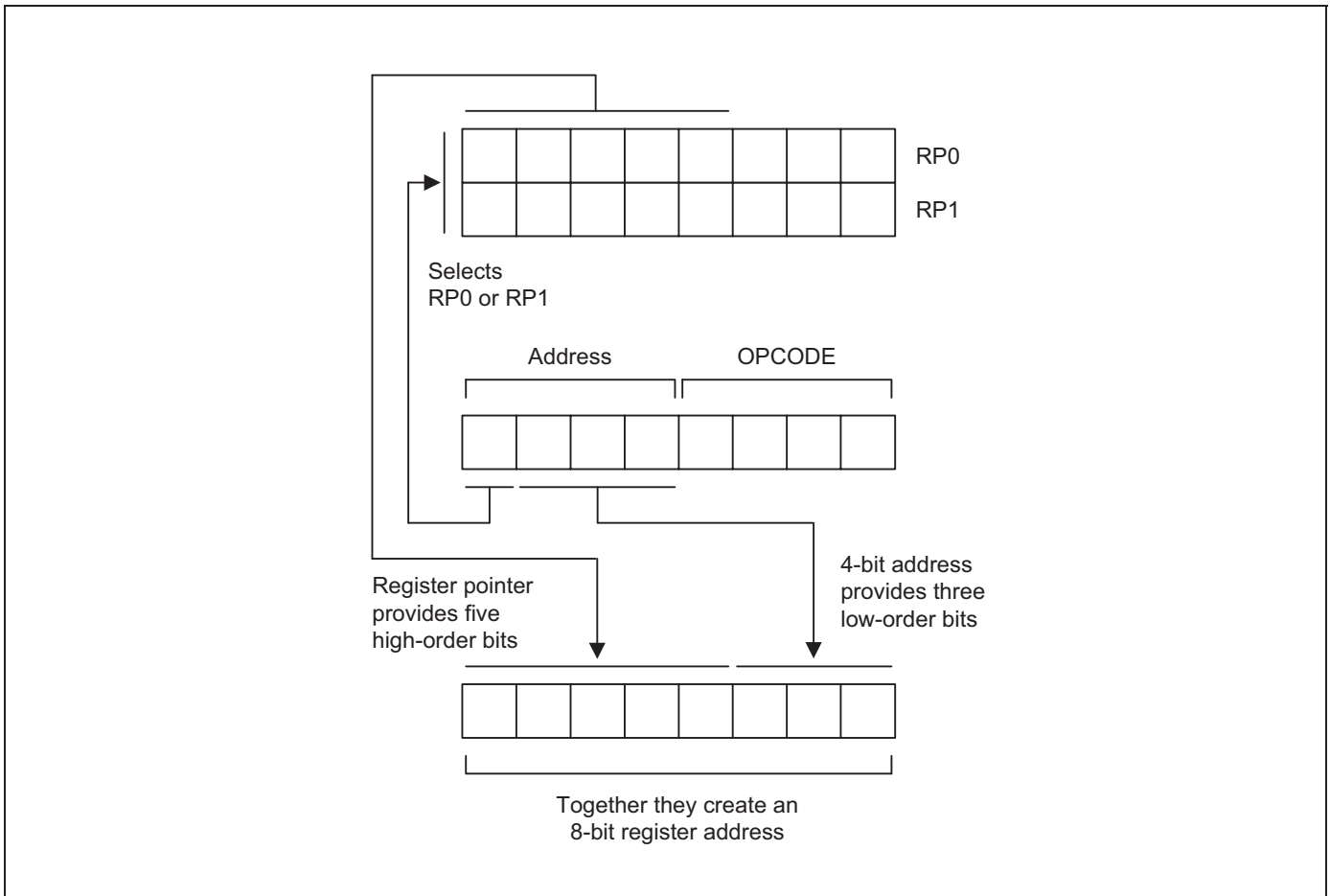


Figure 2-12 4-Bit Working Register Addressing

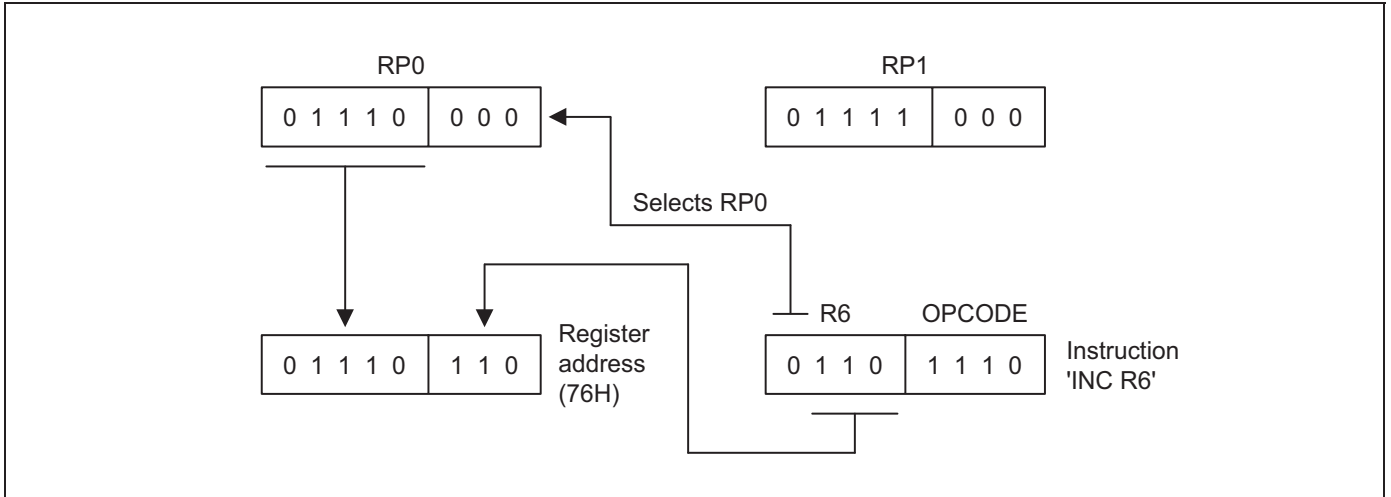


Figure 2-13 4-Bit Working Register Addressing Example

2.4.3 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 8-bit address is concatenated in the same way as 4-bit addressing: Bit 3 selects either RP0 or RP1, which then supplies the five high-order bits of final address; the three low-order bits of 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 instruction address (1100B) specify the 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 register address. The three low-order bits of register address (011) are provided by the three low-order bits of 8-bit instruction address. The five address bits from RP1 and three address bits from instruction are concatenated to form the complete register address, 0ABH (10101011B).

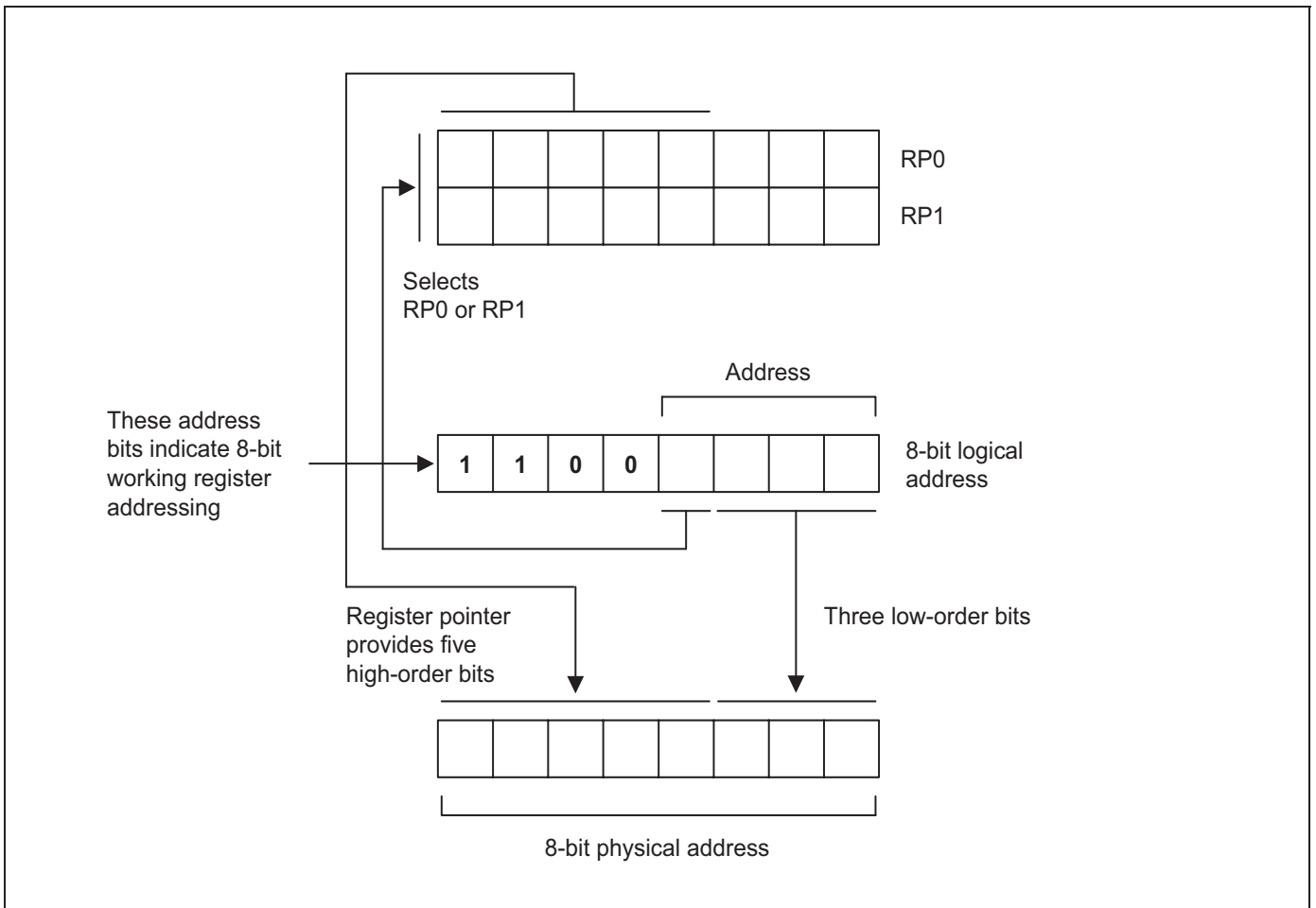


Figure 2-14 8-Bit Working Register Addressing

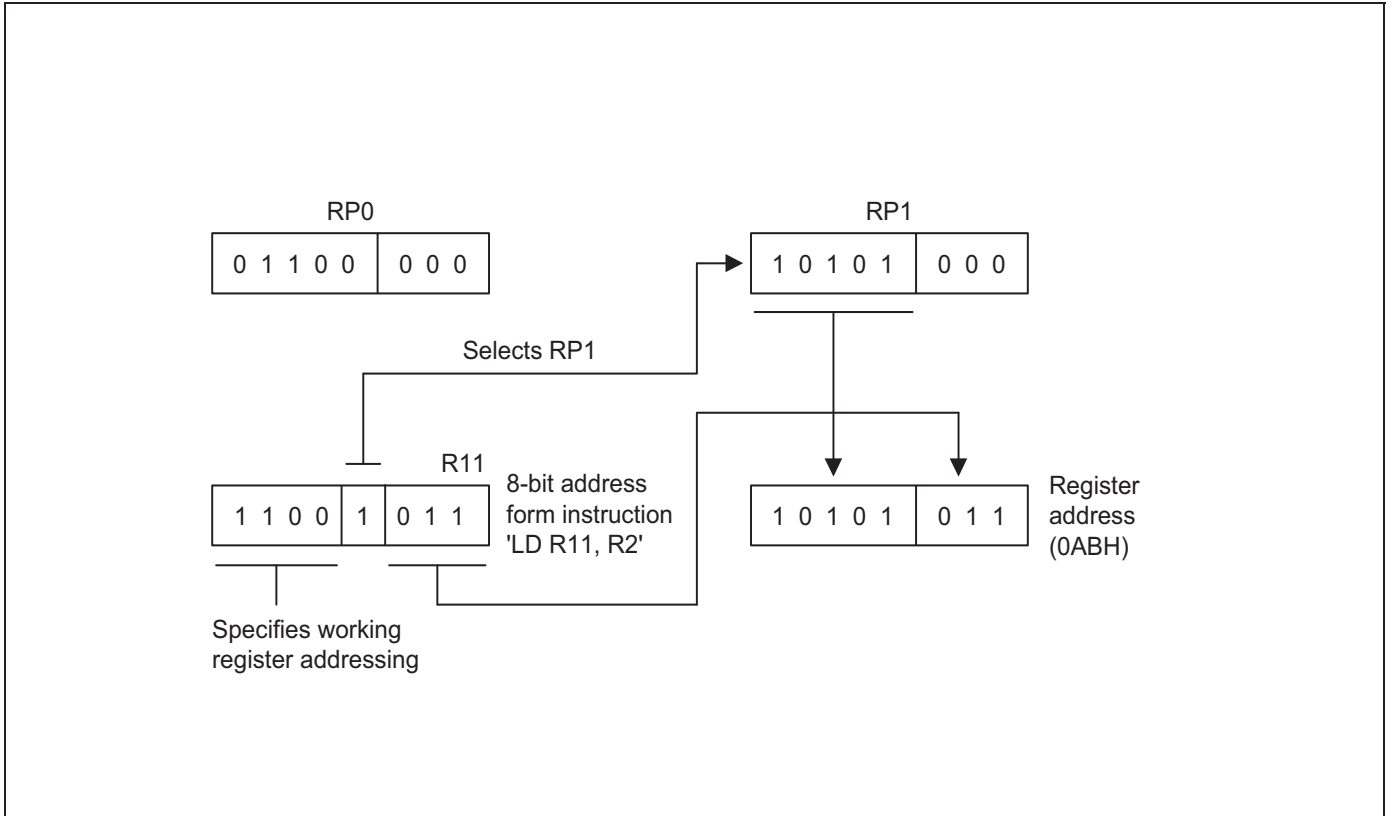


Figure 2-15 8-Bit Working Register Addressing Example

2.4.4 SYSTEM AND USER STACK

The S3F8 series microcontrollers use the system stack for data storage, subroutine calls, and returns. Both PUSH and POP instructions control the system stack operations. The S3F84B8 architecture supports stack operations in internal register file.

2.4.4.1 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 FLAGS registers 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 stack, as shown in [Figure 2-16](#).

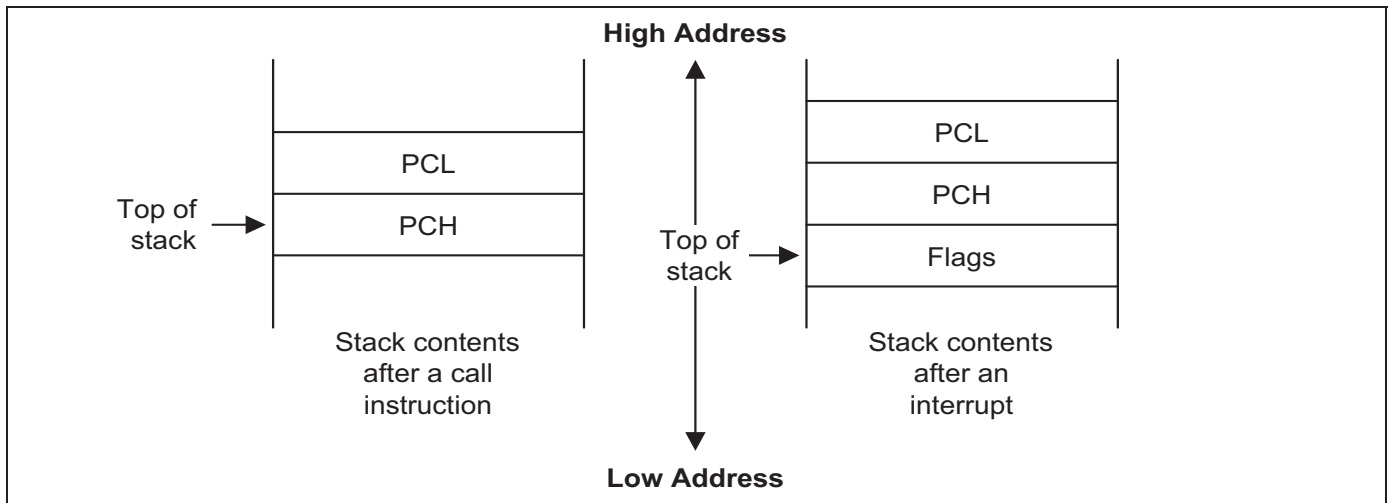


Figure 2-16 Stack Operations

2.4.4.2 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.

2.4.4.3 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.

Since only internal memory space is implemented in 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”.

Example 2-4 Standard Stack Operations Using PUSH and POP

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

```

3 ADDRESSING MODES

3.1 OVERVIEW OF ADDRESSING MODES

The program counter fetches the instructions stored in the program memory for execution. These 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 can include: 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)

3.2 REGISTER (R) ADDRESSING MODE

In Register (R) addressing mode, the operand value is the content of a specified register or register pair (see [Figure 3-1](#)).

Working register addressing differs from Register addressing since 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](#)).

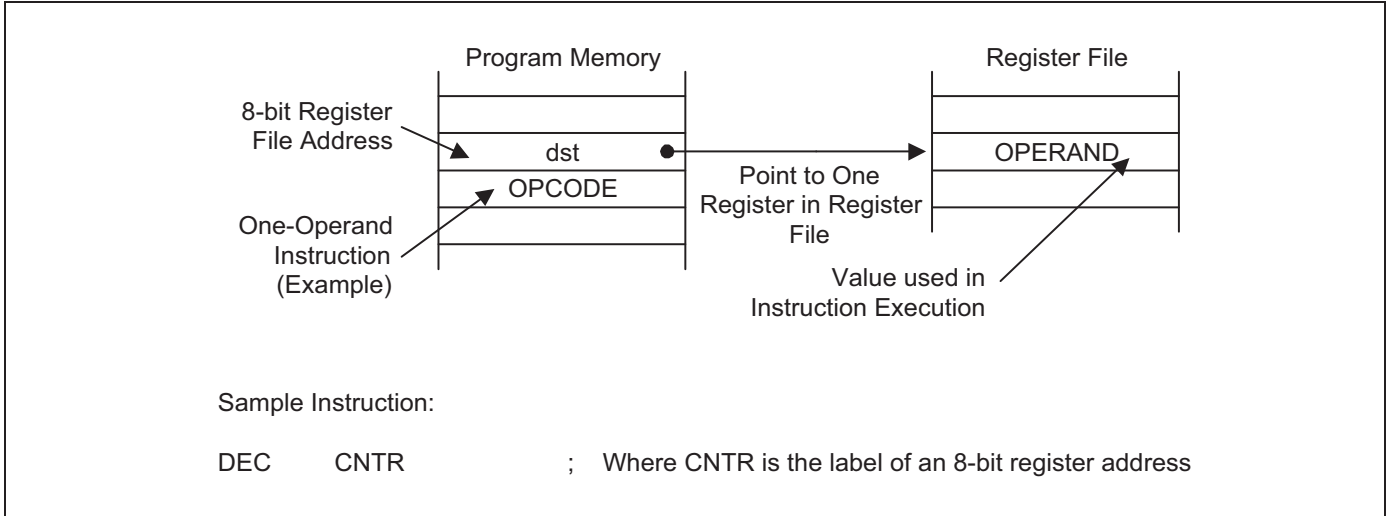


Figure 3-1 Register Addressing

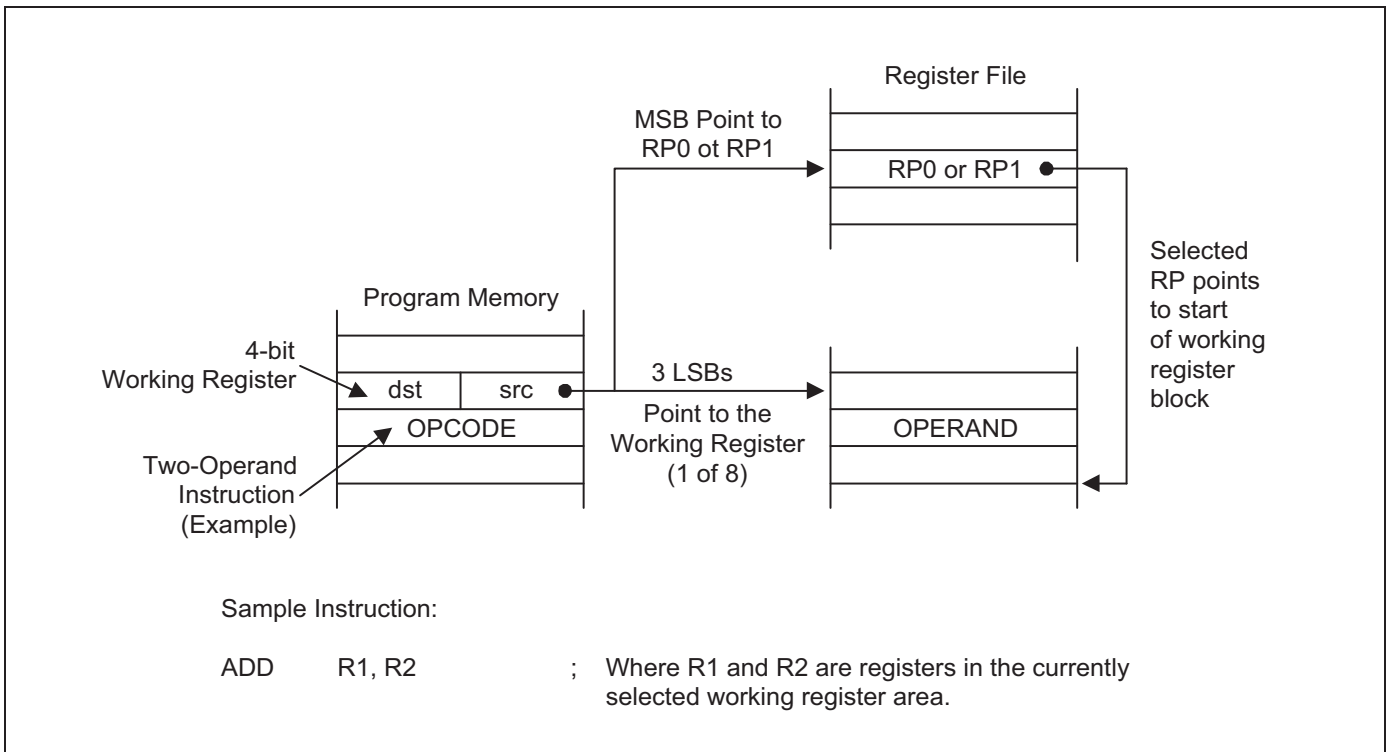


Figure 3-2 Working Register Addressing

3.3 INDIRECT REGISTER (IR) ADDRESSING MODE

In Indirect Register (IR) addressing mode, the content of a specified register or register pair is the address of operand. Depending on the instruction used, the actual address can point to a register in register file, to program memory (ROM), or to an external memory space (see [Figure 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. Note that you cannot access locations C0H–FFH in set 1 using the Indirect Register addressing mode.

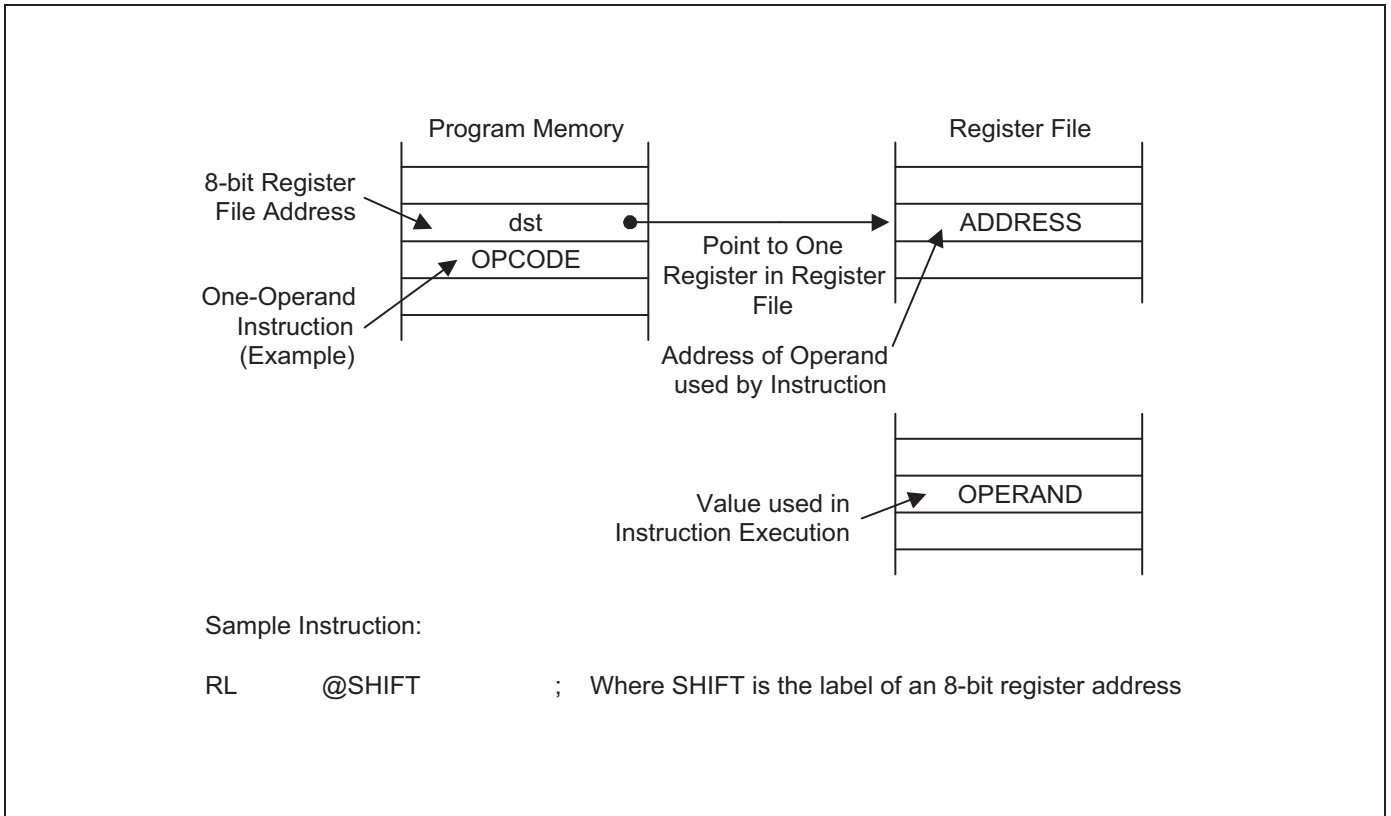


Figure 3-3 Indirect Register Addressing to Register File

3.4 INDIRECT REGISTER (IR) ADDRESSING MODE (CONTINUED)

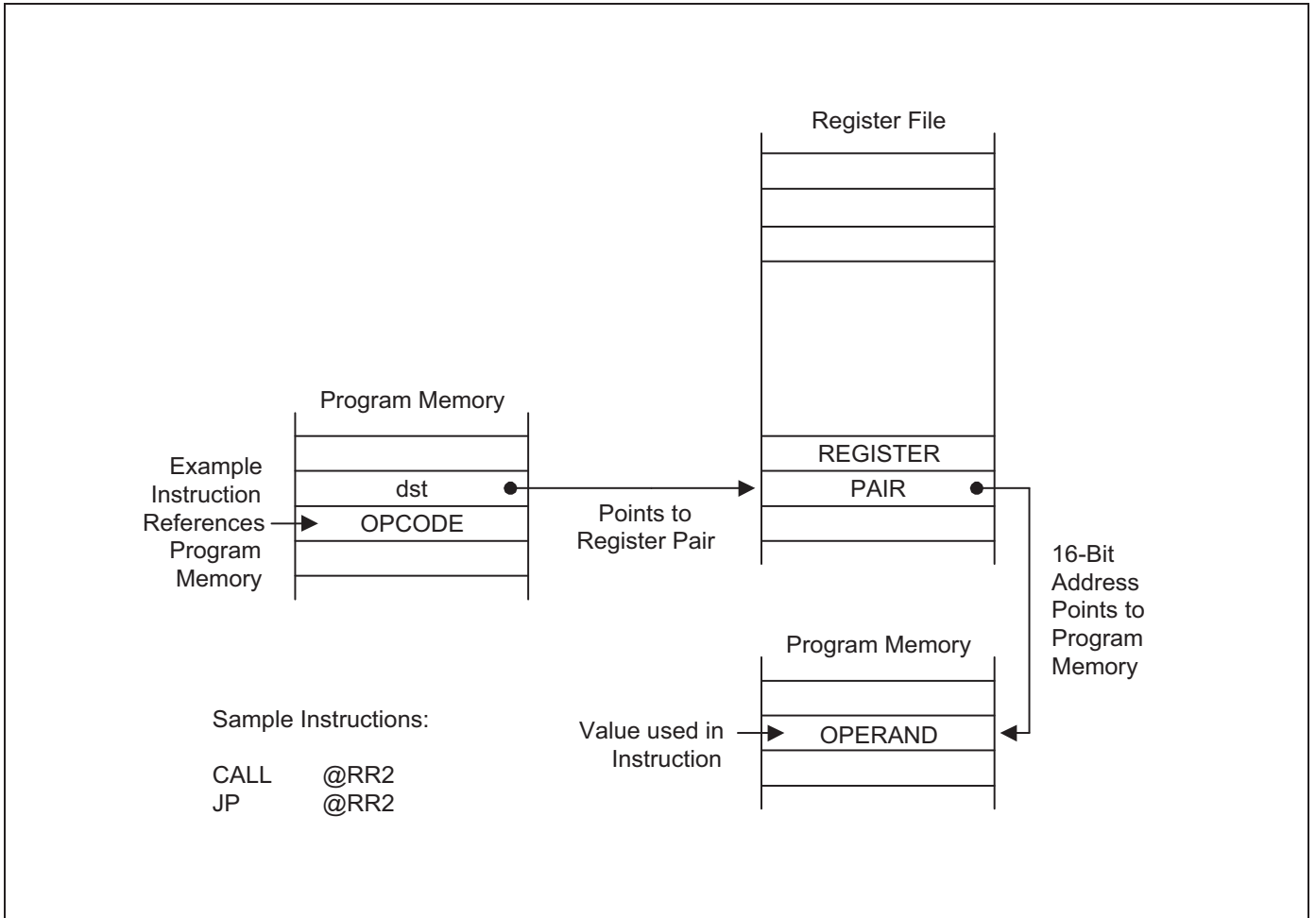


Figure 3-4 Indirect Register Addressing to Program Memory

3.5 INDIRECT REGISTER (IR) ADDRESSING MODE (CONTINUED)

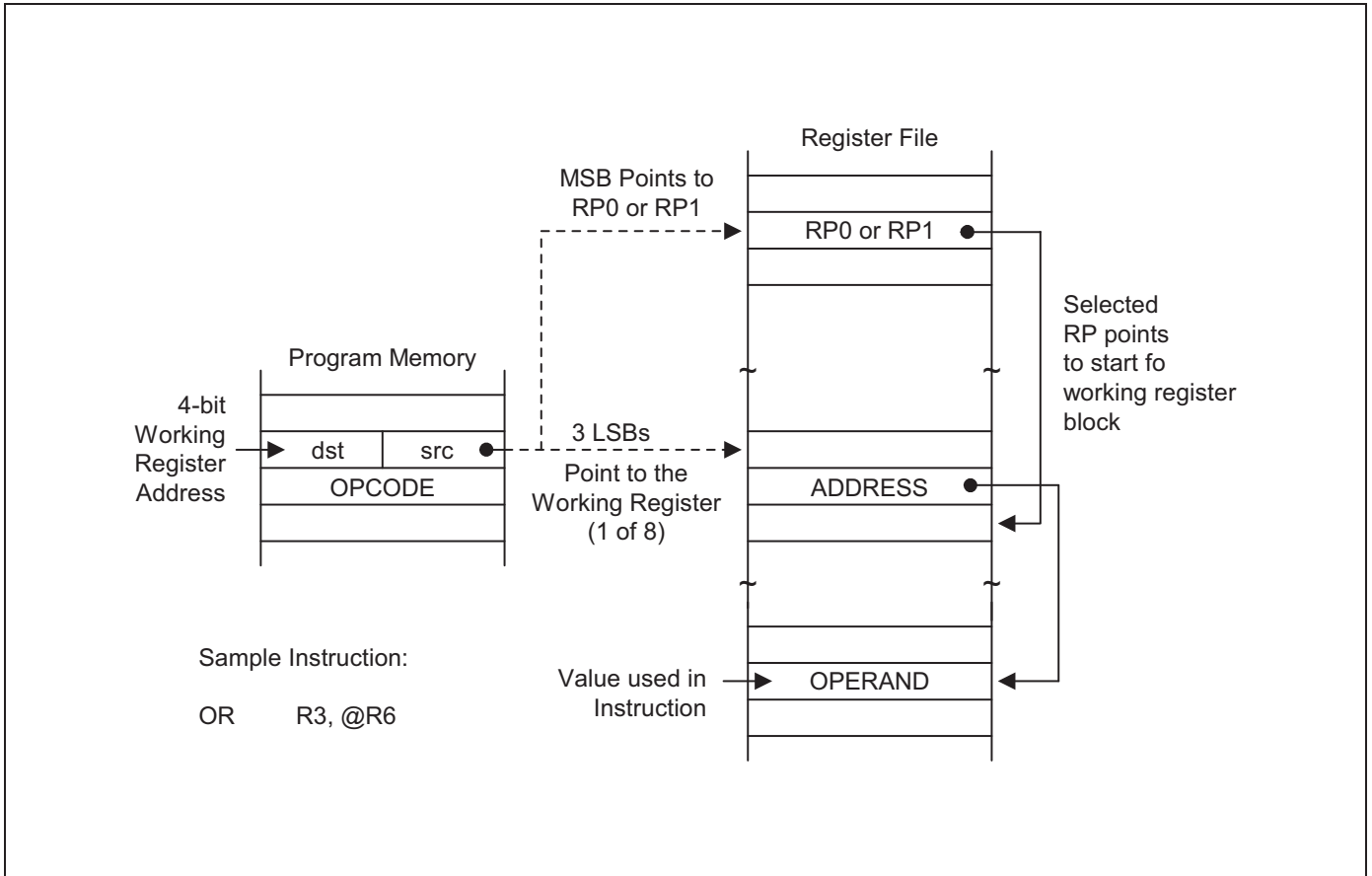


Figure 3-5 Indirect Working Register Addressing to Register File

3.6 INDIRECT REGISTER (IR) ADDRESSING MODE (CONCLUDED)

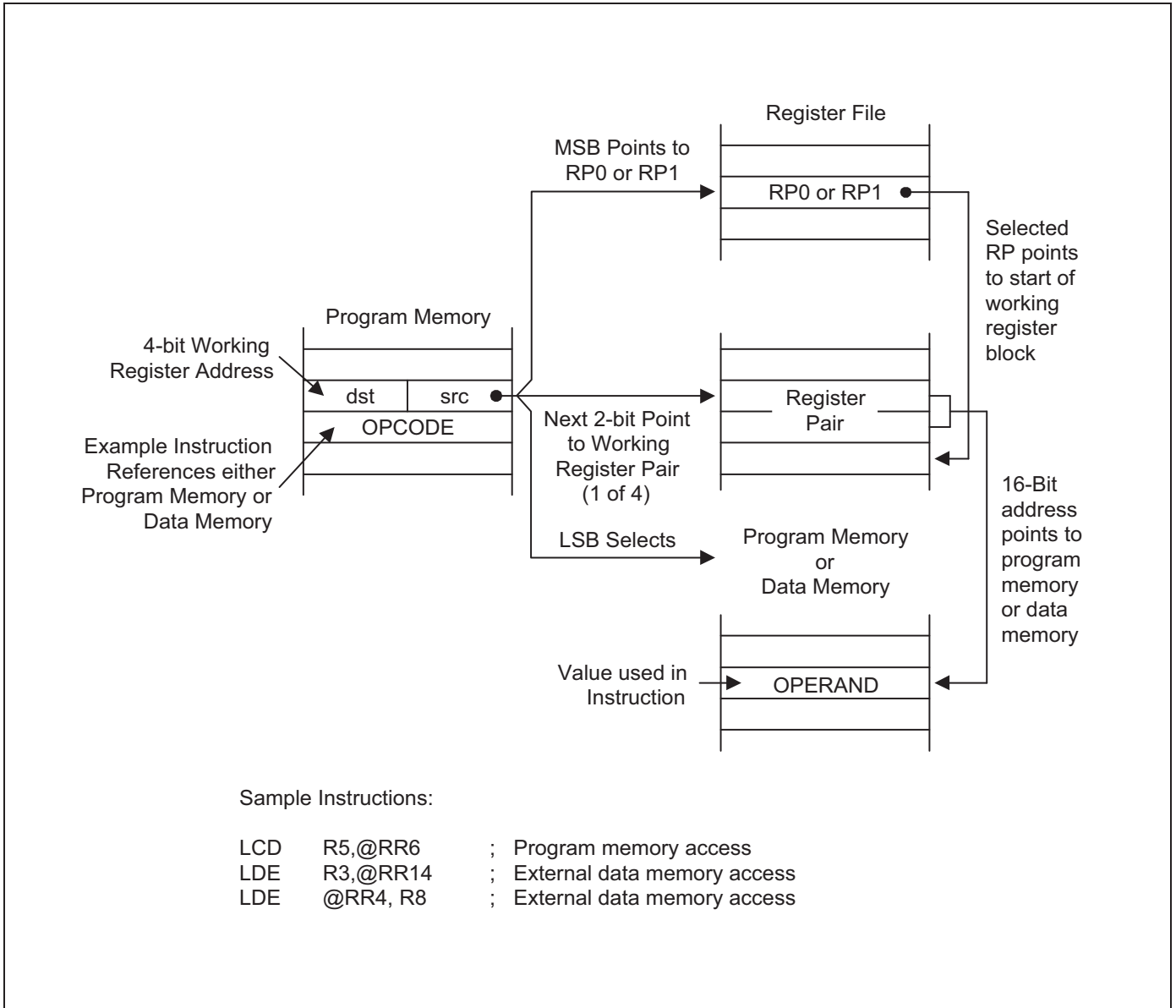


Figure 3-6 Indirect Working Register Addressing to Program or Data Memory

3.7 INDEXED (X) ADDRESSING MODE

Indexed (X) addressing mode adds an offset value to base address while executing an instruction 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 external memory. Note 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 a 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 internal register file is the Load instruction (LD). The LDC and LDE instructions support Indexed addressing mode for internal program memory and external data memory, when implemented.

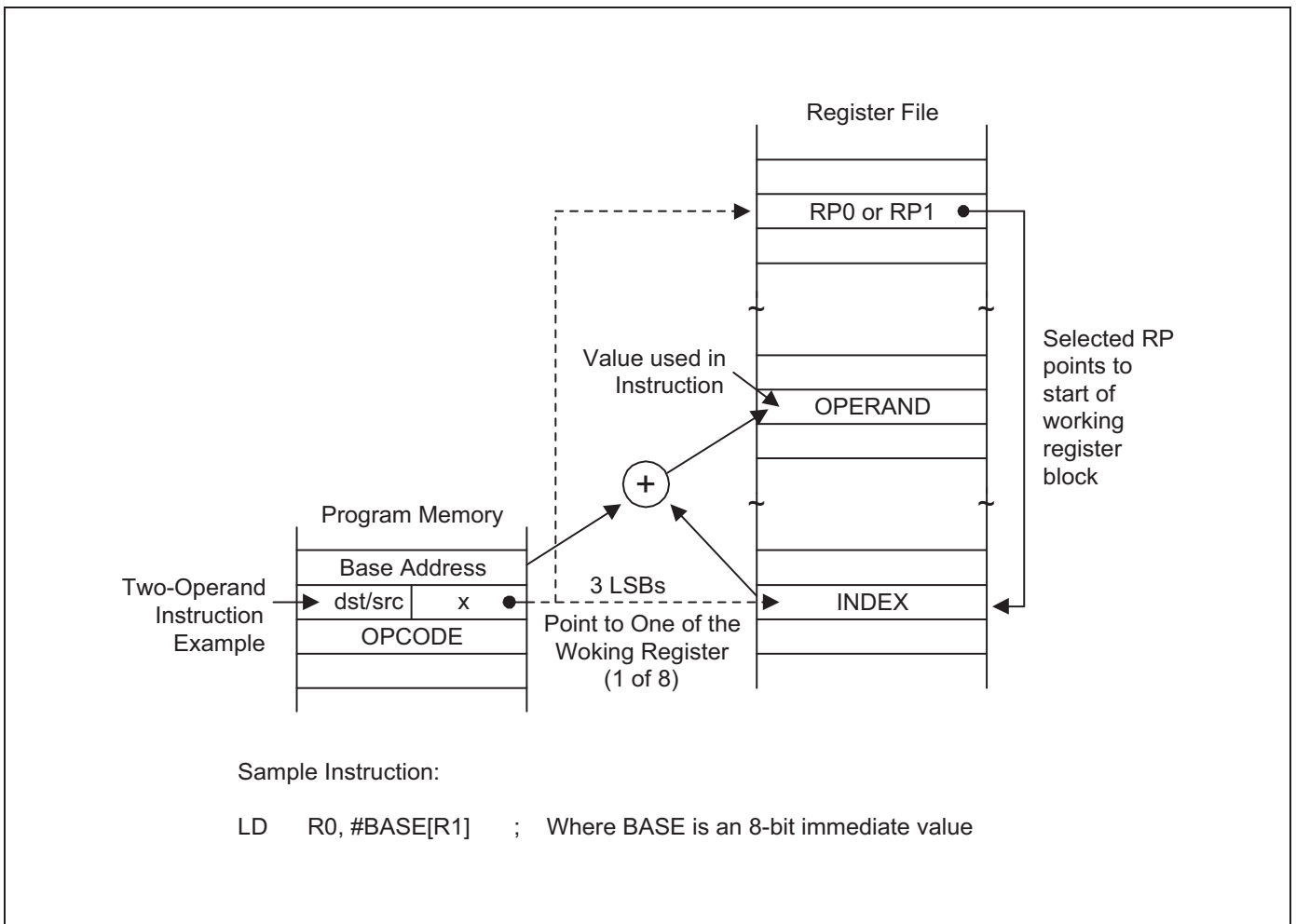


Figure 3-7 Indexed Addressing to Register File

3.13 RELATIVE ADDRESS (RA) MODE

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

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

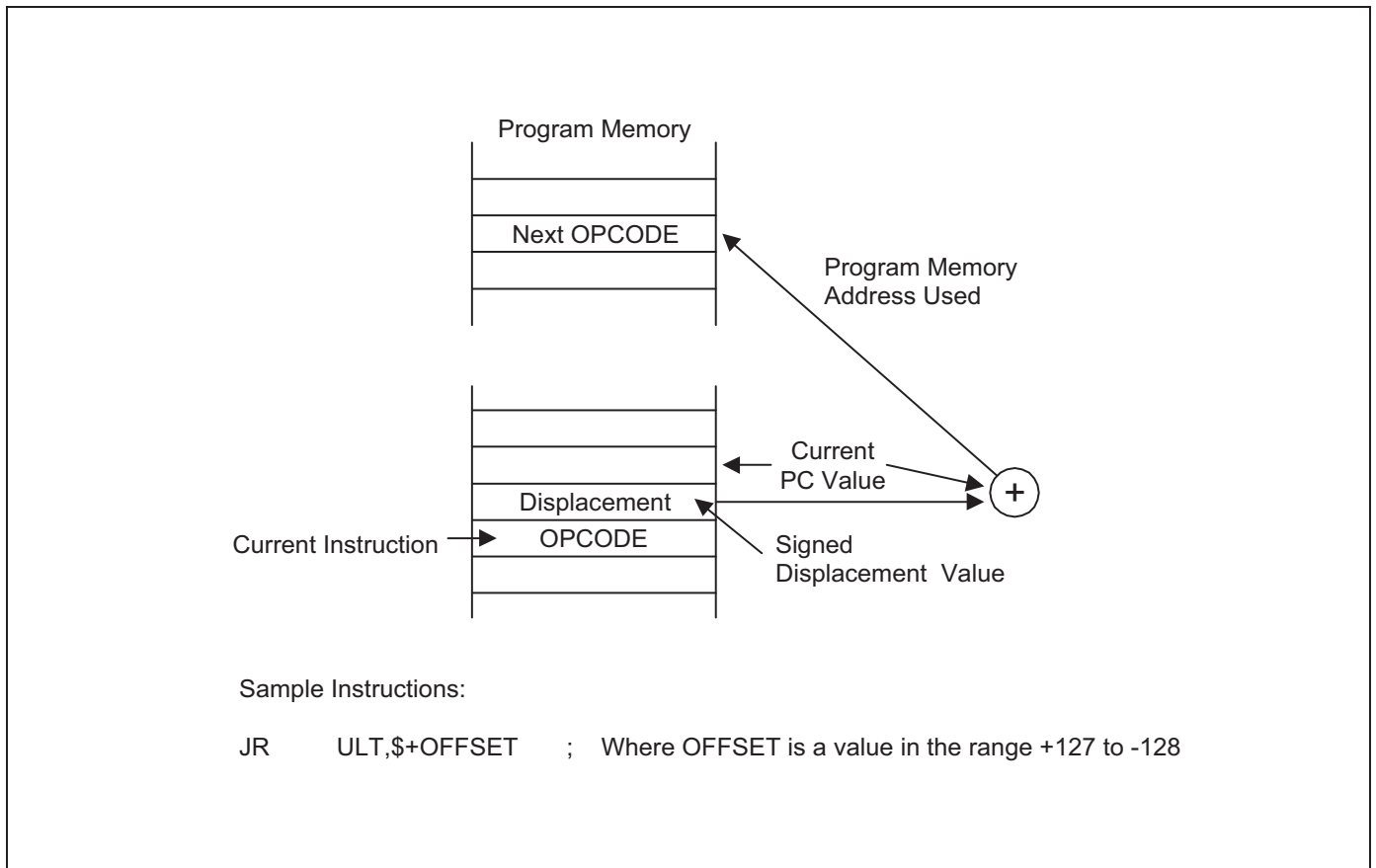


Figure 3-13 Relative Addressing

3.9 INDEXED (X) ADDRESSING MODE (CONCLUDED)

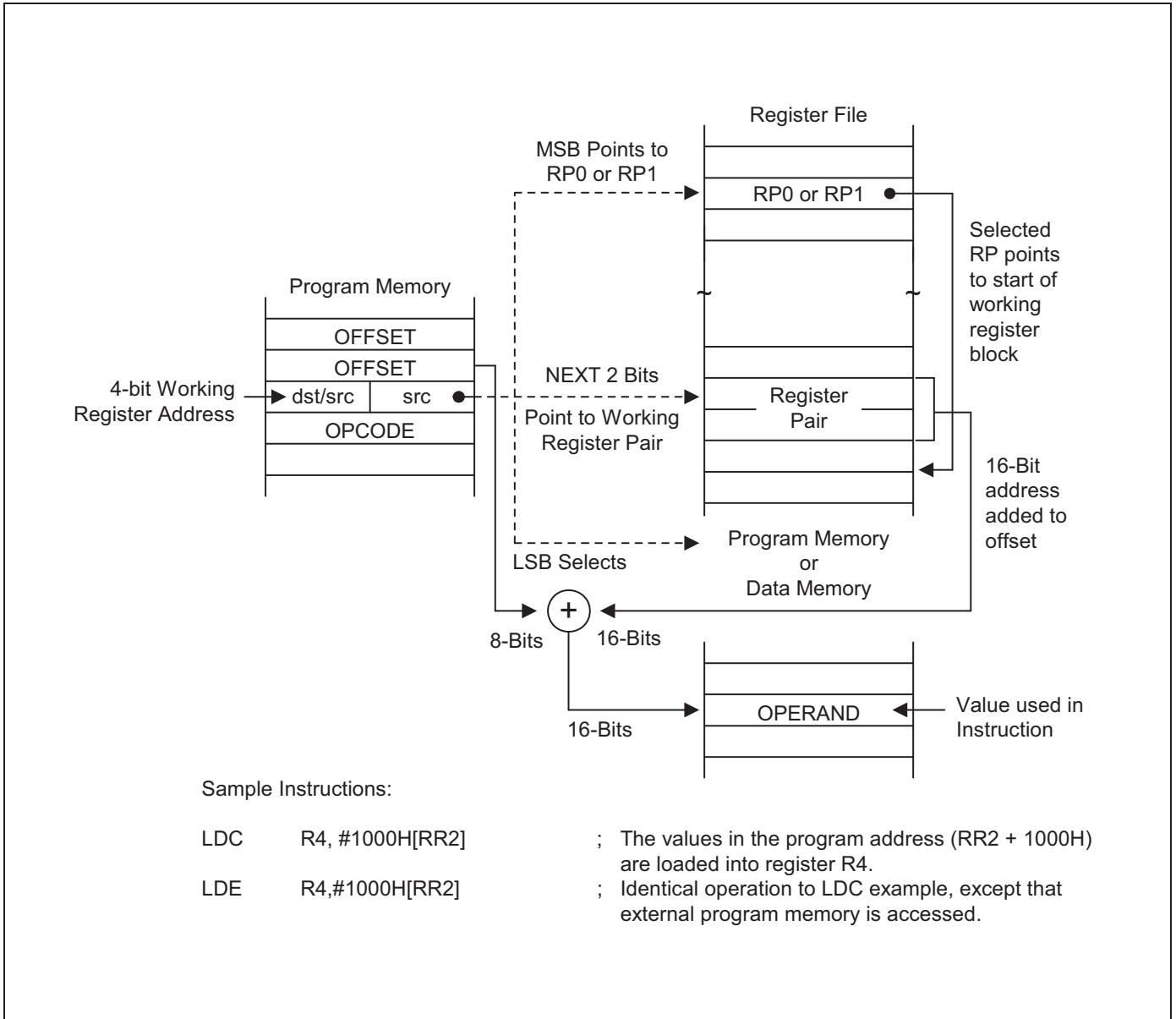


Figure 3-9 Indexed Addressing to Program or Data Memory

3.10 DIRECT ADDRESS (DA) MODE

In Direct Address (DA) mode, the instruction provides an operand's 16-bit memory address. Jump (JP) and Call (CALL) instructions use this addressing mode to specify the 16-bit destination address 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 external data memory (LDE), if implemented.

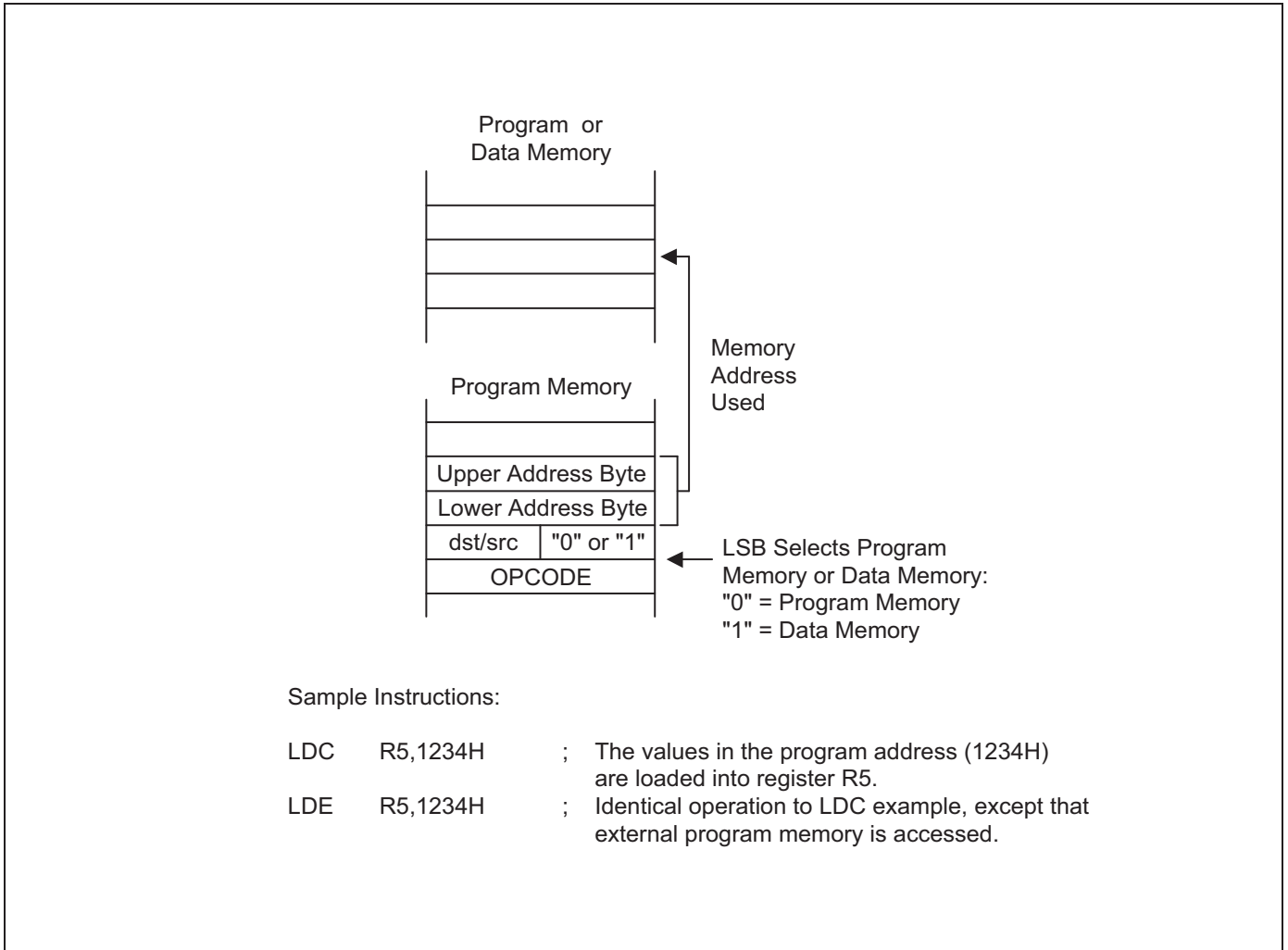


Figure 3-10 Direct Addressing for Load Instructions

3.11 DIRECT ADDRESS (DA) MODE (CONTINUED)

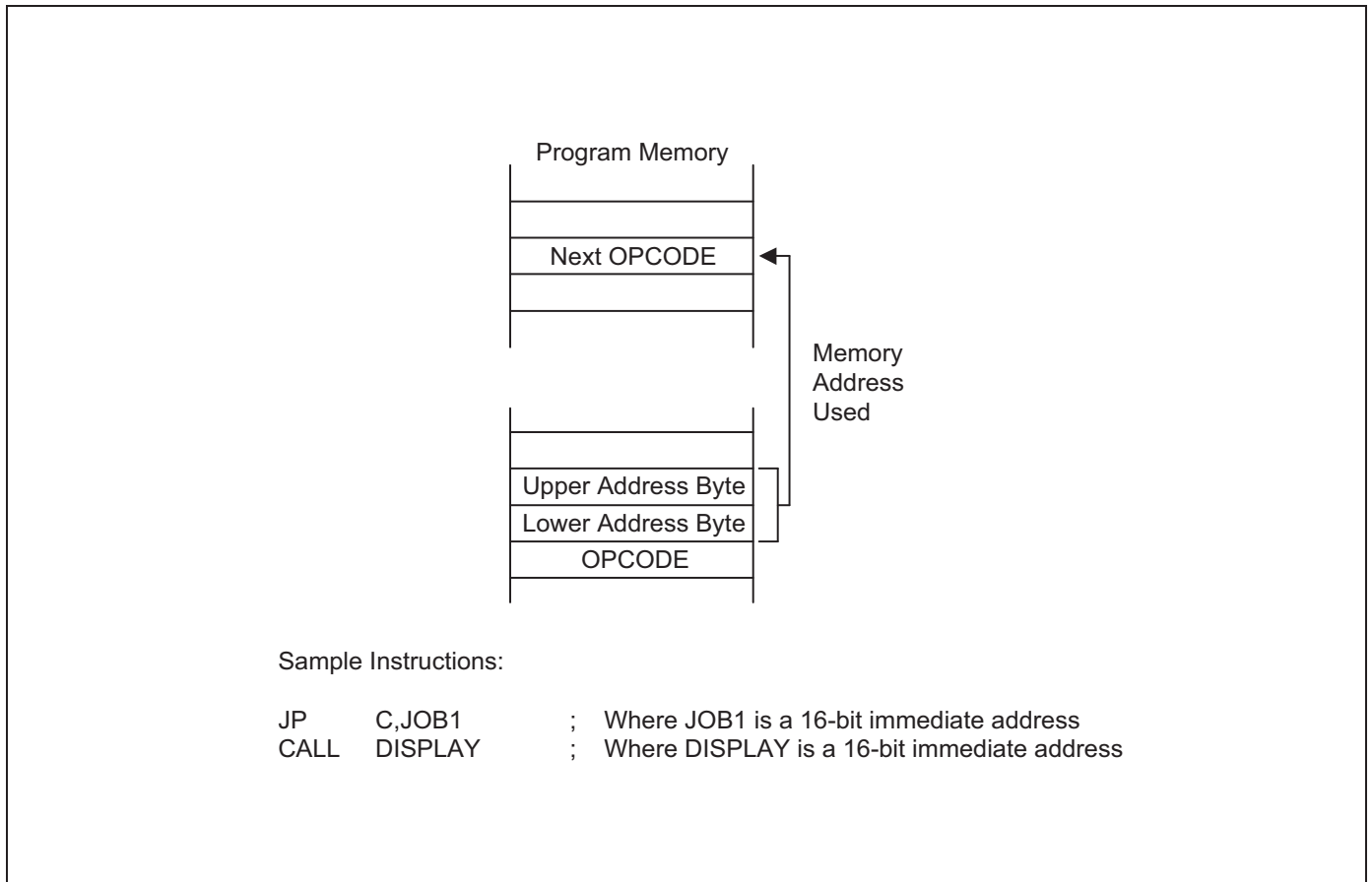


Figure 3-11 Direct Addressing for Call and Jump Instructions

3.12 INDIRECT ADDRESS (IA) MODE

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 next instruction to be executed. Only the CALL instruction can use the Indirect Address mode.

Since the assumption in using Indirect Address mode is 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 destination address are assumed to be all zeros.

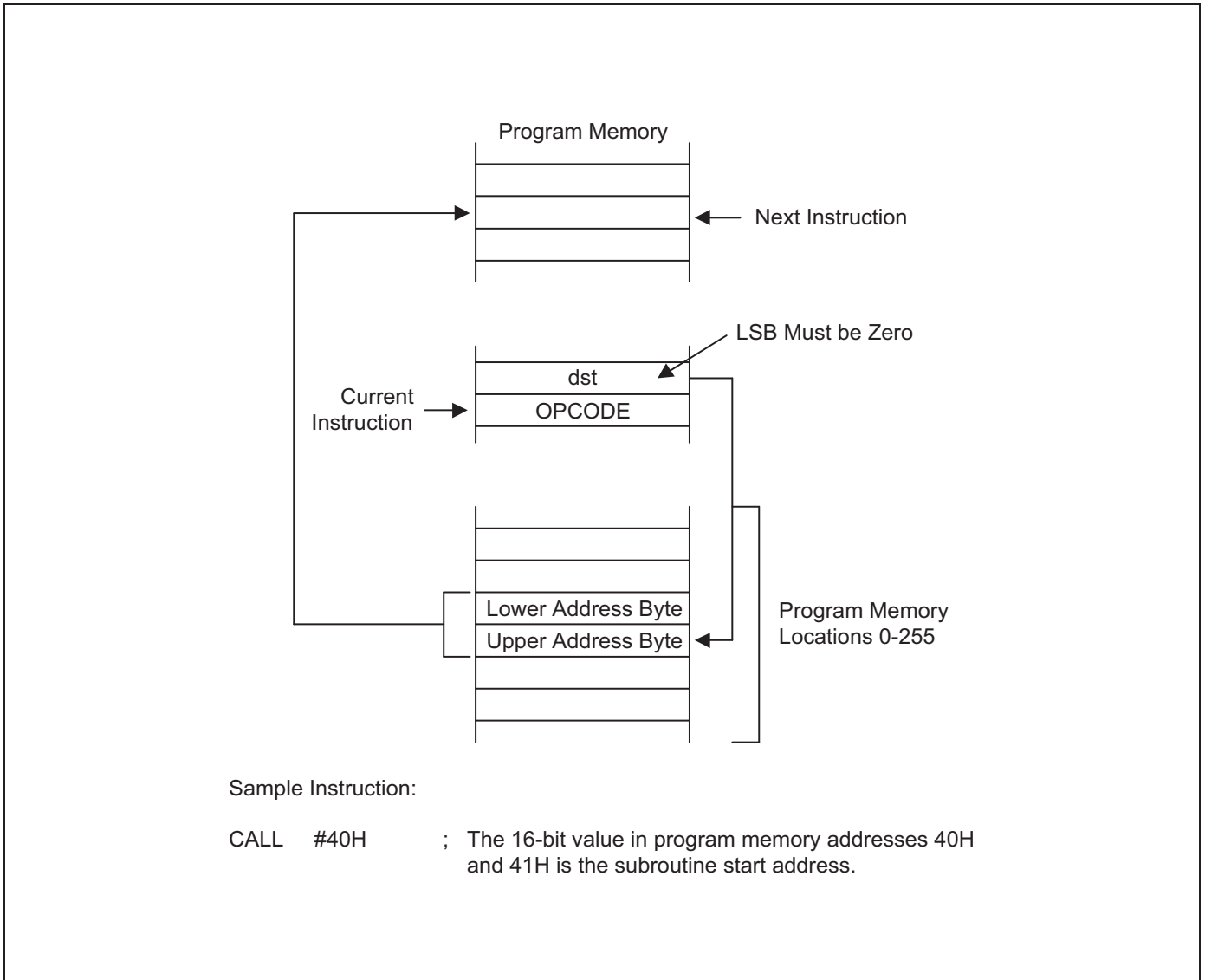


Figure 3-12 Indirect Addressing

3.13 RELATIVE ADDRESS (RA) MODE

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

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

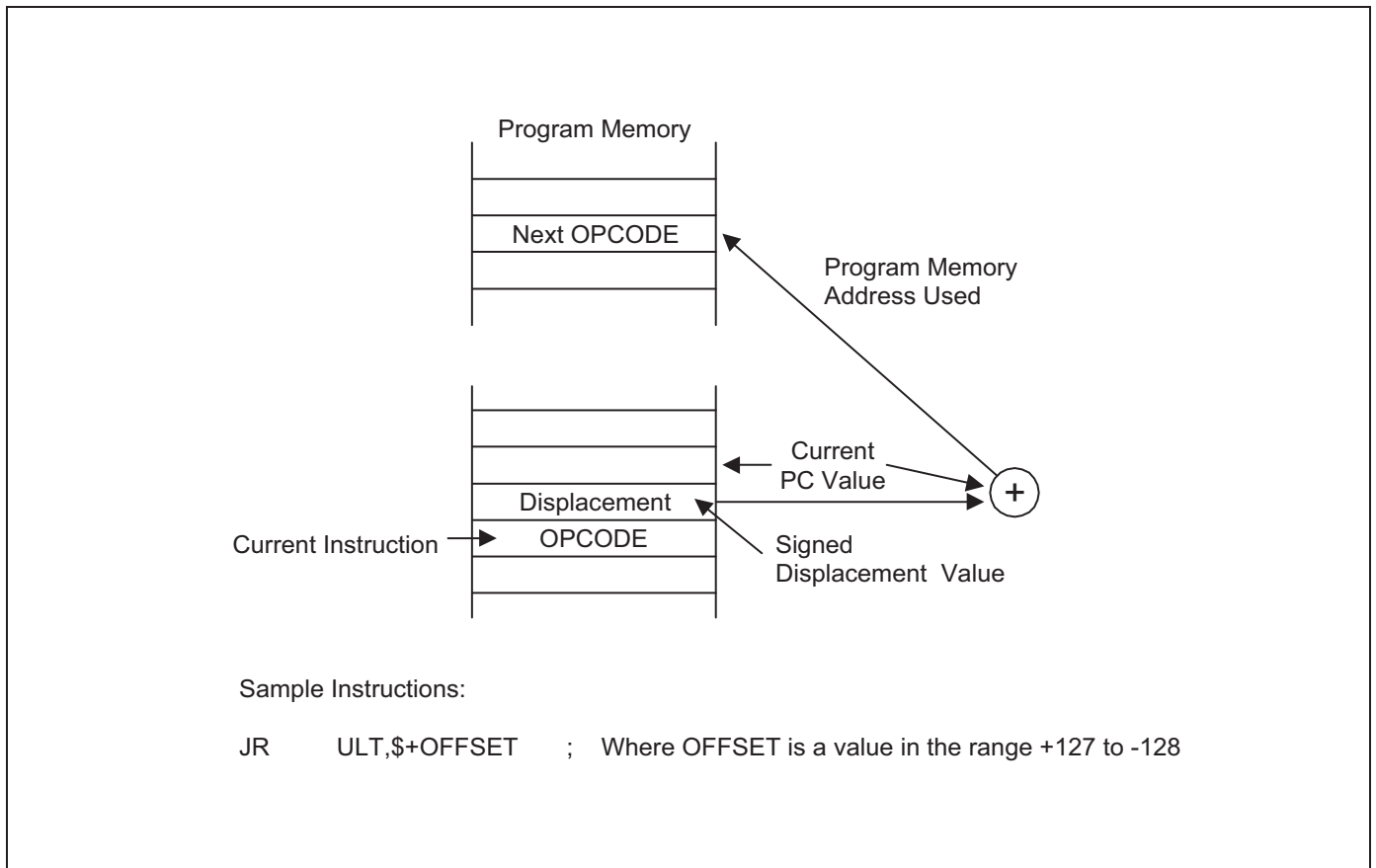


Figure 3-13 Relative Addressing

3.14 IMMEDIATE MODE (IM)

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

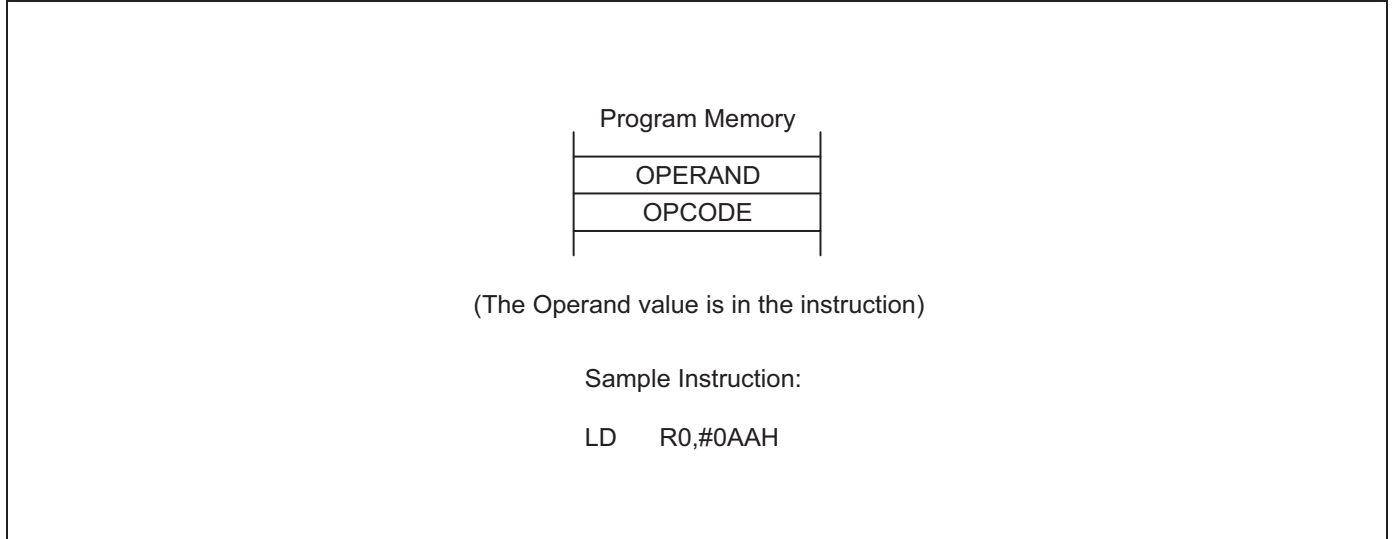


Figure 3-14 Immediate Addressing

4 CONTROL REGISTERS

4.1 OVERVIEW OF CONTROL REGISTERS

This section provides a detailed description of the S3F84B8 control registers in an easy-to-read format to familiarize you with the mapped locations in register file. You can also use them as a quick-reference source when writing application programs.

[Table 4-1](#) summarizes the system and peripheral registers. In addition, [Figure 4-1](#) illustrates the important features of standard register description format.

Control register descriptions are arranged in alphabetical order according to the register mnemonic. More information about control registers is presented in the context of various peripheral hardware descriptions in Part II of this manual.

Table 4-1 System and Peripheral Control Registers Set1 Bank0

Register Name	Mnemonic	Address and 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	0
Clock Control Register	CLKCON	D4H	R/W	0	–	–	0	0	–	–	–	–
System Flags Register	FLAGS	D5H	R/W	x	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	x
Instruction Pointer (High Byte)	IPH	DAH	R/W	x	x	x	x	x	x	x	x	x
Instruction Pointer (Low Byte)	IPL	DBH	R/W	x	x	x	x	x	x	x	x	x
Interrupt Request register	IRQ	DCH	R	0	0	0	0	0	0	0	0	0
Interrupt Mask Register	IMR	DDH	R/W	x	x	x	x	x	x	x	x	x
System Mode Register	SYM	DEH	R/W	0	–	–	x	x	x	0	0	0
Register Page Pointer	PP	DFH	R/W	0	0	0	0	0	0	0	0	0
Port 0 data register	P0	E0H	R/W	–	0	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	R/W	–	–	–	–	–	0	0	0	0
Port 2 data register	P2	E2H	R/W	0	0	0	0	0	0	0	0	0
Port 0 interrupt control register	P0INT	E3H	R/W	–	0	0	0	0	–	0	0	0
Port 0 control register (High byte)	P0CONH	E4H	R/W	–	–	0	0	0	0	0	0	0

Register Name	Mnemonic	Address and Location		RESET Value (Bit)							
		Address	R/W	7	6	5	4	3	2	1	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

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

Table 4-2 System and Peripheral Control Registers Set1 Bank1

Register Name	Mnemonic	Address and 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 detailed 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

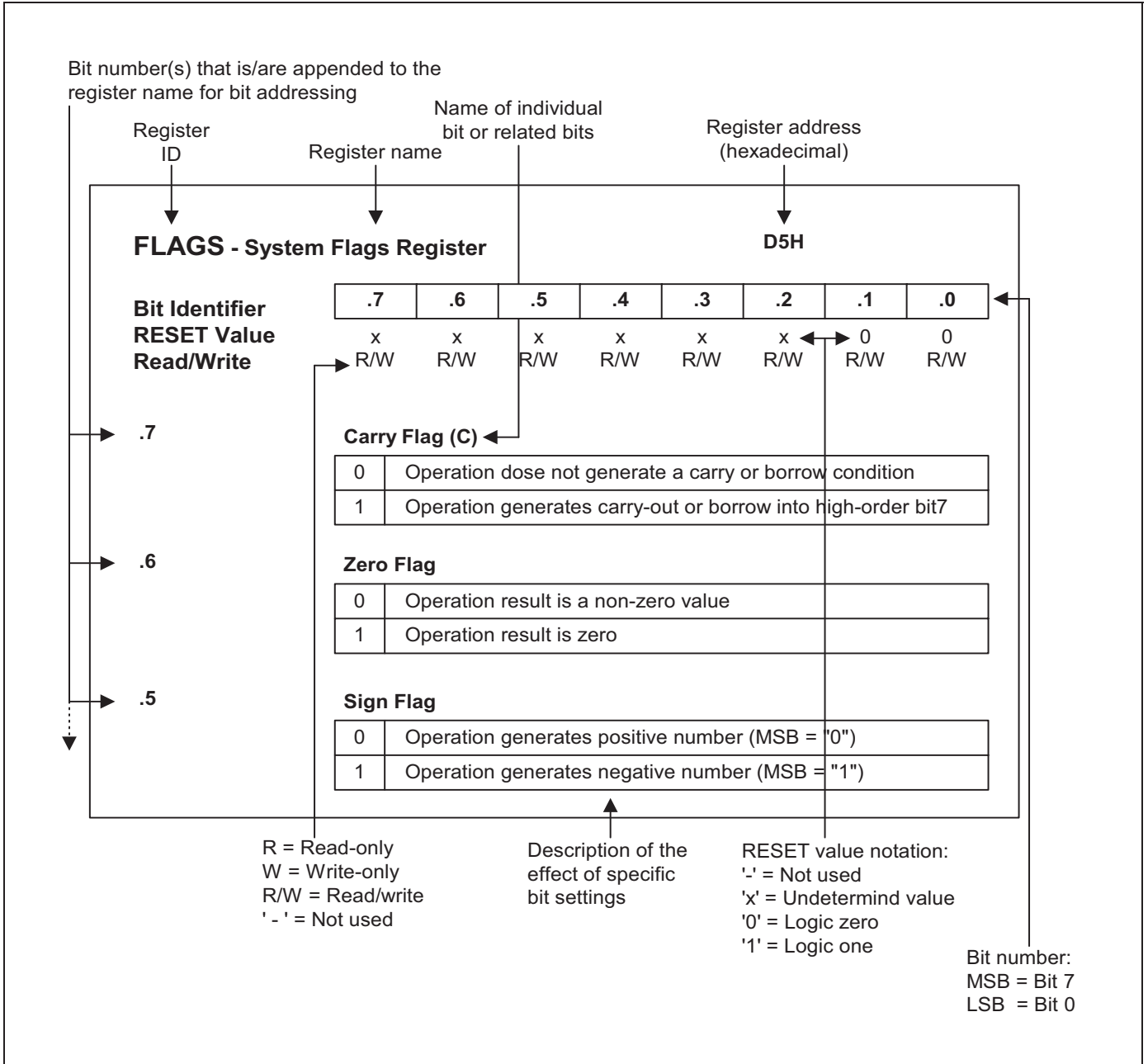


Figure 4-1 Register Description Format

4.1.1 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	Disables ADC Interrupt.						
	1	Enables ADC Interrupt.						
.3	A/DC Interrupt Pending Bit (EOC)							
	0	No interrupt is pending, conversion is in progress (clears pending bit when write).						
	1	Interrupt is pending, conversion has completed (no effect when write).						
.2–.1	Clock Source Selection Bit (Note)							
	0	0	$f_{OSC}/8$ ($f_{OSC} \leq 10\text{MHz}$)					
	0	1	$f_{OSC}/4$ ($f_{OSC} \leq 10\text{MHz}$)					
	1	0	$f_{OSC}/2$ ($f_{OSC} \leq 8\text{MHz}$)					
	1	1	$f_{OSC}/1$ ($f_{OSC} \leq 4\text{MHz}$)					
.0	Conversion Start Bit							
	0	No effect						
	1	Starts A/D conversion.						

NOTE: Maximum ADC clock input = 4MHz.

4.1.2 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 + TST							

NOTE: 0 < TST (setting time) < 4/fpwmclk

4.1.3 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	Disables watchdog timer function.			
	Others			Enables 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	Clears the basic timer counter value.						
.0	Basic Timer Divider Clear Bit							
	0	No effect.						
	1	Clears both the 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 automatically cleared to “0”.

4.1.4 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	$f_{OSC}/16$					
	0	1	$f_{OSC}/32$					
	1	0	$f_{OSC}/64$					
	1	1	$f_{OSC}/128$					
.5	BUZ Enable Bit							
	0	Disables BUZ.						
	1	Enables BUZ.						
.4–.0	BUZ Frequency = $f_{BUZ}/[(BUZCON.4-0)+1] \times 2$							

4.1.5 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	Enables IRQ for main system oscillator wake-up function.						
	1	Disables 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.							

4.1.6 CMP0CON — COMPARATOR0 CONTROL REGISTER: EAH, BANK0

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 ⁽¹⁾							
	0	Does not invert CMP0 output.						
	1	Inverts CMP0 output.						
.3	Comparator0 Enable Bit ⁽²⁾							
	0	Disables CMP0.						
	1	Enables CMP0.						
.2	Comparator0 Interrupt Enable Bit							
	0	Disables CMP0 interrupt.						
	1	Enables CMP0 interrupt.						
.1	Comparator0 Status Bit							
	0	CMP0_N > CMP0_P						
	1	CMP0_N < CMP0_P						
.0	Comparator0 Pending Bit							
	0	No interrupt is pending (clears pending bit when write).						
	1	CMP0 interrupt is pending.						

NOTE:

1. Polarity selection bit (CMP0CON.4) will not affect the interrupt generation logic.
2. Refer to “Programming Tip” in Chapter 14 for proper configuration sequence.

4.1.7 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	Does not invert CMP1 output.						
	1	Inverts CMP1 output.						
.3	Comparator1 Enable Bit							
	0	Disables CMP1.						
	1	Enables CMP1.						
.2	Comparator1 Interrupt Enable Bit							
	0	Disables CMP1 interrupt.						
	1	Enables CMP1 interrupt.						
.1	Comparator1 Status Bit							
	0	CMP1_N > CMP1_P						
	1	CMP1_N < CMP1_P						
.0	Comparator1 Pending Bit							
	0	No interrupt is pending (clears pending bit when write).						
	1	CMP1 interrupt is pending.						

NOTE:

1. Polarity selection bit (CMP1CON.4) will not affect the interrupt generation logic.
2. Refer to “Programming Tip” in Chapter 14 for proper configuration sequence.

4.1.8 CMP2CON — COMPARATOR2 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				
.4	Comparator2 Output Polarity Select Bit							
	0	Does not invert CMP2 output.						
	1	Inverts CMP2 output.						
.3	Comparator2 Enable Bit							
	0	Disables CMP1.						
	1	Enables CMP1.						
.2	Comparator2 Interrupt Enable Bit							
	0	Disables CMP1 interrupt.						
	1	Enables CMP1 interrupt.						
.1	Comparator2 Status Bit							
	0	CMP2_N > CMP2_P						
	1	CMP2_N < CMP2_P						
.0	Comparator2 Pending Bit							
	0	No interrupt is pending (clears pending bit when write).						
	1	CMP2 interrupt is pending.						

NOTE:

1. Polarity selection bit (CMP2CON.4) will not affect the interrupt generation logic.
2. Refer to “Programming Tip” in Chapter 14 for proper configuration sequence.

4.1.9 CMP3CON — COMPARATOR3 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				
.4	Comparator3 Output Polarity Select Bit							
	0	Does not invert CMP3 output.						
	1	Inverts CMP3 output.						
.3	Comparator3 Enable Bit							
	0	Disables comparator3.						
	1	Enables comparator3.						
.2	Comparator3 Interrupt Enable Bit							
	0	Disables CMP3 interrupt.						
	1	Enables CMP3 interrupt.						
.1	Comparator3 Status Bit							
	0	CMP3_N > CMP3_P						
	1	CMP3_N < CMP3_P						
.0	Comparator3 Pending Bit							
	0	No interrupt is pending (clears pending bit when write).						
	1	CMP3 interrupt is pending.						

NOTE:

1. Polarity selection bit (CMP3CON.4) will not affect the interrupt generation logic.
2. Refer to “Programming Tip” in Chapter 14 for proper configuration sequence.

4.1.10 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, the CMPINT register must be set to appropriate value before enabling CMP0/1/2/3.

4.1.11 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 or borrow into high-order bit 7.						
.6	Zero Flag (Z)							
	0	Operation results in a non-zero value.						
	1	Operation results in zero value.						
.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	Completes Add operation.						
	1	Completes Subtraction operation.						
.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.						

4.1.12 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 stops.						
	1	Operation starts (This bit will be cleared automatically just after the corresponding operation is completed).						

4.1.13 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 15 th - 8 th bits selects a sector of flash ROM.							

NOTE: The high byte Flash Memory Sector Address Pointer's value is the higher 8-bits of the 16-bit pointer address.

4.1.14 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 7 th bit selects a sector of flash ROM.							
.6–.0	Bits 6–0							
	Don't care.							

NOTE: The low byte Flash Memory Sector Address Pointer's value is the lower 8-bits of the 16-bit pointer address.

4.1.15 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				Enables user programming mode.			
	Other values				Disables user programming mode.			

4.1.16 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	Disables (mask).						
	1	Enables (unmask).						
.6	Interrupt Level 6 (IRQ6)							
	0	Disables (mask).						
	1	Enables (unmask).						
.5	Interrupt Level 5 (IRQ5)							
	0	Disables (mask).						
	1	Enables (unmask).						
.4	Interrupt Level 4 (IRQ4)							
	0	Disables (mask).						
	1	Enables (unmask).						
.3	Interrupt Level 3 (IRQ3)							
	0	Disables (mask).						
	1	Enables (unmask).						
.2	Interrupt Level 2 (IRQ2)							
	0	Disables (mask).						
	1	Enables (unmask).						
.1	Interrupt Level 1 (IRQ1)							
	0	Disables (mask).						
	1	Enables (unmask).						
.0	Interrupt Level 0 (IRQ0)							
	0	Disables (mask).						
	1	Enables (unmask).						

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

4.1.17 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's value is the upper 8-bits of the 16-bit instruction pointer address (IP15–IP8). The lower byte of the IP address is located in the IPL register (DBH).								

4.1.18 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 8-bits of the 16-bit instruction pointer address (IP7–IP0). The upper byte of the IP address is located in the IPH register (DAH).								

4.1.19 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

4.1.20 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						

4.1.21 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	Disables OP AMP.						
	1	Enables OP AMP.						

4.1.22 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					

4.1.23 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					

4.1.24 POINT — 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	Disables interrupt.						
	1	Enables interrupt.						
.5	P0.5/INT4 Interrupt Enable/Disable Selection Bits							
	0	Disables interrupt.						
	1	Enables interrupt.						
.4	P0.4/INT3 Interrupt Enable/Disable Selection Bits							
	0	Disables interrupt.						
	1	Enables interrupt.						
.3	P0.3/INT2 Interrupt Enable/Disable Selection Bits							
	0	Disables interrupt.						
	1	Enables interrupt.						
.2	Not used for S3F84B8.							
.1	P0.1/ INT1 Interrupt Enable/Disable Selection Bits							
	0	Disables interrupt.						
	1	Enables interrupt.						
.0	P0.0 / INT0 Interrupt Enable/Disable Selection Bits							
	0	Disables interrupt.						
	1	Enables interrupt.						

4.1.25 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 is pending (when read); clears pending bit (when write).						
	1	Interrupt is pending (when read); no effect (when write).						
.5	Port 0.5/INT4 Interrupt Pending Bit							
	0	No interrupt is pending (when read); clears pending bit (when write).						
	1	Interrupt is pending (when read); no effect (when write).						
.4	Port 0.4/INT3 Interrupt Pending Bit							
	0	No interrupt is pending (when read); clears pending bit (when write).						
	1	Interrupt is pending (when read); no effect (when write).						
.3	Port 0.3/INT2 Interrupt Pending Bit							
	0	No interrupt is pending (when read); clears pending bit (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 is pending (when read); clears pending bit (when write).						
	1	Interrupt is pending (when read); no effect (when write).						
.0	Port 0.0/INT0 Interrupt Pending Bit							
	0	No interrupt is pending (when read); clears pending bit (when write).						
	1	Interrupt is pending (when read); no effect (when write).						

4.1.26 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; enables pull-up.					
	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; enables pull-up; 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; enables pull-up; TACK input.					
	1	0	Push pull output					
	1	1	Alternative function: comparator 0 positive input					

4.1.27 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; enables pull-up.					
	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; enables pull-up.					
	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					

4.1.28 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; enables pull-up.					
	1	0	Push-pull output					
	1	1	Alternative function: ADC3 input (NOTE)					
.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/TDOOUT Configuration Bits							
	0	0	Schmitt trigger input					
	0	1	Alternative function: TDOOUT					
	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, regardless of whether you want to use internal ADC module.

4.1.29 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	Clears the PWM counter (when write).						
.3	PWM Counter Enable Bit							
	0	Stops counter.						
	1	Starts counter (unlock operation).						
.2	Anti-Mis-Trigger Enable Bit							
	0	Disables anti-mis-trigger function.						
	1	Enables anti-mis-trigger function.						
.1	PWM Overflow Interrupt Enable Bit							
	0	Disables interrupt.						
	1	Enables interrupt.						
.0	PWM Overflow Interrupt Pending Bit							
	0	No interrupt is pending; clears pending bit (when write).						
	1	Interrupt is pending; no effect (when write).						

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

4.1.30 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	Disables linkage.					
	0	1	Soft Lock					
	1	1	Hard lock					
.5–.4	CMP2 PWM Linkage Mode Selection Bit							
	X	0	Disables linkage.					
	0	1	Soft Lock					
	1	1	Hard lock					
.3–.2	CMP1 PWM Lock Mode Selection Bit							
	X	0	Disables linkage.					
	0	1	Soft Lock					
	1	1	Hard lock					
.1–.0	CMP0 PWM Trigger Mode Selection Bit							
	X	0	Disables 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.

4.1.31 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 + TST							

NOTE: 0 < TST (setting time) < 4/fpwmclk

4.1.32 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 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.

4.1.33 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 the Reset Source								
	.7	.6	.5	.4	.3	.2	.1	.0
LVR	–	–	–	0	–	0	1	–
WDT, or nReset pin	–	–	–	(4)	–	(4)	(3)	–

NOTE:

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

4.1.34 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 and selects the 8-byte working register slice C0H–C7H.								
.2–.0								
Not used for the S3F84B8.								

4.1.35 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 and selects the 8-byte working register slice C8H–CFH.								
.2–.0								
Not used for the S3F84B8.								

4.1.36 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.								

4.1.37 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		Enables STOP instruction.						
Other value		Disables STOP instruction.						

NOTE:

1. Before executing the STOP instruction, set this STPCON register to “10100101b”.
2. When STOPCON register does not have #0A5H value and you use STOP instruction, PC is changed to reset address.

4.1.38 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 (disables tri-state operation).						
	1	Sets the external interface lines to high impedance (enables tri-state operation).						
.6–.5	Not used for 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	Fast Interrupt Enable Bit ⁽³⁾							
	0	Disables fast interrupt processing.						
	1	Enables fast interrupt processing.						
.0	Global Interrupt Enable Bit ⁽⁴⁾							
	0	Disables all interrupt processing.						
	1	Enables all interrupt processing.						

NOTE:

1. Since 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).

4.1.39 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 (captures on rising edge; counter running; OVF can occur)					
	1	0	Capture mode (captures 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	Clears the timer A counter (After clearing, returns to zero).						
.4	Timer A Start/Stop Bit							
	0	Stops Timer A.						
	1	Starts Timer A.						
.3	Timer A Match/Capture Interrupt Enable Bit							
	0	Disables interrupt.						
	1	Enables interrupt.						
.2	Timer A Overflow Interrupt Enable Bit							
	0	Disables interrupt.						
	1	Enables interrupt.						
.1	Timer A Match Interrupt Pending Bit							
	0	No interrupt is pending; clears pending bit (when write).						
	1	Interrupt is pending.						
.0	Timer A Overflow Interrupt Pending Bit							
	0	No interrupt is pending; clears pending bit (when write).						
	1	Interrupt is pending.						

4.1.40 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/ (2TAPS[3-0]); Pre-scalar values above 12 are invalid.								

4.1.41 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	Clears the timer C counter (After clearing, returns to zero).						
.4	Timer C Start/Stop Bit							
	0	Stops Timer C.						
	1	Starts Timer C.						
.3	Timer C Match Interrupt Enable Bit							
	0	Disables Interrupt.						
	1	Enables Interrupt.						
.2	Not used for S3F84B8.							
.1	Timer C Match Interrupt Pending Bit							
	0	No interrupt is pending; clears pending bit (when write).						
	1	Interrupt is pending.						
.0	Not used for S3F84B8.							

4.1.42 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/(2TCPS); 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.

4.1.43 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	Clears the timer D counter (when write).						
.4	Timer D Start/Stop Bit							
	0	Stops Timer D.						
	1	Starts Timer D.						
.3	Timer D Match Interrupt Enable Bit							
	0	Disables interrupt.						
	1	Enables interrupt.						
.2	Timer D Overflow Interrupt Enable Bit							
	0	Disables interrupt.						
	1	Enables interrupt.						
.1	Timer D Match Interrupt Pending Bit							
	0	No interrupt is pending; clears pending bit (when write).						
	1	Interrupt is pending.						
.0	Timer D Overflow Interrupt pending Bit							
	0	No interrupt is pending; clears pending bit (when write).						
	1	Interrupt is pending.						

4.1.44 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/(2TDPS[.3-.0]) Pre-scalar values above 12 are invalid.								

5

INTERRUPT STRUCTURE

5.1 OVERVIEW OF INTERRUPT STRUCTURE

The interrupt structure in S3F8 series 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.

5.1.1 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 allow you to define complex priority relationships between different levels.

5.1.2 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 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.

5.1.3 SOURCES

A source refers to any peripheral that generates an interrupt. It can be an external pin or a counter overflow. Each vector can have several interrupt sources. There are 17 possible interrupt sources in S3F84B8 interrupt structure, which means that every source can have its own vector.

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

5.1.4 INTERRUPT TYPES

The three components of the S3F8 interrupt structure—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 (IRQ_n) + one vector (V₁) + one source (S₁)
- Type 2: One level (IRQ_n) + one vector (V₁) + multiple sources (S₁ – S_n)
- Type 3: One level (IRQ_n) + multiple vectors (V₁ – V_n) + multiple sources (S₁ – S_n, S_{n+1} – S_{n+m})

In the S3F84B8 microcontroller, type 1 and type 2 are implemented.

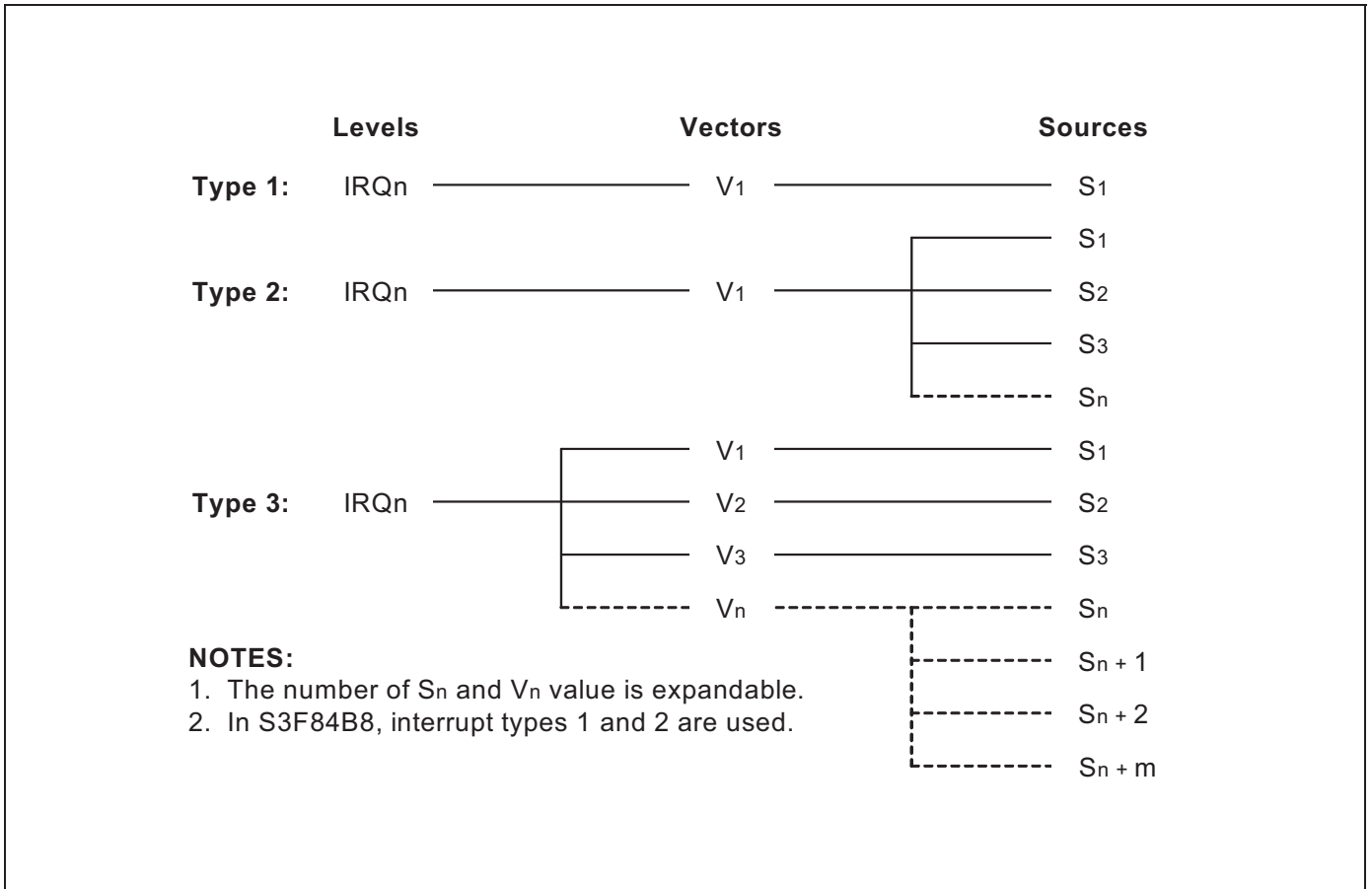


Figure 5-1 S3F8 Series Interrupt Types

5.1.5 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 lowest vector address is usually processed first (The relative priorities of multiple interrupts within a single level are fixed in the 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 service routine is fetched from the appropriate vector address (plus the next 8-bit value to concatenate full 16-bit address) and the service routine is executed.

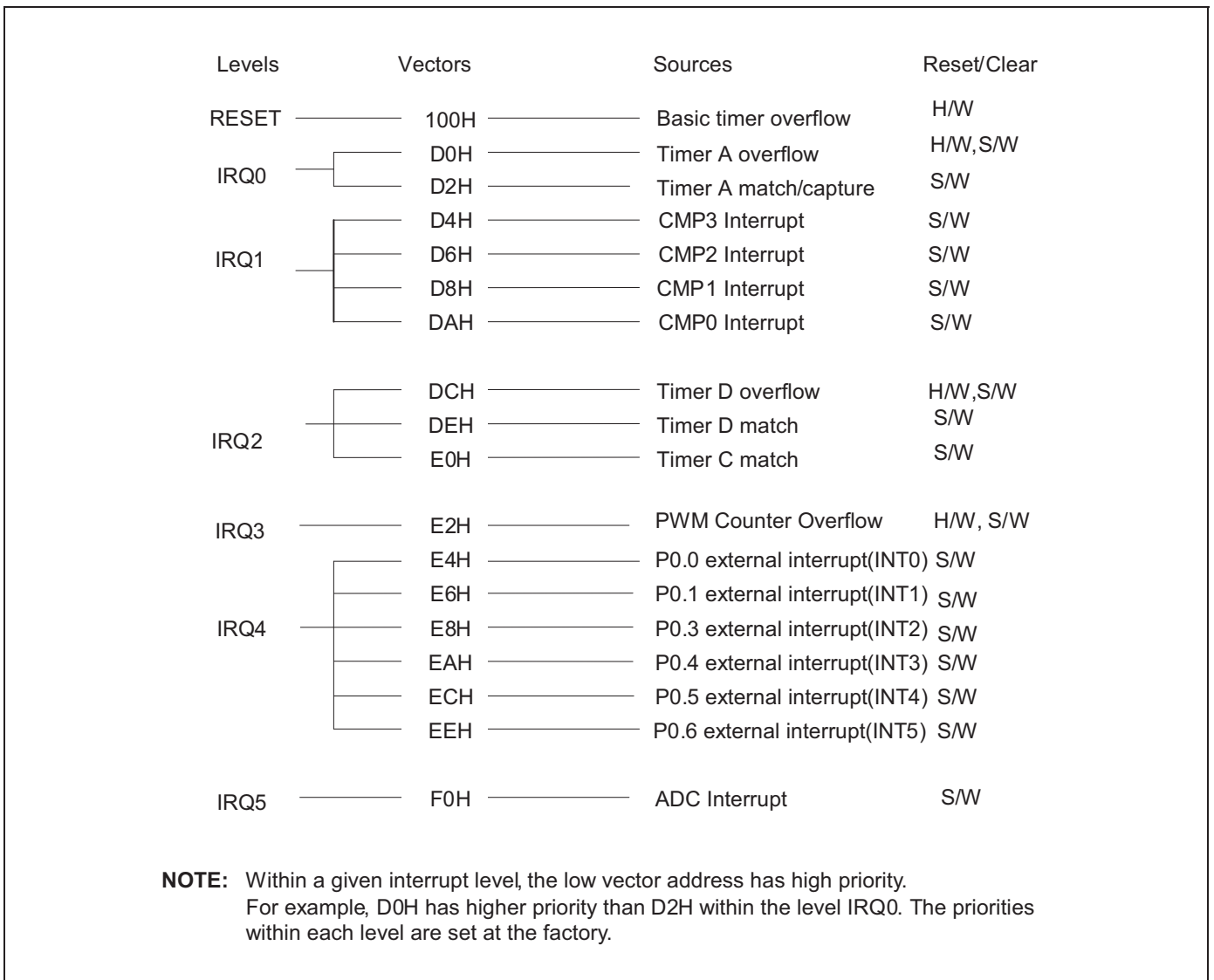


Figure 5-2 S3F84B8 Interrupt Structure

5.1.5.1 Interrupt Vector Addresses

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

You can allocate unused locations in the vector address area as normal program memory. However, do not overwrite any of the stored vector addresses.

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

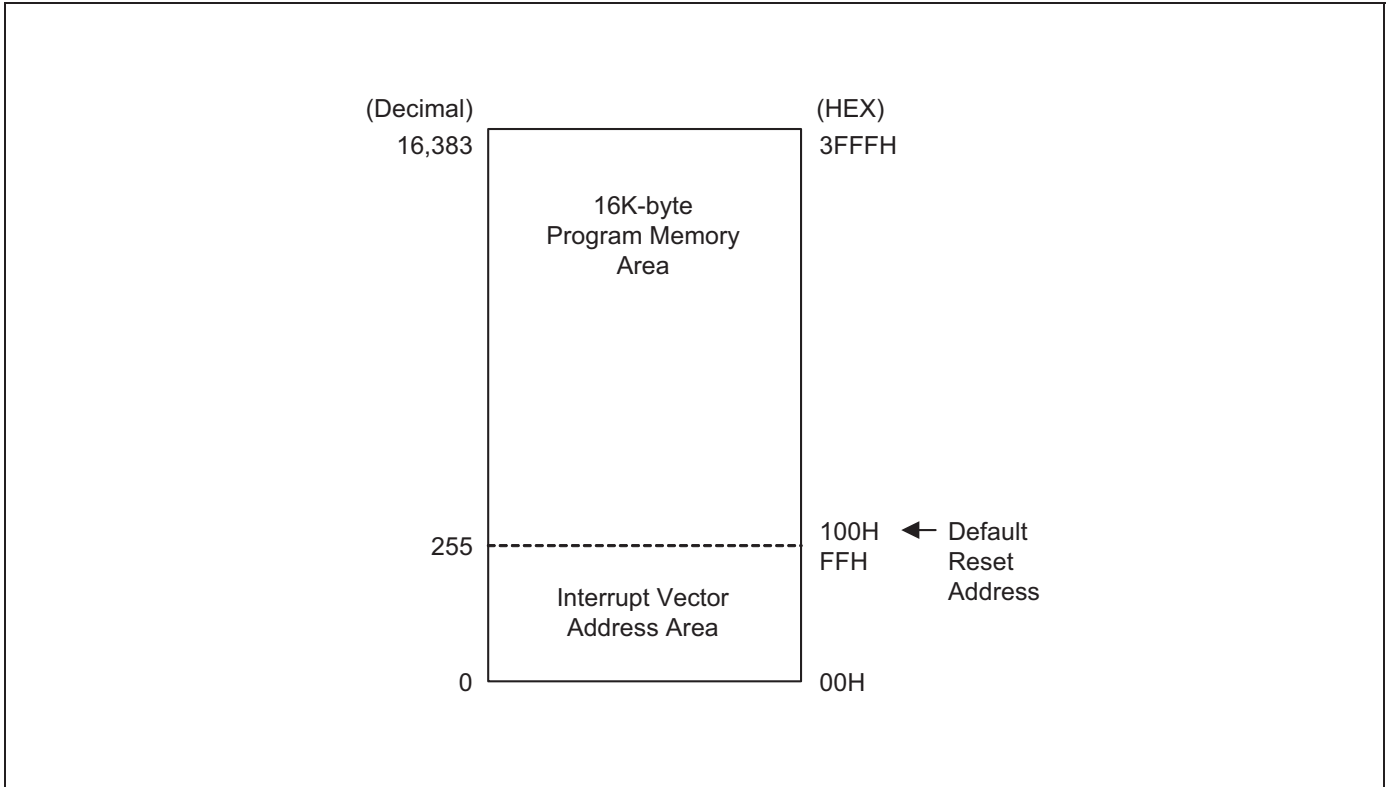


Figure 5-3 ROM Vector Address Area

5.1.5.2 Enable/Disable Interrupt Instructions (EI, DI)

Executing the Enable Interrupts (EI) instruction enables the interrupt structure globally. 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 Disable Interrupt (DI) instruction at any time to disable interrupt processing globally. The EI and DI instructions change the value of bit 0 in the SYM register.

5.1.6 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 the 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.

5.1.7 INTERRUPT PROCESSING CONTROL POINTS

Interrupt processing can 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, make sure to include the necessary register file address (register pointer) information.

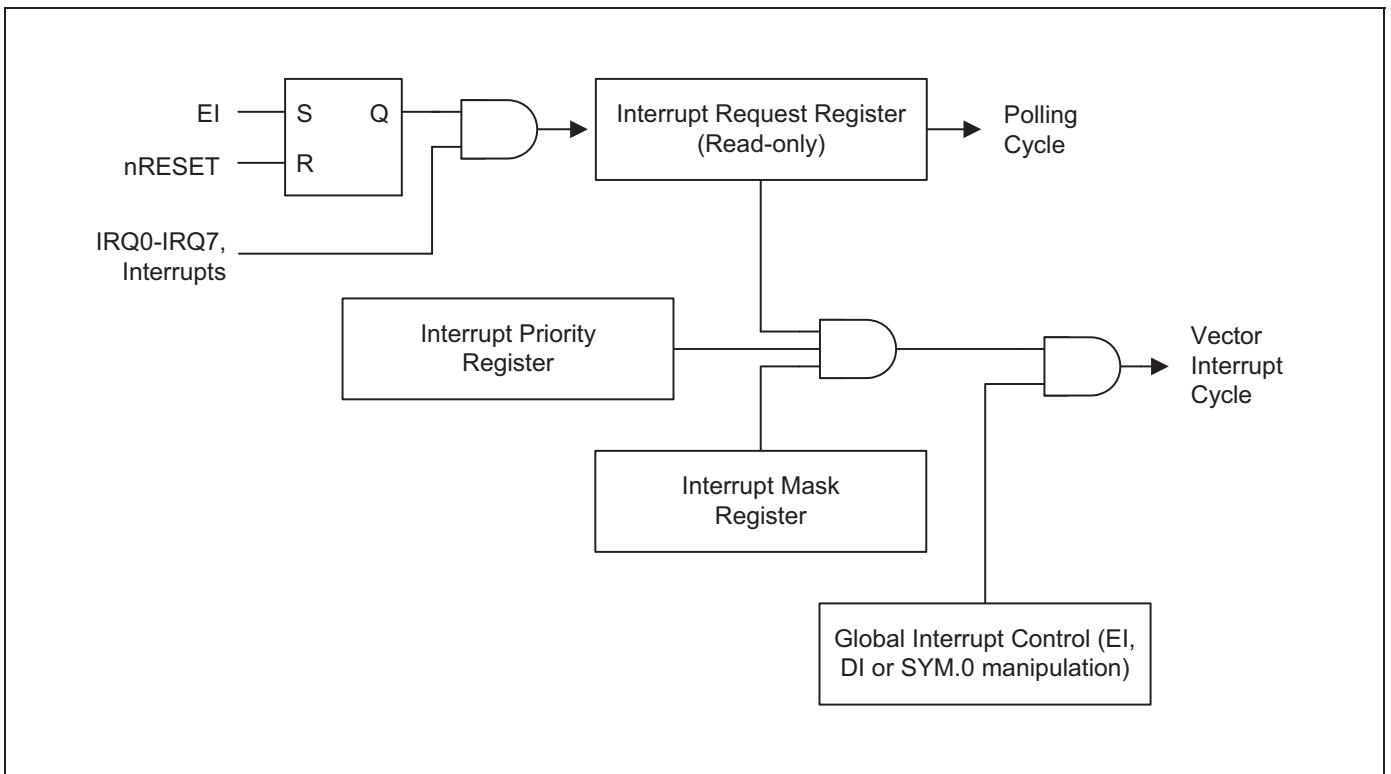


Figure 5-4 Interrupt Function Diagram

5.1.8 PERIPHERAL INTERRUPT CONTROL REGISTERS

For each interrupt source, there is one or more corresponding peripheral control register 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 corresponding interrupt.

5.1.9 SYSTEM MODE REGISTER (SYM)

The system mode register, SYM (DEH, Set1), is used to enable and disable interrupt processing globally 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 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.

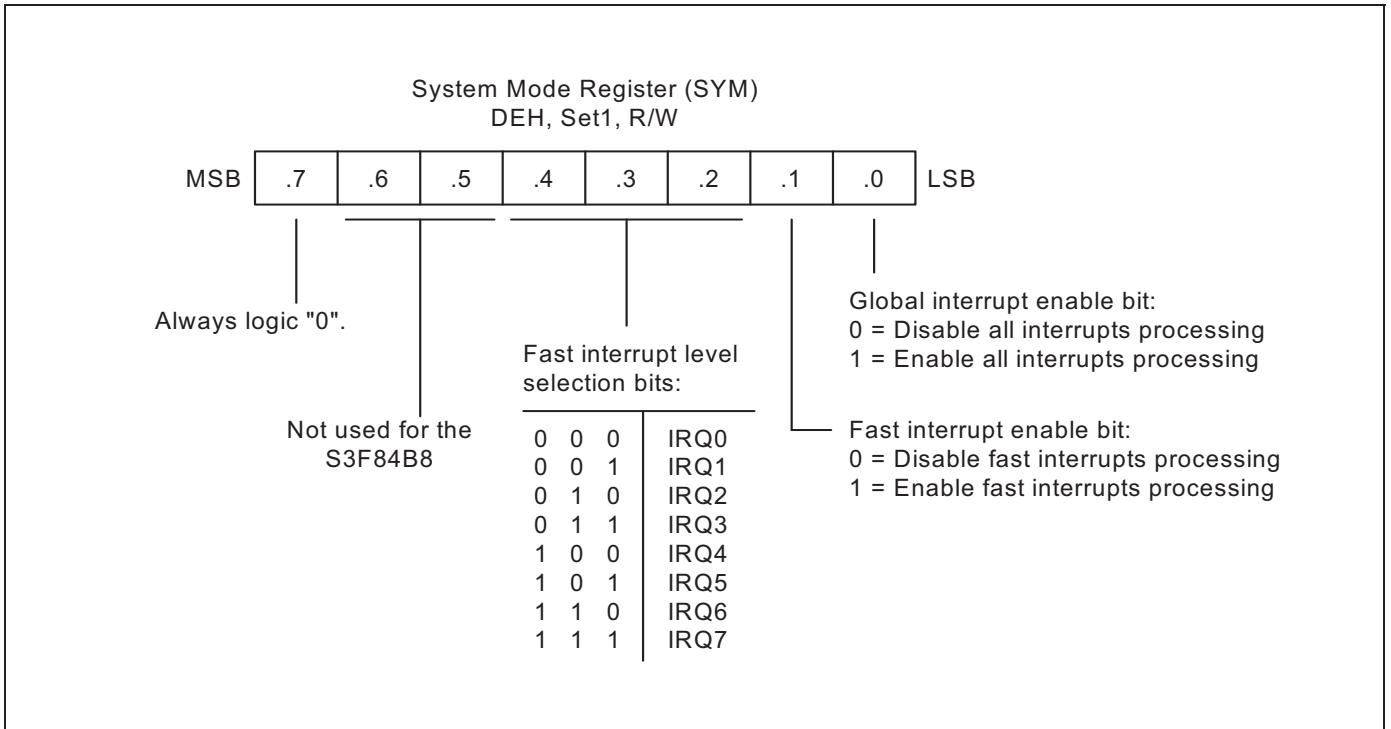


Figure 5-5 System Mode Register (SYM)

5.1.10 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 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.

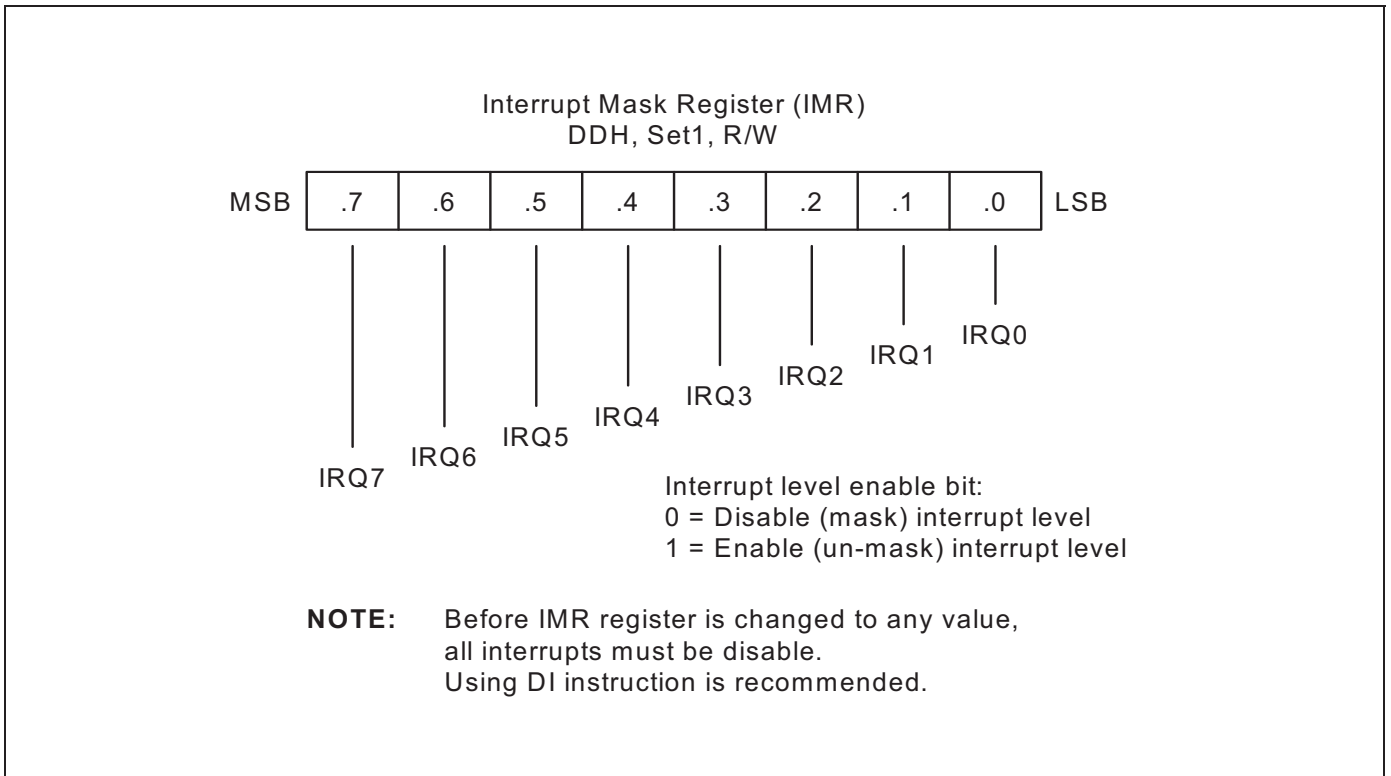


Figure 5-6 Interrupt Mask Register (IMR)

5.1.11 INTERRUPT PRIORITY REGISTER (IPR)

The interrupt priority register, IPR (FFH, Set1, Bank0), is used to set the relative priorities of interrupt levels in the microcontroller’s interrupt structure. After a reset, all IPR bit values are undetermined and must 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 lower vector address has priority (This priority is fixed in the hardware).

To support programming of the relative interrupt level priorities, they are organized into groups and subgroups by the interrupt logic.

NOTE: 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

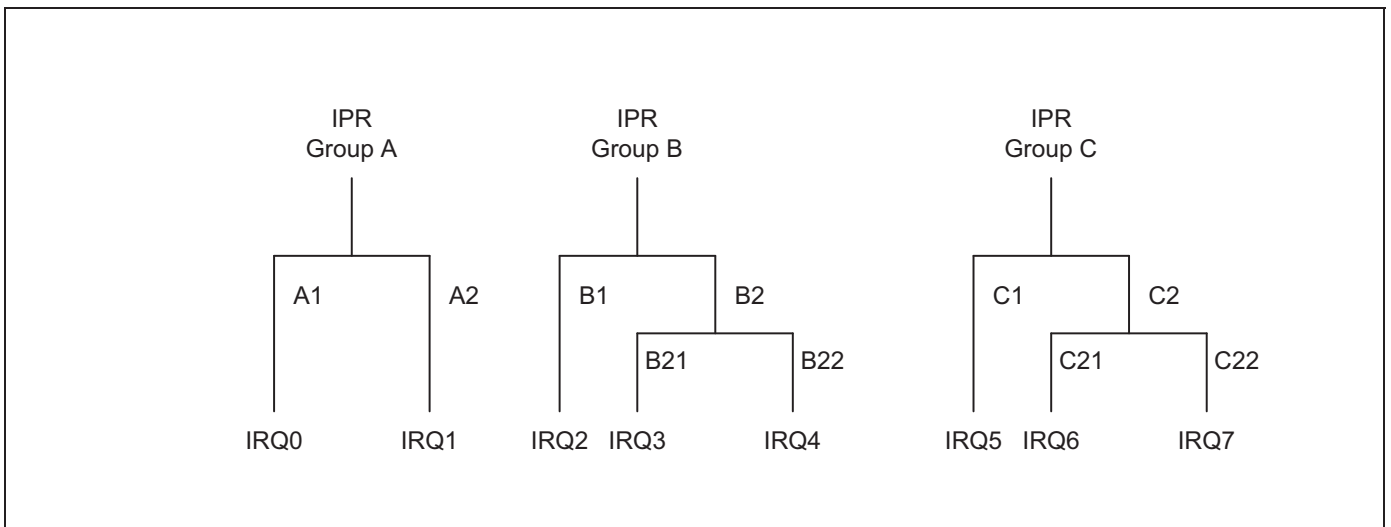


Figure 5-7 Interrupt Request Priority Groups

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 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.

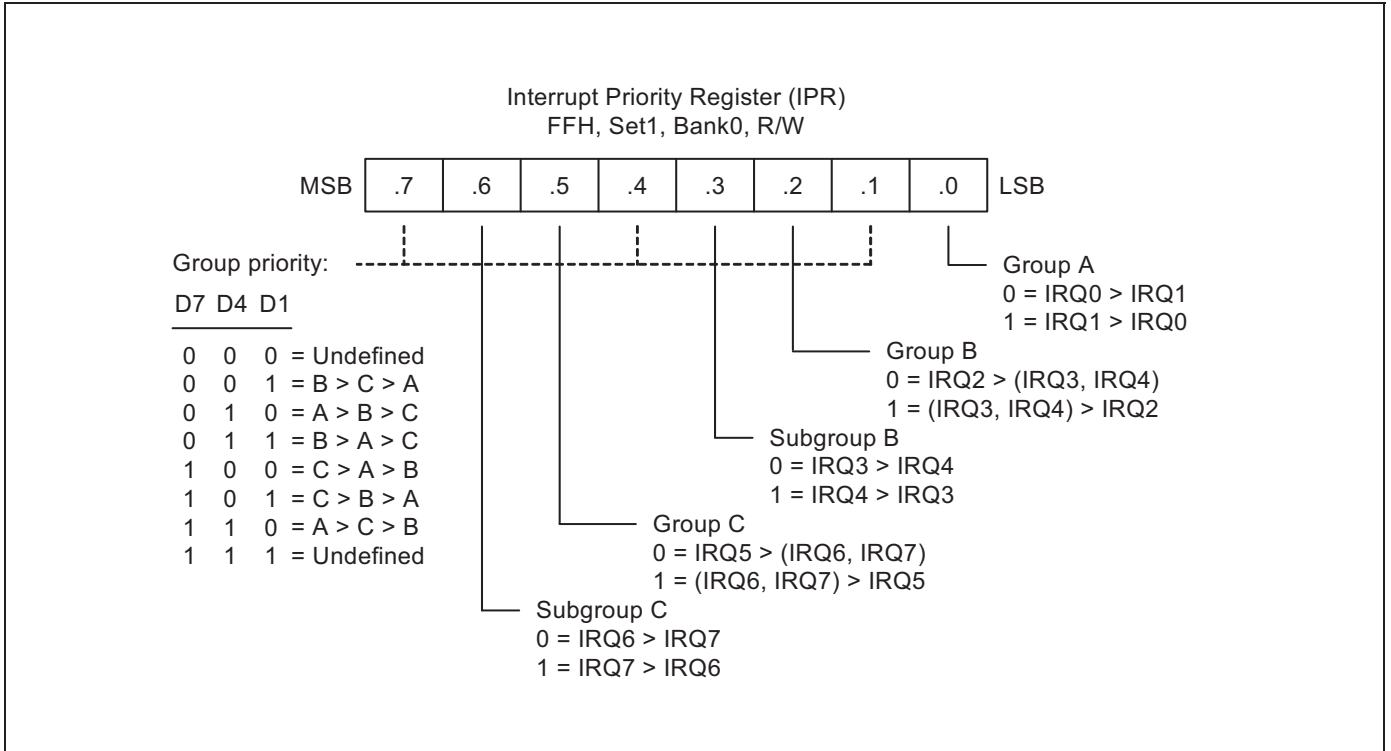


Figure 5-8 Interrupt Priority Register (IPR)

5.1.12 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 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. 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.

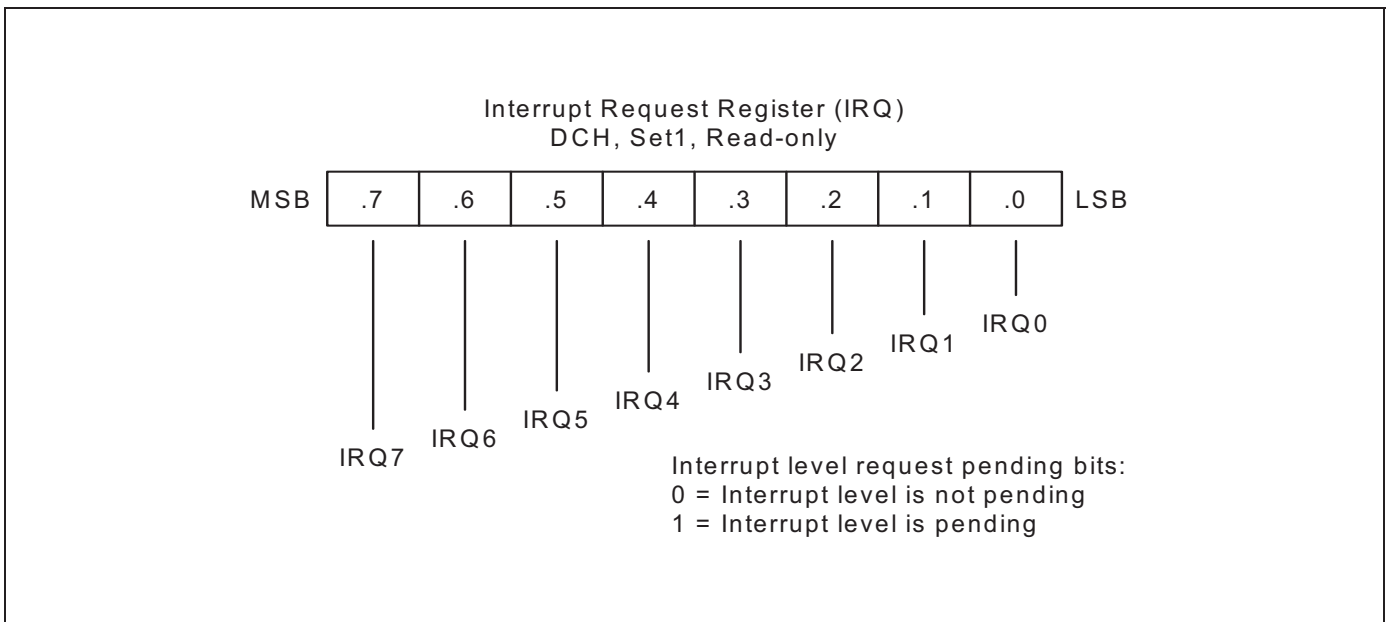


Figure 5-9 Interrupt Request Register (IRQ)

5.1.13 INTERRUPT PENDING FUNCTION TYPES

5.1.13.1 Overview of Interrupt Pending Function Types

There are two types of interrupt pending bits: one that is automatically cleared by the hardware after the interrupt service routine is acknowledged and executed and the other that must be cleared in the interrupt service routine.

5.1.13.2 Pending Bits Cleared Automatically by the Hardware

For interrupt pending bits that are cleared automatically by the 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 be read or written by the application software.

In S3F84B8 interrupt structure, TimerA, TimerD, and PWM counter overflow interrupts belong to this category of interrupts, where pending bits can be cleared automatically by the hardware.

5.1.13.3 Pending Bits Cleared by the Service Routine

The second type of pending bit is the one that should be cleared by the program software. The service routine must clear 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.

5.1.14 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 the hardware or software).
7. The CPU continues polling for interrupt requests.

5.1.15 INTERRUPT SERVICE ROUTINES

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

- Interrupt processing must be enabled globally (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 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 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 and sets SYM.0 to “1”. It allows the CPU to process the next interrupt request.

5.1.16 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.

5.1.17 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 and restore the IMR to its original value by returning the previous mask value from the stack (POP IMR).
5. Execute an IRET.

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

5.1.18 INSTRUCTION POINTER (IP)

The instruction pointer (IP) is adopted by all the 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).

5.1.19 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, write the appropriate 3-bit value to SYM.4–SYM.2. Thereafter, to enable fast interrupt processing for the selected level, set SYM.1 to “1”.

Two other system registers support fast interrupt processing:

- The instruction pointer (IP) contains the starting address of service routine (and is later used to swap the program counter values)
- When a fast interrupt occurs, the contents of 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.

5.1.20 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 “1” to the fast interrupt enable bit in the SYM register.

5.1.21 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.

5.1.22 RELATIONSHIP TO INTERRUPT PENDING BIT TYPES

As described previously, there are two types of interrupt pending bits: One type that is automatically cleared by the hardware after the interrupt service routine is acknowledged and executed and 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 the hardware or software.

5.1.23 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. For more information, refer to the Figure 6-4, “IRET instruction” in Chapter 6.

6 INSTRUCTION SET

6.1 OVERVIEW OF INSTRUCTION SET

The SAM8RC instruction set is specifically designed to support large register files that are typical of most SAM8 microcontrollers. The set contains 78 instructions.

6.1.1 KEY FEATURES OF INSTRUCTION SET

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 to the register file)
- Decimal adjustment is included in the binary-coded decimal (BCD) operations
- 16-bit (word) data can be incremented and decremented
- Flexible instructions for bit addressing, rotate, and shift operations

6.1.1.1 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. Additionally, bits within a byte are numbered from 7 to 0, where bit 0 is the least significant (right-most) bit.

6.1.1.2 Register Addressing

To access an individual register, an 8-bit address in the range of 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, refer to Chapter 2, "Address Spaces."

6.1.1.3 Addressing Modes

There are seven explicit addressing modes, namely, Register (R), Indirect Register (IR), Indexed (X), Direct (DA), Relative (RA), Immediate (IM), and Indirect (IA). For detailed descriptions of these addressing modes, 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)
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

Mnemonic	Operands	Instruction
COM	dst	Complement
OR	dst,src	Logical OR
XOR	dst,src	Logical exclusive OR
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
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		

Mnemonic	Operands	Instruction
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

6.2 FLAGS REGISTER (FLAGS)

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

The flags register (FLAGS) 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. It can be set or reset by instructions as long as its outcome does not affect 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 two write will occur simultaneously to the Flags register, producing an unpredictable result.

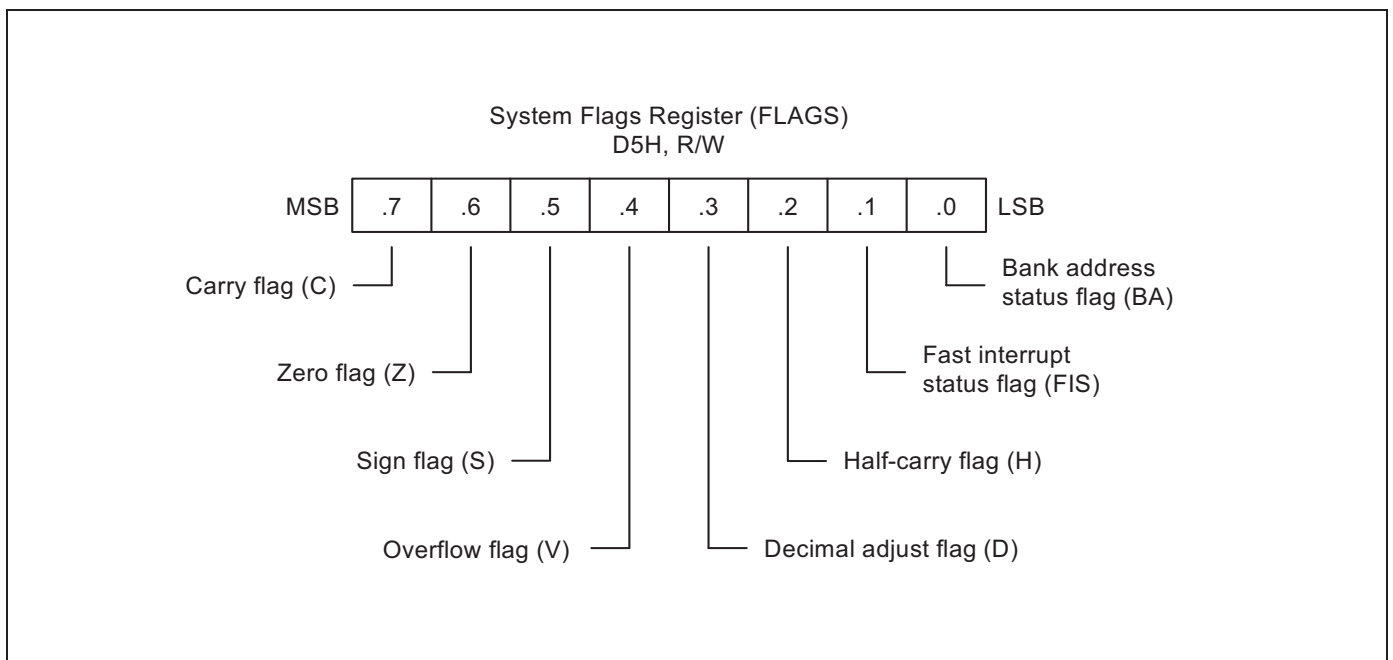


Figure 6-1 System Flags Register (FLAGS)

6.2.1 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 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 the register bits and for operations that require shift and rotate, 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 state of the MSB of the result. A logic zero indicates a positive number, while a logic one indicates a negative number.

V Overflow Flag (FLAGS.4)

The V flag is set to “1” if the result of a two’s-complement operation is greater than +127 or less than -128. It is cleared to “0” following logic operations such as ADD.

D Decimal Adjust Flag (FLAGS.3)

The DA bit is used to specify the last executed instruction during BCD operations, so that a subsequent decimal adjust operation can execute correctly. It 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” if an addition generates a carry-out of bit 3, or if 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 at the end of the interrupt service routine. Once set, it disables 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 set 1 area of internal register file is currently selected. In other words, it indicates whether bank 0 or bank 1 is selected. The BA flag is cleared to “0” (select bank 0) if the SB0 instruction is executed and is set to “1” (select bank 1) if the SB1 instruction is executed.

6.2.2 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,lr2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	BOR r0-Rb
P	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,lr2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	BCP r1.b, R2
P	2	INC R1	INC IR1	SUB r1,r2	SUB r1,lr2	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,lr2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	BTJR r2.b, RA
R	4	DA R1	DA IR1	OR r1,r2	OR r1,lr2	OR R2,R1	OR IR2,R1	OR R1,IM	LDB r0-Rb
	5	POP R1	POP IR1	AND r1,r2	AND r1,lr2	AND R2,R1	AND IR2,R1	AND R1,IM	BITC r1.b
N	6	COM R1	COM IR1	TCM r1,r2	TCM r1,lr2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	BAND r0-Rb
I	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,lr2	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,lr2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, lrr2, xL
E	B	CLR R1	CLR IR1	XOR r1,r2	XOR r1,lr2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, lrr2, xL
	C	RRC R1	RRC IR1	CPIJE lr,r2,RA	LDC r1,lrr2	LDW RR2,RR1	LDW IR2,RR1	LDW RR1,IML	LD r1, lr2
H	D	SRA R1	SRA IR1	CPIJNE lrr,r2,RA	LDC r2,lrr1	CALL IA1		LD IR1,IM	LD lr1, r2
E	E	RR R1	RR IR1	LDCD r1,lrr2	LDCI r1,lrr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, lrr2, xs
X	F	SWAP R1	SWAP IR1	LDCPD r2,lrr1	LDCPI r2,lrr1	CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, lrr1, xs
	-	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

OPCODE MAP									
LOWER NIBBLE (HEX)									
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

6.2.3 CONDITION CODES

The opcode of a conditional jump always contains a 4-bit field called the condition code (cc). This code specifies the conditions under which the jump is executed. For example, a conditional jump with the condition code for “equal” after a compare operation only jumps if the two operands are equal.

[Table 6-6](#) lists the condition codes. The carry (C), zero (Z), sign (S), and overflow (V) flags 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

NOTE:

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

6.3 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 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

6.3.1 ADC — ADD WITH CARRY

ADC dst,src

Operation: dst ← dst + src + c

The source operand, along with the carry flag, is added to the destination operand. The sum is stored in the destination. Contents of the source remain unaffected. Two's-complement addition is performed. In multiple precision arithmetic, this instruction permits the carry from addition of low-order operands 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	3	6	14	R	R
			6	15	R	IR
opc	dst	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, carry flag is set to "1", and source working register R2 contains the value 03H. The statement "ADC R1,R2" adds 03H and carry flag value ("1") to destination value 10H, leaving 14H in register R1.

6.3.2 ADD — ADD

ADD dst,src

Operation: dst ← dst + src

The source operand is added to the destination operand. Their sum is stored in the destination. The contents of source remain 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	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, and 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 the value 12H and source working register R2 contains the value 03H. The statement "ADD R1,R2" adds 03H to 12H, leaving the value 15H in register R1.

6.3.3 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. If the corresponding bits in two operands are both logic ones, AND operation results in a “1” bit being stored in Z bit of FLAG; otherwise a “0” bit value is stored. The contents of the source remain 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	src			
<table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst src</td> </tr> </table>	opc	dst src	2	4	52	r	r	
opc	dst src							
		6	53	r	lr			
<table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst	3	6	54	R	R
opc	src	dst						
		6	55	R	IR			
<table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src	3	6	56	R	IM
opc	dst	src						

Examples: Given R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, and 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 source working register R2 contains the value 03H. The statement “AND R1,R2” logically ANDs the source operand value 03H with the destination operand value 12H, leaving the value 02H in register R1.

6.3.4 BAND — BIT AND

BAND dst,src.b

BAND dst.b,src

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

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

6.3.5 BCP — BIT COMPARE

BCP dst,src.b

Operation: dst(0) – src(b)

The specified bit of source is compared to (subtracted from) bit zero (LSB) of destination. Zero flag is set if the bits are same; otherwise it is cleared. The contents of both operands remain 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 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 the 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). Since the bit values are not identical, the zero flag bit (Z) is cleared in the FLAGS register (0D5H).

6.3.6 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 there.

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 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. Since the result of complement is not “0”, the zero flag (Z) in FLAGS register (0D5H) is cleared.

6.3.7 BITR — BIT RESET

BITR dst.b

Operation: dst(b) ← 0

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

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst b 0	2	4	77	rb

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

Example: Given R1 = 07H:

BITR R1.1 → R1 = 05H

If the value of working register R1 is 07H (00000111B), the statement "BITR R1.1" clears bit one of the destination register R1, leaving the value 05H (00000101B).

6.3.8 BITS — BIT SET

BITS dst.b

Operation: dst(b) ← 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 dst
opc	dst b 1	2	4	77	rb

NOTE: In the second byte of 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 → R1 = 0FH

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

6.3.9 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 source (or destination) is logically ORed with bit zero (LSB) of destination (or source). The resulting bit value is stored in a specified bit of destination. No other bits of the destination are affected. The source remain 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	07	r0	Rb
opc	src b 1	dst	3	6	07	Rb	r0

NOTE: In the second byte of 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 contains 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 contains 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.

6.3.10 BTJRF — BIT TEST, JUMP RELATIVE ON FALSE

BTJRF dst,src.b

Operation: If src(b) is a “0”, then $PC \leftarrow PC + dst$.

The specified bit within source operand is tested. If the bit is “0”, the relative address is added to the program counter and control shifts 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)			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	src b 0	dst	3	10	37	RA	rb

NOTE: In the second byte of 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 the value of working register R1 is 07H (00000111B), the statement “BTJRF SKIP,R1.3” tests bit 3. Since R1.3 is “0”, the relative address is added to the PC, which jumps to the memory location pointed to by SKIP. (Note that the memory location must be within the allowed range of +127 to –128.)

6.3.11 BTJRT — BIT TEST, JUMP RELATIVE ON TRUE

BTJRT dst,src.b

Operation: If src(b) is a “1”, then $PC \leftarrow PC + dst$.

The specified bit within the source operand is tested. If the bit is “1”, the relative address is added to the program counter and control passes to the statement whose address is in the PC; otherwise, the instruction following the BTJRT instruction is executed.

Flags: No flags are affected.

Format:

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

NOTE: In the second byte of 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 the value of working register R1 is 07H (00000111B), the statement “BTJRT SKIP,R1.1” tests bit one in the source register (R1). Since the bit is “1”, the relative address is added to the PC, which then jumps to the memory location pointed to by SKIP. (Note that the memory location must be within the allowed range of + 127 to – 128.)

6.3.12 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 source (or destination) is logically exclusive-ORed with bit zero (LSB) of destination (or source). The result bit is stored in the specified bit of destination. No other bits of the destination are affected. The source remains 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	27	r0	Rb
opc	src b 1	dst	3	6	27	Rb	r0

NOTE: In the second byte of 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 remains unaffected.

6.3.13 CALL — CALL PROCEDURE

CALL dst

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

The current contents of program counter are pushed onto the top of stack. Here, the program counter value used specifies the address of first instruction following the CALL instruction. The specified destination address is then loaded into the program counter and it 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 stack back into the program counter.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode
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 (PC) value is 1A47H and the stack pointer contains the value 0002H, the statement “CALL 3521H” pushes the current PC value onto the top of stack. The stack pointer now points to memory location 0000H. PC is then loaded with the value 3521H, the address of first instruction in the program sequence to be executed.

If the contents of program counter and stack pointer are the same (as specified in the first example), the statement “CALL @RR0” produces the same result, except that 49H is stored in stack location 0001H (because the two-byte instruction format was used). PC is then loaded with the value 3521H, the address of first instruction in the program sequence to be executed.

Assuming the contents of program counter and stack pointer are the same (as specified 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 specified in the second example).

6.3.14 CCF — COMPLEMENT CARRY FLAG

CCF

Operation: $C \leftarrow \text{NOT } C$

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 is equal to “0”:

CCF

If the carry flag is equal to “0”, the CCF instruction complements it in the FLAGS register (0D5H), changing its value from logic zero to logic one.

6.3.15 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		
<table border="1" style="display: inline-table;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	dst		2	4	B0	R
	opc	dst					
			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.

6.3.16 COM — COMPLEMENT

COM dst

Operation: dst ← NOT dst

The contents of destination location are complemented (one’s complement); all “1’s” are changed to “0’s”, 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
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).

6.3.17 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 remain 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 lr	
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

The destination working register R1 contains the value 02H and source register R2 contains the value 03H. The statement “CP R1,R2” subtracts R2 value (source/subtrahend) from R1 value (destination/minuend). When “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, the destination working register R1 contains the value 05H, which is less than the contents of source working register R2 (0AH). The statement “CP R1,R2” generates C = “1” and the JP instruction does not jump to SKIP location. Once the statement “LD R3,R1” is executed, the value 06H remains in working register R3.

6.3.18 CPIJE — COMPARE, INCREMENT, AND JUMP ON EQUAL

CPIJE dst,src,RA

Operation: If $dst - src = "0"$, $PC \leftarrow PC + RA$
 $Ir \leftarrow Ir + 1$

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 is passed 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	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 contains the value 03H, and working register 03 contains the value 02H. The statement "CPIJE R1,@R2,SKIP" compares the @R2 value 02H (00000010B) to 02H (00000010B). Since the result of comparison is equal, the relative address is added to the PC. The PC then jumps to the memory location pointed to by SKIP. The source register (R2) is incremented by one, leaving value of 04H. (Note that the memory location must be within the allowed range of + 127 to - 128.)

6.3.19 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 is passed 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	lr

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

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

6.3.20 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
ADD ADC	0	0–9	0	0–9	00	0
	0	0–8	0	A–F	06	0
	0	0–9	1	0–3	06	0
	0	A–F	0	0–9	60	1
	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
SUB SBC	1	0–3	1	0–3	66	1
	0	0–9	0	0–9	00 = – 00	0
	0	0–8	1	6–F	FA = – 06	0
	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:

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

6.3.21 DA — DECIMAL ADJUST (CONTINUED)

DA

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

6.3.22 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 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	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.

6.3.23 DECW — DECREMENT WORD

DECW dst

Operation: dst ← dst – 1

The contents of destination location (which must be an even address) and the operand following that location are treated as a single 16-bit value 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 contains 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, it is recommend that you use DECW, as shown in the following example:

```
LOOP: DECW RR0
      LD   R2,R1
      OR  R2,R0
      JR  NZ,LOOP
```

6.3.24 DI — DISABLE INTERRUPTS

DI

Operation: SYM (0) ← 0

Bit zero of the system mode control register, SYM.0, is cleared to “0”, globally disabling all interrupt processing. Interrupt requests will continue to set their respective interrupt pending bits, but the CPU will not service them if 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 SYM register is 01H, statement “DI” leaves the new value 00H in register and clears SYM.0 to “0”, disabling interrupt processing.

Before changing IMR, interrupt pending, and interrupt source control register, ensure that all interrupts are disabled.

6.3.25 DIV — DIVIDE (UNSIGNED)

DIV dst,src

Operation: dst ÷ src

dst (UPPER) ← REMAINDER
 dst (LOWER) ← QUOTIENT

Destination operand (16-bits) is divided by source operand (8-bits). The quotient (8-bits) is stored in the lower half of destination, while the remainder (8-bits) is stored in the upper half of destination. When the quotient is ≥ 28 , the numbers stored in the upper and lower halves of 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 the divisor or quotient = "0"; cleared otherwise.
 - S:** Set if the MSB of quotient = "1"; cleared otherwise.
 - V:** Set if the quotient is $\geq 2^8$ or if divisor = "0"; cleared otherwise.
 - D:** Unaffected.
 - H:** Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 5px;">opc</td> <td style="padding: 5px;">src</td> <td style="padding: 5px;">dst</td> </tr> </table>	opc	src	dst			3	26/10	94	RR	R
	opc	src	dst							
					95	RR	IR			
				96	RR	IM				

NOTE: Execution takes 10 cycles if 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 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 destination register RR0 (R0) and the quotient in lower half of destination register (R1).

6.3.26 DJNZ — DECREMENT AND JUMP IF NON-ZERO

DJNZ r,dst

Operation: $r \leftarrow r - 1$

 If $r \neq 0$, $PC \leftarrow PC + dst$

The working register, which is used as a counter, is decremented. If the contents of register are not logic zero after decrementing, the relative address is added to the program counter and control is passed 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 as the address of instruction byte following the DJNZ statement.

NOTE: While using the DJNZ instruction, the working register (which is used as a counter) should be set at the one of the locations 0C0H to 0CFH with SRP, SRP0, or SRP1 instruction.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst			
<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; border-right: 1px solid black; padding: 2px 5px;">r</td> <td style="width: 33%; border-right: 1px solid black; padding: 2px 5px;">opc</td> <td style="padding: 2px 5px;">dst</td> </tr> </table>	r	opc	dst	2	8 (jump taken) 8 (no jump)	rA r = 0 to F	RA
r	opc	dst					

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

```
SRP    #0C0H
DJNZ   R1,LOOP
```

DJNZ controls 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 specifies the label for a relative address.

The statement “DJNZ R1, LOOP” decrements register R1 by one, leaving the value 01H. Since the contents of R1 after the decrement are non-zero, the jump is taken to the relative address specified by the LOOP label.

6.3.27 EI — ENABLE INTERRUPTS

EI

Operation: SYM (0) ← 1

An EI instruction sets the bit zero of system mode register, SYM.0, to “1”. This allows the interrupts to be serviced as they occur (assuming they have the 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)
<div style="border: 1px solid black; display: inline-block; padding: 2px 10px;">opc</div>	1	4	9F

Example: Given SYM = 00H:

EI

If the SYM register contains the value 00H, that is, if the 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.)

6.3.28 ENTER — ENTER

ENTER

Operation: SP ← SP - 2
 @SP ← IP
 IP ← PC
 PC ← @IP
 IP ← IP + 2

This instruction is useful while implementing threaded-code languages. The contents of 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: [Figure 6-2](#) shows an example of how to use an ENTER statement.

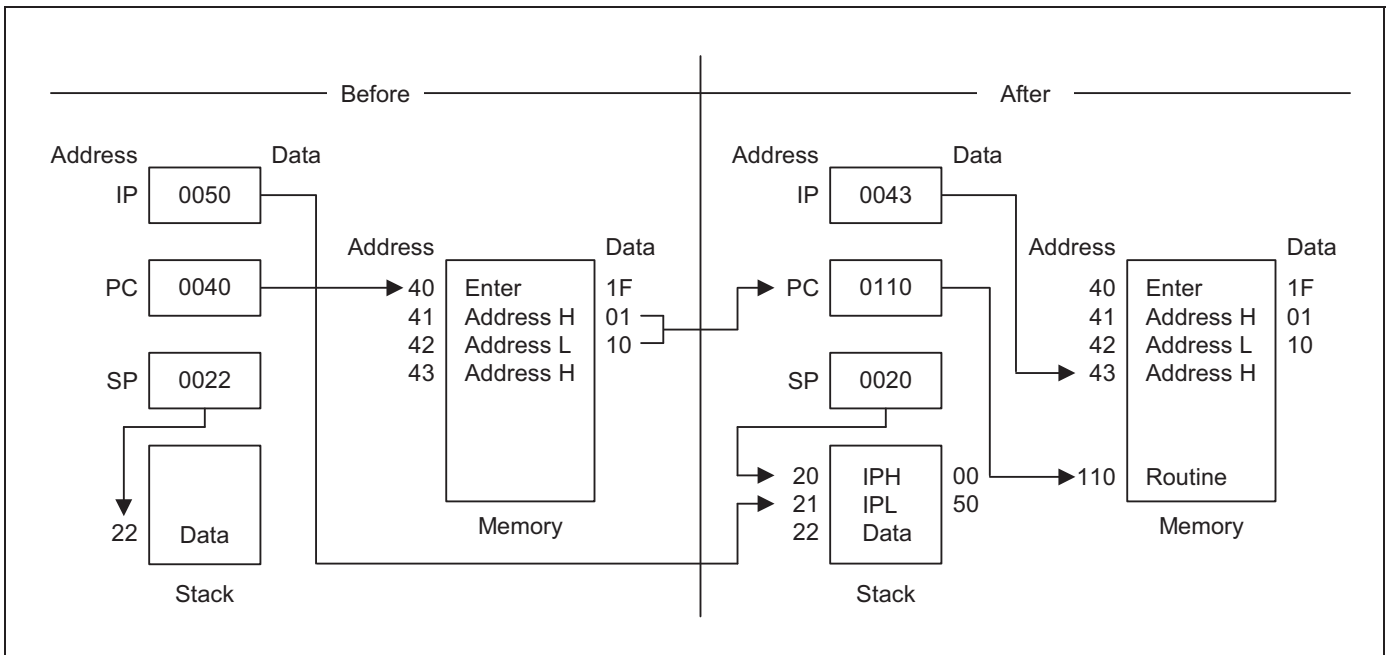


Figure 6-2 Example of the Usage of ENTER Statement

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) 16 (internal stack)	2F

Example: [Figure 6-3](#) shows an example of how to use an EXIT statement.

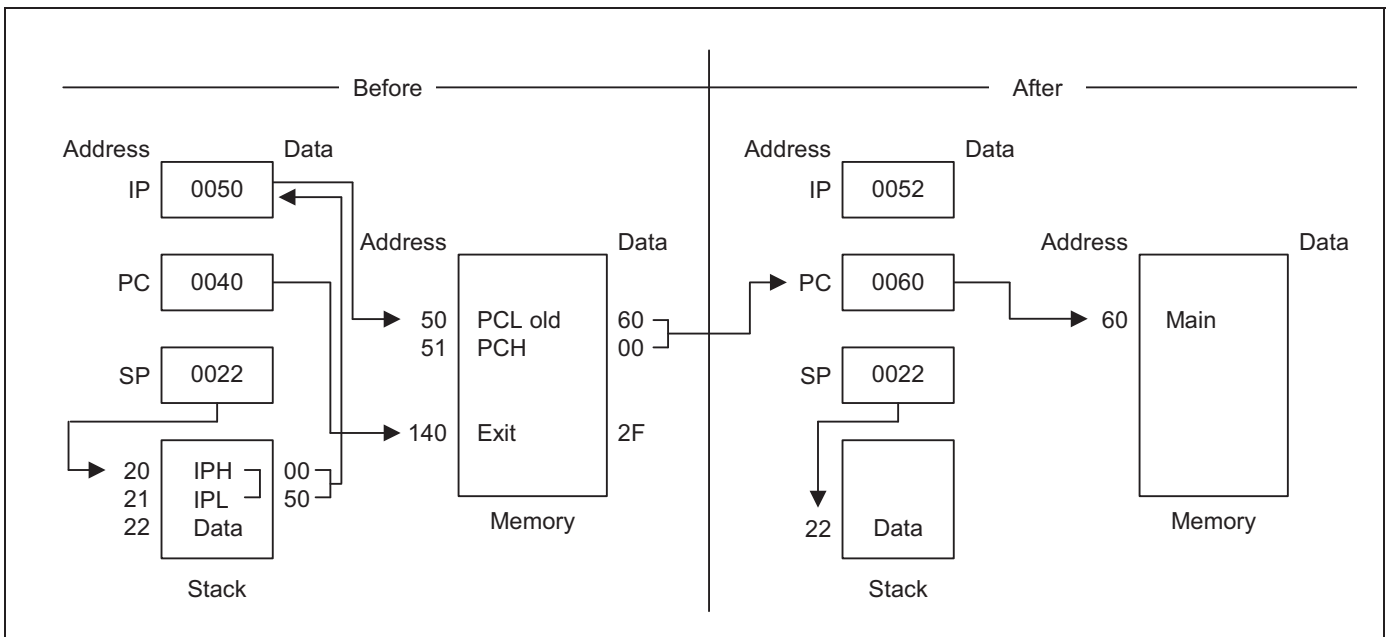


Figure 6-3 Example of the usage of EXIT statement

6.3.29 IDLE — IDLE OPERATION

IDLE

Operation:

The IDLE instruction stops the CPU clock while allowing the system clock oscillation to continue. Idle mode can be released by an interrupt request (IRQ) or an external reset operation. In application programs, an IDLE instruction must be immediately followed by at least three NOP instructions. This ensures an adequate time interval for the clock to stabilize before the next instruction is executed. If three or more NOP instructions 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	src
opc	1	4	6F	-	-

Example: The instruction

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

6.3.30 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
dst opc	1	4	rE r = 0 to F	r
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 the 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.

6.3.31 INCW — INCREMENT WORD

INCW dst

Operation: dst ← dst + 1

The contents of destination (containing an even address) and the byte following that location are treated as a single 16-bit value 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, it is recommend that you use INCW, as shown in the following example:

```
LOOP: INCW RR0
      LD R2,R1
      OR R2,R0
      JR NZ,LOOP
```


6.3.32 IRET — INTERRUPT RETURN

IRET	IRET (Normal)	IRET (Fast)
Operation:	$FLAGS \leftarrow @SP$ $SP \leftarrow SP + 1$ $PC \leftarrow @SP$ $SP \leftarrow SP + 2$ $SYM(0) \leftarrow 1$	$PC \leftrightarrow IP$ $FLAGS \leftarrow FLAGS'$ $FIS \leftarrow 0$

This instruction is used at the end of an interrupt service routine. It restores the flag register and program counter, and enables the global interrupt again. 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) 12 (internal stack)	BF
IRET (Fast)	Bytes	Cycles	Opcode (Hex)
opc	1	6	BF

Example: In [Figure 6-4](#), 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. Typically, the last instruction in service routine 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.

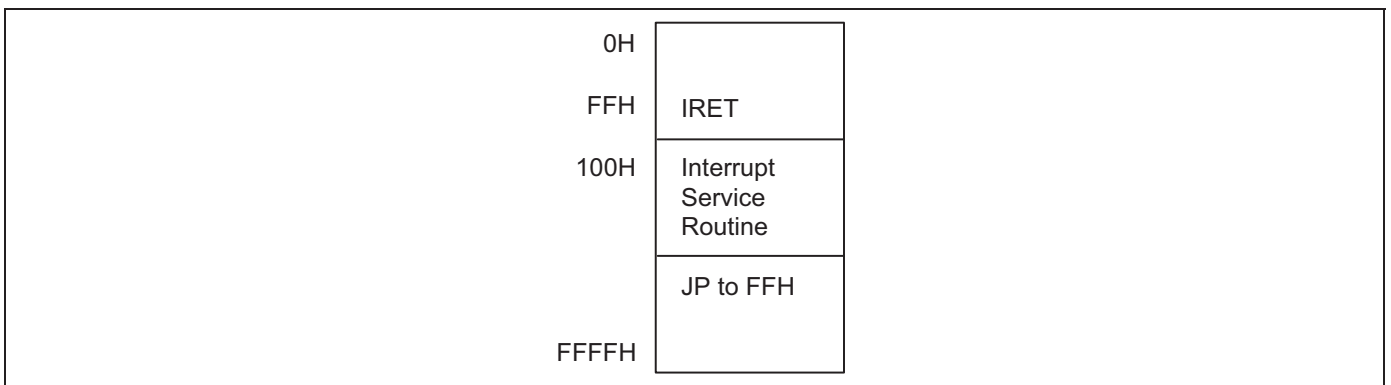


Figure 6-4 Fast interrupt Service Routine

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 last two instructions. The IRET cannot proceed immediately by clearing of the interrupt status (ditto with a reset of the IPR register).

6.3.33 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 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
(2)					
cc		opc	dst		
		3	8	ccD	DA
				cc = 0 to F	
opc		dst			
		2	8	30	IRR

NOTE:

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. If the carry flag was not set, control would 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 register pair 00H and 01H, leaving the value 0120H.

6.3.34 JR — JUMP RELATIVE

JR cc,dst

Operation: If cc is true, PC ← PC + dst.

If the condition specified by condition code (cc) is true, then relative address is added to the program counter and control is passed 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 relative address is from +127 to –128. The original value of the program counter is the address of first instruction byte following the JR statement.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst					
(NOTE)									
<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">cc</td> <td style="padding: 2px 10px;"> </td> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;"> </td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	cc		opc		dst	2	6	ccB	RA
cc		opc		dst					
			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 will be executed.

6.3.35 LD — LOAD

LD dst,src

Operation: dst ← src

The contents of the source are loaded into the destination. The source contents remain unaffected.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode	dst	src
dst opc src			2	4	rC	r		IM
				4	r8	r		R
src opc dst			2	4	r9	R		r
opc dst src			2	4	C7	r		lr
				4	D7	lr		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

6.3.36 LD — LOAD (CONTINUED)

LD

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

6.3.37 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 source is loaded into bit zero (LSB) of destination, or bit zero of source is loaded into the specified bit of destination. No other bits of the destination are affected. The source remains unaffected.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	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 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 contains the value 05H. The statement “LD R0,00H.2” loads the bit two value of 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 destination register, leaving 04H in general register 00H.

6.3.38 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 remain unaffected. LDC refers to program memory, while LDE refers to data memory. The assembler makes 'lrr' or 'rr' values even for program memory and odd for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode	dst	src
1.	opc dst src	2	10	C3	r	lrr	
2.	opc src dst	2	10	D3	lrr		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

NOTE:

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 addresses 'XS [rr]' and source address 'XS [rr]' are one byte each.
3. For formats 5 and 6, the destination address 'XL [rr]' and source address 'XL [rr]' are two bytes each.
4. The DA and r source values for formats 7 and 8 address the program memory; the second set of values used in formats 9 and 10 address the data memory.

6.3.39 LDC/LDE — LOAD MEMORY (CONTINUED)

LDC/LDE

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.

6.3.40 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 memory location is specified by a working register pair. The contents of source location are loaded into the destination location, following which the memory address is decremented. The contents of the source remain unaffected.

LDCD references program memory and LDED references external data memory. The assembler makes 'lrr' 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	src
opc	dst src	2	10	E2	r	lrr

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

6.3.41 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 memory location is specified by a working register pair. The contents of source location are loaded into destination location, following which the memory address is incremented automatically. The contents of source remain unaffected.

LDCI refers to the program memory and LDEI refers to the external data memory. The assembler makes 'lrr' even for program memory and odd for data memory.

Flags: No flags are affected.

Format:

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

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

6.3.42 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 memory location is specified by a working register pair and is first decremented. After this step, the contents of source location are loaded into destination location. The contents of source remain unaffected.

LDCPD refers to program memory and LDEPD refers to external data memory. The assembler makes 'lrr' 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	src
opc	src dst	2	14	F2	lrr	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
```

6.3.43 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 to the register file. The address of memory location is specified by a working register pair and is first incremented. After this step, the contents of source location are loaded into the destination location. The contents of the source remain unaffected.

LDCPI refers to program memory and LDEPI refers to external data memory. The assembler makes 'lrr' 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	src
opc	src dst	2	14	F3	lrr	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
```

6.3.44 LDW — LOAD WORD

LDW dst,src

Operation: dst ← src

The contents of source (word) are loaded into the destination. The contents of source remain unaffected.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode	
						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, note that the statement “LDW 00H,02H” loads the contents of 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.

Other examples show how to use the LDW instruction with various addressing modes and formats.

6.3.45 MULT — MULTIPLY (UNSIGNED)

MULT dst,src

Operation: dst ← dst × src

The 8-bit destination operand (even register of register pair) is multiplied by the source operand (8-bits), and the product (16-bits) is stored in register pair specified by destination address. Both operands are treated as unsigned integers.

- Flags:**
- C:** Set if the result is > 255; cleared otherwise.
 - Z:** Set if the result is “0”; cleared otherwise.
 - S:** Set if the MSB of result is “1”; cleared otherwise.
 - V:** Cleared.
 - D:** Unaffected.
 - H:** Unaffected.

Format:

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

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

- MULT 00H, 02H → Register 00H = 01H, register 01H = 20H, register 02H = 09H
- MULT 00H, @01H → Register 00H = 00H, register 01H = 0C0H
- MULT 00H, #30H → Register 00H = 06H, register 01H = 00H

In the first example, the statement “MULT 00H,02H” multiplies 8-bit destination operand (in the register 00H of register pair 00H, 01H) by the source register 02H operand (09H). The 16-bit product, 0120H, is stored in the register pair 00H, 01H.

6.3.46 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: [Figure 6-5](#) shows an example about how to use the NEXT instruction.

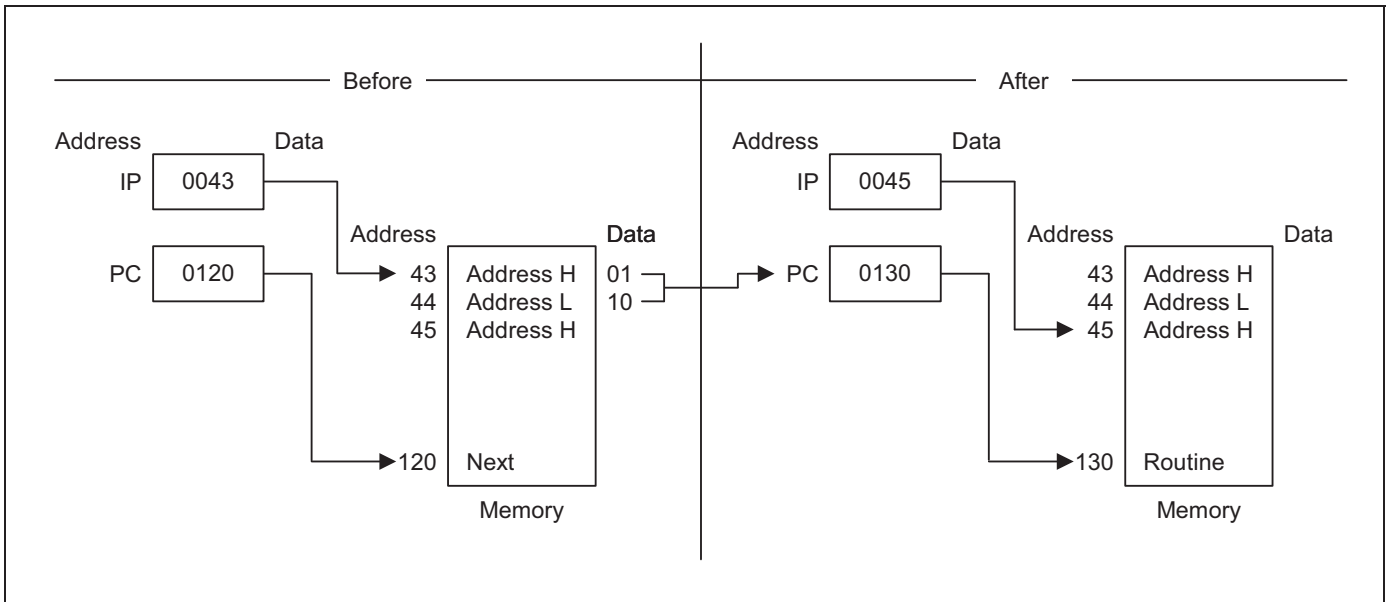


Figure 6-5 Example of the Usage of the NEXT Instruction

6.3.47 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 affect a timing delay of variable duration.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	
<table border="1"><tr><td>opc</td></tr></table>	opc	1	4	FF
opc				

Example: When the instruction NOP is encountered in a program, no operation occurs. Instead, there is a delay in instruction execution time.

6.3.48 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 destination. The contents of source remain unaffected. The OR operation results in a “1” being stored whenever either of the corresponding bits in 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	src
opc	dst src		2	4	42	r	r	
				6	43	r	lr	
opc	src	dst	3	6	44	R	R	
					6	45	R	IR
opc	dst	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 contains 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.

Other examples show the use of logical OR instruction with the various addressing modes and formats.

6.3.49 POP — POP FROM STACK

POP dst

Operation: dst ← @SP
 SP ← SP + 1

The contents of location addressed by the stack pointer are loaded into destination. The stack pointer is then incremented by one.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode		
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 5px;">opc</td> <td style="padding: 5px;">dst</td> </tr> </table>	opc	dst		2	8	50	R
	opc	dst					
			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 contains the value 55H and SP points to location 00FCH.

6.3.50 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 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:

POPUD02H,@00H → Register 00H = 41H, register 02H = 6FH, register 42H = 6FH

If general register 00H contains the value 42H and register 42H contains the value 6FH, the statement “POPUD 02H,@00H” loads the contents of register 42H into destination register 02H. The user stack pointer is then decremented by one, leaving the value 41H.

6.3.51 POPUI — POP USER STACK (INCREMENTING)

POPUI dst,src

Operation: dst ← src

IR ← IR + 1

The POPUI instruction is used for user-defined stacks in register file. The contents of 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	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 contains 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.

6.3.52 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 source (src) into the location addressed by the decremented stack pointer. The operation then adds new value to the top of stack.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode
opc	src	2	8 (internal clock)	70	R
			8 (external clock)		
			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 contains 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 stack.

6.3.53 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 source into register addressed by the decremented stack pointer.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	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.

6.3.54 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 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	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.

6.3.55 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)
opc	1	4	CF

Example: Given C = “1” or “0”:

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

6.3.56 RET — RETURN

RET

Operation: PC ← @SP
 SP ← SP + 2

Typically, the RET instruction is used to return to the previously executed procedure at the end of a procedure entered by a CALL instruction. The contents of 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) 10 (internal stack)	AF

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 program counter. The stack pointer then pops the value in location 00FEH (1AH) to the PC’s low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location 00FEH.

6.3.57 RL — ROTATE LEFT

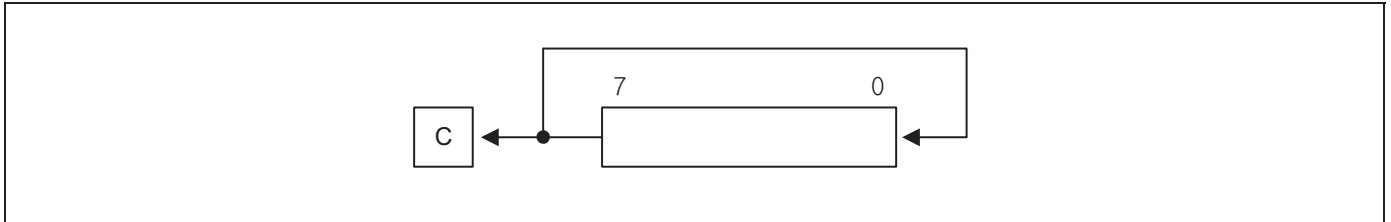
RL dst

Operation: C ← dst (7)

dst (0) ← dst (7)

dst (n + 1) ← dst (n), n = 0–6

The contents of destination operand are rotated left one bit position. The initial value of bit 7 is moved to bit zero (LSB) position. It 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 dst
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.

6.3.58 RLC — ROTATE LEFT THROUGH CARRY

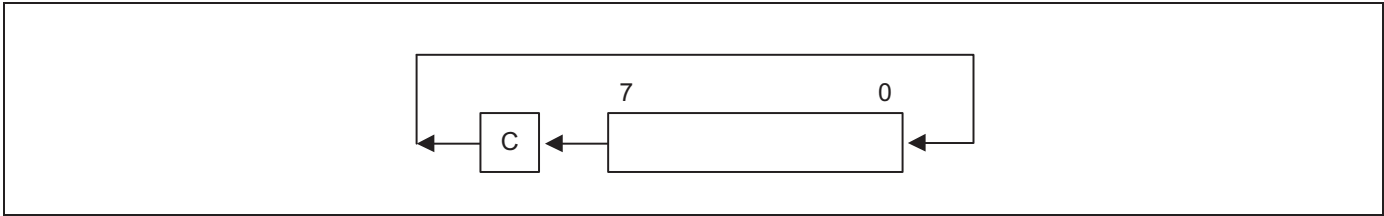
RLC dst

Operation: dst (0) ← C

C ← dst (7)

dst (n + 1) ← dst (n), n = 0–6

The contents of destination operand with the carry flag are rotated left one bit position. The initial value of bit 7 replaces the carry flag (C), while the initial value of 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
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 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.

6.3.59 RR — ROTATE RIGHT

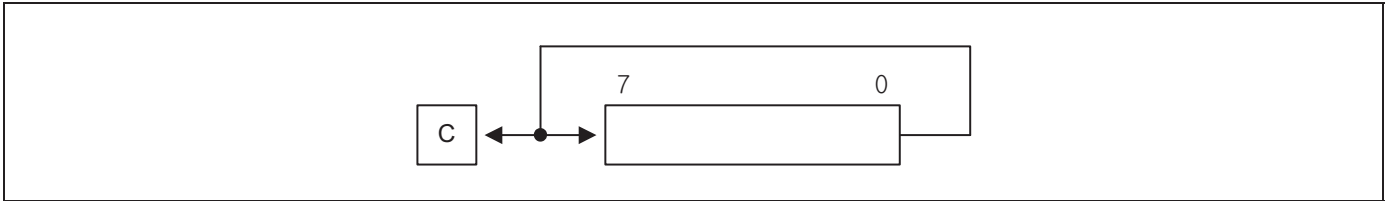
RR dst

Operation: C ← dst (0)

dst (7) ← dst (0)

dst (n) ← dst (n + 1), n = 0–6

The contents of destination operand are rotated right one bit position. The initial value of bit zero (LSB) is moved to bit 7 (MSB). It replaces the carry flag (C).



Flags: **C:** Set if the bit rotated from least significant bit position (bit zero) is “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 destination changed during rotation; cleared otherwise.

D: Unaffected.

H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode
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 destination register. The initial bit zero also resets the C flag to “1”, and sign flag and overflow flag are set to “1”.

6.3.60 RRC — ROTATE RIGHT THROUGH CARRY

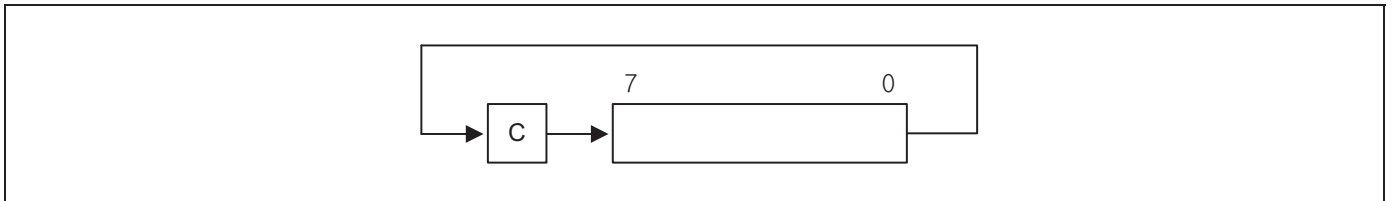
RRC dst

Operation: dst (7) ← C

C ← dst (0)

dst (n) ← dst (n + 1), n = 0–6

The contents of destination operand and carry flag are rotated right one bit position. The initial value of bit zero (LSB) replaces the carry flag; the initial value of carry flag replaces bit 7 (MSB).



Flags: **C:** Set if the bit rotated from the least significant bit position (bit zero) is “1”.

Z: Set if the result is “0”; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Set if the arithmetic overflow occurred, that is, if the sign of destination changes during rotation; cleared otherwise.

D: Unaffected.

H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode
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 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”.

6.3.61 SB0 — SELECT BANK 0

SB0

Operation: BANK ← 0

The SB0 instruction clears the bank address flag in FLAGS register (FLAGS.0) to logic zero and selects bank 0 register addressing in set 1 area of the register file.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	4F

Example: The statement SB0 clears FLAGS.0 to “0” and selects bank 0 register addressing.

6.3.62 SB1 — SELECT BANK 1

SB1

Operation: BANK ← 1

The SB1 instruction sets the bank address flag in FLAGS register (FLAGS.0) to logic one and selects bank 1 register addressing in set 1 area of the register file. (Bank 1 is not implemented in some S3F8-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” and selects bank 1 register addressing, if implemented.

6.3.63 SBC — SUBTRACT WITH CARRY

SBC dst,src

Operation: $dst \leftarrow dst - src - c$

The source operand, along with the current value of carry flag, is subtracted from the destination operand. The result is stored in destination. The contents of source remain unaffected. Subtraction is performed by adding the two's-complement of source operand to destination operand. In multiple precision arithmetic, this instruction allows the carry ("borrow") from subtraction of low-order operands to be subtracted from 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 result is same as the sign of source; cleared otherwise.

D: Always set to "1".

H: Cleared if there is a carry from the most significant bit of low-order four bits of the result; set otherwise, indicating a "borrow".

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode	
					dst	src
opc	dst src	2	4	32	r	r
			6	33	r	lr
opc	src	3	6	34	R	R
			6	35	R	IR
opc	dst	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 contains 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.

6.3.64 SCF — SET CARRY FLAG

SCF

Operation: $C \leftarrow 1$

The carry flag (C) is set to logic one, regardless of its previous value.

Flags: **C:** Set to "1".

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	DF

Example: The statement SCF sets the carry flag to logic one.

6.3.65 SRA — SHIFT RIGHT ARITHMETIC

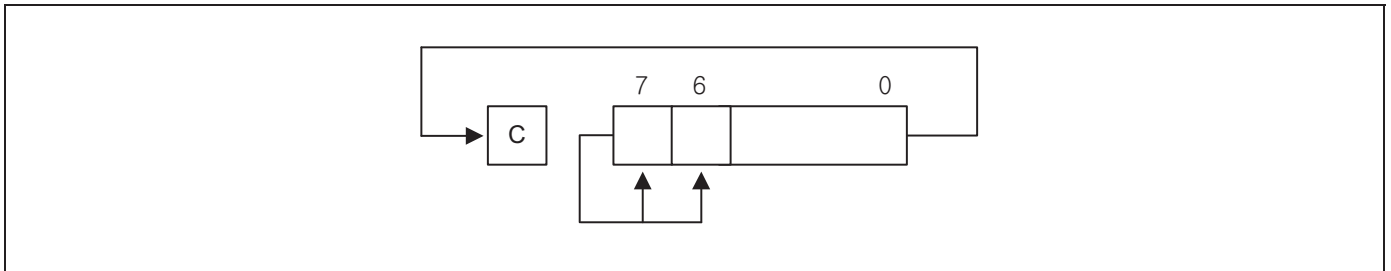
SRA dst

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) is “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 to bit 6 position (bit 7 remains unchanged). This leaves the value 0CDH (11001101B) in destination register 00H.

6.3.66 RP/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, except when 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 src
opc	2	4	31	IM
src				

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.

6.3.67 STOP — STOP OPERATION

STOP

Operation:

The STOP instruction stops both the CPU clock and system clock and causes the microcontroller to enter the Stop mode. In the 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 adequate time interval for the clock to stabilize before the next instruction is executed. If three or more NOP instructions are not used after STOP instruction, leakage current will not flow because of floating state in the internal bus.

Flags: No flags are affected.

Format:

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

Example:

The statement

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

6.3.68 SUB — SUBTRACT

SUB dst,src

Operation: dst ← dst – src

Once source operand is subtracted from destination operand, the result is stored in destination. The contents of source remain unaffected. Subtraction is performed by adding the two's complement of source operand to 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 result is same as the sign of source operand; cleared otherwise.
 - D:** Always set to “1”.
 - H:** Cleared if there is a carry from the most significant bit of low-order four bits of result; else set to indicate a “borrow”.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode	
						dst	src
opc	dst	src	2	4	22	r	r
				6	23	r	lr
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”

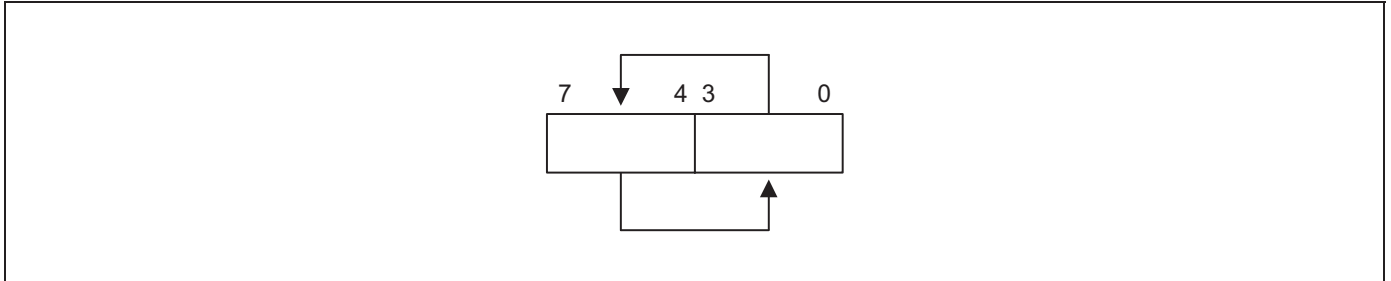
In the first example, if working register R1 contains the value 12H and if register R2 contains the value 03H, the statement “SUB R1,R2” subtracts source value (03H) from destination value (12H) and stores the result (0FH) in destination register R1.

6.3.69 SWAP — SWAP NIBBLES

SWAP dst

Operation: dst (0 – 3) ↔ dst (4 – 7)

The contents of 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 dst
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).

6.3.70 TCM — TEST COMPLEMENT UNDER MASK

TCM dst,src

Operation: (NOT dst) AND src

This instruction tests selected bits in destination operand for a logic one value. The bits to be tested are specified by setting a “1” bit in the corresponding position of source operand (mask). The TCM statement complements destination operand, which is then ANDed with source mask. The zero (Z) flag can then be checked to determine the result. The destination and source operands remain 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	src
opc	dst src	2	4	62	r	r
			6	63	r	lr
opc	src	3	6	64	R	R
			6	65	R	IR
opc	dst	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 contains the value 02H (00000010B), the statement “TCM R0,R1” tests bit one in the destination register for a “1” value. Since 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 TCM operation.

6.3.71 TM — TEST UNDER MASK

TM dst,src

Operation: dst AND src

This instruction tests selected bits in destination operand for logic zero value. The bits to be tested are specified by setting a “1” bit in the corresponding position of source operand (mask), which is ANDed with destination operand. The zero (Z) flag can then be checked to determine the result. The destination and source operands remain 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
<div style="border: 1px solid black; padding: 2px; display: inline-block; margin-right: 5px;">opc</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">dst src</div>		2	4	72		r	r
			6	73		r	lr
<div style="border: 1px solid black; padding: 2px; display: inline-block; margin-right: 5px;">opc</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-right: 5px;">src</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">dst</div>		3	6	74		R	R
			6	75		R	IR
<div style="border: 1px solid black; padding: 2px; display: inline-block; margin-right: 5px;">opc</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-right: 5px;">dst</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">src</div>		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 contains the value 02H (00000010B), the statement “TM R0,R1” tests bit one in the destination register for a “0” value. Since 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 TM operation.

6.3.72 WFI — WAIT FOR INTERRUPT

WFI

Operation:

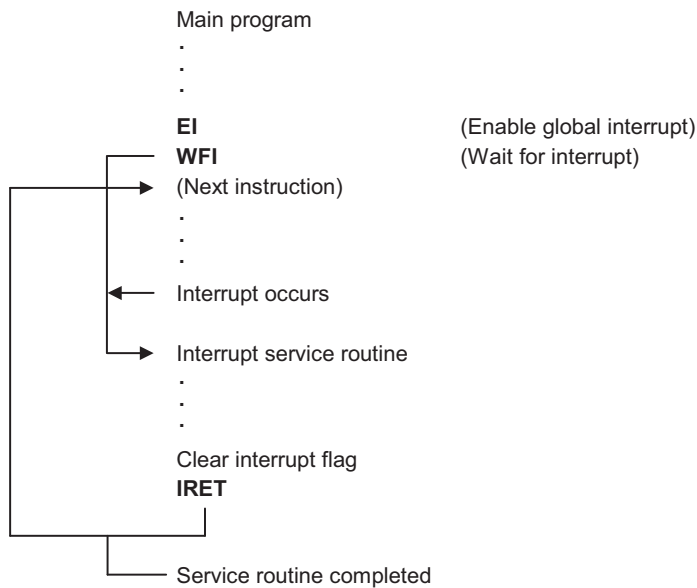
The CPU is halted until an interrupt occurs; even though DMA transfers can still take place during the wait state. The WFI status can be released by an internal interrupt, including a fast interrupt.

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:



6.3.73 XOR — LOGICAL EXCLUSIVE OR

XOR dst,src

Operation: dst ← dst XOR src

Source operand is logically exclusive-ORed with destination operand. The result is stored in 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	src			
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30px; text-align: center;">opc</td> <td style="width: 60px; text-align: center;">dst src</td> </tr> </table>	opc	dst src			2	4	B2	r	r	
	opc	dst src								
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30px; text-align: center;">opc</td> <td style="width: 30px; text-align: center;">src</td> <td style="width: 30px; text-align: center;">dst</td> </tr> </table>	opc	src	dst			3	6	B4	R	R
	opc	src	dst							
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30px; text-align: center;">opc</td> <td style="width: 30px; text-align: center;">dst</td> <td style="width: 30px; text-align: center;">src</td> </tr> </table>	opc	dst	src			3	6	B6	R	IM
	opc	dst	src							

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 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.

7

CLOCK CIRCUIT

7.1 OVERVIEW OF CLOCK CIRCUIT

Using the Smart option (3FH.1–.0 in ROM), you can select the internal RC oscillator, external RC oscillator, or external oscillator. In internal oscillator, X_{IN} (P0.0) and X_{OUT} (P0.1) can be used by normal I/O pins.

An internal RC oscillator can provide a typical frequency of 8MHz or 0.5MHz for S3F84B8, depending on the Smart option. On the other hand, an external RC oscillator can provide a typical frequency of 8MHz clock for S3F84B8.

An internal capacitor supports the RC oscillator circuit. In addition, an external crystal or ceramic oscillation source provides 10MHz clock (maximum). The X_{IN} and X_{OUT} pins connect oscillation source to the on-chip clock circuit.

[Figure 7-1](#) and [Figure 7-2](#) show a simplified external RC oscillator and crystal/ceramic oscillator circuits. When you use external oscillator, P0.0 and P0.1 must be set to output port to prevent current consumption.

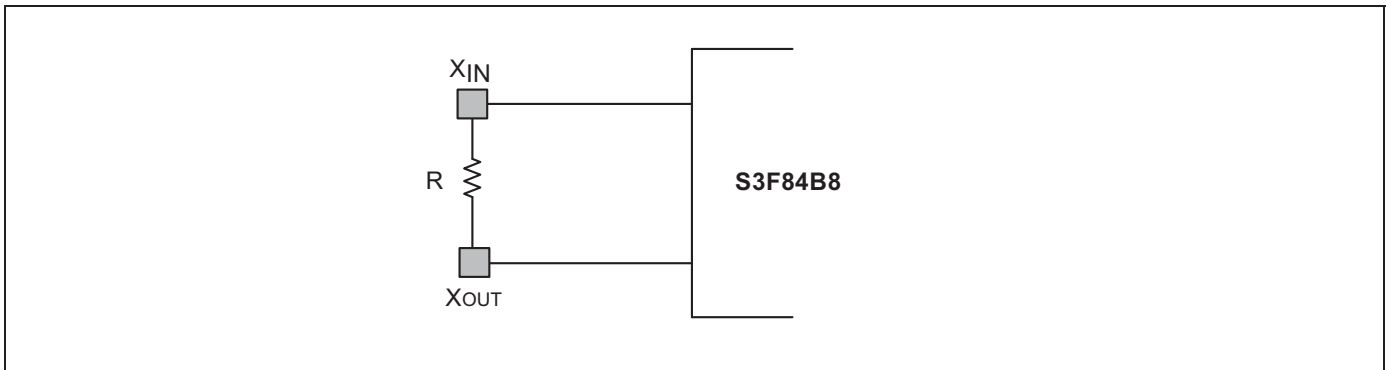


Figure 7-1 Main Oscillator Circuit (RC Oscillator with Internal Capacitor)

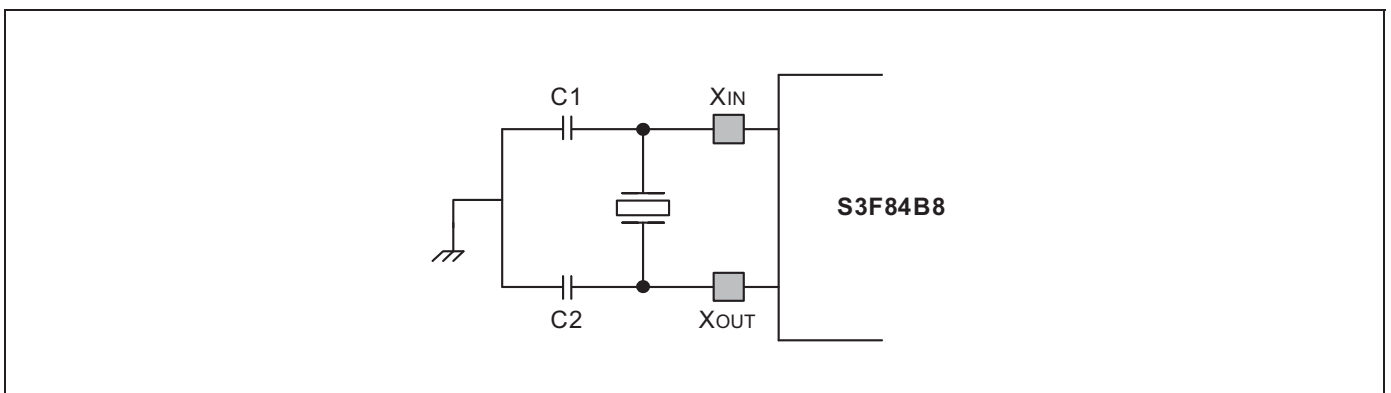


Figure 7-2 Main Oscillator Circuit (Crystal/Ceramic Oscillator)

7.1.1 CLOCK STATUS DURING POWER-DOWN MODES

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

- In Stop mode, the main oscillator “freezes”. This in turn halts the CPU and peripherals. The contents of register file and current system register values are retained. Using a reset operation or an external interrupt with RC-delay noise filter (for S3F84B8, INT0–INT5), the Stop mode is released and oscillation is started.
- In Idle mode, the internal clock signal is gated off to the CPU, but not to the interrupt control and timer. The current CPU status is retained, including stack pointer, program counter, and flags. Data in the register file is retained. Using a reset or an interrupt (external or internally-generated), the Idle mode is released.

7.1.2 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:

- Enables/disables the oscillator IRQ wake-up function (CLKCON.7).
- Divides oscillator frequency by value: non-divided, 2, 8, or 16 (CLKCON.4 and CLKCON.3)

The CLKCON register controls whether an external interrupt can be used to trigger a Stop mode release. (This function is known as “IRQ wake-up”.) The IRQ wake-up enable bit is CLKCON.7.

After a reset, the external interrupt oscillator wake-up function is enabled, and $f_{OSC}/16$ (slowest clock speed) is selected as the CPU clock. If necessary, you can increase the CPU clock speed to f_{OSC} , $f_{OSC}/2$ or $f_{OSC}/8$.

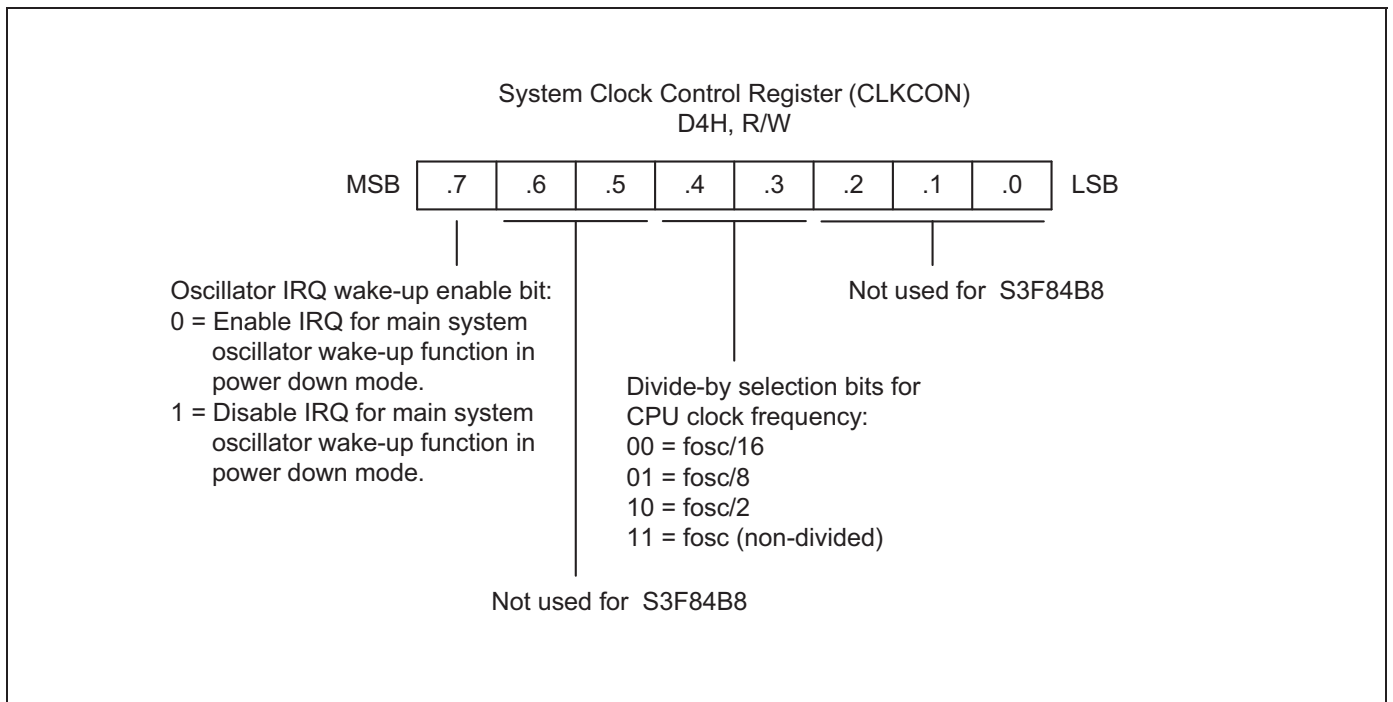


Figure 7-3 System Clock Control Register (CLKCON)

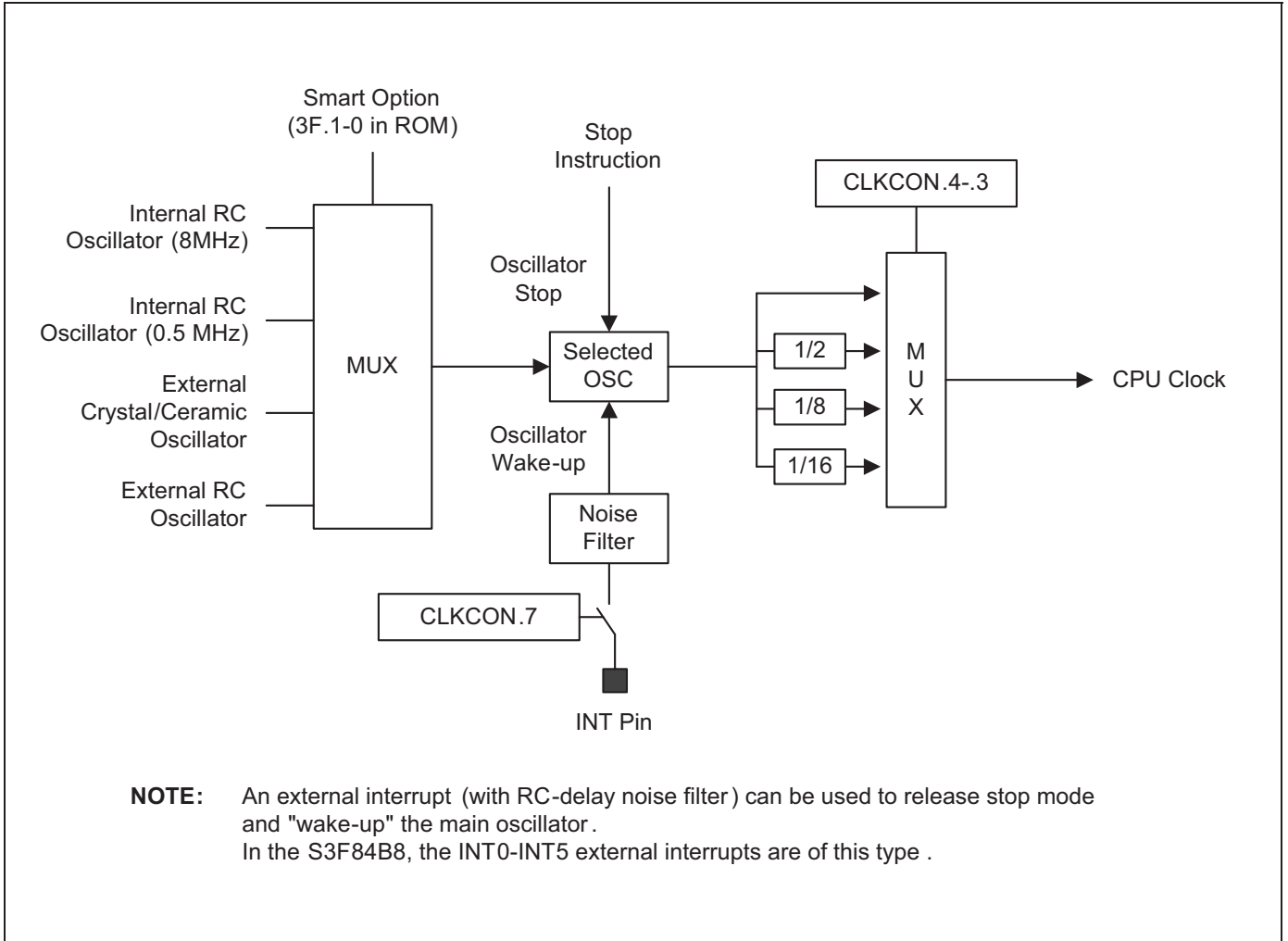


Figure 7-4 System Clock Circuit Diagram

8

RESET AND POWER-DOWN

8.1 OVERVIEW OF SYSTEM RESET

Using the Smart option (3FH.7 in ROM), you can choose the Reset source as internal (LVR) or external.

S3F84B8 can be RESET in the following four ways:

- External power-on-reset
- External nRESET input pin pulled low
- Digital watchdog peripheral time out
- Low Voltage Reset (LVR)

During an external power-on reset, the voltage at V_{DD} is set to high level and the nRESET pin stays low level for some time. The nRESET signal is inputted through a Schmitt trigger circuit, where it is then synchronized to the CPU clock. This brings the S3F84B8 into a known operating status. To ensure correct start-up, you should make sure that the nRESET signal is not released before the V_{DD} level is sufficient. This allows the MCU to operate at the chosen frequency.

The nRESET pin must be held at low level for a minimum time interval, after the power supply comes within the tolerance level. This allows 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 to 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](#), [Table 8-2](#)).

The MCU provides a watchdog timer function to ensure recovery from any 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 (LVR) features static Reset when supply voltage is below a reference value (Typical voltages are 1.9V, 2.3V, 3.0V, 3.6V, and 3.9V). Owing to this feature, the external reset circuit can be removed while keeping the application safe. As long as supply voltage is below reference value, there is an internal and static RESET. The MCU can start only when supply voltage rises over reference value.

While calculating power consumption, remember that static current of LVR circuit should be added to the CPU operating current in any operating modes such as Stop, Idle, and Normal Run mode.

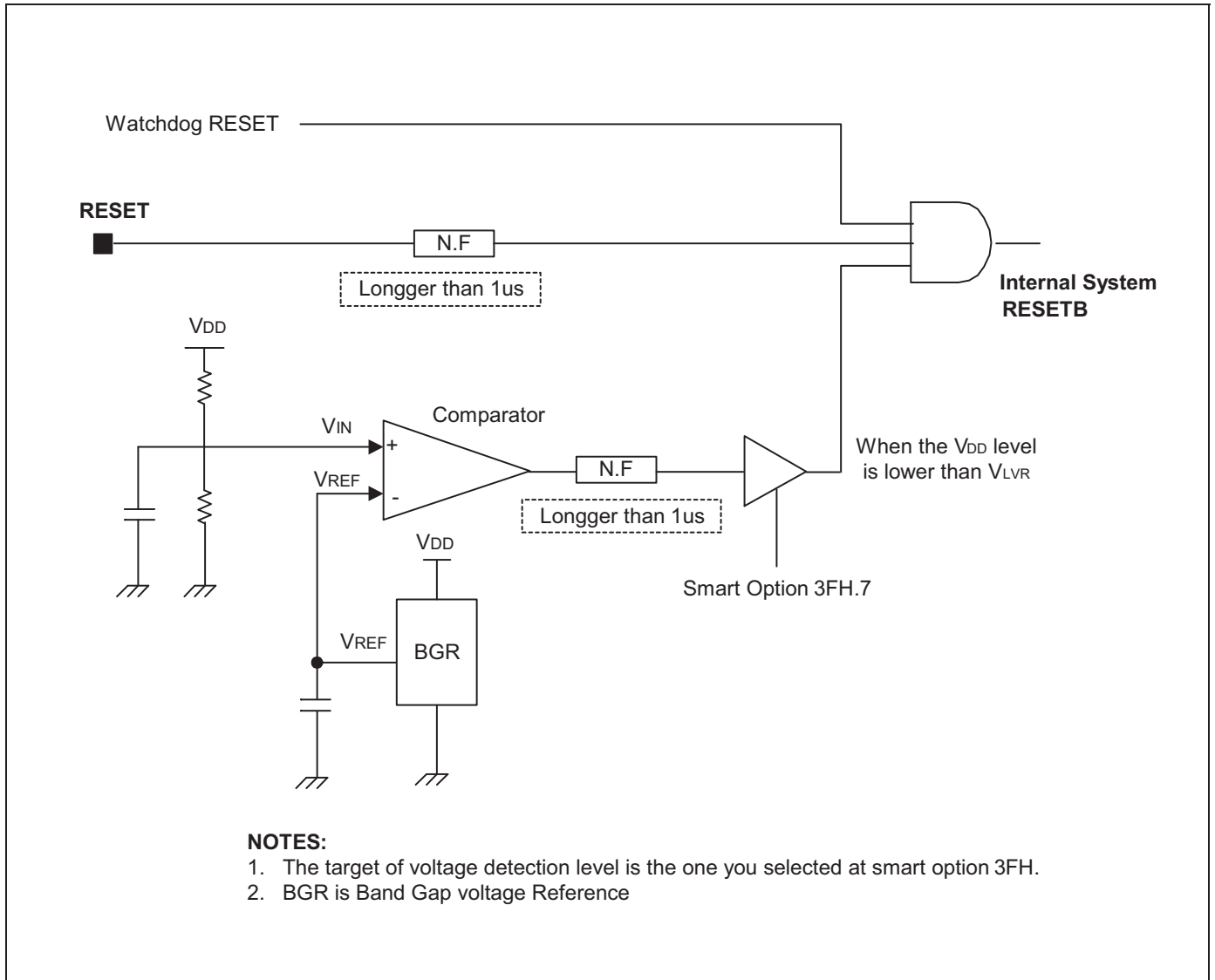


Figure 8-1 Low Voltage Reset Circuit

NOTE: To program the duration of oscillation stabilization interval, you must set the basic timer control register, BTCON, before entering the Stop mode. If you do not want to use the basic timer watchdog function (which causes a system reset when a basic timer counter overflow occurs), you can disable it by writing "1010B" to the upper nibble of BTCON.

8.1.1 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 are reset to their initial values (see [Table 8-1](#), [Table 8-2](#)).
- The program counter is loaded with ROM reset address (0100H) or other values set by the 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.

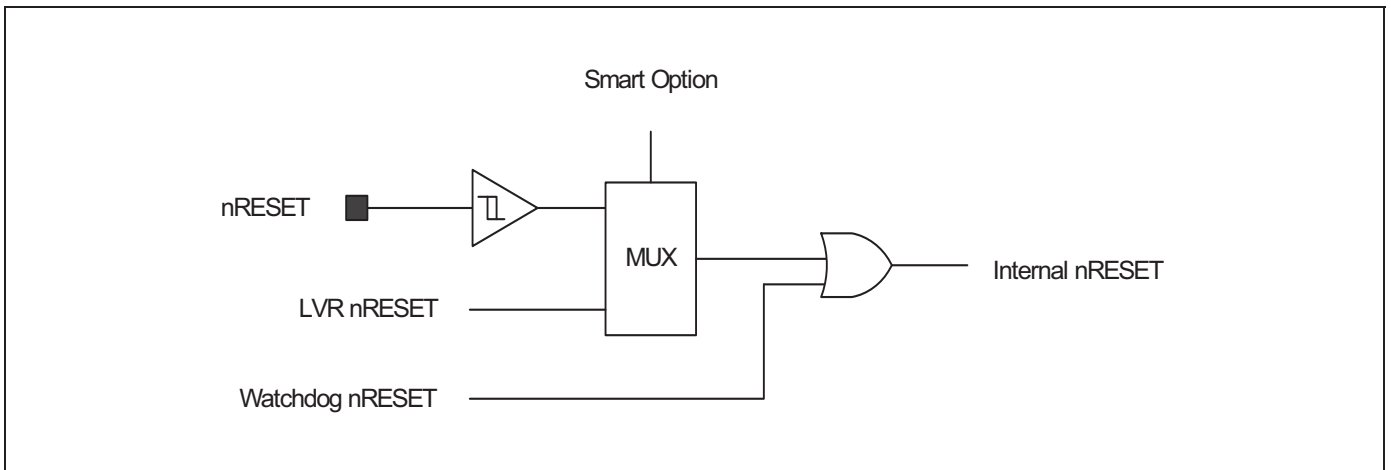


Figure 8-2 Reset Block Diagram

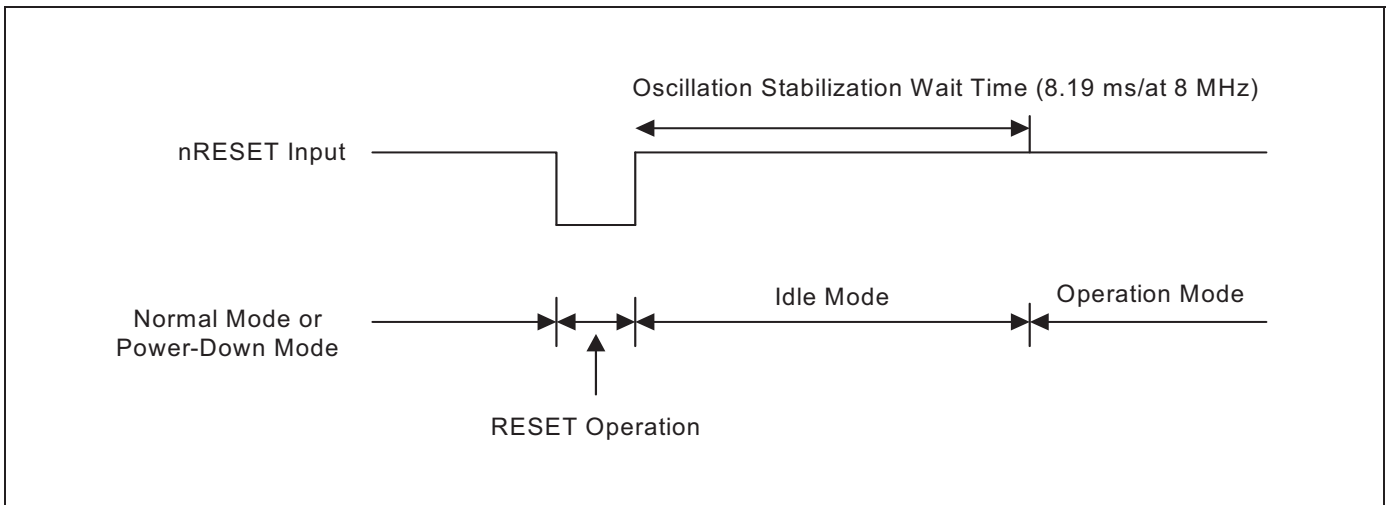


Figure 8-3 Timing for S3F84B8 after RESET

8.2 POWER-DOWN MODES

8.2.1 STOP MODE

Stop mode is invoked by the STOP (opcode 7FH) instruction. In Stop mode, the operation of the CPU and all peripherals is halted. In other words, the on-chip main oscillator is stopped and the supply current is reduced to less than 2 μ A when Low Voltage Reset (LVR) is disabled. All system functions are halted when the clock “freezes,” but the data stored in internal register file is retained. Stop mode can be released in one of the two ways: by an nRESET signal or an external interrupt.

NOTE: Before executing the STOP instruction, STPCON register must be set to “10100101B”.

8.2.1.1 Using RESET to Release Stop Mode

Stop mode is released when the nRESET signal is released and returned 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 system initialization routine by fetching the 16-bit address stored in the first and second bytes of RESET address in ROM.

8.2.1.2 Using an External Interrupt to Release Stop Mode

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

NOTE: When Stop mode is released by an external interrupt, values in system and peripheral control registers remain unchanged. Also, when you use an interrupt to release Stop mode, the CLKCON.3 and CLKCON.4 register values remain unchanged, and the selected clock value is used. Thus, you can also program the duration of oscillation stabilization interval by putting the appropriate value to BTCON register before entering Stop mode.

The external interrupt is serviced after the Stop mode is released. The interrupt service routine will then return to the instruction immediately following the STOP instruction.

8.2.1.3 Idle Mode

Idle mode is invoked by the IDLE (opcode 6FH) instruction. In Idle mode, the 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 the interrupt logic and timer/counters. Port pins retain the mode (input or output) they had at the time of entering Idle mode.

There are two ways to release the 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, 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 selected clock value is used. The interrupt is then serviced. The interrupt service routine will then return to the instruction immediately following the STOP instruction.

NOTE: Only external interrupts that are not related to clock can be used to release the Stop mode. To release the Idle mode, any type of interrupt (that is, internal or external) can be used.
Before entering the STOP or IDLE mode, ADC must be disabled. Otherwise, the STOP or IDLE current will increase significantly.

8.2.2 HARDWARE RESET VALUES

[Figure 8-1](#) shows the reset values of the CPU and system registers, peripheral control registers, and peripheral data registers, following a reset operation.

Following notation is used to represent the 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 and 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	0
Clock control register	CLKCON	D4H	R/W	0	–	–	0	0	–	–	–	–
System flags register	FLAGS	D5H	R/W	x	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	x
Instruction Pointer (High Byte)	IPH	DAH	R/W	x	x	x	x	x	x	x	x	x
Instruction Pointer (Low Byte)	IPL	DBH	R/W	x	x	x	x	x	x	x	x	x
Interrupt Request register	IRQ	DCH	R	0	0	0	0	0	0	0	0	0
Interrupt Mask Register	IMR	DDH	R/W	x	x	x	x	x	x	x	x	x
System Mode Register	SYM	DEH	R/W	0	–	–	x	x	x	0	0	0
Register Page Pointer	PP	DFH	R/W	0	0	0	0	0	0	0	0	0
Port 0 data register	P0	E0H	R/W	–	0	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	R/W	–	–	–	–	–	0	0	0	0
Port 2 data register	P2	E2H	R/W	0	0	0	0	0	0	0	0	0
Port 0 interrupt control register	P0INT	E3H	R/W	–	0	0	0	0	–	0	0	0
Port 0 control register (High byte)	P0CONH	E4H	R/W	–	–	0	0	0	0	0	0	0
Port 0 control register (Low byte)	P0CONL	E5H	R/W	0	0	–	–	0	0	0	0	0
Port 0 interrupt pending register	P0PND	E6H	R/W	–	0	0	0	0	–	0	0	0
Port 1 control register (Low byte)	P1CON	E7H	R/W	–	–	0	0	0	0	0	0	0
Port 2 control register (High byte)	P2CONH	E8H	R/W	0	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	0
Comparator 0 control register	CMP0CON	EAH	R/W	–	–	–	0	0	0	1	0	0
Comparator 1 control register	CMP1CON	EBH	R/W	0	0	0	0	0	0	1	0	0

Register Name	Mnemonic	Address and Location		RESET Value (Bit)							
		Address	R/W	7	6	5	4	3	2	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

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

Table 8-2 System and Peripheral Control Registers Set1 Bank1

Register Name	Mnemonic	Address and 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

9 I/O PORTS

9.1 OVERVIEW OF I/O PORTS

The S3F84B8 microcontroller has three bit-programmable I/O ports (P0, P1, and P2) and 17 I/O pins. Each port can be easily configured to meet the application design requirements. The CPU accesses ports by directly writing or reading the port registers. No special I/O instructions are required.

[Table 9-1](#) provides 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 the 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 the software. Pins can also be assigned individually as alternative function pins.
2	I/O port with bit-programmable pins. Configurable to input mode or push-pull output mode. Pins can also be assigned individually as alternative function pins.

For better Electrical Fast transients Test (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.

9.1.1 PORT DATA REGISTERS

[Table 9-2](#) provides 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, as 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

9.1.1.1 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.

9.1.1.1.1 Port 0 Control Register (*P0CONH, P0CONL*)

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

When you select the output mode, a push-pull or an open-drain circuit is configured. Different selections are available such as:

- 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)

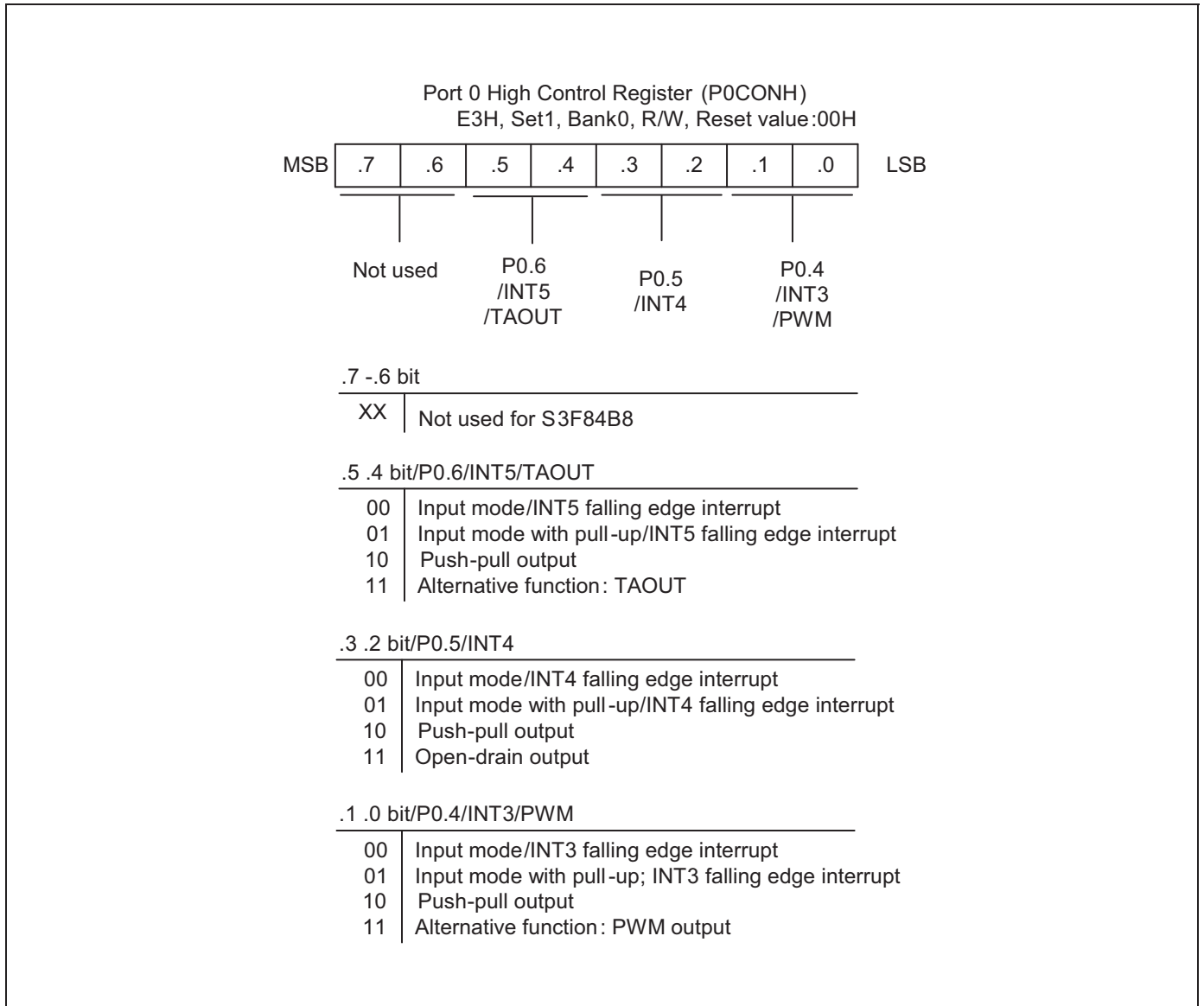


Figure 9-1 Port 0 Control Register High Byte (P0CONH)

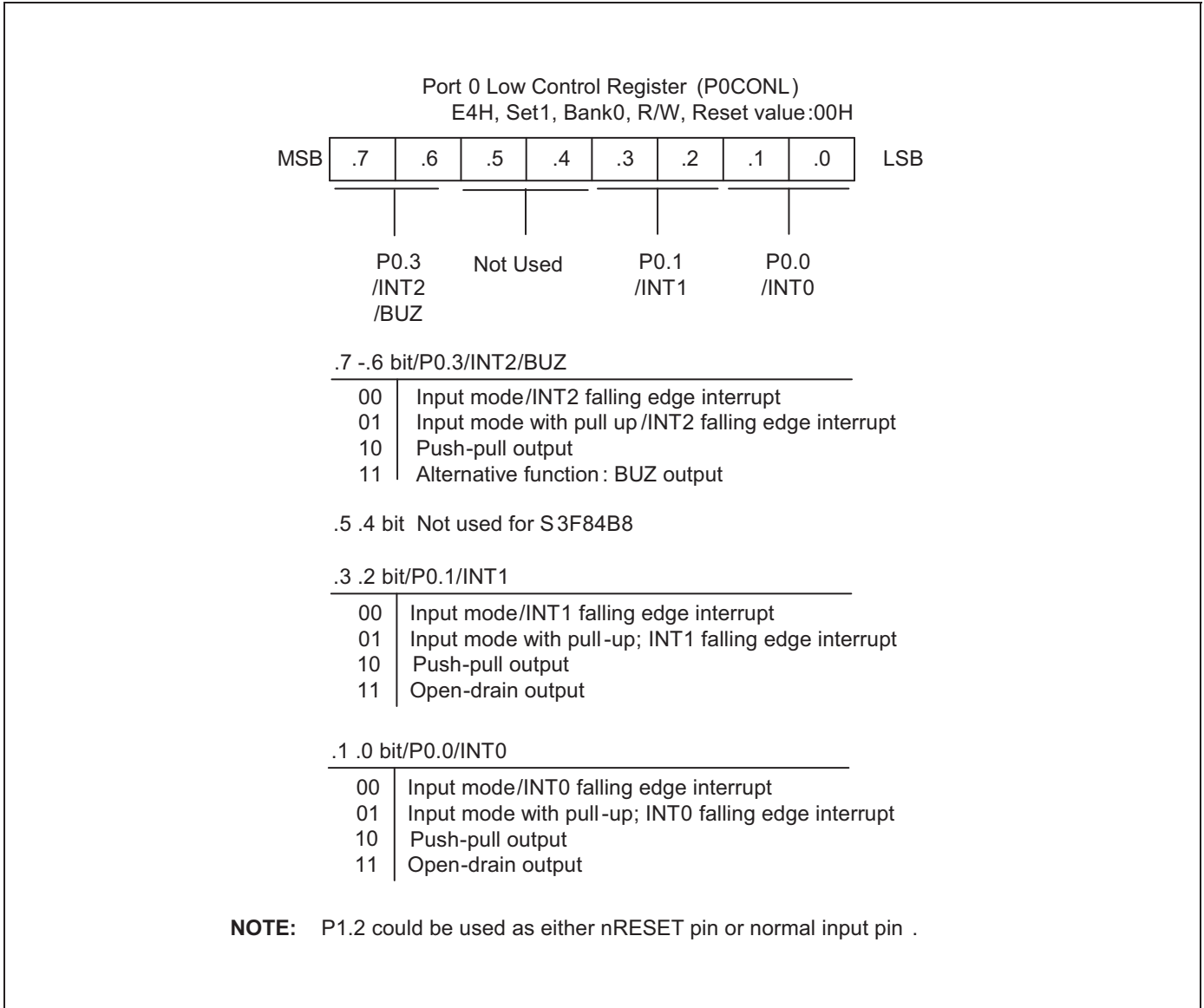


Figure 9-2 Port 0 Control Register Low Byte (P0CONL)

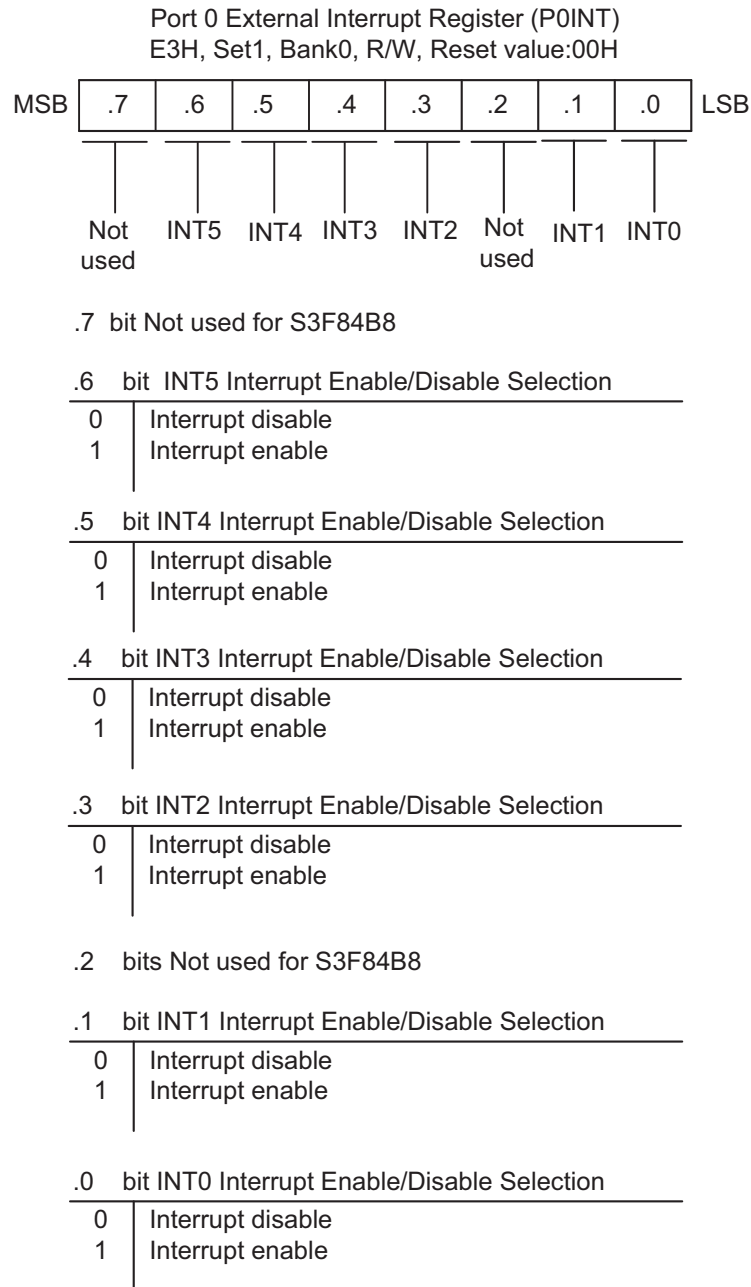


Figure 9-3 Port 0 Interrupt Control Register (P0INT)

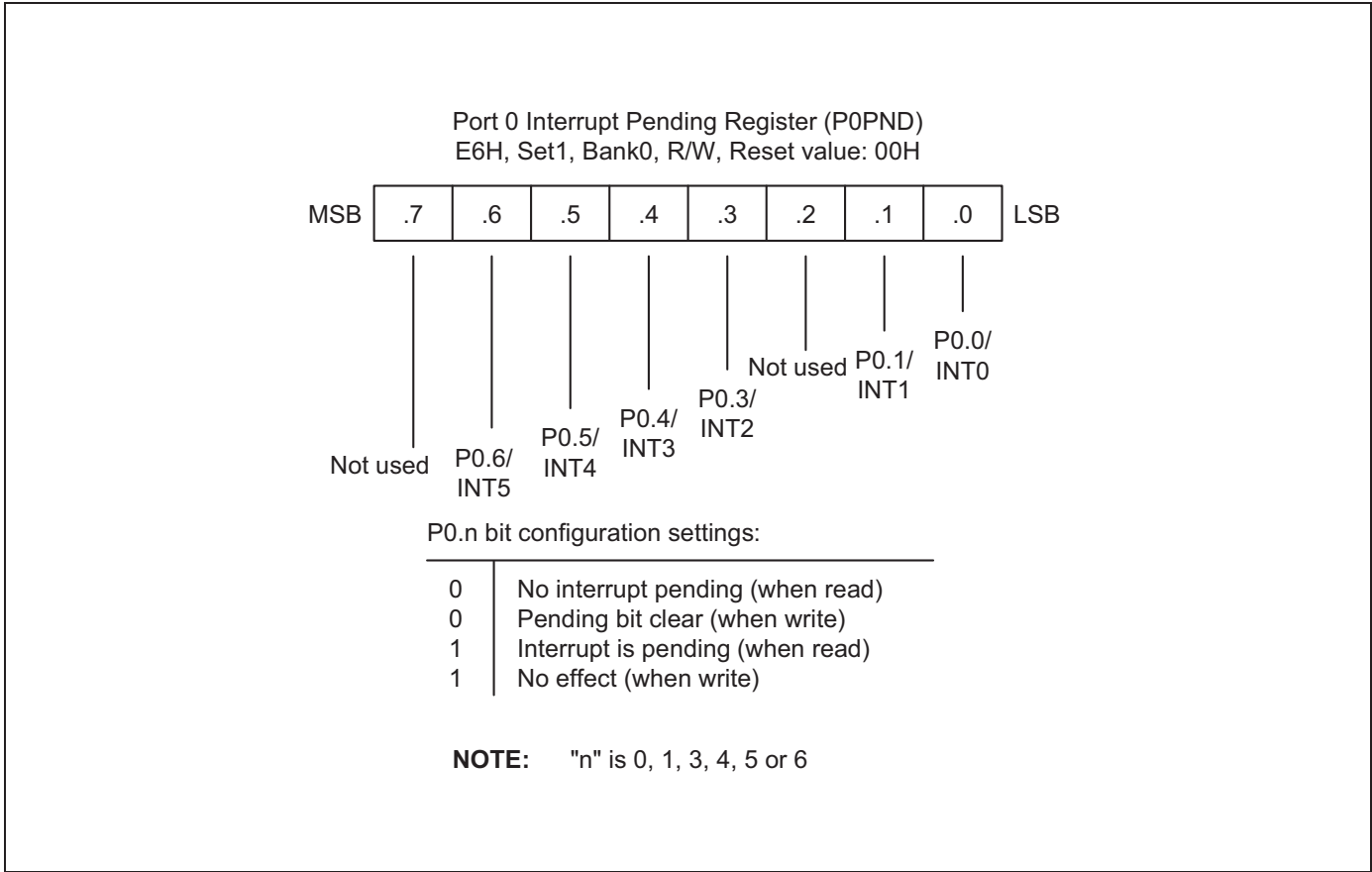


Figure 9-4 Port 0 Interrupt Pending Register (P0PND)

9.1.1.2 Port 1

Port 1 is a 3-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.

9.1.1.2.1 Port 1 Control Register (P1CON)

Port 1 pins are configured by setting the control registers located at P1CON (E7H, Set1 Bank0).

When you select the output mode, push-pull circuit can be configured. In the input mode, pull-up resistor can be configured as on or off. For alternative functions, different selections are available such as:

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

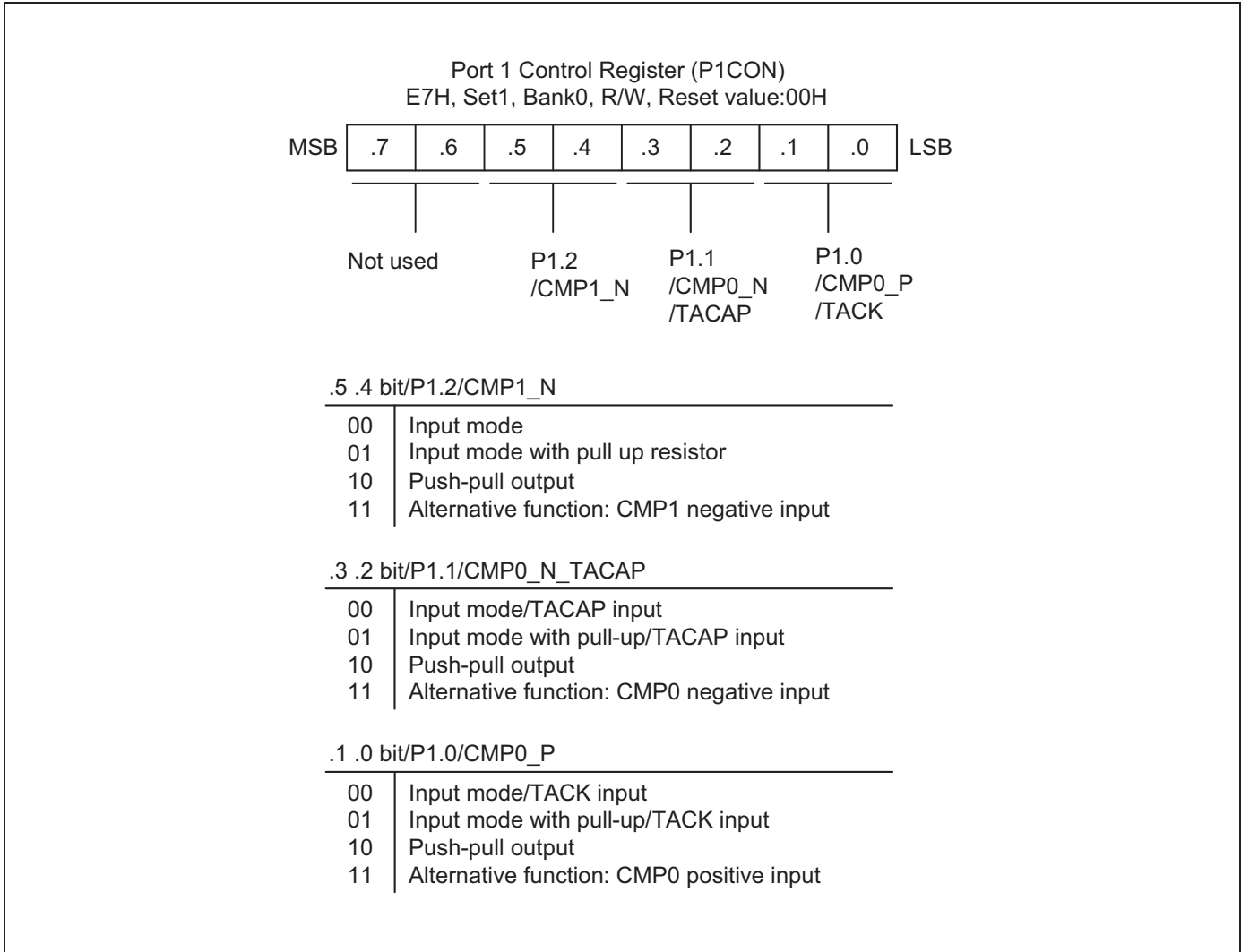


Figure 9-5 Port 1 Control Register (P1CON)

9.1.1.3 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.

9.1.1.3.1 Port 2 Control Register (*P2CONH, P2CONL*)

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

When you select the output mode, a push-pull circuit is configured. In the input mode, pull-up resistor can be configured as on or off. Different selections are available such as:

- 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

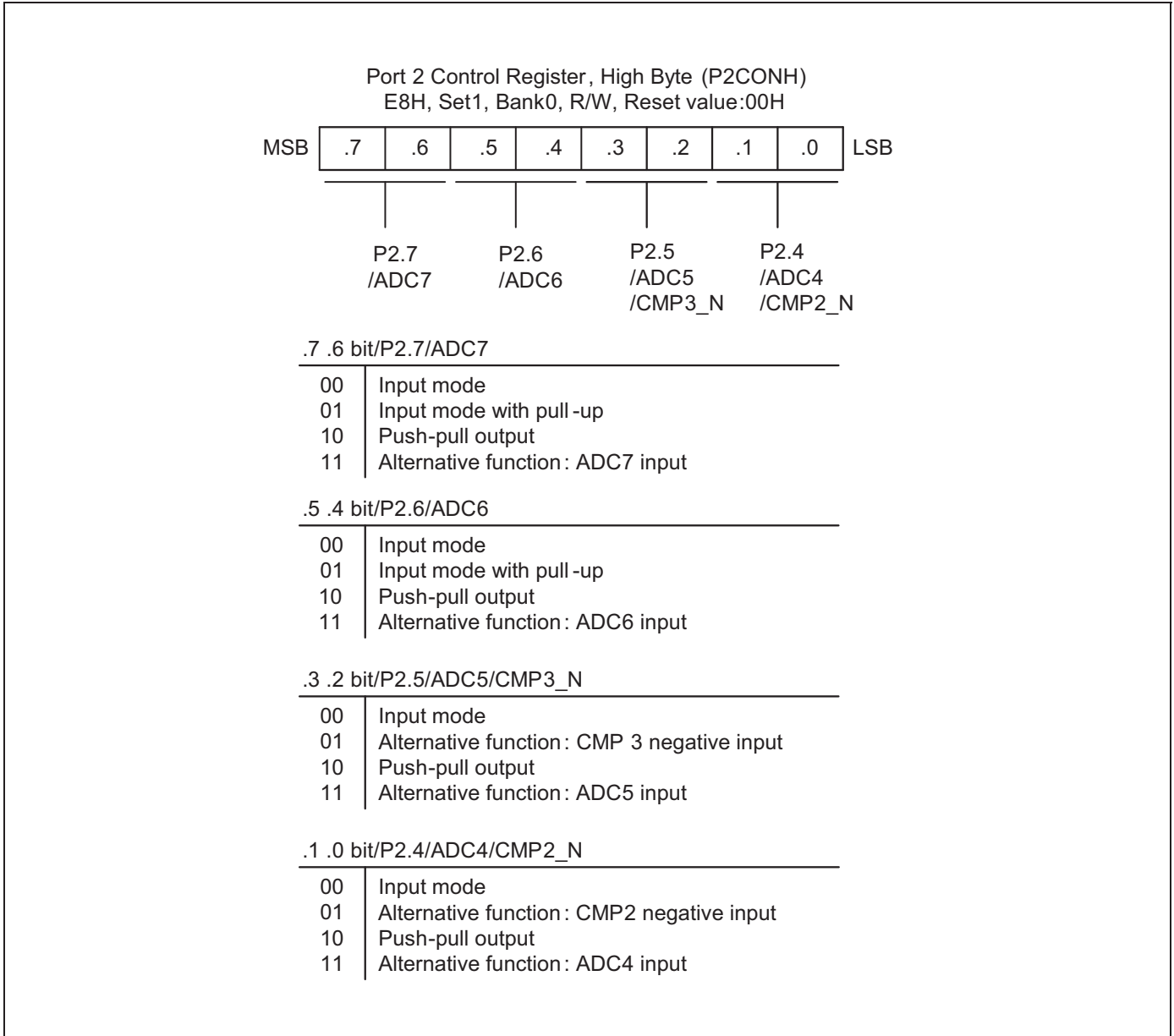


Figure 9-6 Port 2 High-Byte Control Register (P2CONH)

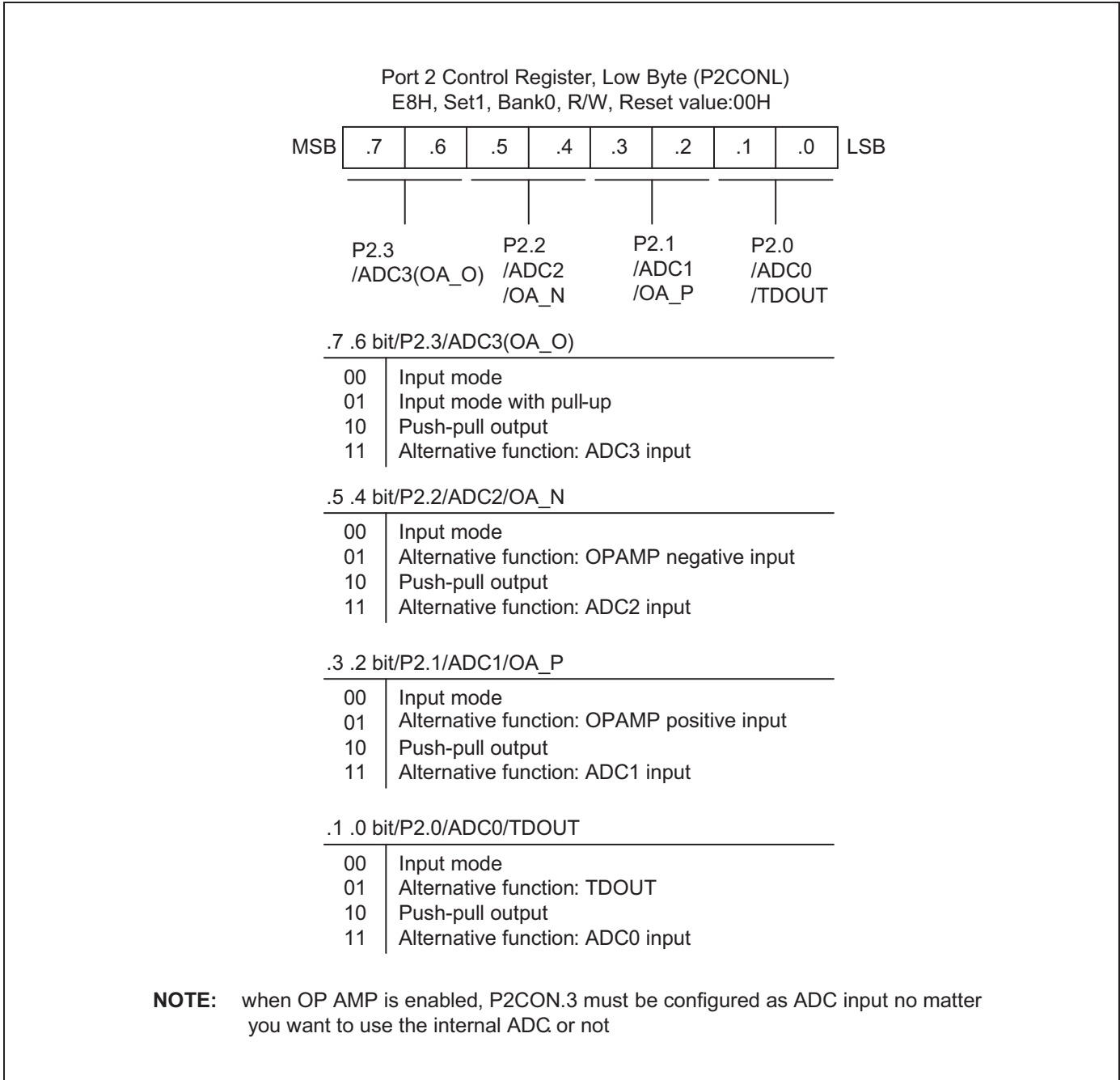


Figure 9-7 Port 2 Low-Byte Control Register (P2CONL)

10 BASIC TIMER

10.1 OVERVIEW OF BASIC TIMER

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 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)

10.2 BASIC TIMER CONTROL REGISTER (BTCON)

The basic timer control register, BTCON, selects the input clock frequency to clear the basic timer counter and frequency dividers and enable (or disable) the watchdog timer function.

A reset clears BTCON to “00H”. This enables the watchdog function to select a basic timer clock frequency of $f_{OSC}/4096$. To disable the watchdog function, you must write the signature code “1010B” to 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 basic timer input clock, write a “1” to BTCON.0.

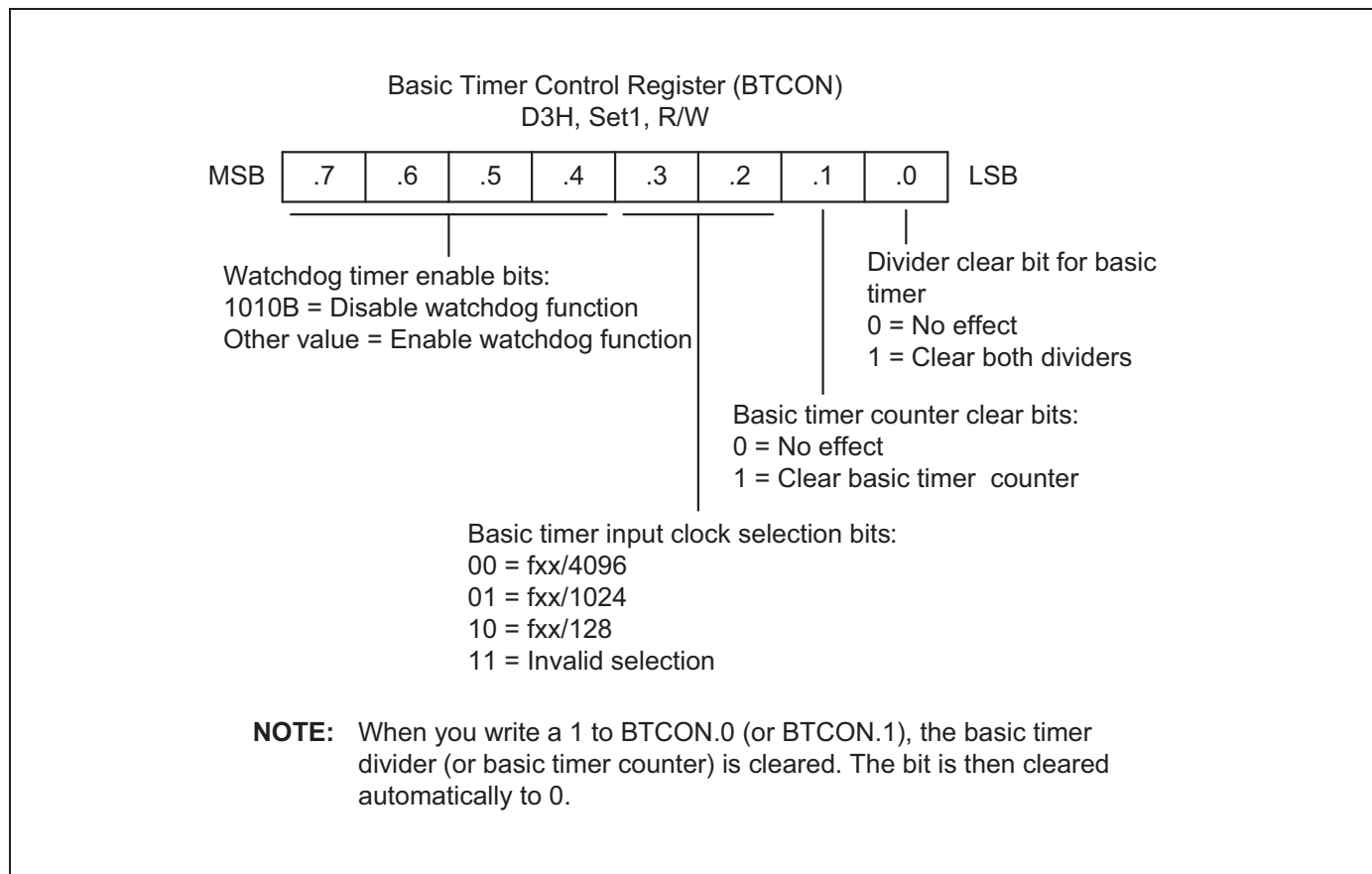


Figure 10-1 Basic Timer Control Register (BTCON)

10.2.1 BASIC TIMER FUNCTION DESCRIPTION

10.2.1.1 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. It also selects the oscillator clock divided by 4096 as the BT clock.

A reset occurs whenever a basic timer counter overflows. During normal operation, the application program must prevent the overflow and its 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 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 8-bit basic timer counter, BTCNT) is always broken by a BTCNT clear instruction. If a malfunction occurs, a reset is triggered automatically.

10.2.1.2 Oscillation Stabilization Interval Timer Function

You can 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 the 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 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, leading to the start of oscillation.
2. If external power-on reset occurs, the basic timer counter will increase at the rate of $f_{OSC}/4096$. If an external interrupt releases the Stop mode, the BTCNT value increases at the rate of 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 is resumed.

[Figure 10-2](#) and [Figure 10-3](#) show the oscillation stabilization time on RESET and STOP mode release.

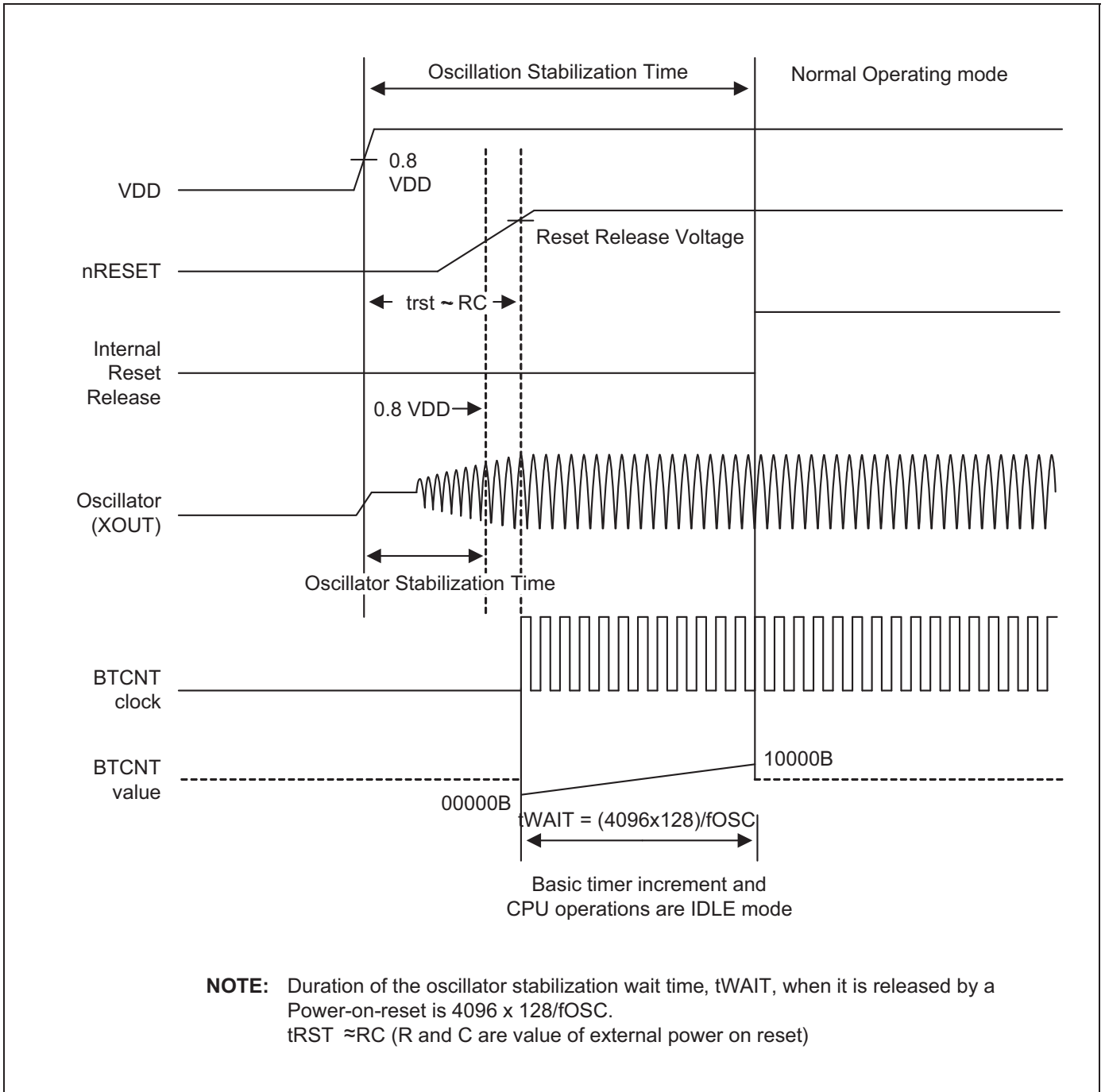
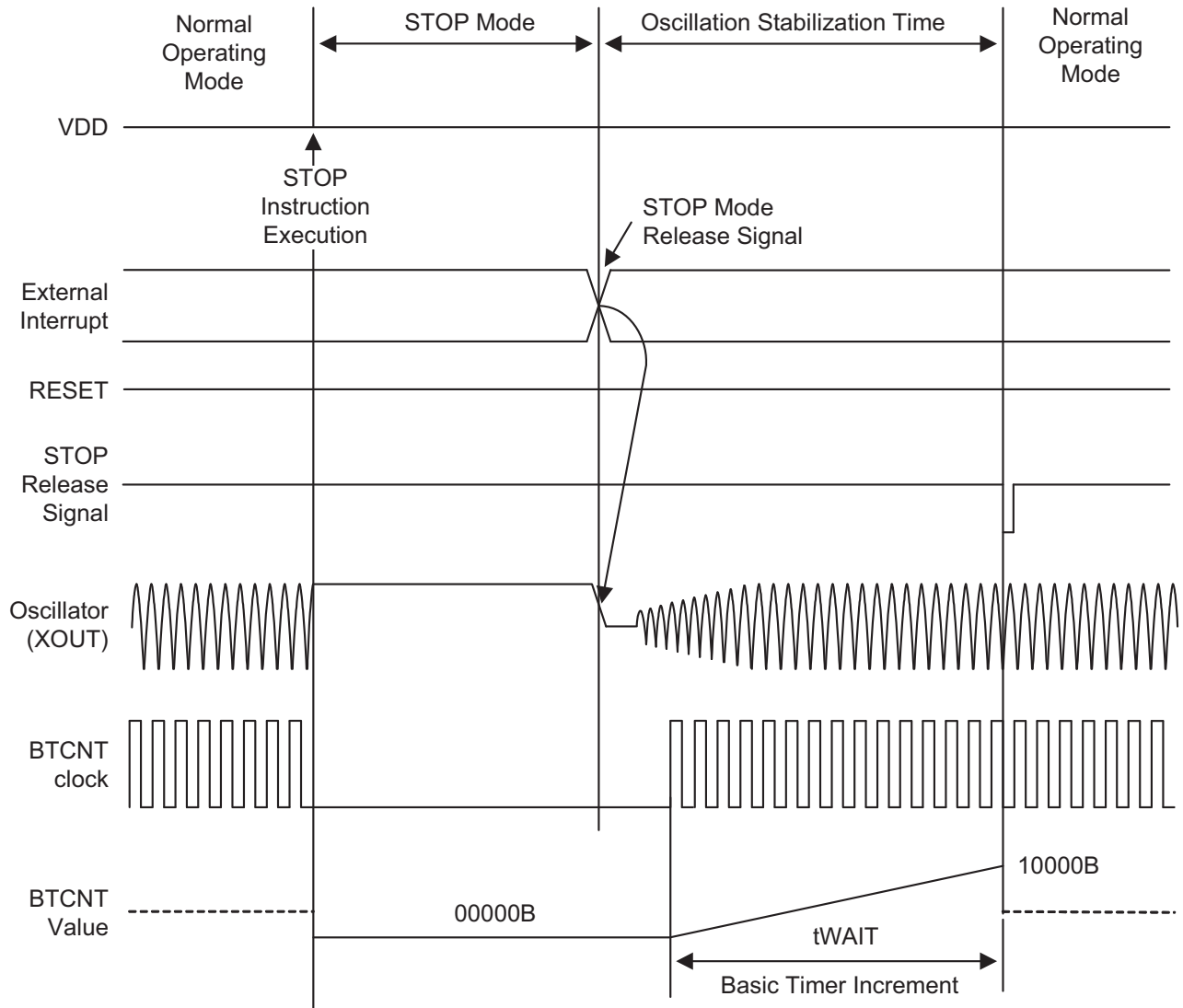


Figure 10-2 Oscillation Stabilization Time on RESET



NOTE: Duration of the oscillator stabilization wait time, t_{WAIT} , it is released by an interrupt is determined by the setting in basic timer control register, BTCON.

BTCON.3	BTCON.2	t_{WAIT}	t_{WAIT} (When f_{OSC} is 8 MHz)
0	0	$(4096 \times 128)/f_{osc}$	65.536 ms
0	1	$(1024 \times 128)/f_{osc}$	16.384 ms
1	0	$(128 \times 128)/f_{osc}$	2.048 ms
1	1	Invalid setting	—

Figure 10-3 Oscillation Stabilization Time on STOP Mode Release

Example 10-1 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, enables LVR, enables nRESET pin

;-----<< Initialize System and Peripherals >>
        ORG    0100H
RESET:   DI                     ; Disables interrupt
        LD     CLKCON, #00011000B ; Selects non-divided CPU clock
        LD     SPL, #0FFH        ; Stack pointer must be set
        .
        .
        LD     BTCON, #02H       ; Enables watchdog function
                                       ; Basic timer clock: fOSC/4096
                                       ; Clears basic counter (BTCNT)
        .
        .
        EI                     ; Enable interrupt

;-----<< Main loop >>
MAIN:    .
        LD     BTCON, #02H       ; Enables watchdog function
                                       ; Clears basic counter (BTCNT)
        .
        .
        JR     T, MAIN           ;

```

11

8-BIT TIMER A

11.1 OVERVIEW OF 8-BIT TIMER A

The 8-bit Timer A is a general-purpose timer/counter. It has three operating modes, and you can select one of the modes using the appropriate TACON setting.

The three operating modes are:

- Interval timer mode (Toggles output at TAOUT pin)
- Capture input mode with a rising or falling edge trigger at the TACAP pin
- PWM mode (TAOUT)

Timer A comprises of the following functional components:

- Prescaler 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), PWM, or Match Output (TAOUT)
- Timer A overflow interrupt and match/capture interrupt
- Timer A control register, TACON (E1H, Set1 Bank1, read/write)

11.1.1 FUNCTIONAL DESCRIPTION

11.1.1.1 Timer A Interrupts

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

Timer A overflow interrupt (TAOVF) can be cleared by both software and hardware. On the other hand, Timer A match/capture interrupt (TAINT) pending conditions are cleared by software when it has been serviced.

11.1.1.2 Interval Timer Function

The Timer A module can generate the Timer A match interrupt (TAINT).

When Timer A interrupt occurs, it is serviced by the CPU. The pending condition should be cleared by the 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.

For example, if 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, counter value is reset, and counting is resumed.

11.1.1.3 Pulse Width Modulation Mode

Pulse width modulation (PWM) mode allows you to program the width (duration) of pulse that is outputted at the TAOUT pin. As in the interval timer mode, a match signal is generated when the counter value is similar to the value written to 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.

Even though 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 as long as the data value is greater than ($>$) the counter value. One pulse width is equal to $tCLK \cdot 256$.

11.1.1.4 Capture Mode

In capture mode, a signal edge 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, that is, signal edge at the TACAP pin. You can select the capture input by setting the value of 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, the output is selected.

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

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

11.1.2 TIMER A CONTROL REGISTER (TACON)

You can use the Timer A control register (TACON) for the following purposes:

- 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 the Timer A match/capture interrupt pending conditions

You can use Timer A prescaler register (TAPS) for the following purposes:

- Select the 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 the 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 the 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, it is serviced by the CPU. 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 the counter and interrupt pending bit. When interrupt service routine is served, the pending condition must be cleared by the software by writing a '0' to the interrupt pending bit.

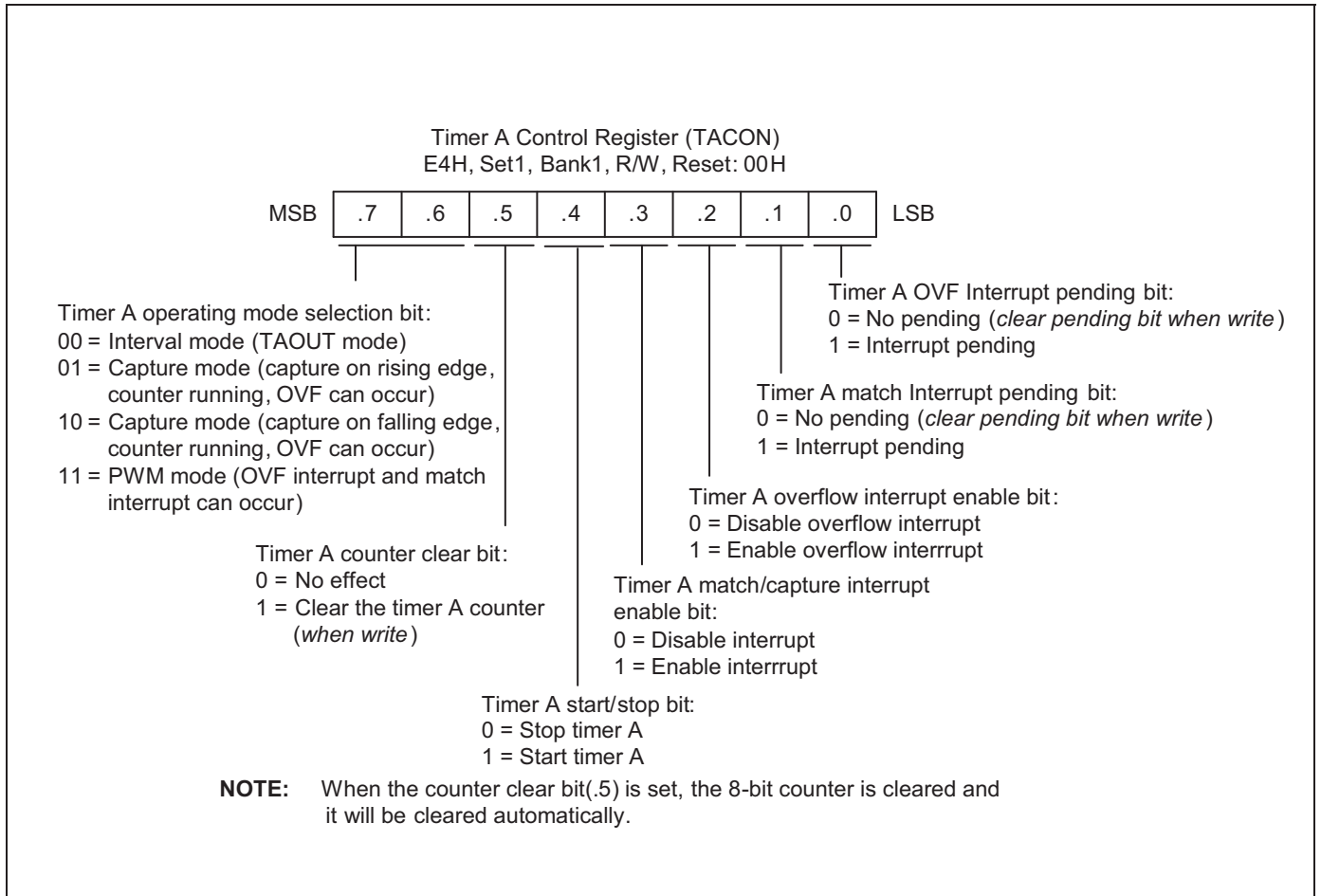


Figure 11-1 Timer A Control Register (TACON)

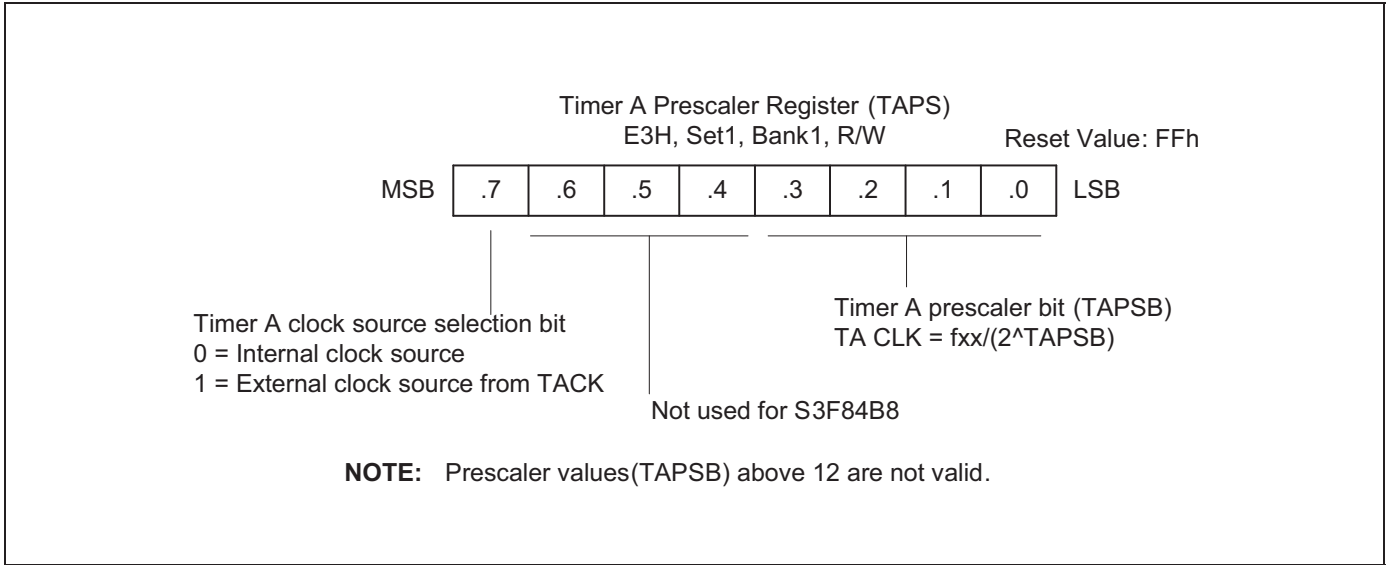


Figure 11-2 Timer A Prescaler Register (TAPS)

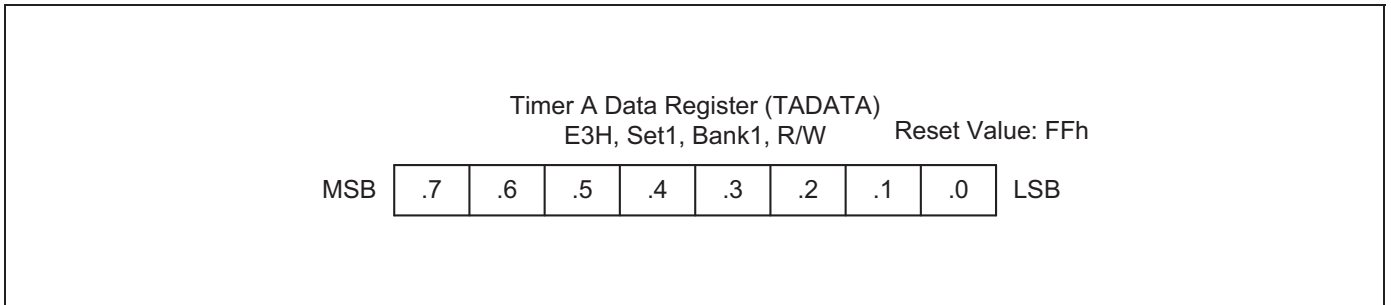


Figure 11-3 Timer A DATA Register (TADATA)

11.1.3 BLOCK DIAGRAM OF TIMER A

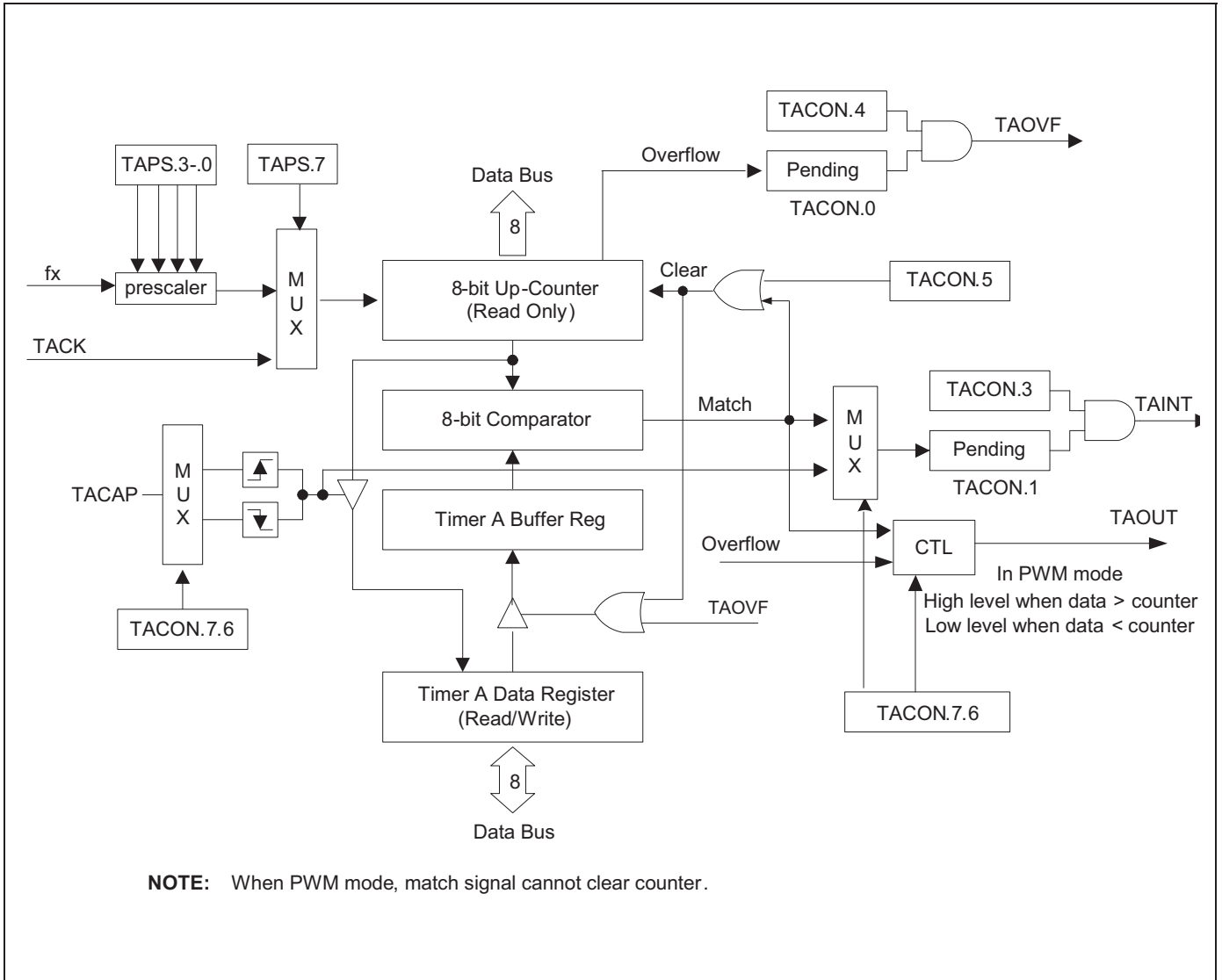


Figure 11-4 Simplified Timer A Functional Block Diagram

12

TIMER 0

12.1 ONE 16-BIT TIMER MODE (TIMER 0)

The 16-bit Timer 0 is used in one 16-bit Timer mode or two 8-bit Timers mode. If TCCON.7 is set to “1”, Timer 0 is used as a 16-bit Timer. On the other hand, 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 (Timers C and D)

12.1.1 OVERVIEW OF ONE 16-BIT TIMER MODE (TIMER 0)

Timer 0 is a 16-bit general-purpose timer. It works in the interval timer mode by using the appropriate TCCON setting.

Timer 0 has the following functional components:

- Prescaler for clock frequency programmable from f_x to $f_x/4096$
- 16-bit comparator and 16-bit reference data register (TCDATA and TDDATA)
- Timer 0 match interrupt generation (Interrupt vector address: E0H)
- Timer 0 control register, TCCON (E5H, Set1 Bank1, read/write)

12.1.2 FUNCTIONAL DESCRIPTION OF ONE 16-BIT TIMER MODE (TIMER 0)

12.1.2.1 Interval Timer Function

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

In interval timer mode, a match signal is generated when the counter value is identical to the values written to Timer 0 reference data registers, TCDATA and TDDATA. The match signal generates a Timer 0 match interrupt and clears the counter. For example, if you write the values 32H to TCDATA, 10H to TDDATA, 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 is resumed.

12.1.2.2 Timer 0 Control Register (TCCON)

You can use the Timer 0 control register, TCCON, for the following purposes:

- Enable the Timer 0 operation (interval timer).
- Clear the Timer 0 counter.
- Enable the Timer 0 interrupt.
- Clear the Timer 0 interrupt pending conditions.

You can use the Timer0 prescaler register, TCPS, for the following purposes:

- Select the clock source. Comparator 0's output can be configured as the clock source of Timer0.
- Program the 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 the Timer 0 to work in 16-bit Timer mode and disables the 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. This clears the counter and interrupt pending bit.

When Timer 0 is disabled, the interrupt pending bit can still be set when it meets the interrupt condition. Application program can poll for the pending bit, TCCON.1. When a "1" is detected, Timer 0 interrupt is pending. The pending condition must be cleared by the software by writing a "0" to Timer 0 interrupt pending bit, TCCON.1.

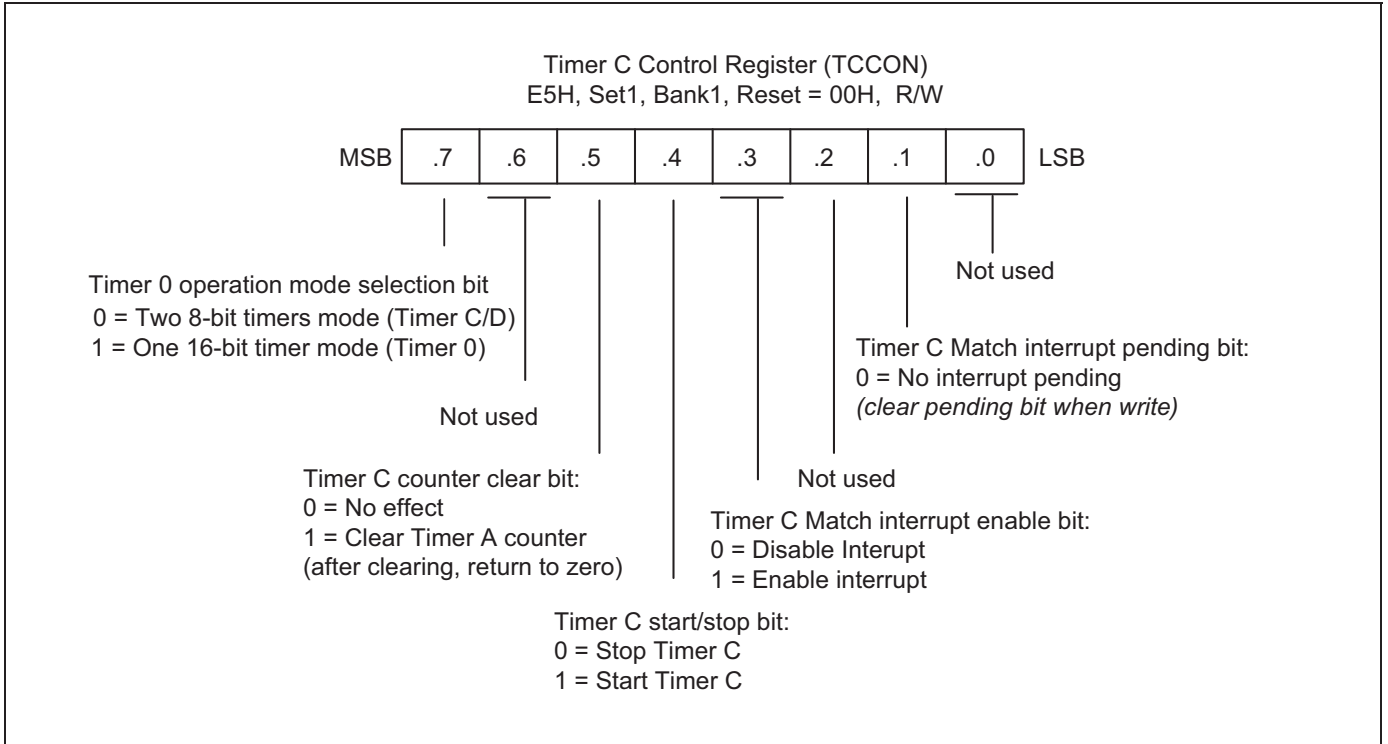


Figure 12-1 Timer 0 Control Register (TCCON)

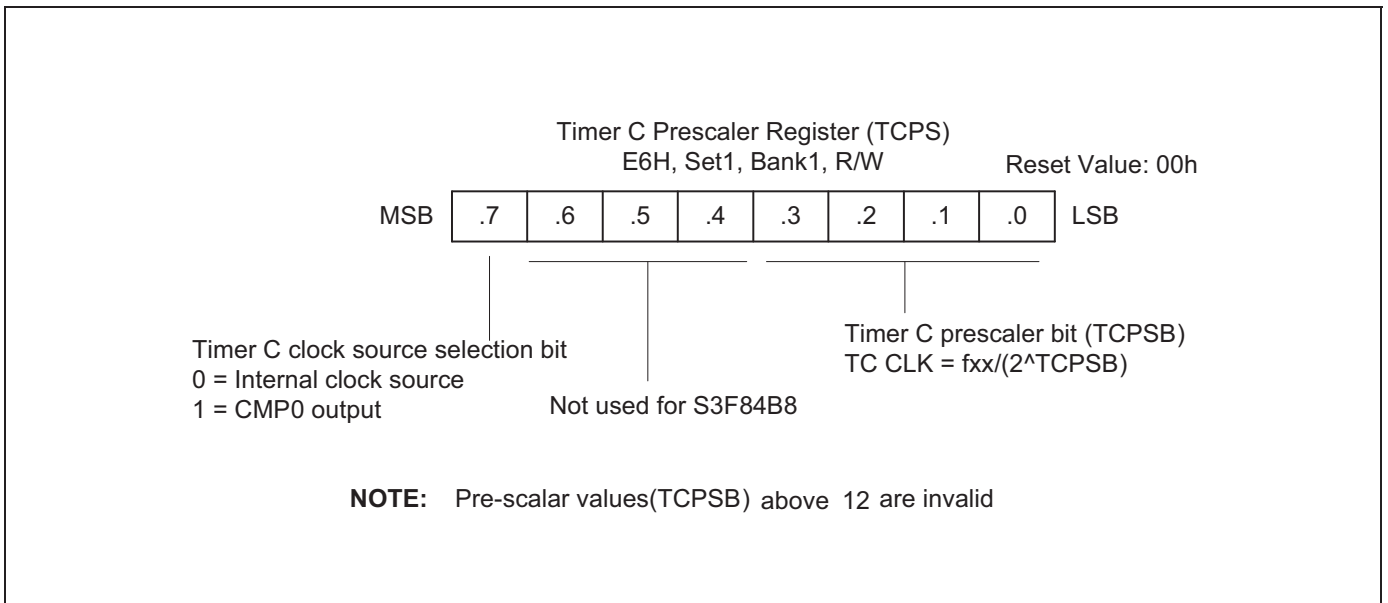


Figure 12-2 Timer 0 Prescaler Register (TCPS)

12.1.3 BLOCK DIAGRAM OF TIMER 0

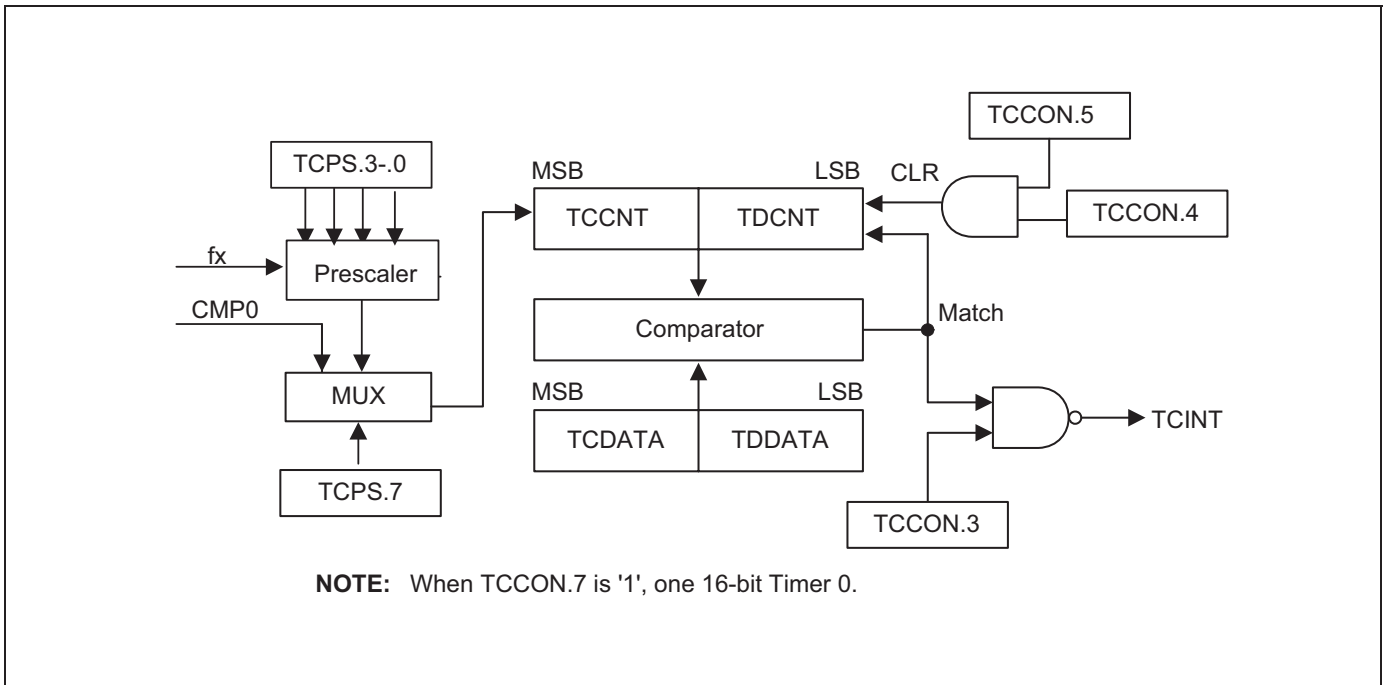


Figure 12-3 Timer 0 Functional Block Diagram

12.2 TWO 8-BIT TIMERS MODE (TIMER C AND D)

12.2.1 OVERVIEW OF TWO 8-BIT TIMERS MODE (TIMER C AND D)

Timers C and D are 8-bit general-purpose timers. Timer C works in the interval timer mode, while Timer D works in the interval timer and PWM modes by using the appropriate TCCON and TDCON setting, respectively.

Timers C and D have the following functional components:

- Prescaler for Timer C clock frequency programmable from fx to $fx/4096$
Prescaler for Timer D clock frequency programmable from fx to $fx/4096$
- 8-bit counter (TCCNT and TDCNT), 8-bit comparator, and 8-bit reference data register (TCDATA and TDDATA)
- Timer C match interrupt generation
- Timer C control register, TCCON (E5H, bank1, read/write)
- Timer D has an 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)

12.2.2 TIMER C AND D CONTROL REGISTER (TCCON, TDCON)

You can use the Timers C and D control registers, TCCON and TDCON, for the following purposes:

- Enable the Timer C (interval timer mode) and Timer D operation (interval timer mode and PWM mode).
- Select the Timer C clock source.
- Clear the Timers C and D counter, TCCNT and TDCNT.
- Enable the Timers C and D interrupt.
- Clear the Timers C and D interrupt pending conditions.

You can use Timer C prescaler register, TCPS, for the following purposes:

- Select the clock source. Comparator 0's output can be configured to the clock source of Timer C.
- Select the clock prescaler.
- You can use Timer D prescaler register, TDPS, for the following purpose:
- 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 the Timer C interrupt. You can clear the Timer C counter at any time during normal operation by writing a “1” to TCCON.5.

A reset clears TDCON to “00H”. This sets the Timer D to work in interval Timer mode, and disables the 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” and 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 clears the 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-routines have been serviced, the pending condition must be cleared by the software by writing a “0” to the Timers C and D interrupt pending bit, TCCON.1 and TDCON.1, respectively.

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

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

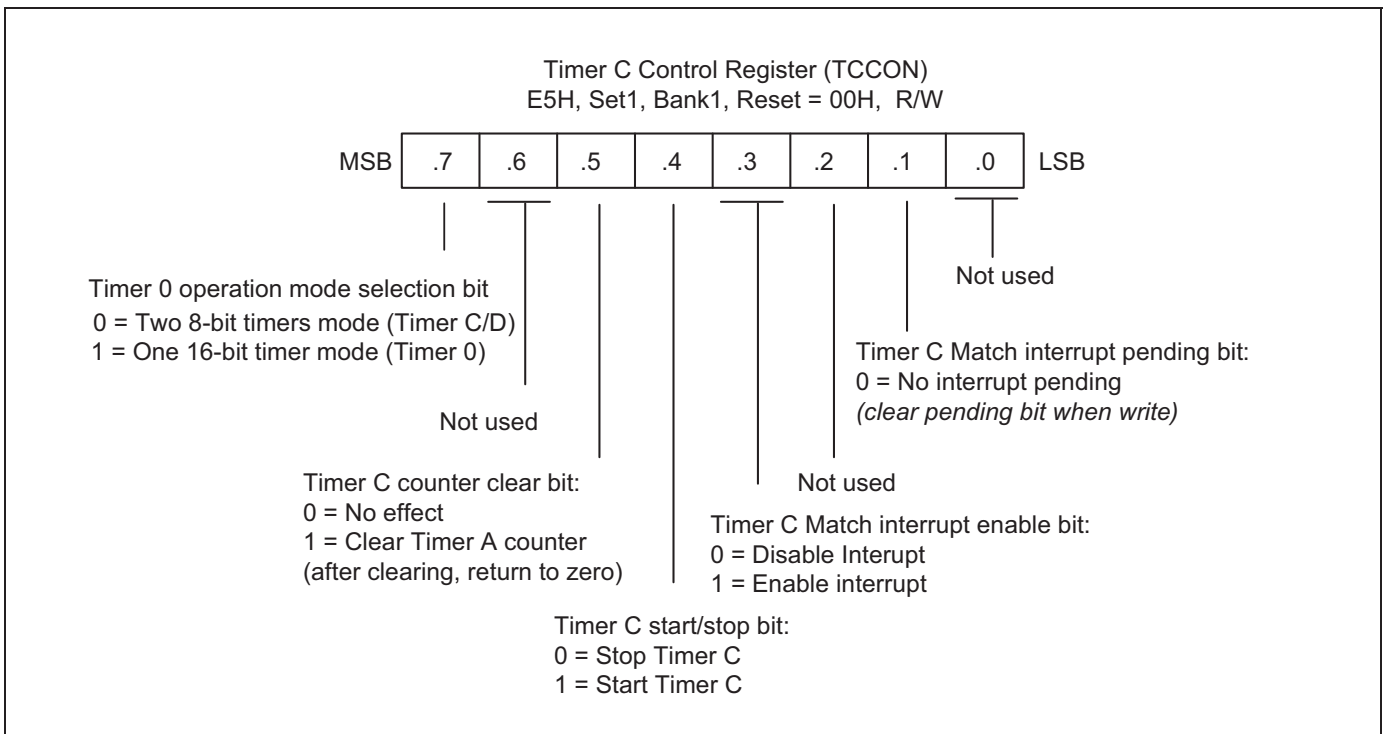


Figure 12-4 Timer C Control Register (TCCON)

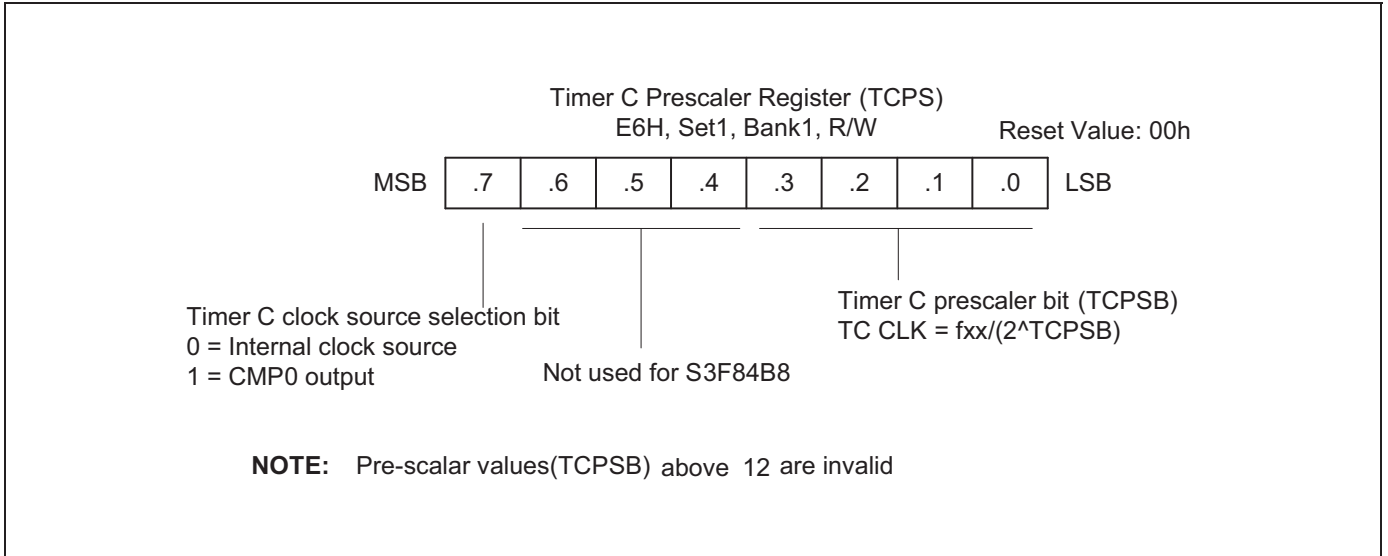


Figure 12-5 Timer C Prescaler Register (TCPS)

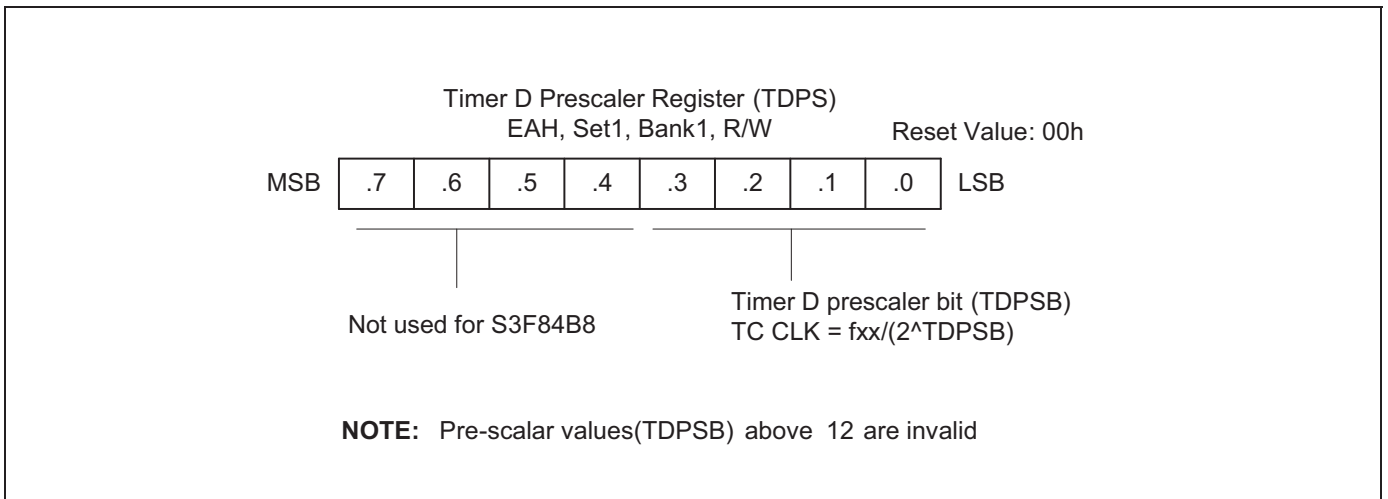


Figure 12-6 Timer D Prescaler Register (TDPS)

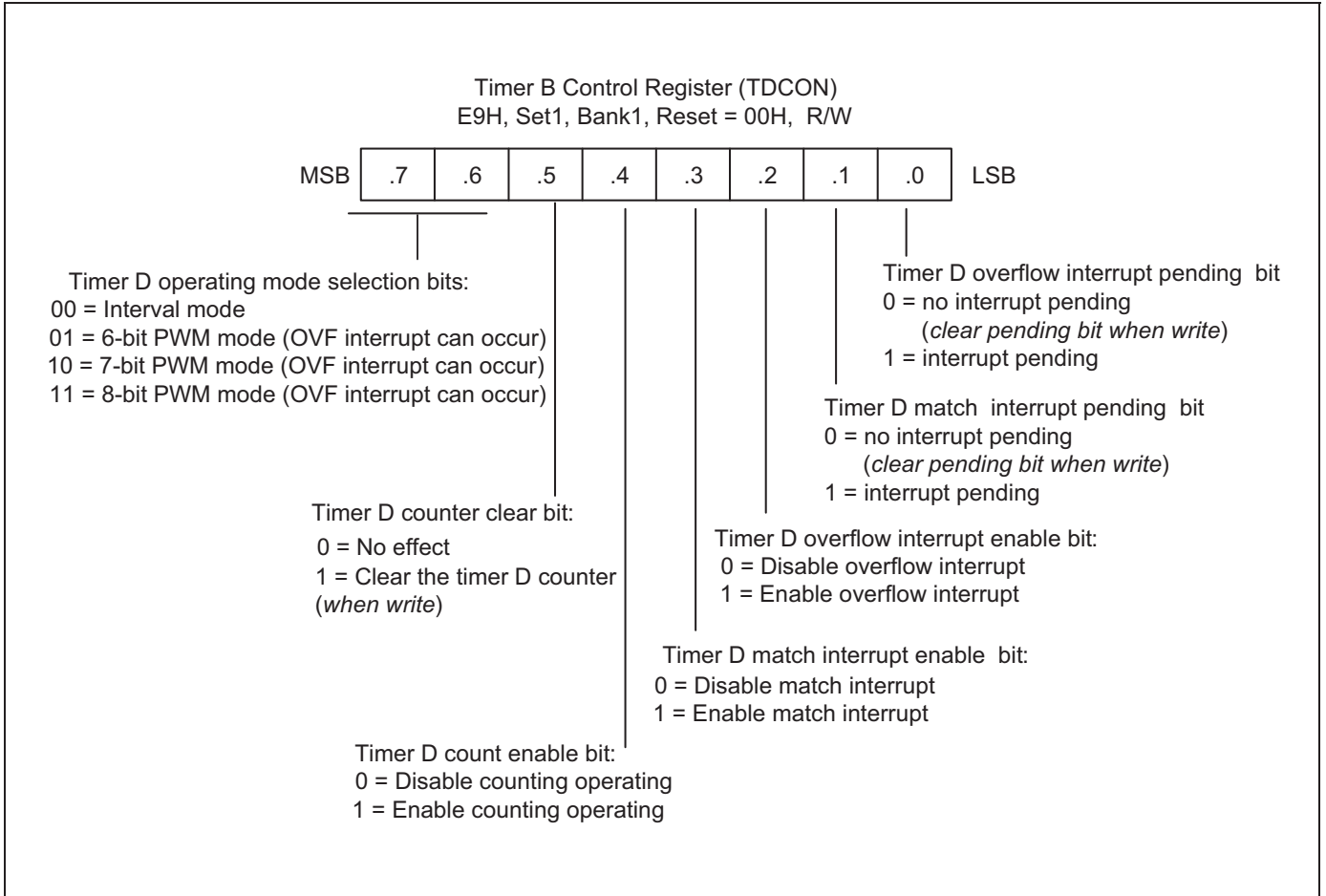


Figure 12-7 Timer D Control Register (TDCON)

12.2.3 FUNCTIONAL DESCRIPTION OF TWO 8-BIT TIMERS MODE (TIMER C AND D)

12.2.3.1 Interval Timer Function (Timers C and D)

Timers C and D module can generate the Timer C match interrupt (TCINT) and Timer D match interrupt (TDINT). Timer C match interrupt pending condition (TCCON.1) and Timer D match interrupt pending condition (TDCON.1) must be cleared by the software in interrupt service routine by means of writing a “0” to TCCON.1 and TDCON.1 interrupt pending bits.

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 Timer C or Timer D reference data registers, TCDATA or TDDATA. The match signal generates corresponding match interrupts (TCINT and TDINT) and clears the counter.

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

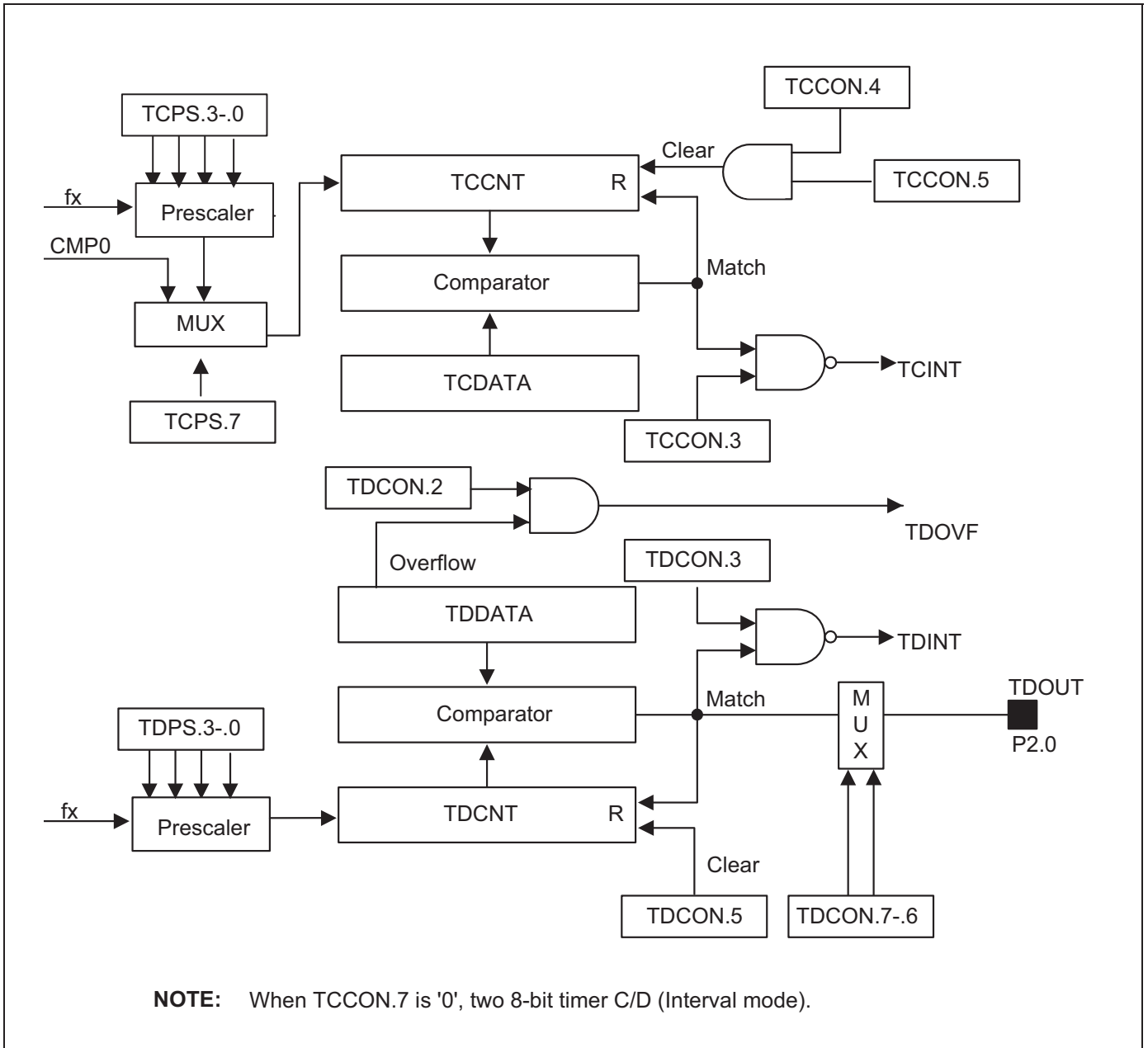


Figure 12-8 Timers C and D Function Block Diagram

12.2.3.2 Pulse Width Modulation Mode (Timer D)

Pulse width modulation (PWM) mode allows you to program the width (duration) of pulse that is outputted 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 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 to increment from “00H”.

Even though 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 TDOUT pin is held to Low level as long as the reference data value is less than or equal to (\leq) the counter value. The pulse is then held to High level as long as the data value is greater than ($>$) the counter value. One pulse width is equal to $t_{CLK} \times 256$ in case 8-bit PWM mode is selected (see [Figure 12-6](#)).

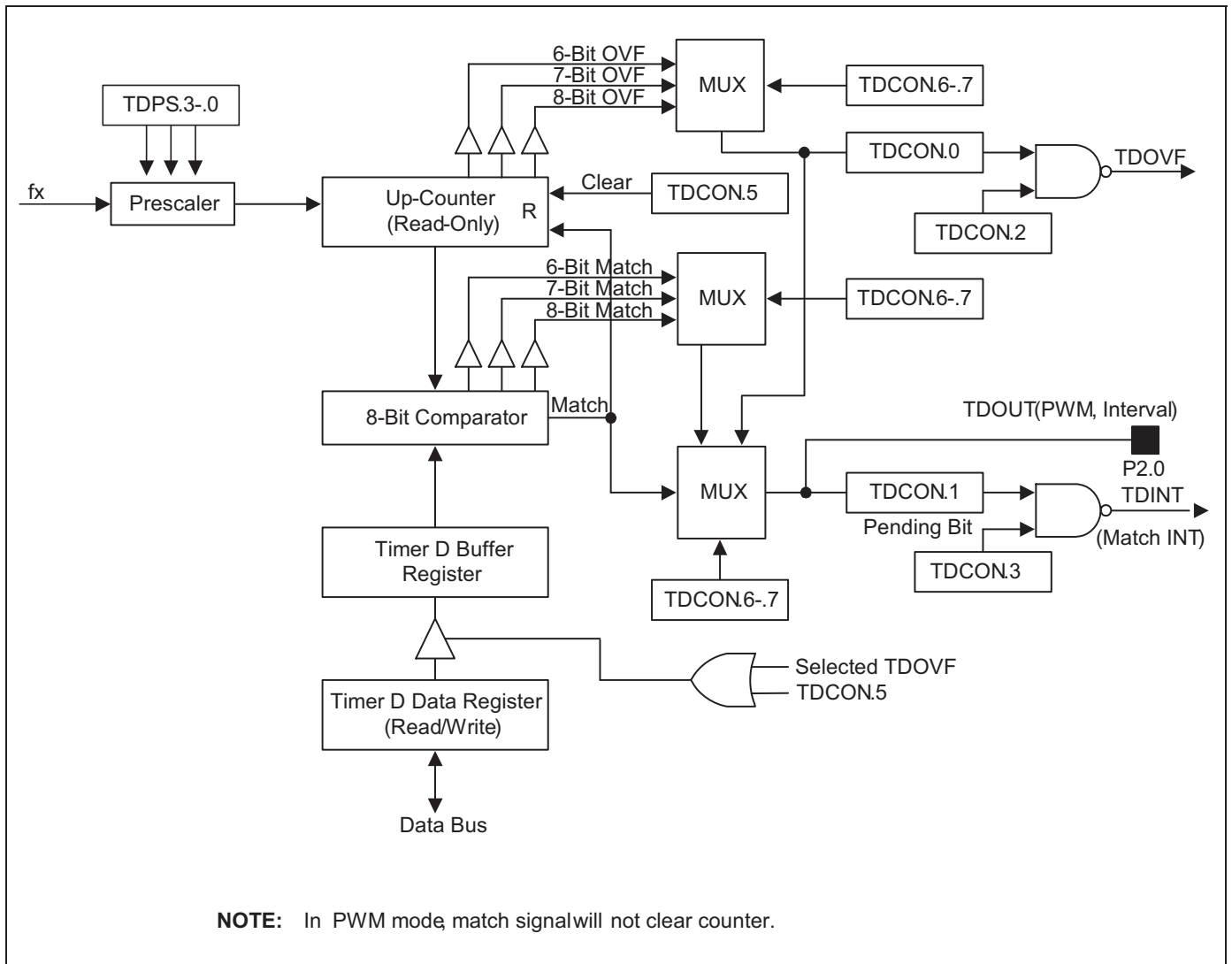


Figure 12-9 Timer D PWM Function Block Diagram

13

A/D CONVERTER

13.1 OVERVIEW OF A/D CONVERTER

The 10-bit analog-to-digital (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. Analog input level must lie between the V_{DD} and V_{SS} values.

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, write the channel selection data in the A/D converter control register (ADCON). This way you can select one of the eight analog input pins (ADC $_n$, $n = 0-7$) and set the conversion start or enable bit (ADCON.0). The read-write ADCON register is located at the FAH address.

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 the conversion is complete, the end-of-conversion (EOC) bit (ADCON.3) is automatically set to 1; the result is dumped into the ADDATA register, where it can be read. If the ADC interrupt is enabled (ADCON.4 = 1), an interrupt request will be generated. The A/D converter then enters an Idle state. The contents of ADDATA must be read before another conversion starts; else the previous result will be overwritten by next conversion result.

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

13.1.1 USING A/D PINS FOR STANDARD DIGITAL INPUT

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

13.1.1.1 A/D Converter Control Register (ADCON)

The A/D converter control register, ADCON, is located at FAH address.

ADCON has five functions:

- Bits 7-5 select an analog input pin (ADC0–ADC7).
- Bit 4 enables/disables the ADC interrupt.
- Bit 3 indicates the status of 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.

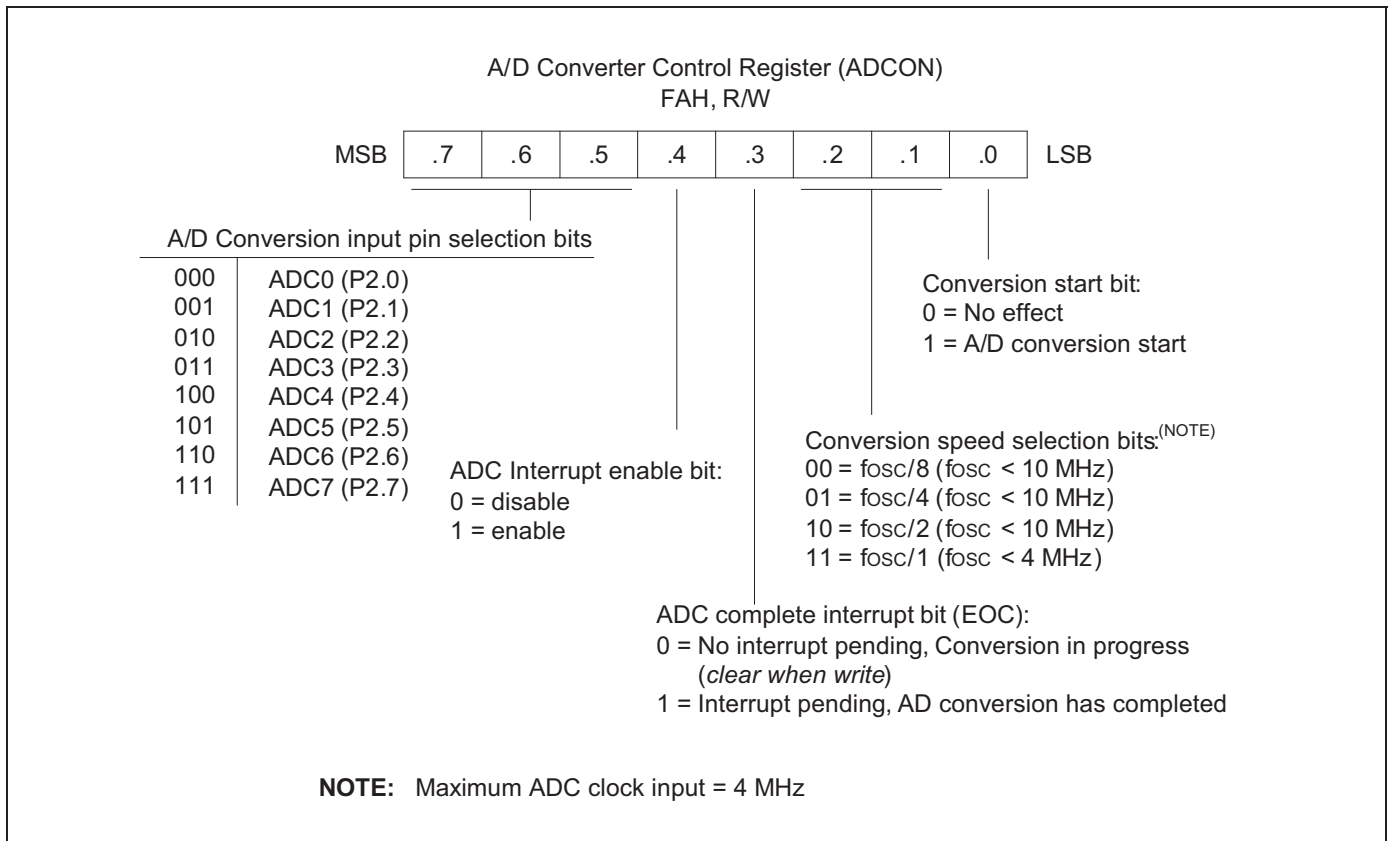


Figure 13-1 A/D Converter Control Register (ADCON)

13.1.2 INTERNAL REFERENCE VOLTAGE LEVELS

In the ADC function block, the analog input voltage level is compared to the reference voltage. The reference voltage is internally connected to VDD in S3F84B8. Thus, the analog input level must remain within the range of VSS to VDD.

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 VDD.

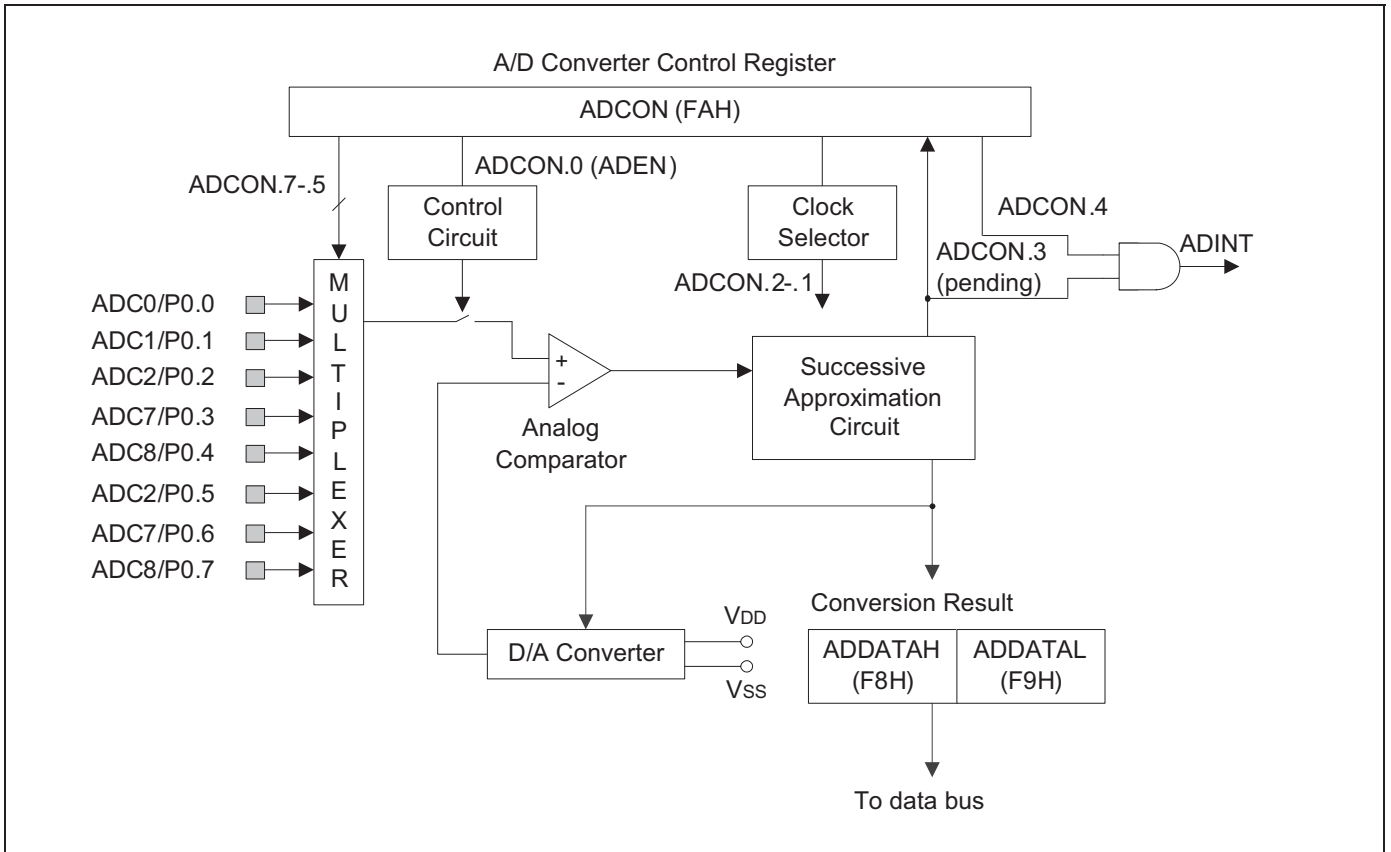


Figure 13-2 A/D Converter Circuit Diagram

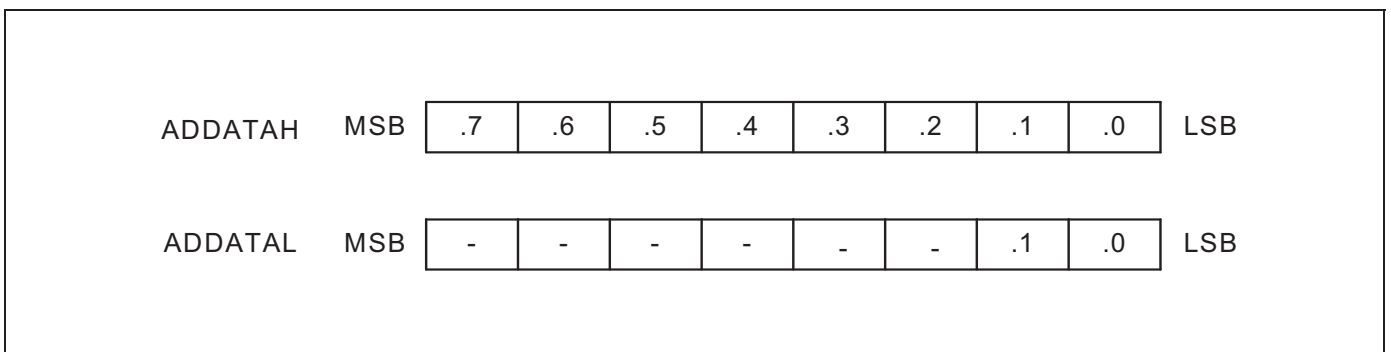


Figure 13-3 A/D Converter Data Register (ADDATAH/L)

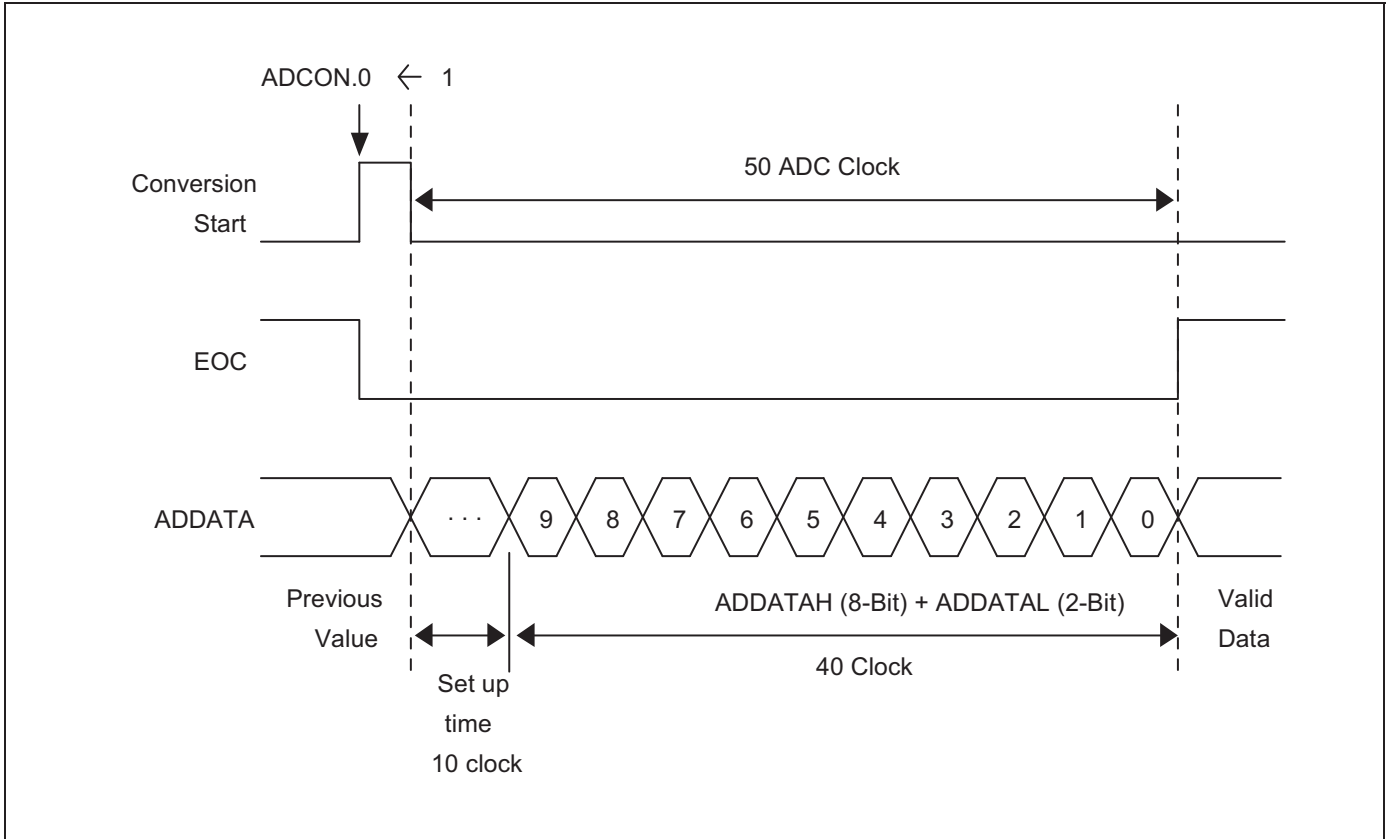


Figure 13-4 A/D Converter Timing Diagram

13.1.3 CONVERSION TIMING

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

$$4 \text{ clocks/bit} \times 10\text{-bits} + \text{step-up time (10 clock)} = 50 \text{ clocks}$$

$$50 \text{ clocks} \times 500\text{ns} = 25\mu\text{s at 8MHz, 1 clock time} = 4/f_{xx} \text{ (assuming ADCON.2-.1} = 01)$$

13.1.4 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 setting the P2CONH and P2CONL registers.
3. Before the conversion operation starts, you must select one of the eight input pins (ADC0-ADC7) by writing the appropriate value to the ADCON register.
4. When conversion is complete (that is 50 clocks have elapsed), the Interrupt pending bit (EOC flag) is set to "1". If ADC interrupt is enabled, a request will be sent to the 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). The ADC module then enters an Idle state.
6. The digital conversion result can now be read from ADDATAH and ADDATAL registers.

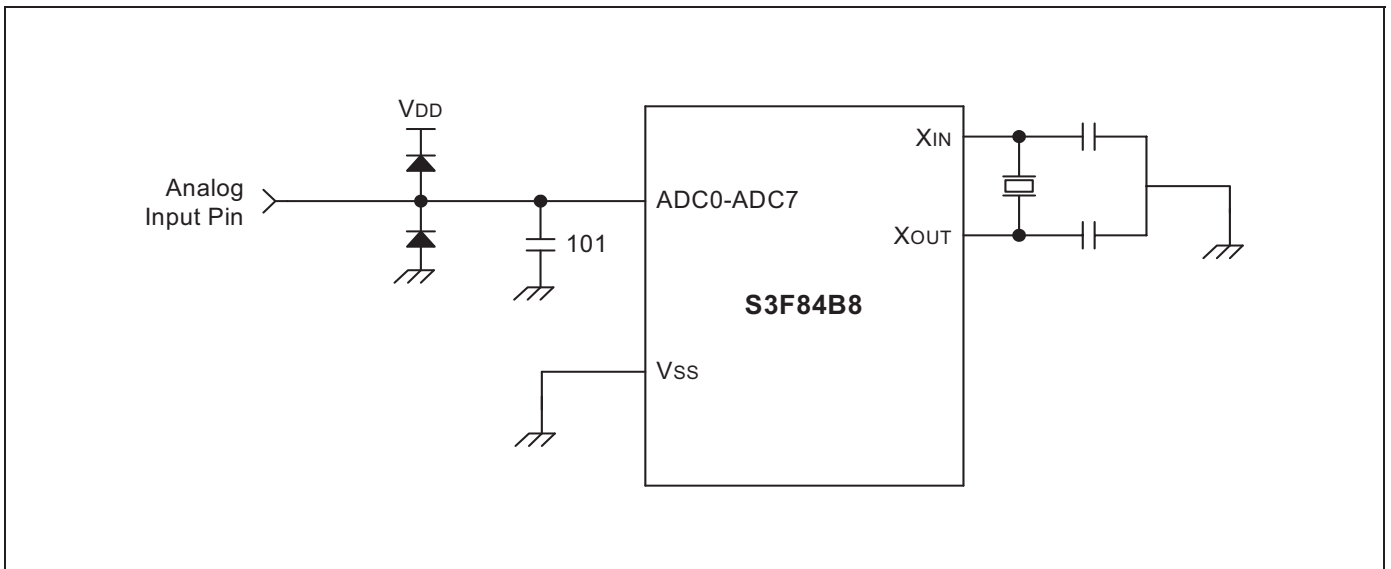


Figure 13-5 Recommended A/D Converter Circuit for Highest Absolute Accuracy

Example 13-1 Configuring A/D Converter

```

;-----<< Interrupt Vector Address >>
      VECTORF0H, 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, disables LVR and internal RC oscillator

      ORG    0100H
RESET:  DI          ; Disables interrupt
        LD    BTCON,#10100011B ; Disables Watchdog
        .
        .
        .
        LD    P2CONH,#11111111B ; Configures P2.4–P2.7 AD input
        LD    P2CONL,#11111111B ; Configures P2.0–P2.3 AD input
        EI          ; Enables interrupt
;-----<< Main loop >>
MAIN:   .
        .
        .
        .
        .
        JR    t, MAIN
AD_CONV: LD    ADCON, #00110001B ; Selects analog input channel → P2.1
        ; Enables ADC interrupt
        ; Selects conversion speed → fOSC/8
        ; Sets conversion start bit

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

INT_ADC:
        LD    R2, ADDATAH      ;
        LD    R3, ADDATAL      ;
AND     ADCON, #11110111B      ; Clears pending bit
        .
        .
        .
        IRET                    ;
        .
        .
        END
  
```

14

COMPARATOR

14.1 OVERVIEW OF COMPARATOR

The S3F84B8 microcontroller has four comparators (Comparator 0, 1, 2, and 3). The operation of these four comparators is controlled by four registers, namely, CMP0CON, CMP1CON, CMP2CON, and CMP3CON. The interrupt control register (CMPINT) controls the interrupt mode of four comparators.

14.1.1 FUNCTIONAL DESCRIPTION OF COMPARATOR

14.1.1.1 Comparator 0

In Comparator 0, both positive and negative inputs act as chip pins. The polarity of comparator 0 output can be set to inverted or non-inverted. You 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. It can have a programmable delay to realize delay trigger by configuring the AMTDATA register, which is useful when realizing timing adjustment.

14.1.1.1.1 Comparator 0 Control Register (CMP0CON)

You can use comparator 0 control register (CMP0CON) for the following purposes:

- 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.

CMP0CON.1 represents the real status of two inputs, read as '0' when $CMP0_N > CMP0_P$ or '1' when $CMP0_N < CMP0_P$.

Comparator 0 can generate an interrupt to indicate the alternation of two input pins. The interrupt trigger mode (rising/falling/rising and falling edge) can be configured in CMPINT register. To enable the interrupt, write '1' in CMP0CON.2. On the other hand, to clear the interrupt pending bit, write '0' to CMP0CON.0. The interrupt pending bit must be cleared by the software.

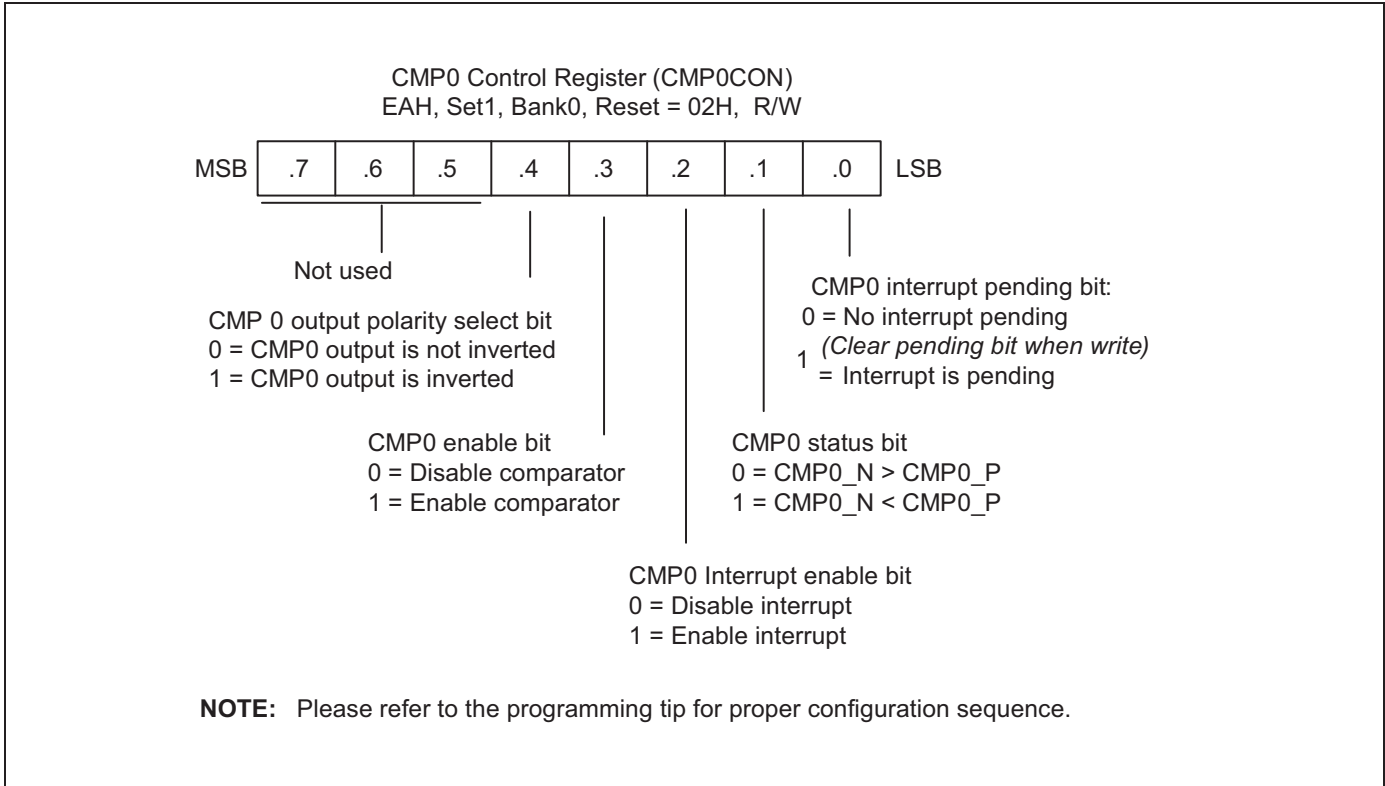


Figure 14-1 CMP0 Control Register (CMP0CON)

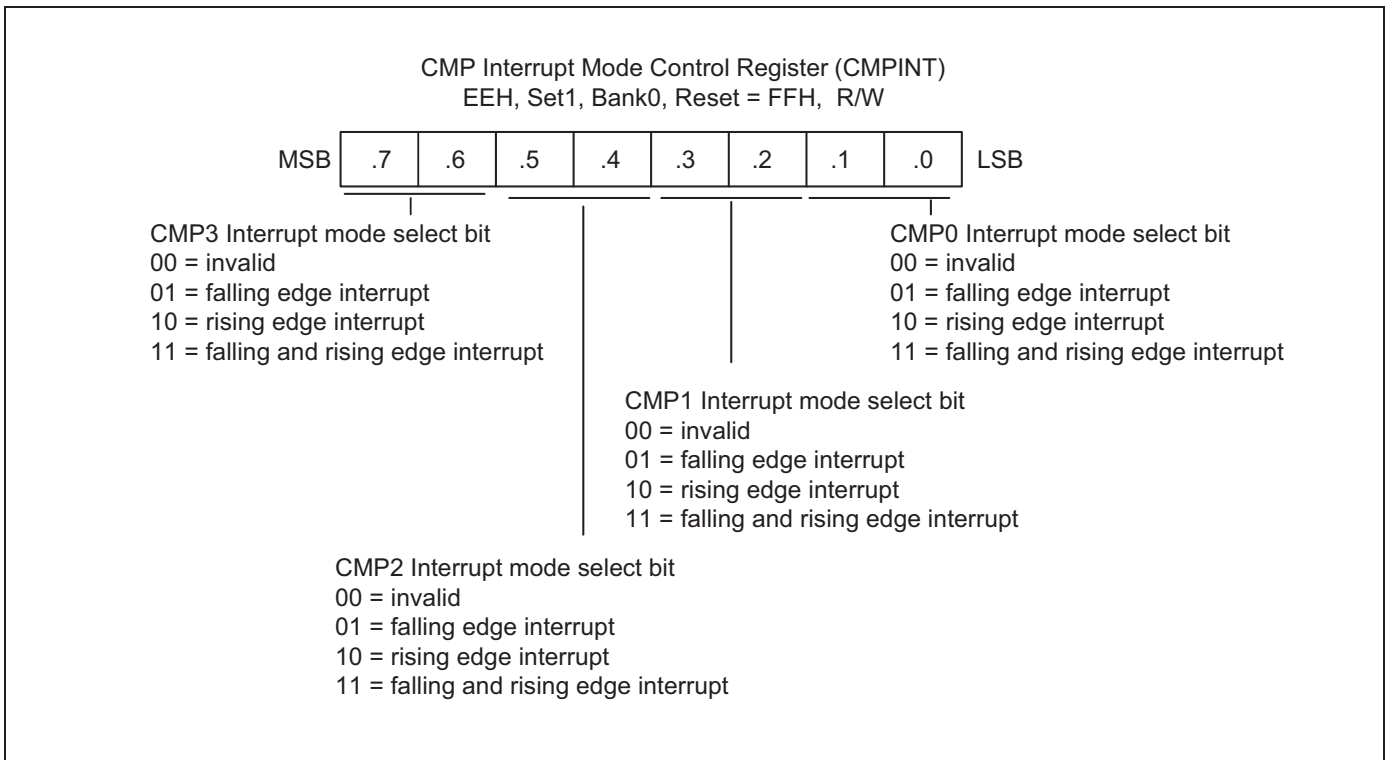


Figure 14-2 CMP Interrupt Mode Control Register (CMPINT)

14.1.1.1.2 Block Diagram of Comparator 0

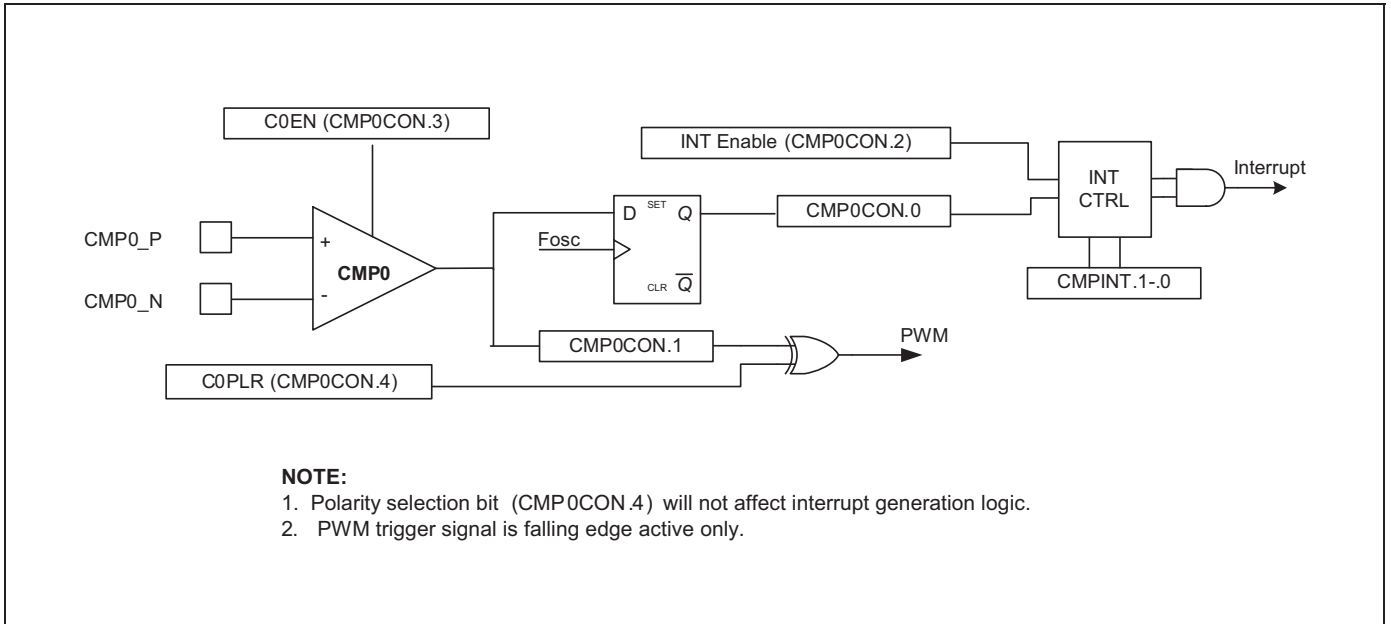


Figure 14-3 Block Diagram of Comparator 0

14.1.1.2 Comparator 1/2/3

Comparator 1, 2, and 3 have 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 trigger signal (PWMCCON.1–.2/.3–0.4/.5/.6 = 11) or soft lock trigger signal (PWMCCON.1–.2/.3–0.4/.5/.6 = 01).

In case of hard lock, PWM output will stop immediately (stop voltage level is determined by PWM output polarity bit, that is, when PWMCON.5 = 0, PWM output is '0' and when PWMCON.5 = 1, PWM output is '1'). To unlock the hard lock, write '1' to PWMCON.3.

On the other hand, in case of soft lock, PWM output will stop immediately (stop voltage level is determined by PWM output polarity bit, that is, when PWMCON.5 = 0, PWM output is '0' and when PWMCON.5 = 1, PWM output is '1'). The PWM output will then reload PWMDATA with PWMPDATA. Soft lock will be automatically unlocked in the next PWM cycle.

14.1.1.2.1 Comparator Control Register (CMP1CON, COM2CON, CMP3CON)

You can use comparator control registers for the following purposes:

- 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. The positive input of comparator is internally connected with reference voltage, programmable from 0.45VDD to 0.8VDD with step length of 0.05VDD.

The output polarity is programmable by configuring CMP1/2/3CON.4.

CMP1/2/3CON.1 represents the real status of two inputs, read as '0' when CMP1/2/3_N > reference voltage or '1' when CMP1/2/3_N < reference voltage.

Comparator 1/2/3 can generate an interrupt to indicate the alternation of two input pins. You can choose the falling edge, rising edge, or falling and rising edge to trigger the comparator interrupt by configuring CMPINT register. To enable the interrupt, write '1' in CMP1/2/3CON.2. On the other hand, to clear the interrupt pending bit, write '0' to CMP1/2/3CON.0. The interrupt pending bit must be cleared by the software.

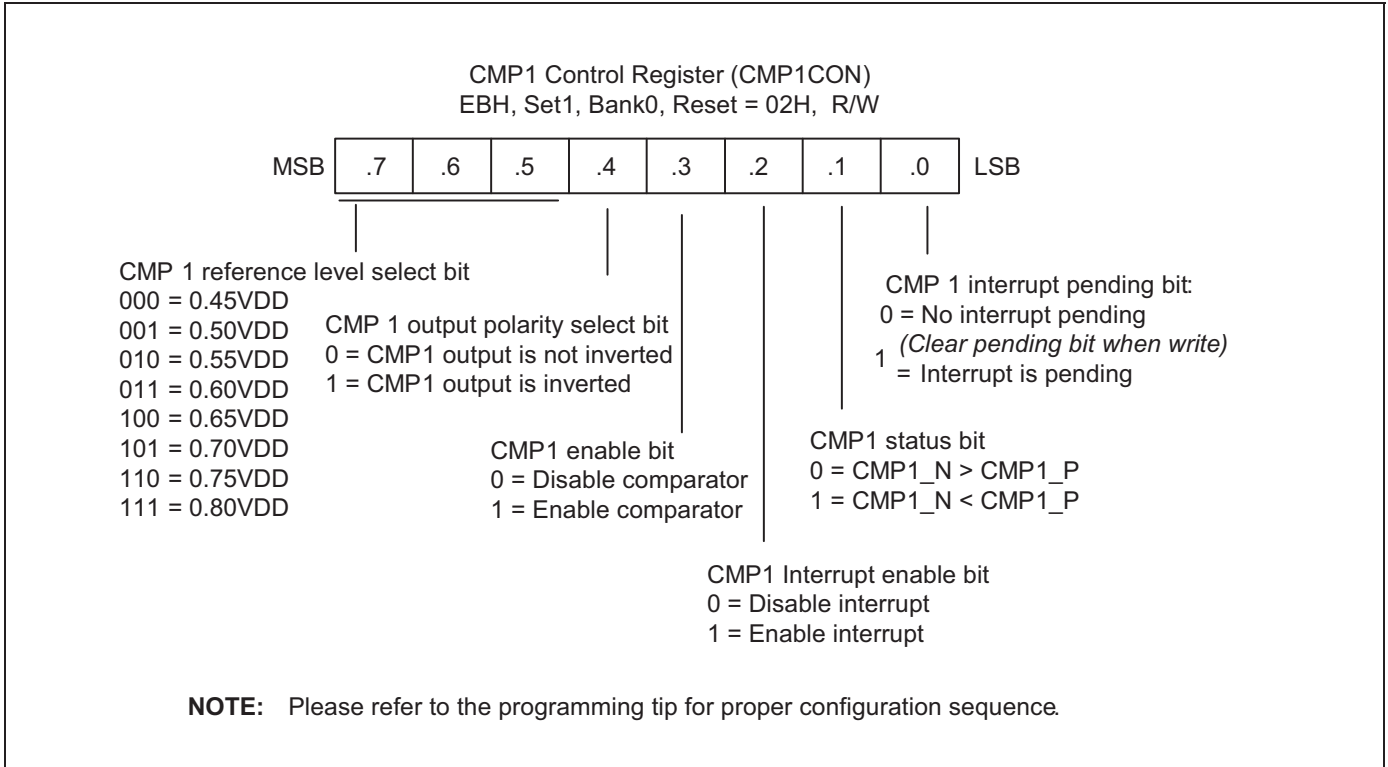


Figure 14-4 CMP1 Control Register (CMP1CON)

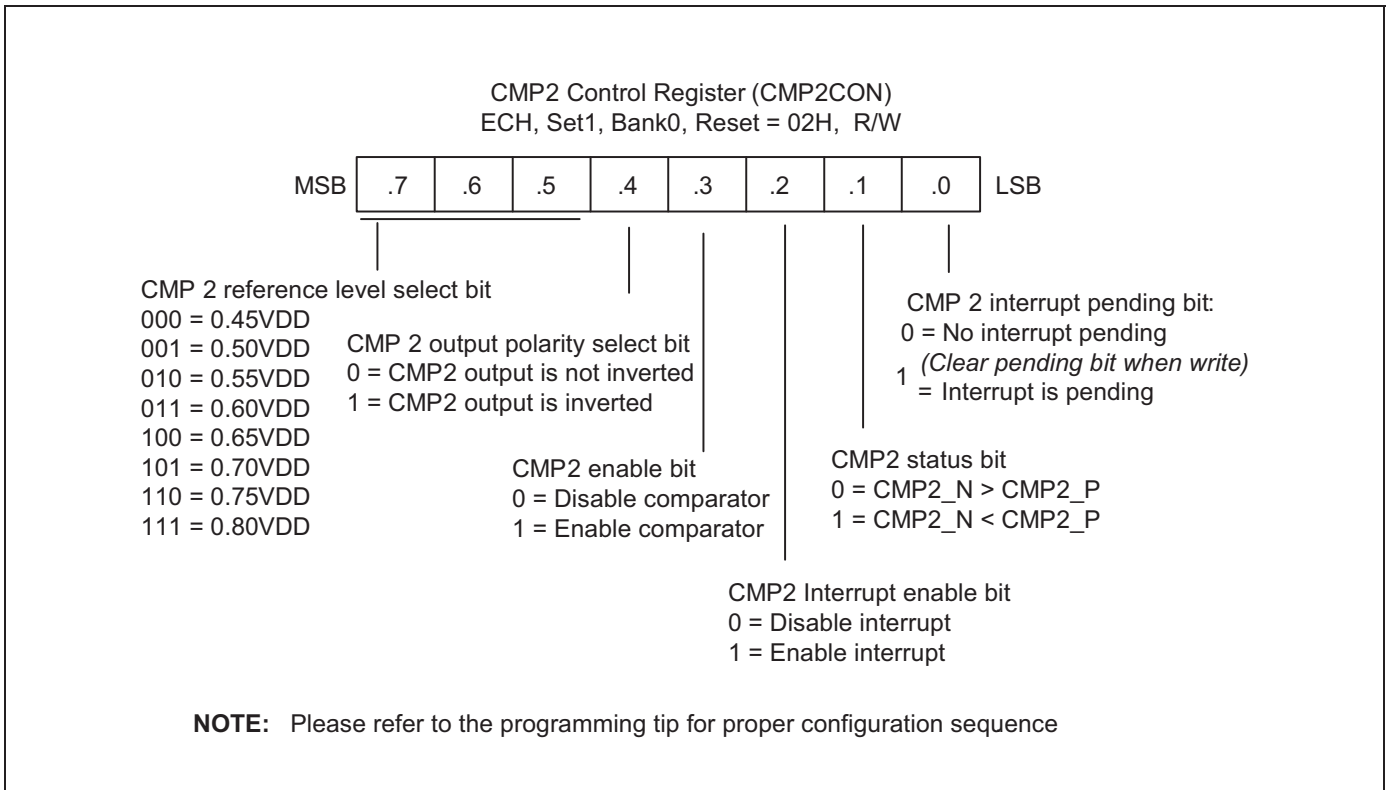


Figure 14-5 CMP2 Control Register (CMP2CON)

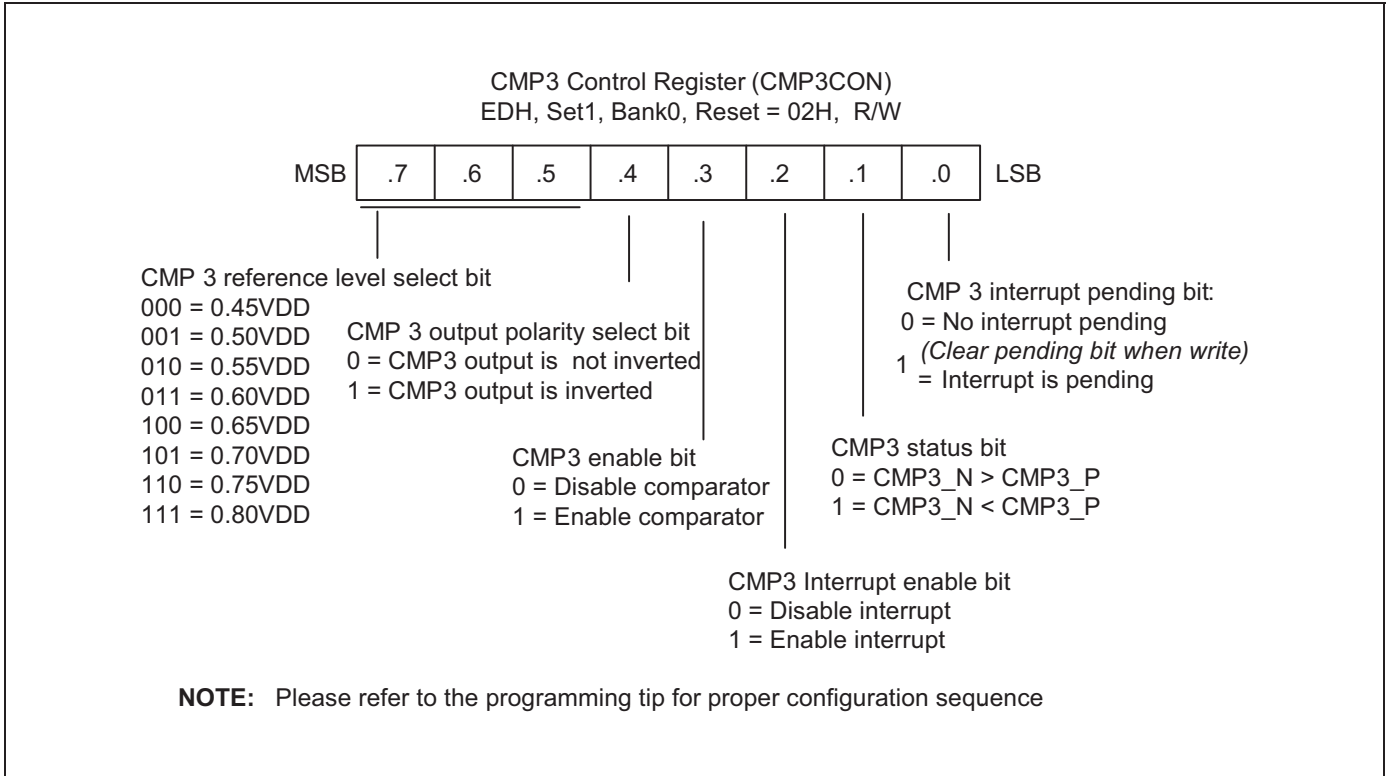


Figure 14-6 CMP3 Control Register (CMP3CON)

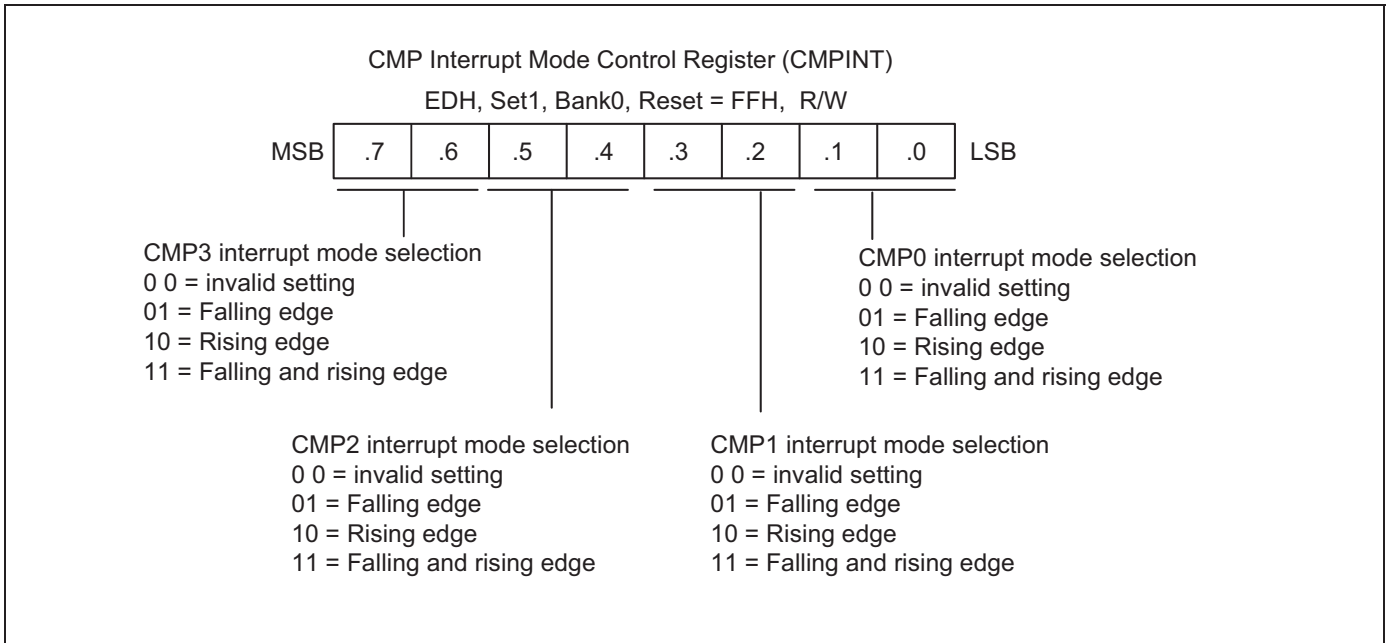


Figure 14-7 CMP Interrupt Mode Control Register (CMPINT)

14.1.1.2.2 Block Diagram of Comparator 1/2/3

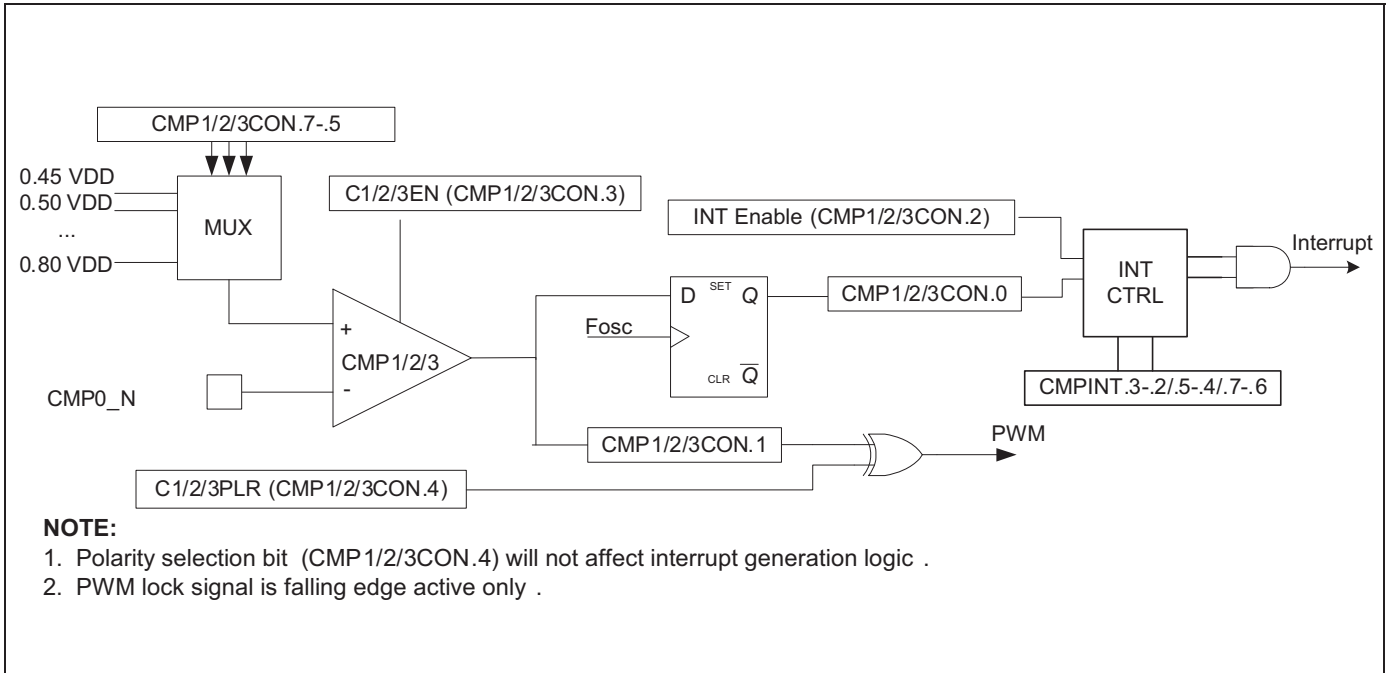


Figure 14-8 Block Diagram of Comparator 1/2/3

Example 14-1 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    ; Enables CMP, enables interrupt
EI
•
•

```

15

OPERATIONAL AMPLIFIER

15.1 OVERVIEW OF OPERATIONAL AMPLIFIER

The S3F84B8 microcontroller has an Operational Amplifier (OP AMP), which is controlled by a control register (OPACON).

15.1.1 FUNCTIONAL DESCRIPTION OF OPERATIONAL AMPLIFIER

The OP AMP has two operation modes, namely, on chip mode and off chip mode.

On chip mode: Positive input is internally connected to the 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 amplifier or a non-inverting amplifier.

15.1.2 OPAMP CONTROL REGISTER

You can use the OPAMP control register, OPACON, for the following purposes:

- 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.

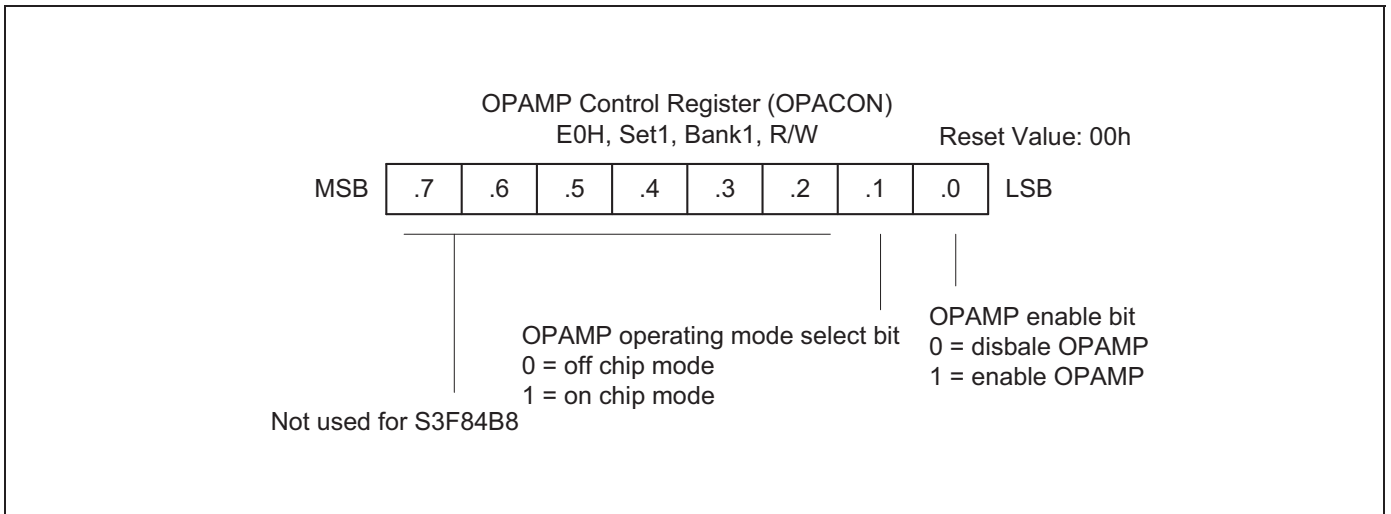


Figure 15-1 OPAMP Control Register (OPACON)

15.1.3 BLOCK DIAGRAM OF OPAMP

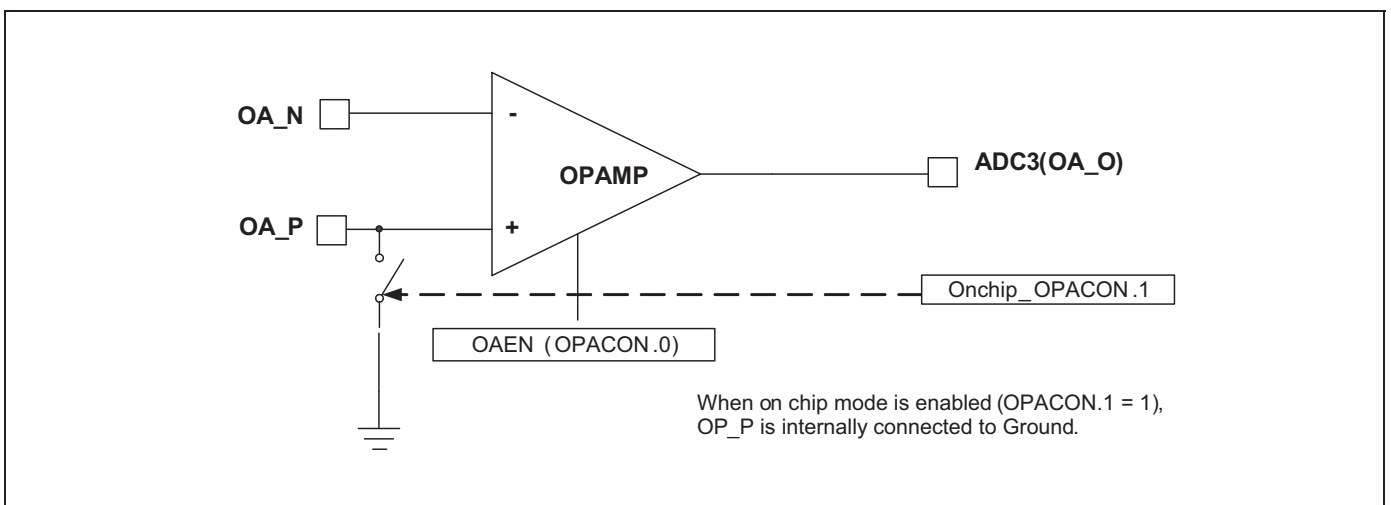
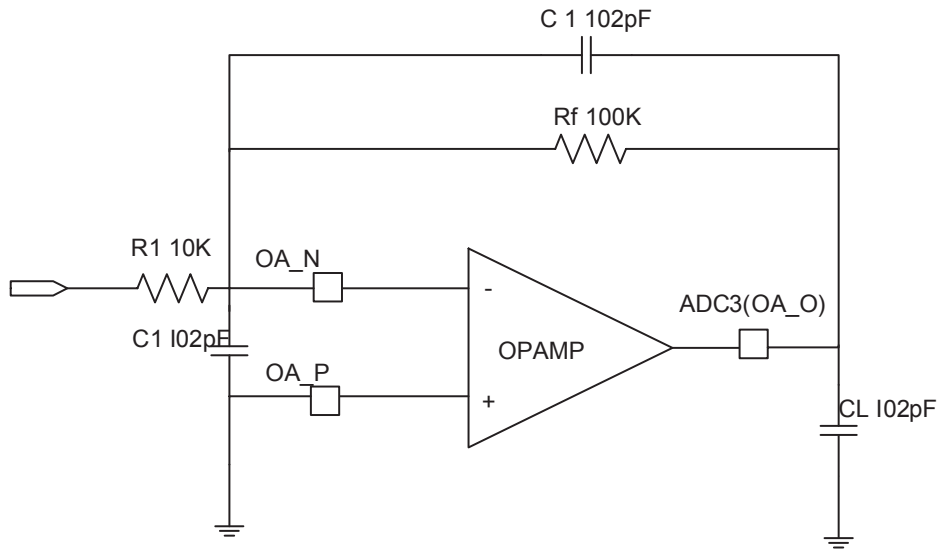


Figure 15-2 Block Diagram of OPAMP

15.1.4 REFERENCE CIRCUIT



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

16

10-BIT IH-PWM

16.1 OVERVIEW OF 10-BIT IH-PWM

The S3F84B8 microcontroller has a 10-bit IH-PWM circuit that can cooperate with the comparators. This circuit is exclusively designed for the IH cooker application.

The operation of all PWM circuits is controlled by a control register (PWMCON). The linkage of comparators and PWM is controlled by another control register (PWMCCON).

PWM can work in the following modes:

- Normal 10-bit PWM mode (When all the linkages with comparators are disabled)
- Comparator-cooperation mode

In comparator-cooperation mode, the PWM circuit can perform the following functions:

- Delay trigger
- Anti-mis-trigger
- Hard/soft lock

16.2 FUNCTIONAL DESCRIPTION OF 10-BIT IH-PWM

16.2.1 PWM

The 10-bit PWM circuit has 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)

16.2.2 PWM CLOCK RATE

The timing characteristic of PWM output is based on the f_{OSC} clock frequency. Additionally, the PWM counter clock value is determined by setting PWMCON.6–.7.

Table 16-1 PWM Control and Data Registers

Register Name	Mnemonic	Address	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
	PWMPDATAH	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, and so on
PWM CMP Register	PWMCCON	F0H	Set1, Bank0	PWM CMP Linkage Settings

16.2.3 PWM FUNCTIONAL 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, if linkage is enabled (PWMCCON.6/4/2/0 = 1), the PWM will work according to the outputs of four comparators. If all the comparators do not generate valid trigger signals, the PWM will work as normal 10-bit PWM.

For comparator0, the output falling edge will clear PWM counter. It will restart one PWM cycle immediately (maximum delay = $4/f_{PWM}$) or after some programmable delay period (known as delay trigger function; enabled when PWMCCON.0 = 1). The delay period is programmable through PWMDL register.

Anti-mis-trigger function can be used to prevent the 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 start to up count at PWM clock. Before match happens, signals from Comparator 0 will be neglected. Thus, they will not trigger PWM to start another new cycle.

For comparator1, 2, and 3, the output falling edge will either directly stop the PWM (hard lock), or stop the current PWM cycle and restart PWM when the next cycle begins with a preset PWM data called PWMPDATA (soft lock).

To avoid invalid trigger or lock, register PWMCCON must be set to appropriate value before enabling PWM module.

You can select a clock for the PWM counter by setting PWMCON.6–7. Clocks that you can select are $f_{OSC}/64$, $f_{OSC}/8$, $f_{OSC}/2$, $f_{OSC}/1$.

16.2.4 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:

- Selects the PWM counter clock
- Selects the PWM output polarity
- Clears the PWM counter
- Disables/enables/resumes PWM counter operation
- Selects the Anti-Mis-Trigger function
- Controls the PWM counter overflow interrupt

A reset clears all the PWMCON bits to logic zero, disabling the entire PWM module.

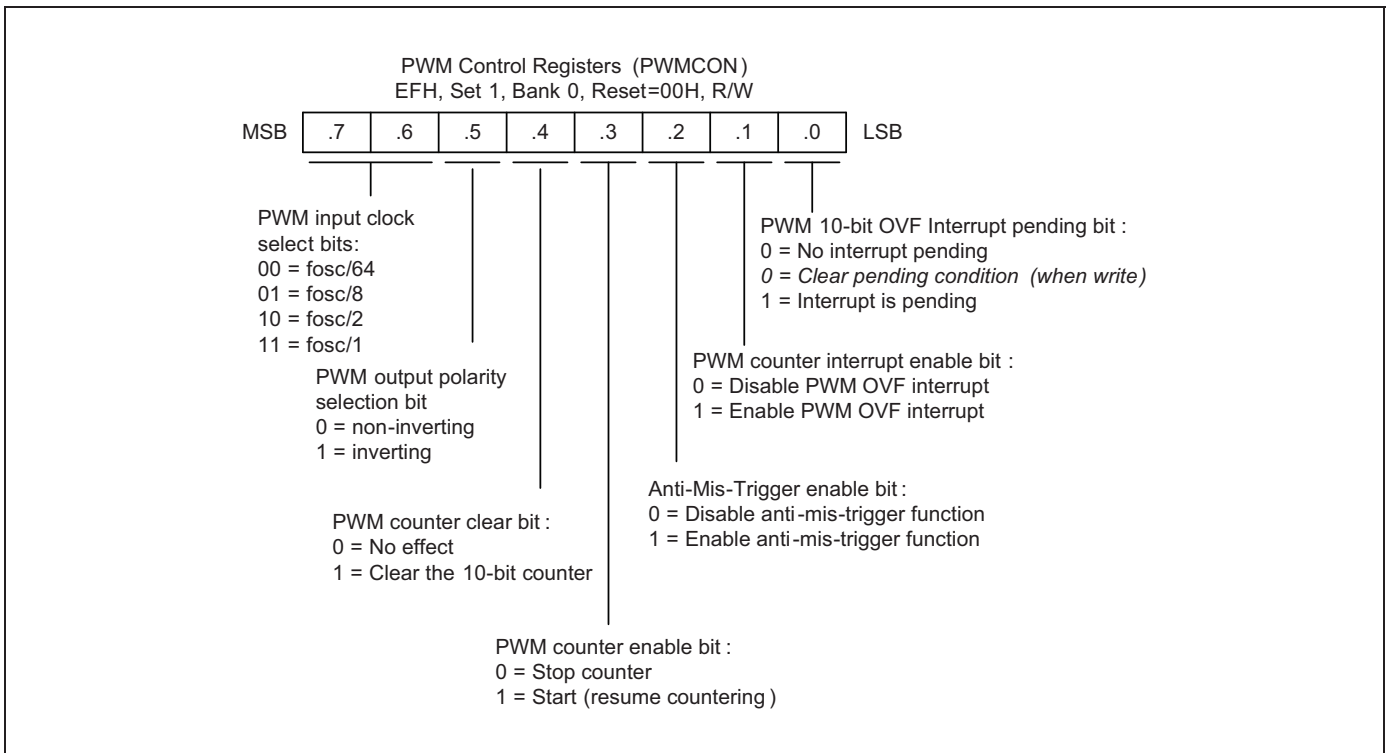


Figure 16-1 PWM Module Control Register (PWMCON)

16.2.5 PWM CMP LINKAGE CONTROL REGISTER (PWMCCON)

The control register for 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 linkage configuration of PWM CMP0, PWM CMP1, PWM CMP2, and PWM CMP3.

A reset clears all the PWMCCON bits to logic zero, disabling the entire linkage.

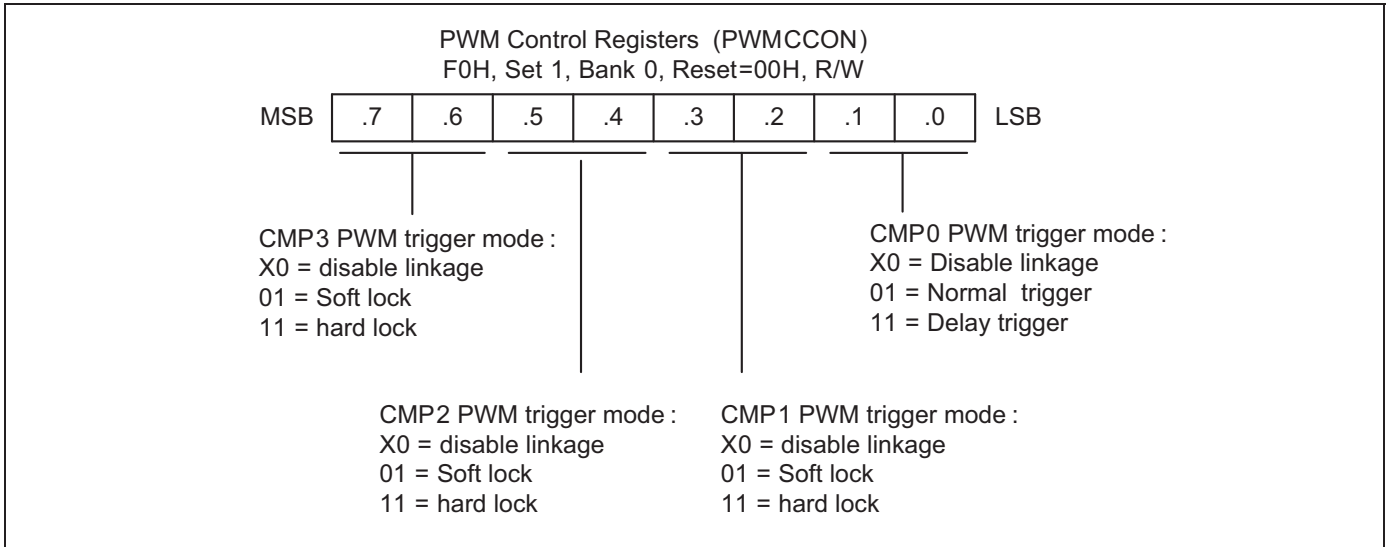


Figure 16-2 PWM CMP Linkage Control Register (PWMCCON)

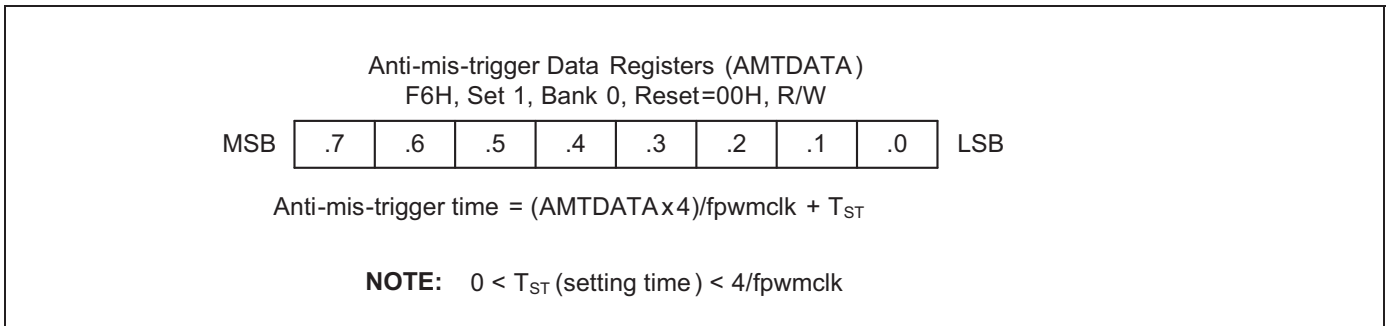


Figure 16-3 Anti-mis-trigger Data Register (AMTDATA)

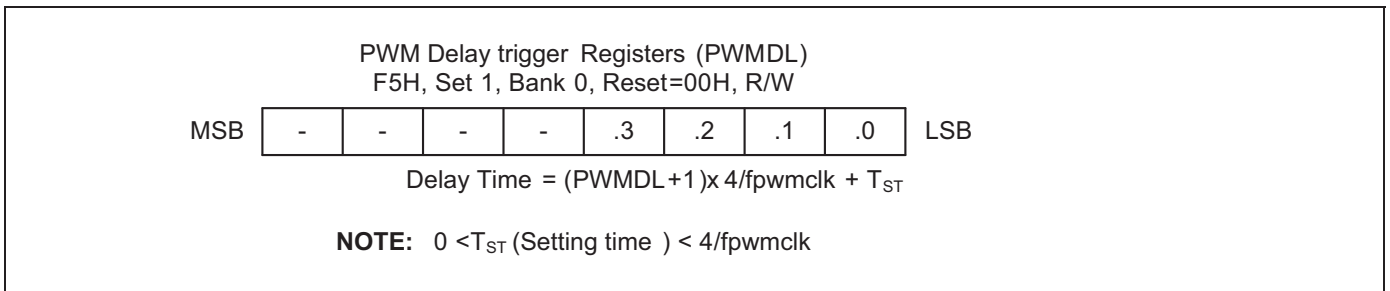


Figure 16-4 Delay trigger Data Register (PWMDL)

16.2.6 BLOCK DIAGRAM OF PWM MODULE

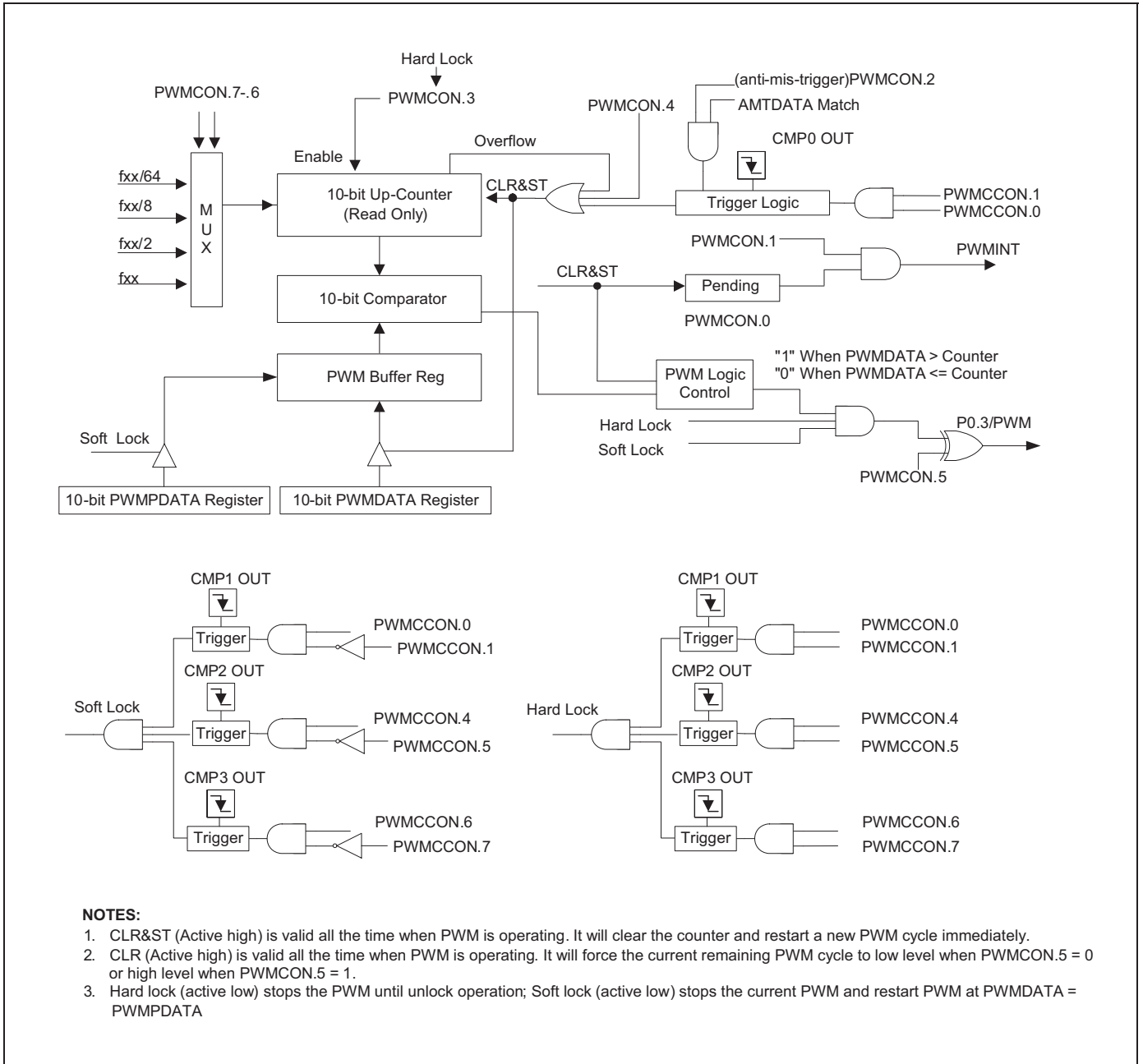


Figure 16-5 Functional Block Diagram of PWM Module

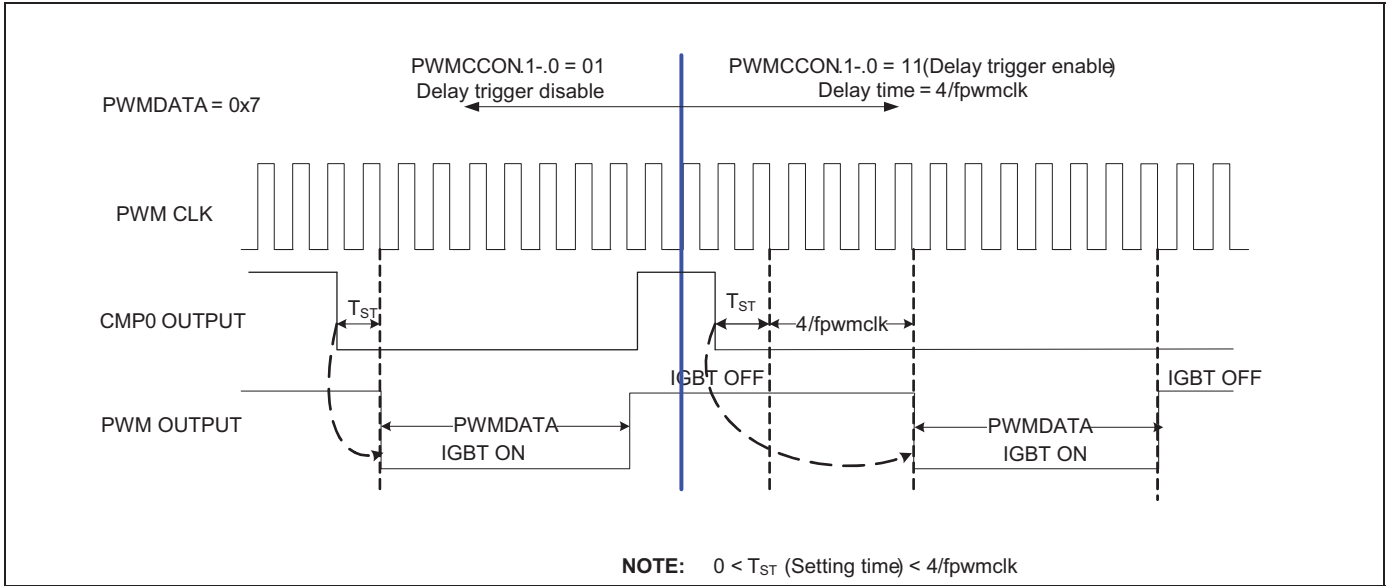


Figure 16-6 Example of the cooperation of PWM and Comparator 0_Delay Trigger

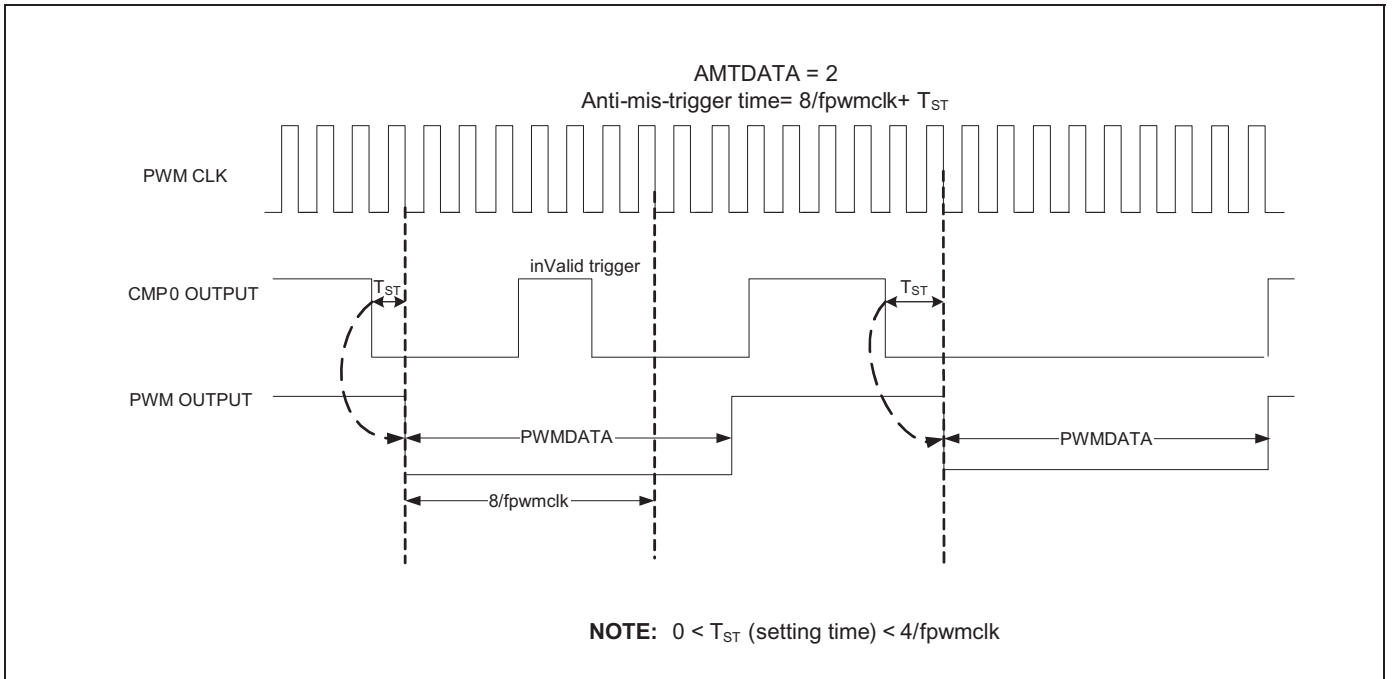


Figure 16-7 Example of the cooperation of PWM and Comparator 0_Anti-mis-Trigger

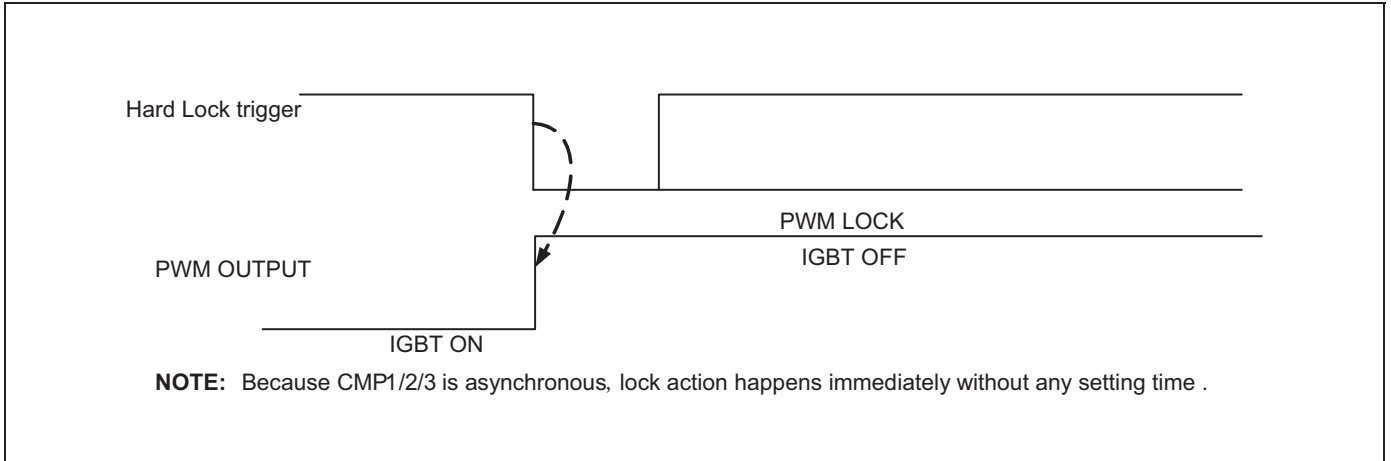


Figure 16-8 Example of the Cooperation of PWM and Comparator 1/2/3_ Hard Lock

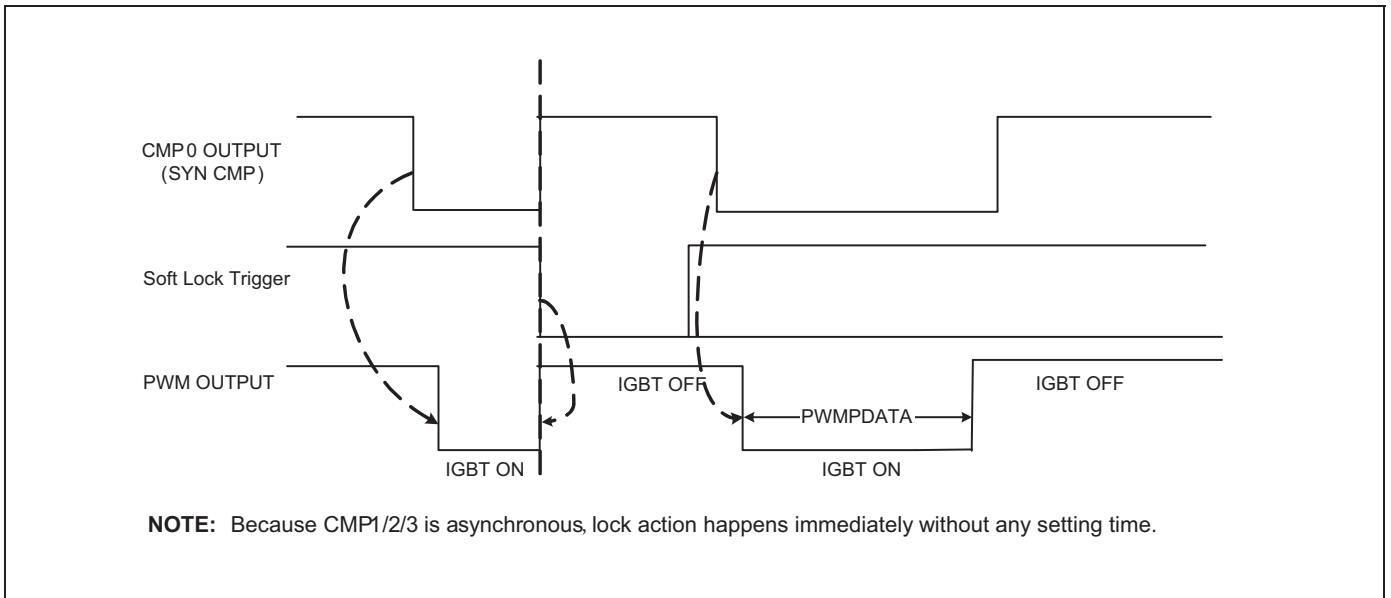


Figure 16-9 Example of the Cooperation of PWM and Comparator 1/2/3_ Soft Lock

17 PROGRAMMABLE BUZZER

17.1 OVERVIEW OF PROGRAMMABLE BUZZER

The S3F84B8 microcontroller has a built-in programmable buzzer, whose operation is controlled by a single control register, BUZCON.

17.2 FUNCTIONAL DESCRIPTION OF PROGRAMMABLE BUZZER

The buzzer's output in S3F84B8 is a square wave with wide frequency range.

- 0.488kHz – 125kHz @ $f_{OSC} = 4\text{MHz}$

17.2.1 BUZ CONTROL REGISTERS (BUZCON)

You can use the BUZ control register, BUZCON, for the following purposes:

- Enable BUZ
- Select input clock frequency
- Program output frequency

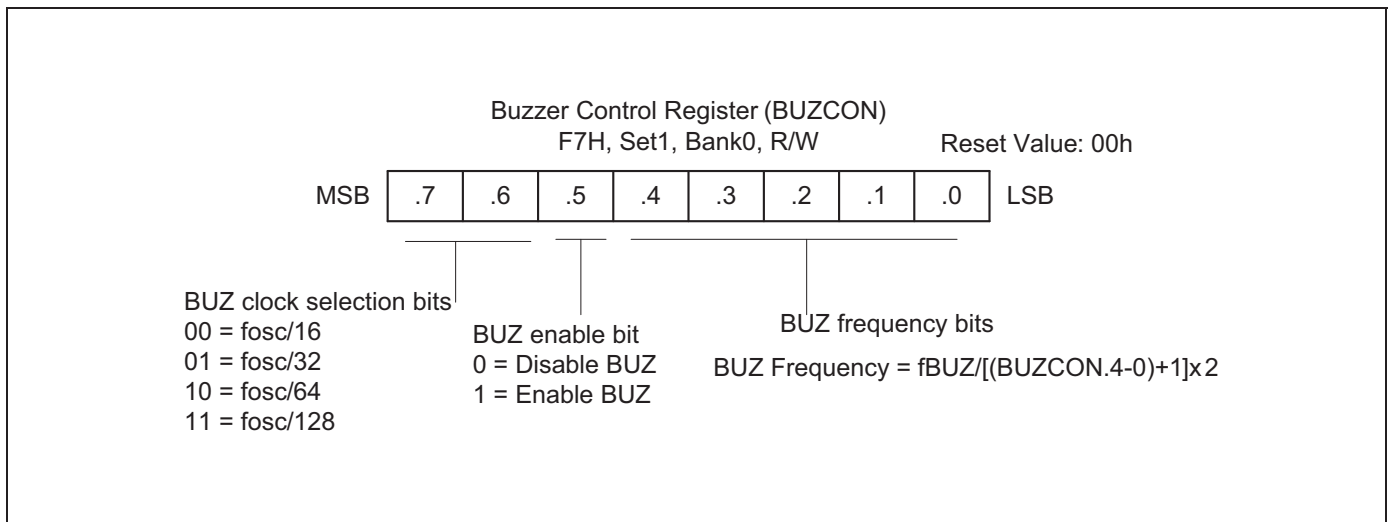


Figure 17-1 Buzzer Control Register (BUZCON)

17.2.2 BUZ FREQUENCY TABLE (@4MHZ)

Table 17-1 Buzzer Frequency Table (@4MHz)

BUZCON .4-.0	Output Frequency (kHz)				BUZCON .4-.0	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

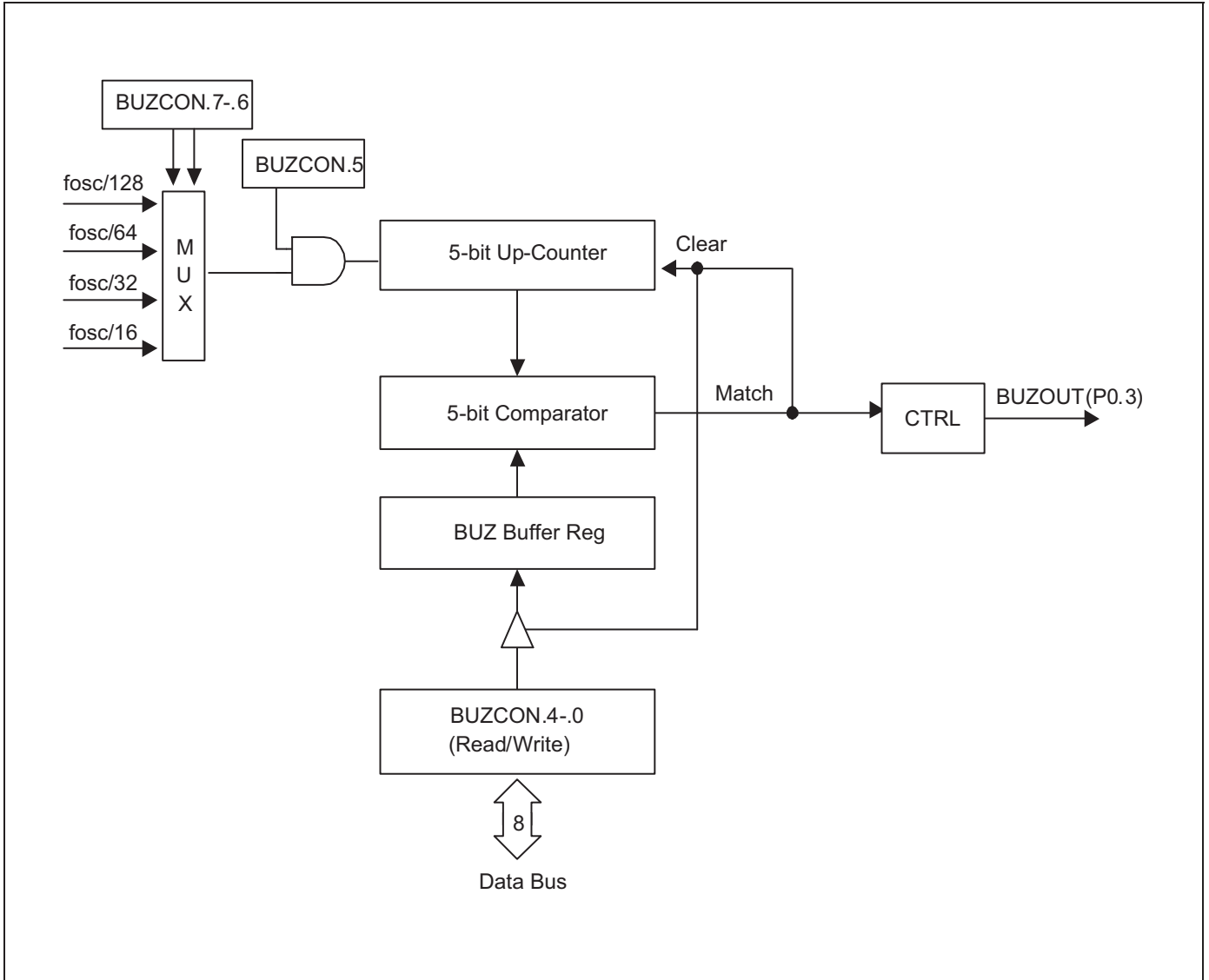


Figure 17-2 BUZ Functional Block Diagram

18 FLASH MCU ROM

18.1 OVERVIEW OF FLASH MCU ROM

The S3F84B8 single-chip CMOS microcontroller has an on-chip Flash MCU ROM that can be accessed by serial data format.

NOTE: This section only discusses about the Tool Program Mode of Flash MCU ROM. For more details about the User Program Mode, refer to Chapter 19, “Embedded Flash Memory Interface”.

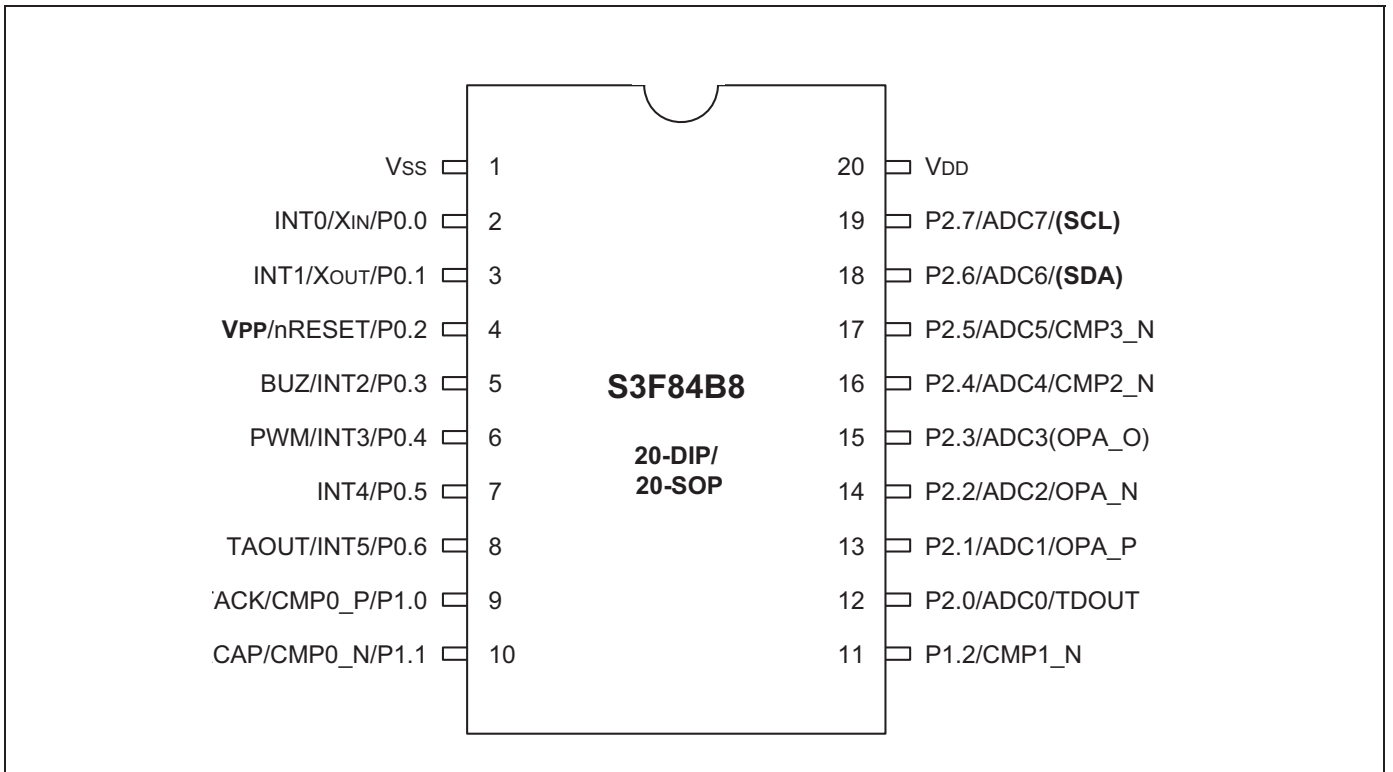


Figure 18-1 Pin Assignment Diagram (20-Pin SOP/DIP Package)

Table 18-1 Descriptions of Pins Used to Read/Write the Flash ROM

Main Chip	During Programming			
Pin Name	Pin Name	Pin Number	I/O	Function
P2.6	SDAT	18	I/O	Serial data pin. Specifies the output port while reading and input port while writing.
P2.7	SCLK	19	I	Serial clock pin.
RESET/P0.2	VPP	4	I	Power supply pin for Flash ROM Cell Writing. Using this pin, the MTP can enter into the writing mode. If 11 V is applied, the MTP enters into the Tool Program 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. Therefore, this pin must not be directly connected to Vpp (12.5V) generated from some OTP/MTP writer. A specific adapter board for S3F84B8 must be used while using these OTP/MTP writers.

19

EMBEDDED FLASH MEMORY INTERFACE

19.1 OVERVIEW OF EMBEDDED FLASH MEMORY INTERFACE

S3F84B8 microcontroller supports an internal (on-chip) flash memory instead of a masked ROM. The sector erasable and byte programmable flash memory in S3F84B8 can also be programmed by the user using 'LDC' instruction. You can program the data in flash memory area at any time.

The embedded 8KB memory in S3F84B8 has two operating modes, namely:

- User Program Mode
- Tool Program Mode

NOTE: For more information about Tool Program Mode, refer to the Chapter 18, "Flash MCU".

19.1.1 FLASH ROM CONFIGURATION

The flash memory in S3F84B8 consists of 64 sectors. Each sector, in turn, consists of 128 bytes. So, the total size of flash memory is 128×64 bytes (8KB). You can erase the flash memory by a sector unit at any time, and even write data into the flash memory by a byte unit at any time.

19.1.2 KEY FEATURES OF EMBEDDED FLASH MEMORY INTERFACE

The key features of embedded flash memory interface include:

- 8kB internal flash memory
- Sector size: 128 bytes
- 10 years of data retention
- Fast programming Time: Sector Erase: 4ms (minimum) and Byte Program: 20us (minimum)
- Byte programmable
- User programmable by 'LDC' instruction
- Sector erase (128 bytes)
- External serial programming
- Endurance: 10,000 Erase/Program cycles (minimum)
- Expandable On Board Program (OBP)

19.1.3 USER PROGRAM MODE

This mode supports sector erase, byte programming, byte read, and protection mode (Hard Lock Protection). S3F84B8 has an internal pumping circuit to generate high voltage. Therefore, there is no need to supply high programming voltage to the Vpp (Test) pin. To program flash memory in this mode, several control registers are used.

19.1.4 SMART OPTION

Smart option specifies the Program Memory option for starting condition of the chip. The Program Memory Addresses used by the Smart option range from 003CH to 003FH. However, S3F84B8 only uses 003FH. The default value of Smart option bits in Program Memory is 0FFH. Before executing the Program Memory code, you can set the Smart option bits according to the hardware option you want to select.

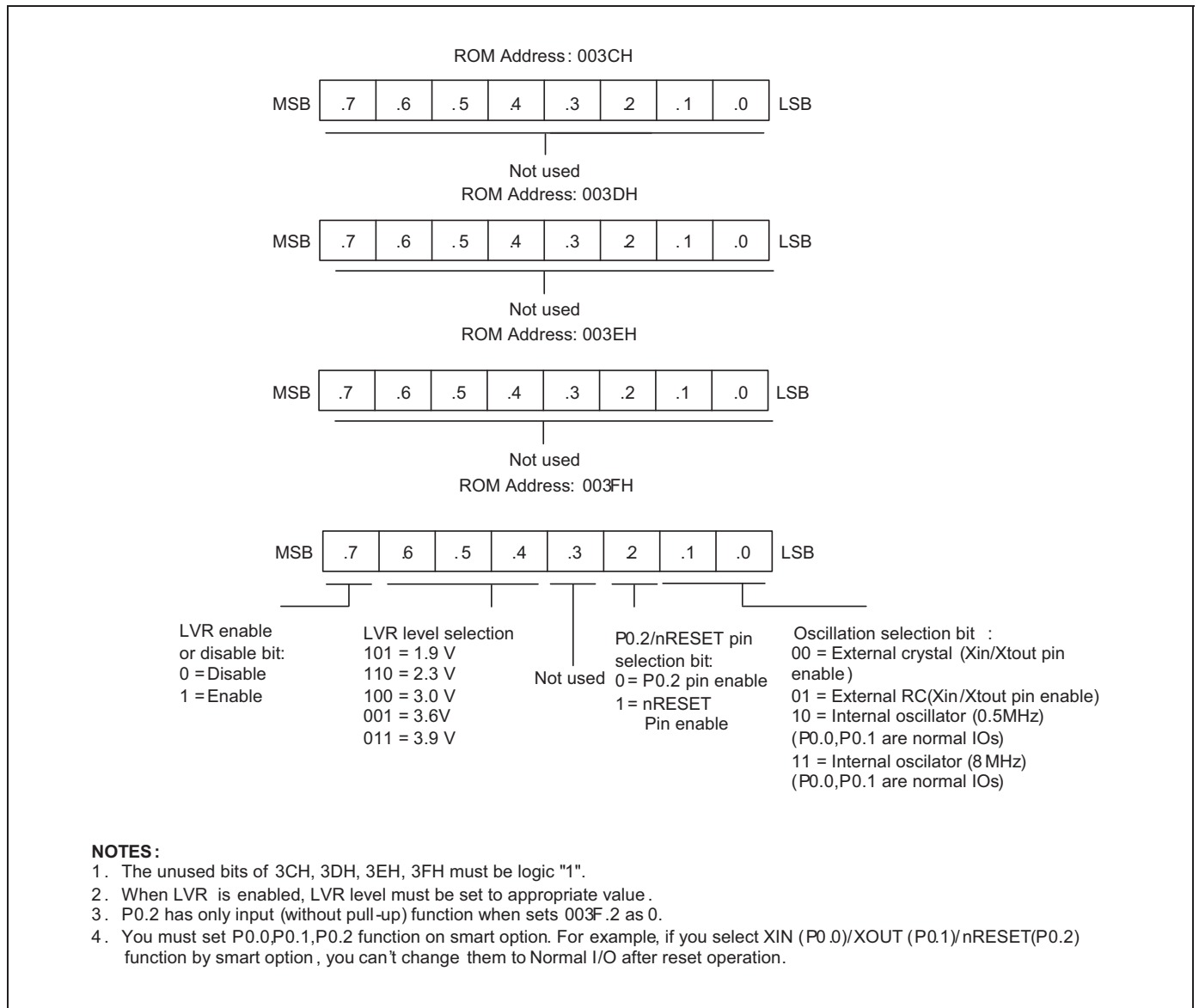


Figure 19-1 Smart Option

19.1.5 FLASH MEMORY CONTROL REGISTERS (USER PROGRAM MODE)

19.1.5.1 Flash Memory Control Register (FMCON)

The FMCON register is only available in User Program mode to select the Flash Memory Operation mode, sector erase, and byte programming, and to make the status of flash memory as hard lock protection.

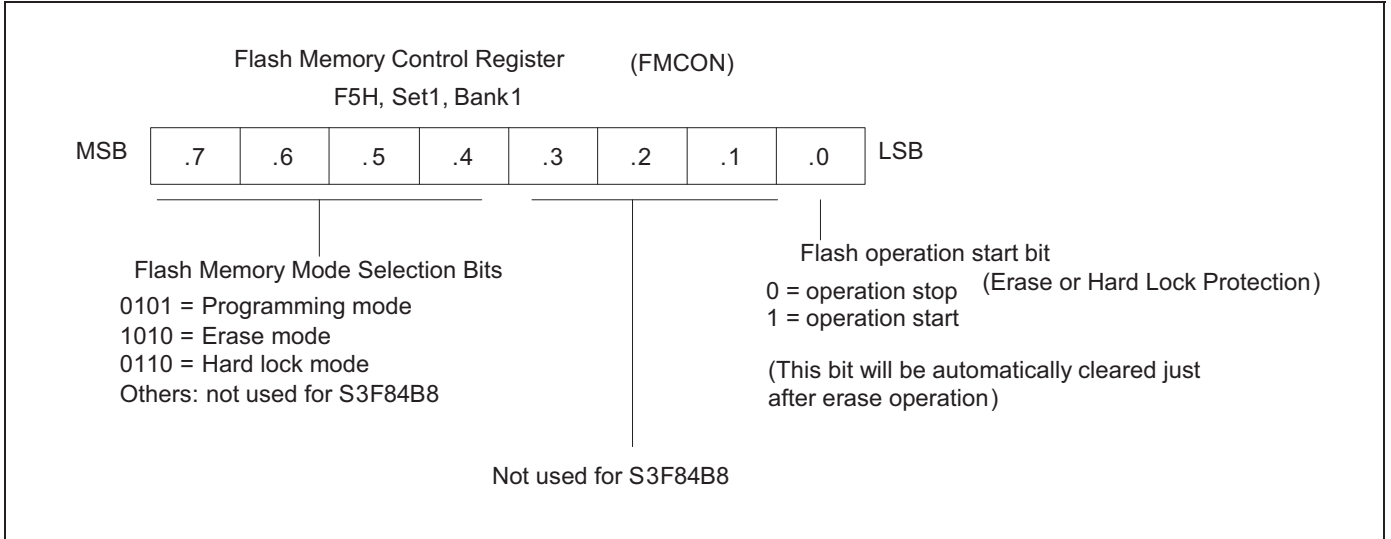


Figure 19-2 Flash Memory Control Register (FMCON)

The bit 0 of FMCON register (FMCON.0) specifies a bit for the start of Erase and Hard Lock Protection operations. Therefore, both Erase and Hard Lock Protection operations are activated when you set FMCON.0 to “1”. If you write FMCON.0 to 1 for erasing, the CPU is stopped automatically for erasing time (minimum for 4ms). After erasing time, the CPU is restarted automatically. When you read or program a byte data from or into flash memory, you do not need to touch this bit.

19.1.5.2 Flash Memory User Programming Enable Register (FMUSR)

The FMUSR register is used for safe operation of the flash memory. This register will protect undesired erase or program operation from malfunctioning of the CPU caused by electrical noise. After reset, the User Program mode is disabled because the value of FMUSR becomes “00000000B” due to reset operation. If it is necessary to operate the flash memory, you can use the User Program mode by setting the value of FMUSR to “10100101B”. If the value of FMUSR is other than “10100101B,” User Program mode is disabled.

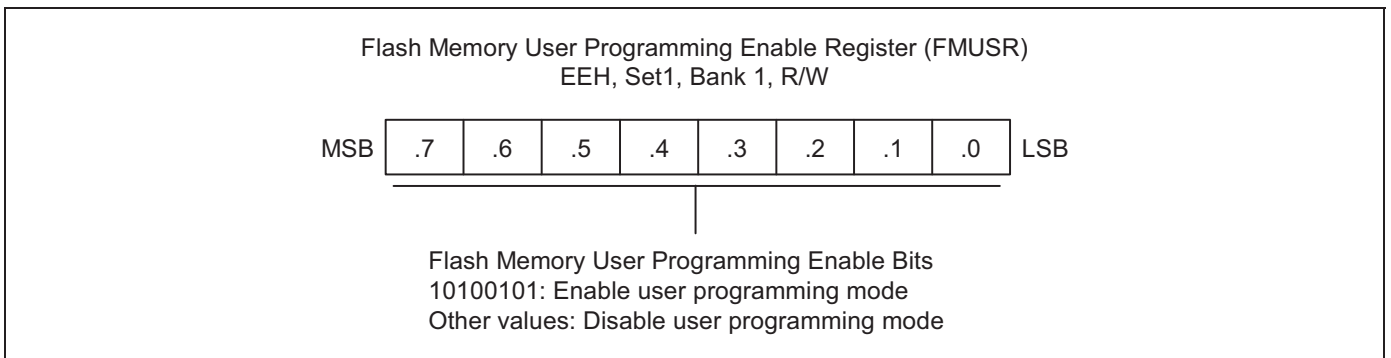


Figure 19-3 Flash Memory User Programming Enable Register (FMUSR)

19.1.5.3 Flash Memory Sector Address Registers

There are two sector address registers for erasing or programming flash memory, namely:

- Flash Memory Sector Address Register Low Byte (FMSECL)
- Flash Memory Sector Address Register High Byte (FMSECH)

FMSECL indicates the low byte of sector address, whereas FMSECH indicates the high byte of sector address.

One sector consists of 128 bytes. Each sector's address starts XX00H or XX80H, that is, the base address of sector is XX00H or XX80H. Thus, bit.6-.0 of FMSECL is meaningless. While programming the flash memory, you should load the sector base address before program. If the next operation is to write one byte data, you should check whether the next destination address is located in the same sector. In case of other sectors, you should reload the sector address to FMSECH and FMSECL registers. (For more information, refer to page 19-10 for "[Example 19-1](#) — Programming".)

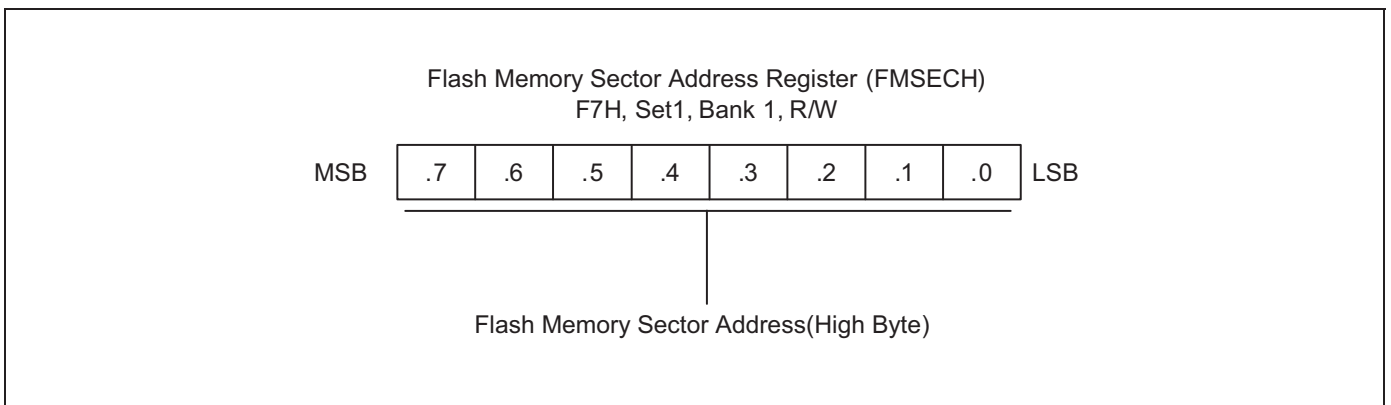


Figure 19-4 Flash Memory Sector Address Register (FMSECH)

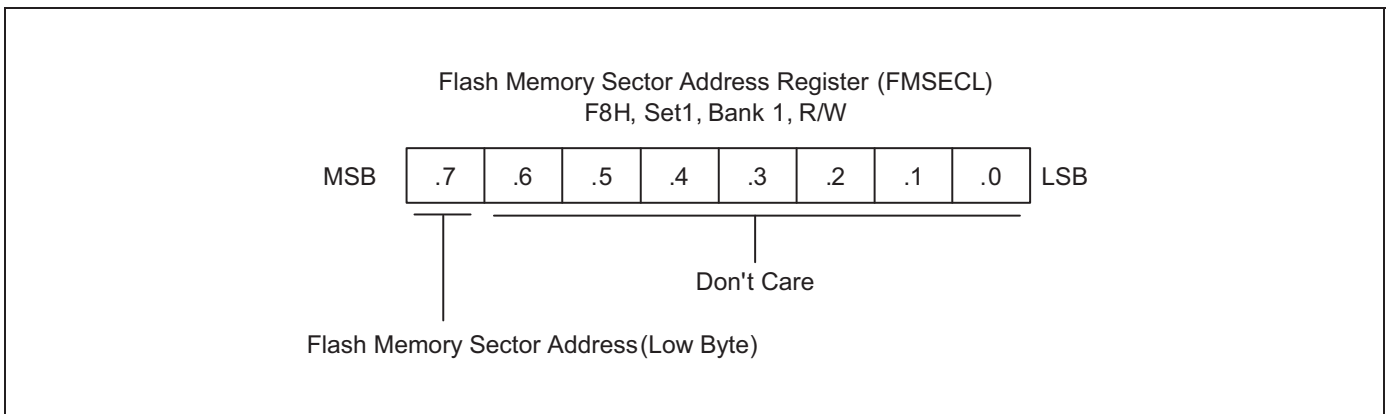


Figure 19-5 Flash Memory Sector Address Register (FMSECL)

Sector Erase Procedure in User Program Mode

To erase sector in User Program mode, follow these steps:

1. Set Flash Memory User Programming Enable Register (FMUSR) to “10100101B”.
2. Set Flash Memory Sector Address Registers (FMSECH and FMSECL).
3. Set Flash Memory Control Register (FMCON) to “10100001B”.
4. Set Flash Memory User Programming Enable Register (FMUSR) to “00000000B”.

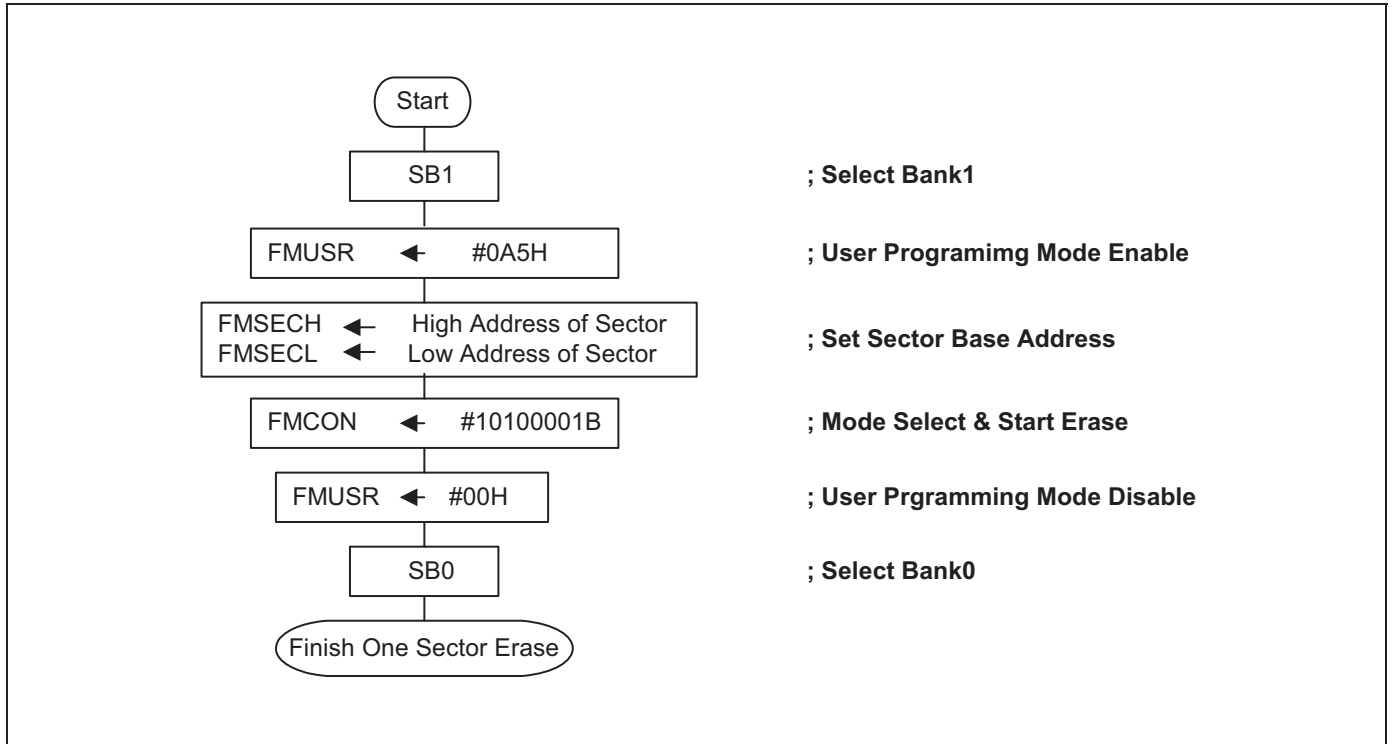


Figure 19-7 Sector Erase Flowchart in User Program Mode

NOTE:

1. If you erase a sector selected by Flash Memory Sector Address Registers (FMSECH and FMSECL), FMUSR should be enabled just before starting the sector erase operation. To erase a sector, Flash Operation Start Bit of FMCON register is written from stop operation ‘0’ to start operation ‘1’. This bit will be cleared automatically just after the erase operation is completed. In other words, when S3F84B8 is in a condition where Flash Memory User Programming Enable Bit is enabled and sector erase is started, the erase operation will start at the selected sector. The Flash Operation Start Bit of FMCON register will be cleared automatically thereafter.
2. If you disable FMUSR before the sector erase operation, the Flash Operation Start Bit (FMCON.0 bit) remains ‘high’. This specifies erase or hard lock start operation. FMCON.0 bit is not cleared even though the next instruction is executed. Therefore, you should be careful while setting FMUSR for sector erase. It should not have any effect on other flash sectors.

Example 19-1 Sector Erase

Case 1. Erase one sector

•

•

ERASE_ONESECTOR:

```
LD    FMUSR,#0A5H      ; Enables user program mode
LD    FMSECH,#04H      ; Set sector address 0400H, sector 8,
LD    FMSECL,#00H      ; among sector 0–32
LD    FMCON,#10100001B ; Select erase mode enable and Start sector erase
```

```
ERASE_STOP: LD    FMUSR,#00H      ; Disables user program mode
```

19.1.7 PROGRAMMING

Flash memory is programmed in one-byte unit after sector erase. The write operation of programming is executed using the LDC instruction.

Program Procedure in User Program Mode

To program Flash memory in User Program mode, follow these steps:

1. Erase target sectors before programming (mandatory).
2. Set Flash Memory User Programming Enable Register (FMUSR) to “10100101B”.
3. Set Flash Memory Control Register (FMCON) to “0101000XB”.
4. To write data, set Flash Memory Sector Address Registers (FMSECH and FMSECL) to the sector base address of destination address.
5. Load transmission data into working register.
6. Load flash memory upper address into upper register of pair working register.
7. Load flash memory lower address into lower register of pair working register.
8. Load transmission data to flash memory location area using ‘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’.

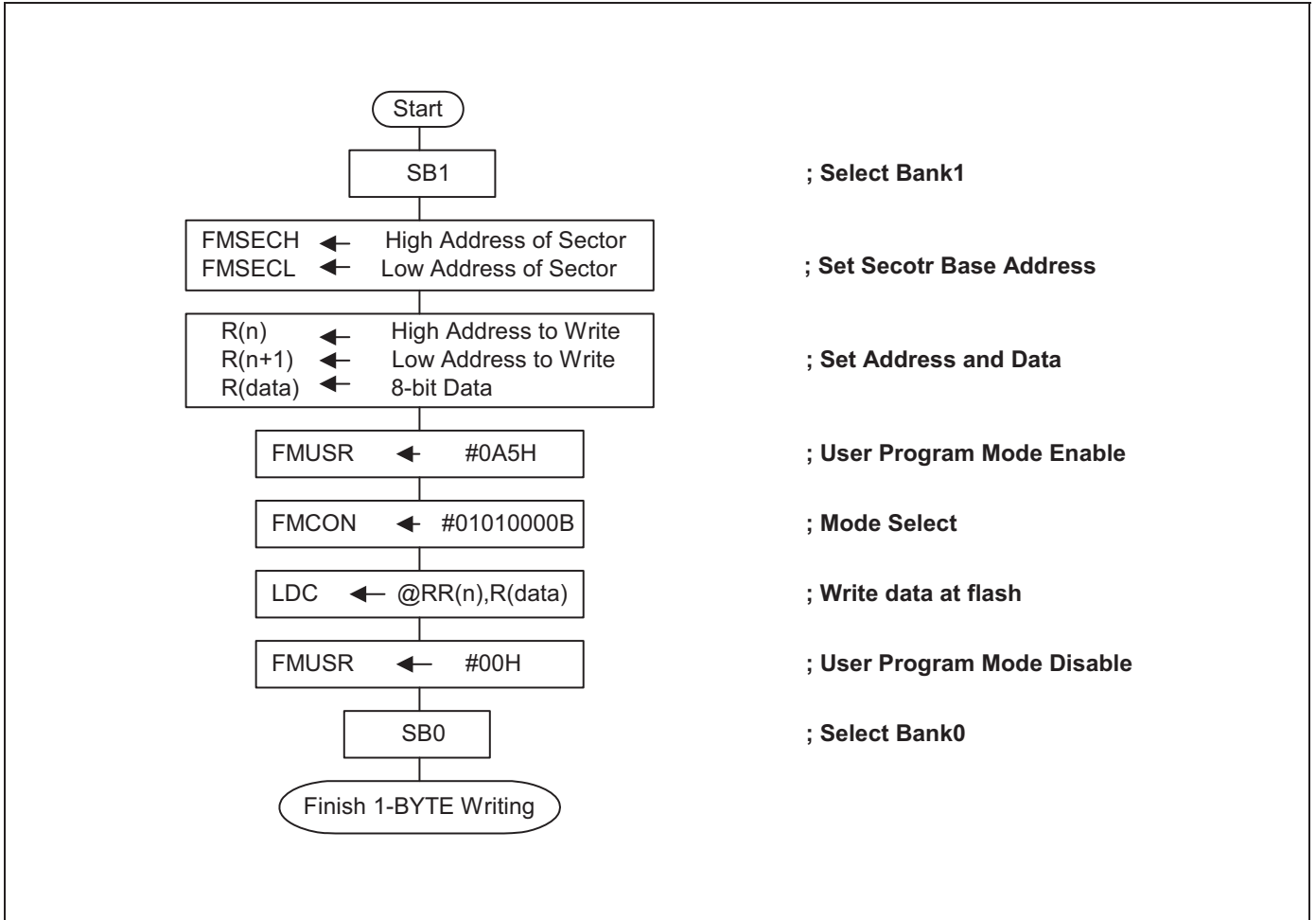


Figure 19-8 Byte Program Flowchart in a User Program Mode

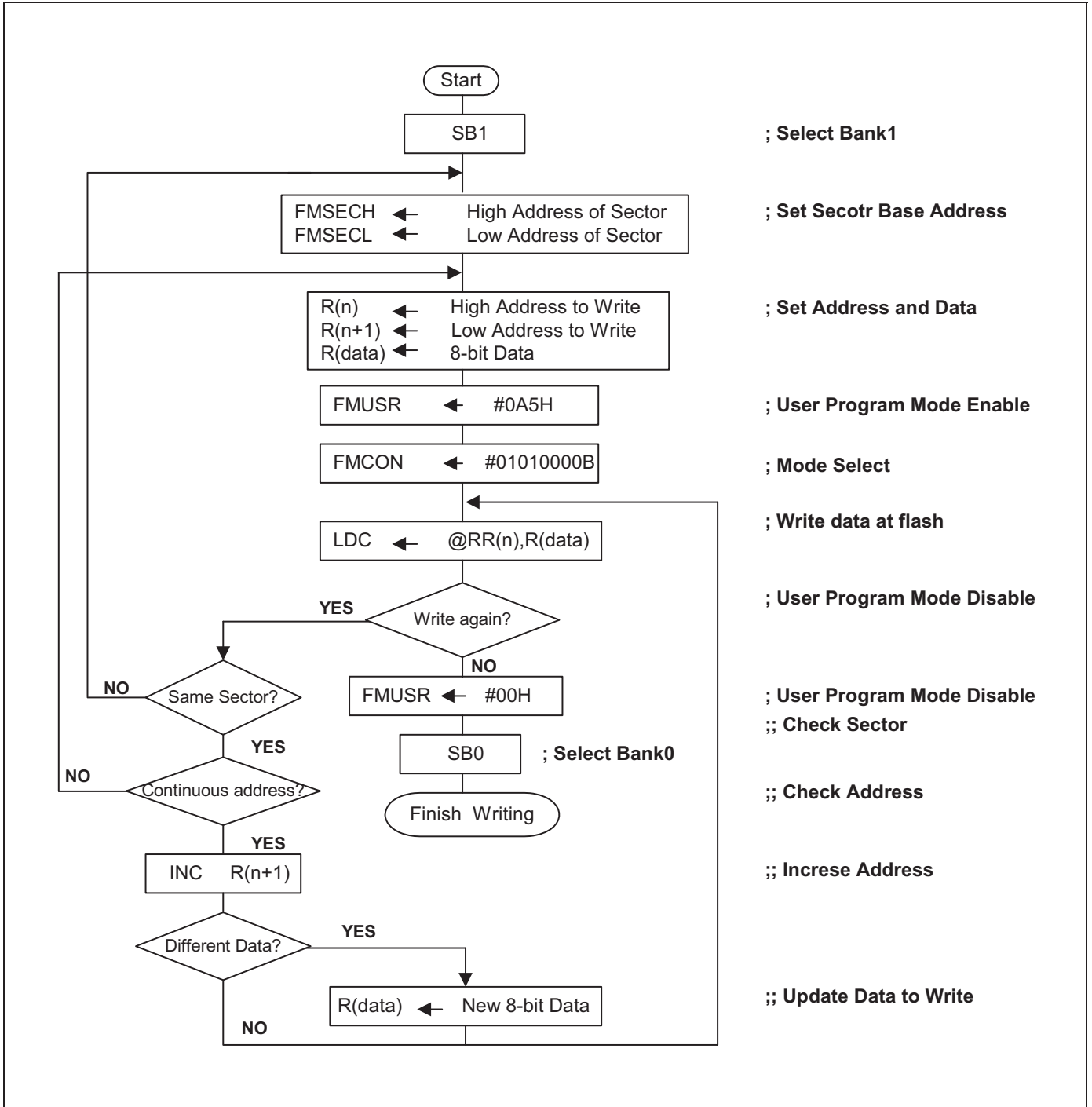


Figure 19-9 Program Flowchart in a User Program Mode

Example 19-2 Programming

Case1. 1-Byte Programming

•
•

```

WR_BYTE:                                ; Writes data "AAH" to destination address 0310H

LD    FMUSR,#0A5H                        ; Enables User Program mode
LD    FMCON,#01010000B                   ; Selects Programming mode
LD    FMSECH, #03H                       ; Sets the base address of sector (0300H)
LD    FMSECL, #00H
LD    R9,#0AAH                           ; Loads data "AA" to write
LD    R10,#03H                           ; Loads flash memory upper address into upper register of pair working
                                           ; register
LD    R11,#10H                           ; Loads flash memory lower address into lower register of pair working
                                           ; register
LDC   @RR10,R9                           ; Writes data "AAH" to flash memory location (0310H)

LD    FMUSR,#00H                         ; Disables User Program mode
  
```

Case2. Programming in the same sector

•
•

```

WR_INSECTOR:                            ; RR10-->Address copy (R10-high address,R11-low address)

LD    R0,#40H

LD    FMUSR,#0A5H                        ; Enables User Program mode
LD    FMCON,#01010000B                   ; Selects Programming mode and starts programming
LD    FMSECH,#06H                        ; Sets the base address of sector located in target address to write data
LD    FMSECL,#00H                        ; Sector 12's base address is 0600H.
LD    R9,#33H                            ; Loads data "33H" to write
LD    R10,#06H                           ; Loads flash memory upper address into upper register of pair working
                                           ; register
LD    R11,#00H                           ; Loads flash memory lower address into lower register of pair working
                                           ; register

WR_BYTE:
LDC   @RR10,R9                           ; Writes data "33H" to flash memory location
INC   R11                                 ; Resets address in the same sector by INC instruction
  
```

```

DEC  R0
JP   NZ, WR_BYTE      ; Checks whether the end address for programming reaches 0640H.

LD   FMUSR,#00H       ; Disables User Program mode
  
```

Case3. Programming to the flash memory space located in other sectors

-
-

WR_INSECTOR2:

```

LD   R0,#40H
LD   R1,#40H

LD   FMUSR,#0A5H      ; Enables User Program mode
LD   FMCON,#01010000B ; Selects Programming mode and starts programming
LD   FMSECH,#01H      ; Sets the base address of sector located in target address to write data
LD   FMSECL,#00H      ; Sector 2's base address is 100H
LD   R9,#0CCH         ; Loads data "CCH" to write
LD   R10,#01H         ; Loads flash memory upper address into upper register of pair working
                          ; register
LD   R11,#40H         ; Loads flash memory lower address into lower register of pair working
                          ; register

CALL WR_BYTE

LD   R0,#40H
  
```

WR_INSECTOR5:

```

LD   FMSECH,#02H      ; Sets the base address of sector located in target address to write data
LD   FMSECL,#80H      ; Sector 5's base address is 0280H
LD   R9,# 55H         ; Loads data "55H" to write
LD   R10,#02H         ; Loads flash memory upper address into upper register of pair working
                          ; register
LD   R11,#90H         ; Loads flash memory lower address into lower register of pair working
                          ; register

CALL WR_BYTE
  
```

WR_INSECTOR12:

```

LD    FMSECH,#06H      ; Sets the base address of sector located in target address to write data
LD    FMSECL,#00H      ; Sector 12's base address is 0600H
LD    R9,#0A3H         ; Loads data "A3H" to write
LD    R10,#06H         ; Loads flash memory upper address into upper register of pair working
                        ; register
LD    R11,#40H         ; Loads flash memory lower address into lower register of pair working
                        ; register

```

WR_BYTE1:

```

LDC   @RR10,R9         ; Writes data "A3H" to flash memory location
INC   R11
DEC   R1
JP    NZ, WR_BYTE1
LD    FMUSR,#00H       ; Disables User Program mode
.
.

```

WR_BYTE:

```

LDC   @RR10,R9         ; Writes data written by R9 to flash memory location
INC   R11
DEC   R0
JP    NZ, WR_BYTE
RET

```


19.1.8 READING

The read operation starts using the 'LDC' instruction.

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 using 'LDC' instruction by indirectly addressing mode.

Example 19-3 Reading

```

      •
      •
      LD   R2,#03H           ; Loads flash memory's upper address
                          ; to upper register of pair working register
      LD   R3,#00H           ; Loads flash memory's lower address
                          ; to lower register of pair working register
LOOP:  LDC  R0,@RR2          ; Reads data from flash memory location
                          ; (Between 300H and 3FFH)
      INC  R3
      CP   R3,#0FFH
      JP   NZ,LOOP
      •
      •
      •
      •

```

19.1.9 HARD LOCK PROTECTION

You can set Hard Lock Protection by writing '0110B' in FMCON7–4. This function prevents data change in flash memory area. If this function is enabled, you cannot write or erase data in flash memory anymore. This protection can be released by chip erase in Tool Program mode. To set Hard Lock Protection in Tool mode, refer to the “Serial Program Writer Tool Manual”.

Program Procedure in User Program Mode

To set Hard Lock Protection in User Program mode, follow these steps:

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”.

Example 19-4 Hard Lock Protection

```

•
•
SB1
LD    FMUSR,#0A5H           ; Enables User Program mode
LD    FMCON,#01100001B     ; Selects Hard Lock mode and starts protection
LD    FMUSR,#00H           ; Disables User Program mode
SB0
•
•

```

20

LOW VOLTAGE RESET

20.1 OVERVIEW OF LOW VOLTAGE RESET

Using the Smart option (3FH.7 in ROM), you can choose the reset source as internal (LVR) or external.

The S3F84B8 microcontroller can be reset in four ways using:

- External power-on-reset
- External reset input pin pulled low
- Digital watchdog time out
- Low Voltage Reset circuit (LVR)

During an external power-on reset, the voltage V_{DD} is set to High level and the RESETB pin stays low level for some time. The RESETB signal is inputted through a Schmitt trigger circuit, where it is then synchronized with the CPU clock. This brings the S3F84B8 microcontroller to a known operating status.

To ensure the correct start up, you should ensure that reset signal is not released before the V_{DD} level is sufficient. This allows the MCU to operate at the chosen frequency.

The RESETB pin must be held to Low level for a minimum time interval of 10us after the power supply comes within tolerance level. This allows time for internal CPU clock oscillation to stabilize.

If a reset occurs during normal operation (with both V_{DD} and RESETB at High level), the signal at RESETB pin is forced to Low level and the reset operation starts. All system and peripheral control registers are then set to their default hardware reset values (see [Figure 20-1](#)).

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

S3F84B8 has a built-in low voltage reset (LVR) circuit that detects voltage drop of external V_{DD} input level and prevents the MCU from malfunctioning whenever it encounters fluctuation in power level. This voltage detector is used to reset the MCU.

The LVR circuit includes an analog comparator and VREF circuit. The value of detection voltage is set internally by the hardware.

The on-chip LVR circuit features static reset when supply voltage is below reference voltage value (Typical 1.9/2.3/3.03.6/3.9 V). Owing 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 only start when the supply voltage rises over the reference voltage.

To calculate power consumption, static current of LVR circuit should be added to the CPU operating current in operating modes such as Stop, Idle, and Normal Run.

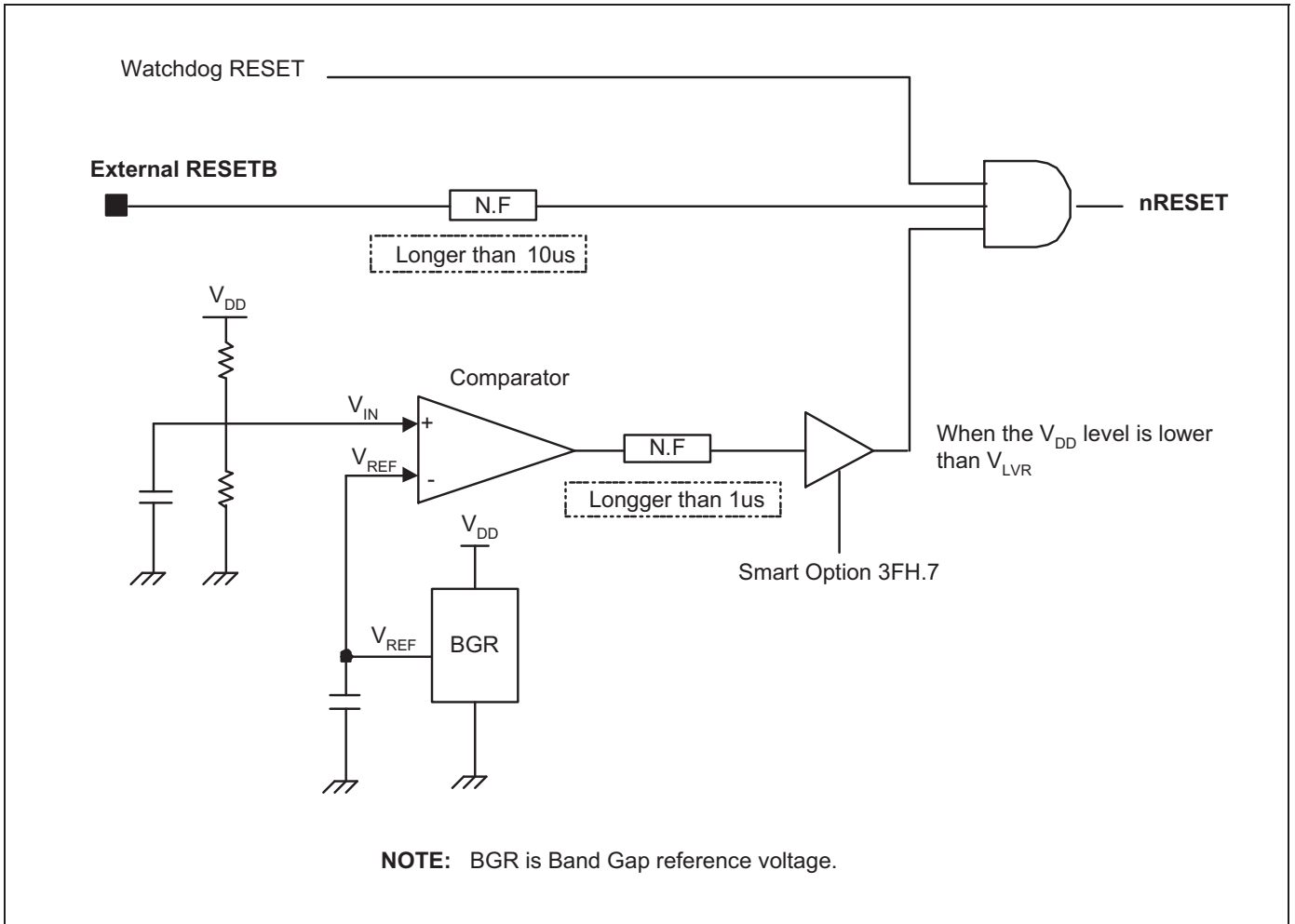


Figure 20-1 Low Voltage Reset Circuit

NOTE: To program the duration of the oscillation stabilization interval, set the basic timer control register, BTCON, before entering the Stop mode. If you do not want to use the basic timer watchdog function (which causes a system reset if basic timer counter overflows), you can disable it by writing '1010B' to the upper nibble of BTCON.

21

ELECTRICAL DATA

21.1 OVERVIEW OF ELECTRICAL DATA

This section describes the electrical characteristics of S3F84B8 in the form of tables and graphs. The following information has been provided:

- Absolute maximum ratings
- DC electrical characteristics
- AC 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

($T_A = 25^\circ\text{C}$)

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 is active	–25	mA
		All I/O pins are active	–80	
Output current low	I_{OL}	One I/O pin is active	+30	mA
		All I/O pins are active	+100	
Operating temperature	T_A	–	–40 to + 85	$^\circ\text{C}$
Storage temperature	T_{STG}	–	–65 to + 150	$^\circ\text{C}$

Table 21-2 DC Electrical Characteristics

($T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$, $V_{DD} = 1.8\text{V}$ to 5.5V)

Parameter	Symbol	Conditions		Minimum	Typical	Maximum	Unit
Operating Voltage	V_{DD}	Fmain = 0.4 – 2MHz		1.8	–	5.5	V
		Fmain = 0.4 – 4MHz		2.0	–	5.5	
		Fmain = 0.4 – 10MHz		2.7	–	5.5	
Main crystal or ceramic frequency	f_{main}	$V_{DD} = 2.7\text{V}$ to 5.0V		0.4	–	10	MHz
		$V_{DD} = 1.8\text{V}$ to 2.7V		0.4	–	4	
Input high voltage	V_{IH1}	Ports 0,1, 2, and RESET	$V_{DD} = 1.8$ to 5.5V	0.8 VDD	–	VDD	V
	V_{IH2}	X_{IN}		VDD-0.1			
Input low voltage	V_{IL1}	Ports 0, 1, 2, and RESET	$V_{DD} = 1.8$ to 5.5V	–	–	0.2 VDD	V
	V_{IL2}	X_{IN}				0.1	
Output high voltage	V_{OH}	$I_{OH} = -10\text{mA}$ Ports 0,1, and 2	$V_{DD} = 4.5$ to 5.5V	VDD-1.5	VDD-0.4	–	V
Output low voltage	V_{OL}	$I_{OL} = 25\text{mA}$ Ports 0, 1, and 2	$V_{DD} = 4.5$ to 5.5V	–	0.4	2.0	V
Input high leakage current	I_{LH1}	All input pins (except P0.2 and I_{LH2})	$V_{IN} = V_{DD}$	–	–	1	μA
	I_{LH2}	X_{IN}	$V_{IN} = V_{DD}$			20	
Input low leakage current	I_{LIL1}	All input pins (except P0.2 and I_{LIL2})	$V_{IN} = 0\text{V}$	–	–	–1	μA
	I_{LIL2}	X_{IN}	$V_{IN} = 0\text{V}$			–20	
Output high leakage current	I_{LOH}	All output pins	$V_{OUT} = V_{DD}$	–	–	2	μA
Output low leakage current	I_{LOL}	All output pins	$V_{OUT} = 0\text{V}$	–	–	–2	μA
Pull-up resistors	R_{P1}	$V_{IN} = 0\text{V}$, Ports 0, 1, and 2	$V_{DD} = 5\text{V}$ $T_A = 25^\circ\text{C}$	25	50	100	k Ω
Supply current	I_{DD1}	Run mode (10MHz CPU clock)	$V_{DD} = 4.5$ to 5.5V	–	3	6	mA
	I_{DD2}	Idle mode (10MHz CPU clock)	$V_{DD} = 4.5$ to 5.5V	–	2	4	
	I_{DD3}	Stop mode	$V_{DD} = 4.5$ to 5.5V (LVR disabled) $T_A = -40^\circ\text{C} \sim 85^\circ\text{C}$	–	0.6	4.0	μA

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Unit
		$V_{DD} = 4.5$ to $5.5V$ (LVR enabled) $T_A = -40^{\circ}C \sim 85^{\circ}C$		40	100	

NOTE: Supply current does not include the current drawn through internal pull-up resistors or external output current loads and ADC module.

Table 21-3 AC Electrical Characteristics

($T_A = -40^{\circ}C$ to $+85^{\circ}C$, $V_{DD} = 1.8V$ to $5.5V$)

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Unit
Interrupt input high, low width	t_{INTH} t_{INTL}	INT0, INT1 $V_{DD} = 5V \pm 10\%$	–	200	–	ns
RESET input low width	t_{RSL}	Input $V_{DD} = 5V \pm 10\%$	10	–	–	us

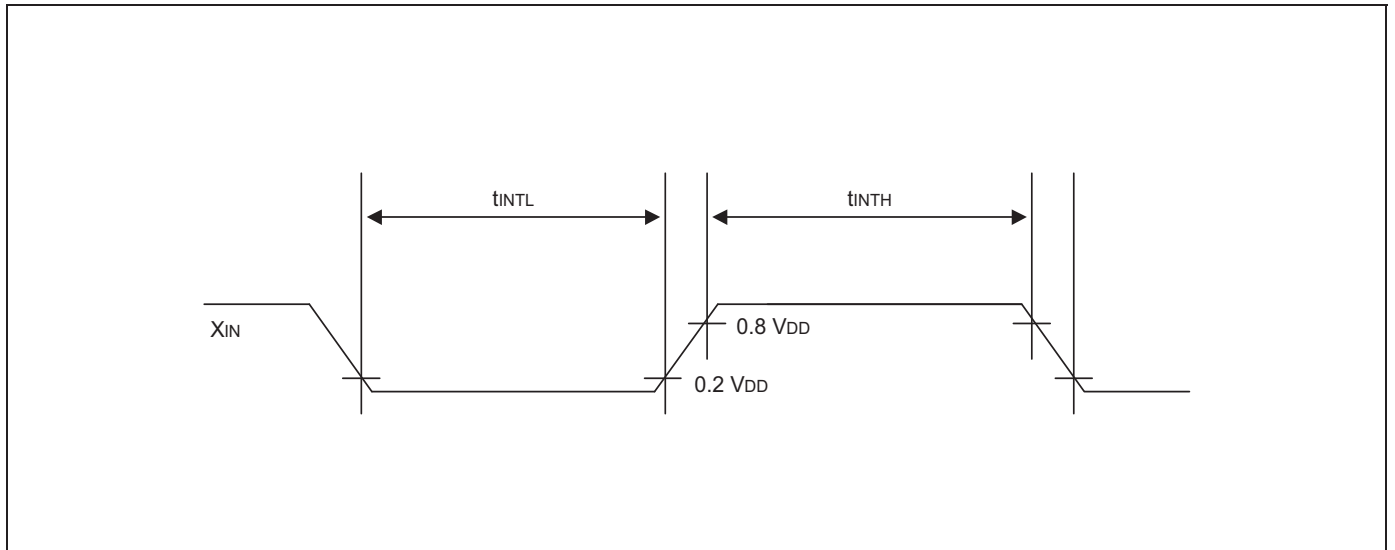


Figure 21-1 Input Timing Measurement Points

Table 21-4 Oscillator Characteristics

($T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$)

Oscillator	Clock Circuit	Test Condition	Minimum	Typical	Maximum	Unit
Main crystal or ceramic		$V_{DD} = 2.7$ to 5.5V	0.4	–	10	MHz
		$V_{DD} \text{ (NOTE)} = 2.0$ to 2.7V	0.4	–	4	MHz
		$V_{DD} \text{ (NOTE)} = 1.8$ to 2.0V	0.4	–	2	MHz
External clock (Main System)		$V_{DD} = 2.7$ to 5.5V	0.4	–	10	MHz
		$V_{DD} = 1.8$ to 2.7V	0.4	–	4	MHz
External RC oscillator	–	$V_{DD} = 5.0\text{V}$	–	8	–	
Tolerance of Internal RC	–	Factory calibrated at 25°C , 5.0V	–	–	± 3	%
	–	$V_{DD} = 5.0\text{V}$ $T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$	–	–	± 6	%
	–	$V_{DD} = 2.0$ to 5.5V $T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$	–	–	± 9	%

NOTE: Refer to [Figure 21-2](#), “Operating Voltage Range @ External clock”.

Table 21-5 Oscillation Stabilization Time

($T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$, $V_{DD} = 1.8\text{V}$ to 5.5V)

Oscillator	Test Condition	Minimum	Typical	Maximum	Unit
Main crystal stabilization time	$f_{\text{OSC}} > 1.0\text{MHz}$	–	–	20	ms
Main ceramic stabilization time	Oscillation stabilization is achieved 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⁾	–	$219/f_{\text{OSC}}$	–	ms
	t_{WAIT} when released by an interrupt ⁽²⁾	–	–	–	ms

NOTE:

- f_{OSC} specifies the oscillator frequency.
- When released by an interrupt, the duration of oscillator stabilization wait time (t_{WAIT}) is determined by the settings in asic timer control register, BTCON.

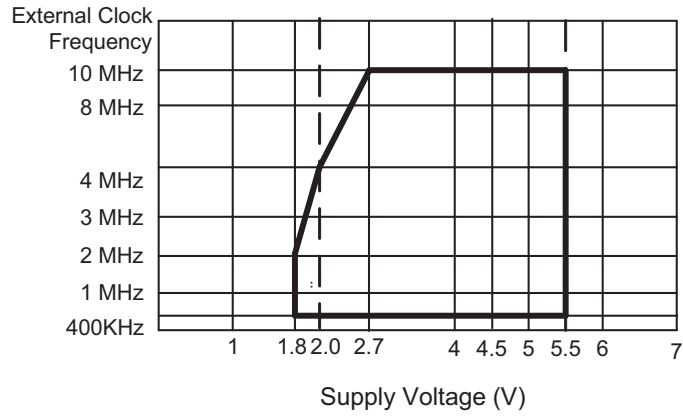


Figure 21-2 Operating Voltage Range @ External clock

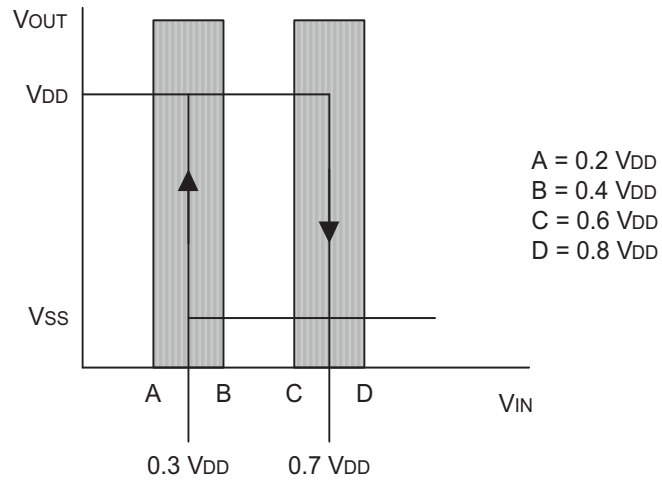


Figure 21-3 Schmitt Trigger Input Characteristics Diagram

Table 21-6 Data Retention Supply Voltage in Stop Mode

($T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$, $V_{DD} = 1.8\text{V}$ to 5.5V)

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Unit
Data retention supply voltage	V_{DDDR}	Stop mode	1.0	–	5.5	V
Data retention supply current	I_{DDDR}	Stop mode; $V_{DDDR} = 1.8\text{V}$	–	–	1	uA

NOTE: Supply current does not include the current drawn through internal pull-up resistors or external output current loads.

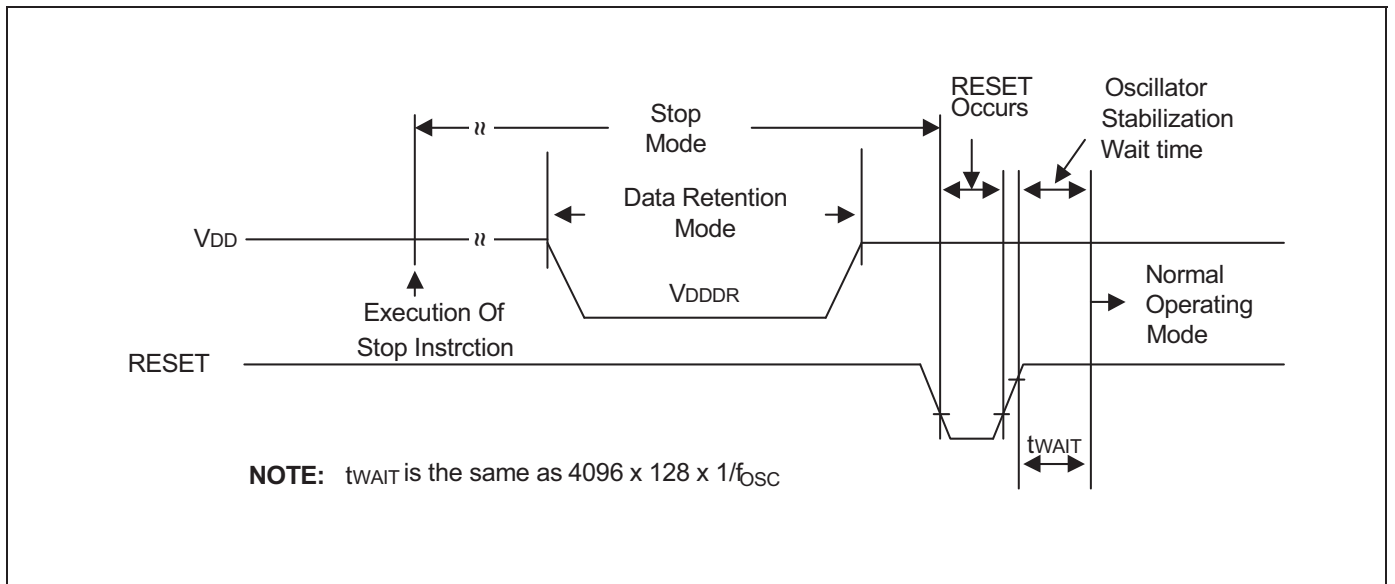


Figure 21-4 Stop Mode Release Timing When Initiated by a RESET

Table 21-7 A/D Converter Electrical Characteristics

 ($T_A = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$, $V_{DD} = 1.8\text{V}$ to 5.5V , $V_{SS} = 0\text{V}$)

Parameter	Symbol	Test Conditions	Minimum	Typical	Maximum	Unit
Resolution			–	10	–	bit
Total accuracy		$V_{DD} = 5.12\text{V}$ (1) CPU clock = 10MHz $V_{SS} = 0\text{V}$	–	–	± 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	–	$\text{M}\Omega$
Analog input current	I_{ADIN}	$V_{DD} = 5\text{V}$	–	–	10	μA
Analog block current (3)	I_{ADC}	$V_{DD} = 5\text{V}$	–	0.5	1.5	mA
		$V_{DD} = 5\text{V}$ Power down mode		100	500	mA

NOTE:

1. When $V_{DD} = 2.7\text{V}$ to 5.5V , the total accuracy is characterized to be maximum 3LSB, but not tested.
2. “Conversion time” specifies the time required from the moment a conversion operation starts until it ends.
3. I_{ADC} specifies the operating current during A/D conversion.

Table 21-8 OP AMP Electrical Characteristics

($T_A = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$, $V_{DD} = 2.0\text{V}$ to 5.5V)

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Unit
Input offset voltage	$ V_{io} $	$V_{DD} = 2.0\text{V}$	–	10	30 ⁽¹⁾	mV
		$V_{DD} = 5.5\text{V}$	–	10	30 ⁽¹⁾	mV
Input common-mode voltage range ⁽²⁾	V_{cm}		GND	–	$V_{DD}-0.1$	V
Output voltage	V_{out}		GND+0.1	–	$V_{DD}-0.1$	V

NOTE:

1. For the hardware and software calibration methods, refer to the Application Note.
2. The input signal voltage should not go below -0.3V .

Table 21-9 Comparator Electrical Characteristics

($T_A = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$, $V_{DD} = 2.0\text{V}$ to 5.5V)

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Unit
CMP0 input offset voltage ⁽¹⁾	$ V_{io} $	$V_{DD} = 2\text{V}$	–	10	20	mV
		$V_{DD} = 5.5\text{V}$	–	10	20	mV
CMP1/2/3 input offset voltage ^{(1), (2)}	$ V_{io} $	$V_{DD} = 2\text{V}$	–	15	30	mV
		$V_{DD} = 5.5\text{V}$	–	15	30	mV
CMP0 input common mode voltage range	V_{cm}		GND	–	$V_{DD}-0.1$	V

NOTE:

1. These parameters are characterized only, but not tested.
2. Parameter includes the tolerance level of internal voltage reference.

Table 21-10 LVR Circuit Characteristics

($T_A = 25^\circ\text{C}$, $V_{DD} = 1.8\text{V to } 5.5\text{V}$)

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Unit
Low voltage reset	V_{LVR}	–	1.8	1.9	2.0	V
			2.1	2.3	2.5	
			2.8	3.0	3.2	
			3.4	3.6	3.8	
			3.7	3.9	4.1	

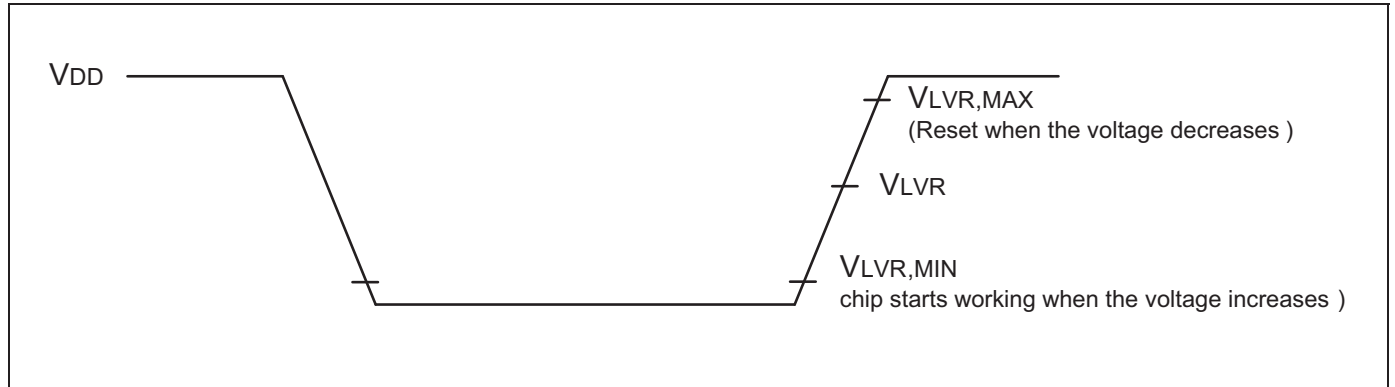


Figure 21-5 LVR Reset Timing

Table 21-11 Flash Memory AC Electrical Characteristics

($T_A = -40^\circ\text{C to } +85^\circ\text{C}$ at $V_{DD} = 1.8\text{V to } 5.5\text{V}$)

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Unit
Flash Erase/Write/Read voltage	Fewrv	V_{DD}	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	FtRS	$V_{DD} = 2.0\text{V}$	–	250	–	nS
Number of writing/erasing	FNwe	–	10,000	–	–	Times
Data retention	Ftdr	–	10	–	–	Years

NOTE:

1. Programming time specifies the time during which one byte (8-bit) is programmed.
2. Chip erasing time specifies the time during which the entire program memory is erased.
3. Sector erasing time specifies the time during which the 128 byte block is erased.
4. Chip erasing is available in Tool Program mode only.

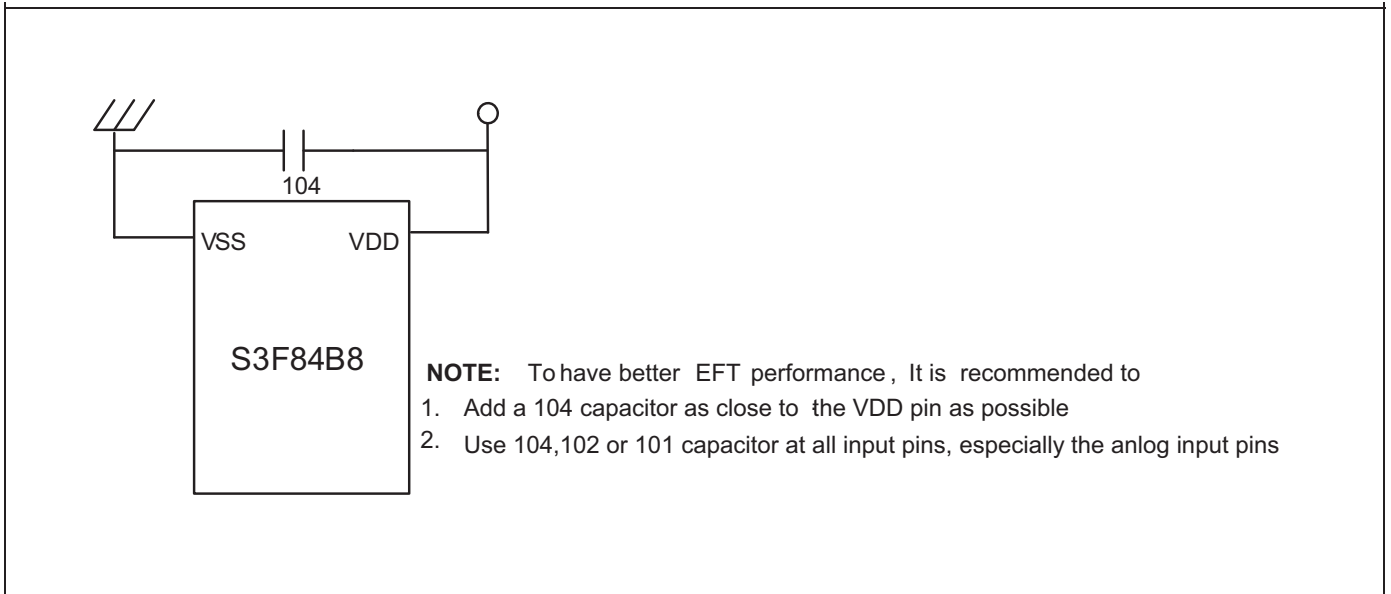


Figure 21-6 Circuit Diagram to Improve the EFT Characteristics

Table 21-12 ESD Characteristics

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Unit
Electrostatic discharge	V_{ESD}	HBM	2000	–	–	V
		MM	200	–	–	V
		CDM	500	–	–	V

22

DEVELOPMENT TOOLS

22.1 OVERVIEW OF DEVELOPMENT TOOLS

Zilog provides a powerful and easy-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 in both the hardware and software such as in-circuit emulator, OPENice-i500, and SK-1200, for the S3F7-, S3F9-, and S3F8- microcontroller families, respectively. Zilog also offers supporting software that includes a debugger, an assembler, and a program for setting options.

22.1.1 TARGET BOARDS

Target boards are available for all the 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.

22.1.2 PROGRAMMING SOCKET ADAPTER

When you program S3F84B8's flash memory by using an emulator or an OTP/MTP writer, you need a specific programming socket adapter for S3F84B8.

22.2 DEVELOPMENT SYSTEM CONFIGURATION

Figure 22-1 shows the Development System Configuration.

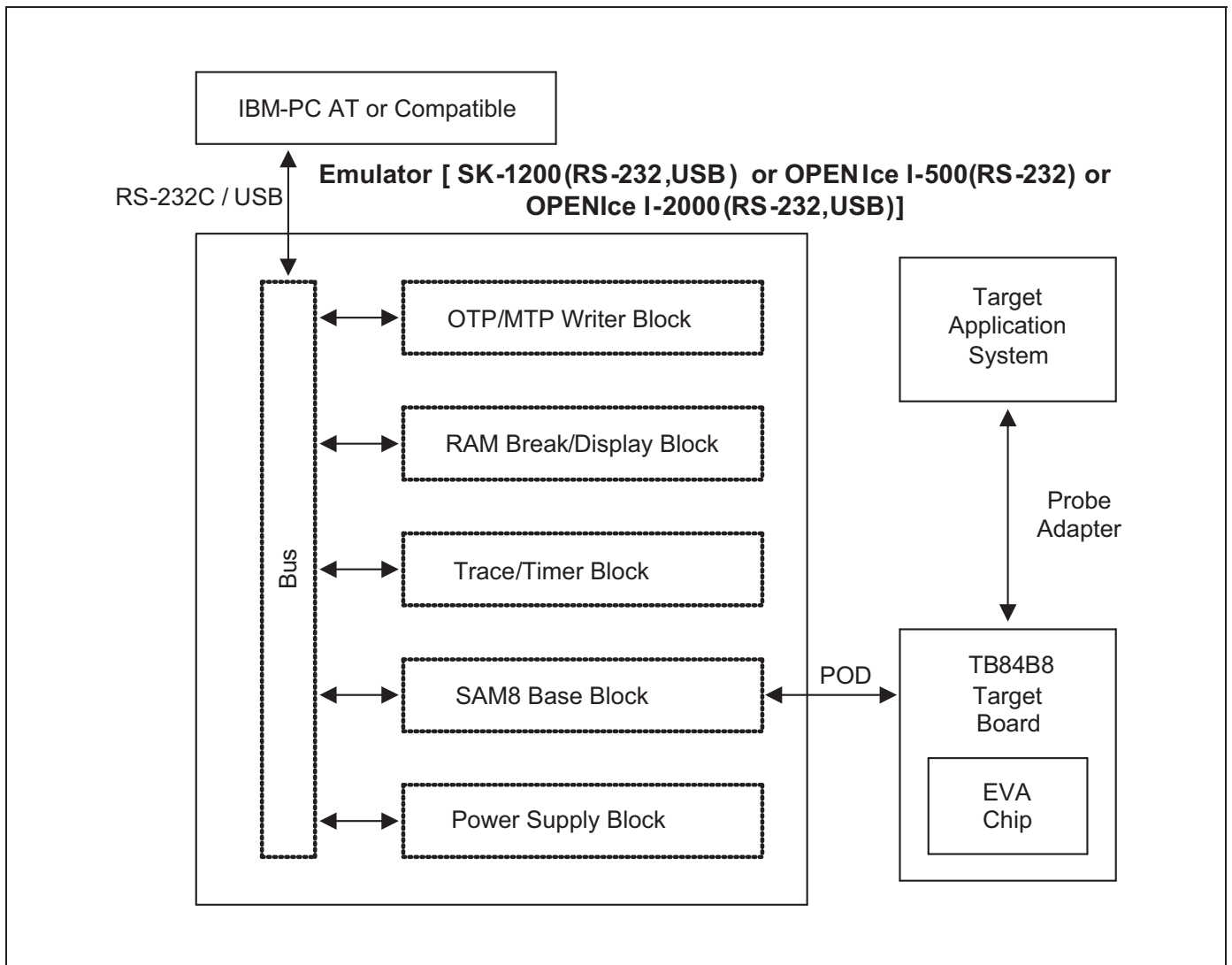


Figure 22-1 Development System Configuration

22.3 TB84B8 TARGET BOARD

The TB84B8 target board is used for S3F84B8 microcontrollers. It is operated as a target CPU with emulator (OPENIce I-500/2000 or SK-1200).

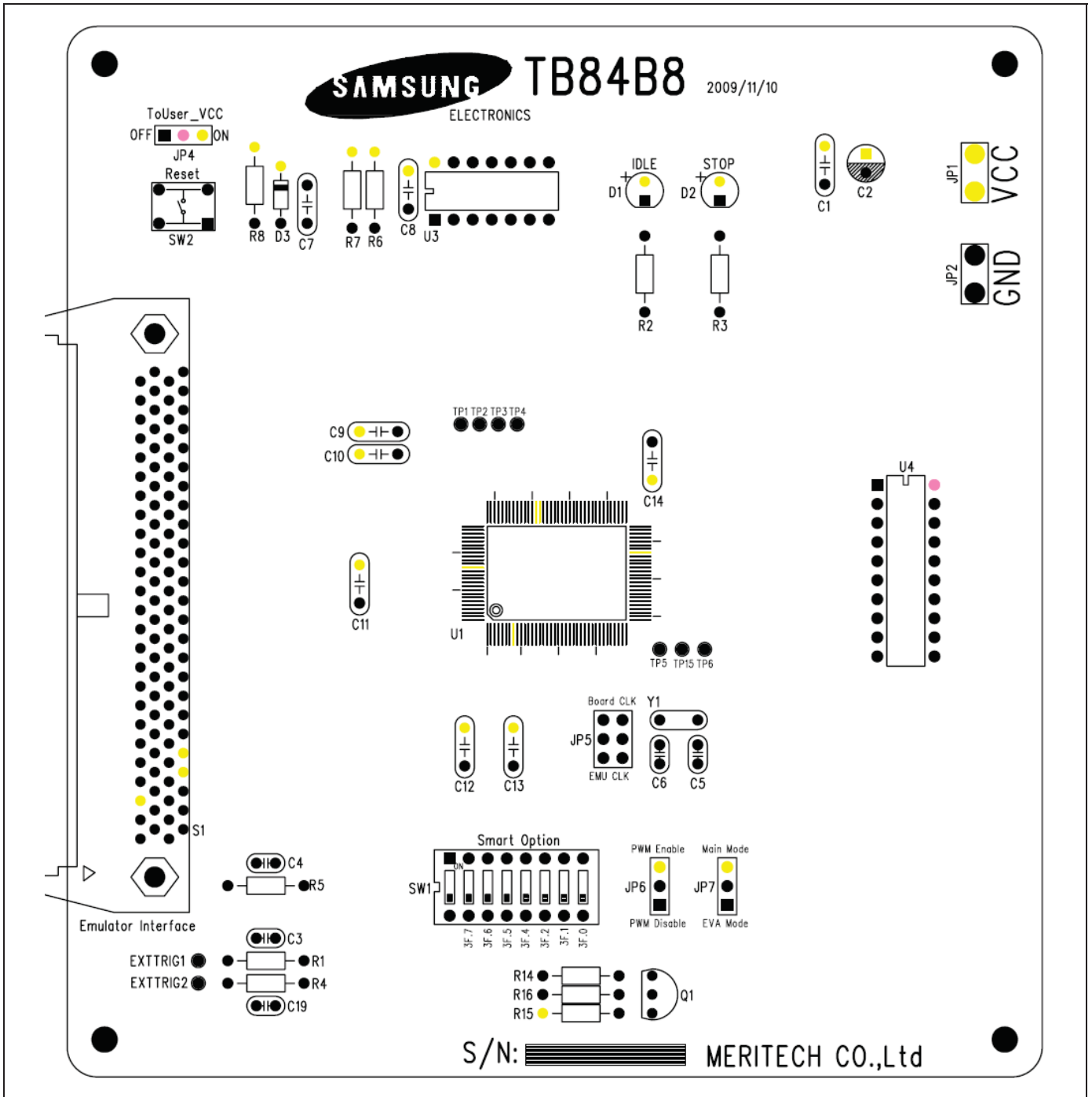



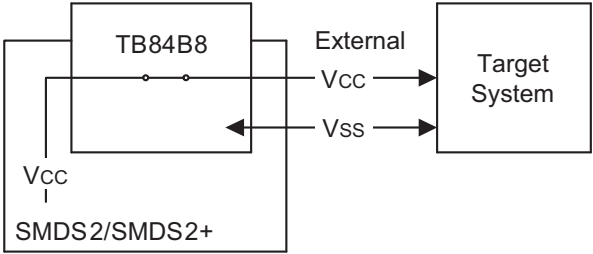

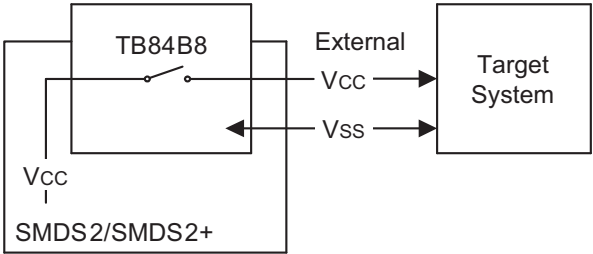
Figure 22-2 TB84B8 Target Board Configuration

NOTE: TB84B8 should be supplied with 5V normally. Thus, the power supply from Emulator should be set to 5V for the target board operation.

Table 22-1 TB84B8 Components

Mark	Usage	Description
S1	100-cable interface	Connect the emulator and TB84B8
U4	20-cable interface	Connect TB84B8 and user system
SW1	8- channel switch	Smart Option configuration of S3E84B0
RESET	Key	Generate reset signal to S3E84B0
VCC, GND	Power in	Power supply for TB84B8
IDLE, STOP LED	STOP/IDLE display	Indicate S3E84B0 work status
JP5	Clock source selection	Select clock source as from the emulator or board
JP7	Mode selection	EVA /Main mode selection of S3E84B0
JP6	PWM selection	Select whether PWM keeps output as the emulator pauses
JP4	User power selection	Select user power supply

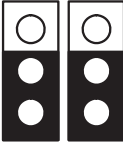
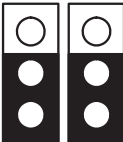




Table 22-2 Power Selection Settings for TB84B8

“To User_Vcc” Setting	Operating Mode	Comments
To user_Vcc off  on		The SMDS2/SMDS2+ main board supplies V _{CC} and V _{SS} to the target board (evaluation chip) and target system.
To user_Vcc off  on		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:



Table 22-3 Using Single Header Pins to Select Clock Source and Enable/Disable PWM

Target Board Part	Comments
<p>Board CLK</p>  <p>JP5 Clock Source</p> <p>Inner CLK</p>	<p>Use SMDS2/SMDS2+ internal clock source as the system clock (Default setting).</p>
<p>Board CLK</p>  <p>JP5 Clock Source</p> <p>Inner CLK</p>	<p>Use external crystal or ceramic oscillator as the system clock.</p>
<p>PWM Enable</p>  <p>JP6</p> <p>PWM Disable</p>	<p>PWM stops output as the emulator pauses</p>
<p>PWM Enable</p>  <p>JP6</p> <p>PWM Disable</p>	<p>PWM keeps output as the emulator pauses (Default setting).</p>
<p>Main Mode</p>  <p>JP7</p> <p>EVA Mode</p>	<p>S3E84B0 runs in the Main mode, similar to S3F84B8. The debug interface is not available.</p>
<p>Main Mode</p>  <p>JP7</p> <p>EVA Mode</p>	<p>S3E84B0 runs in the EVA mode (Default setting). While debugging the program, set the jumper in this mode.</p>

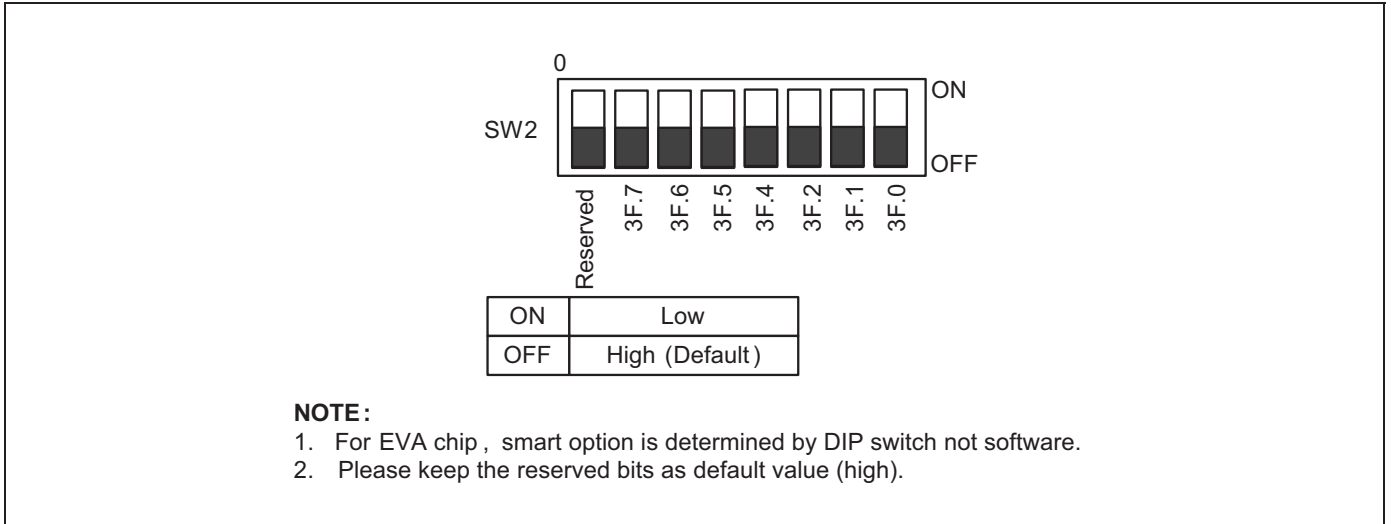


Figure 22-3 DIP Switch for Smart Option

- IDLE LED**
 This LED is ON when the evaluation chip (S3E84B0) is in the Idle mode.
- STOP LED**
 This LED is ON when the evaluation chip (S3E84B0) is in the Stop mode.

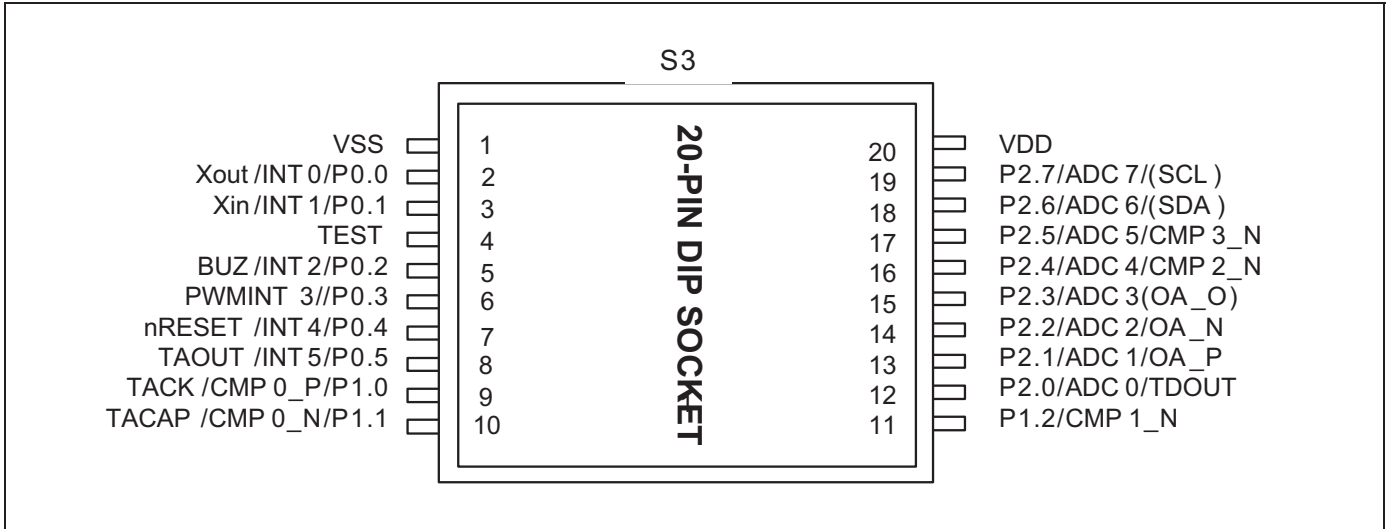


Figure 22-4 40-Pin Connector for TB84B8

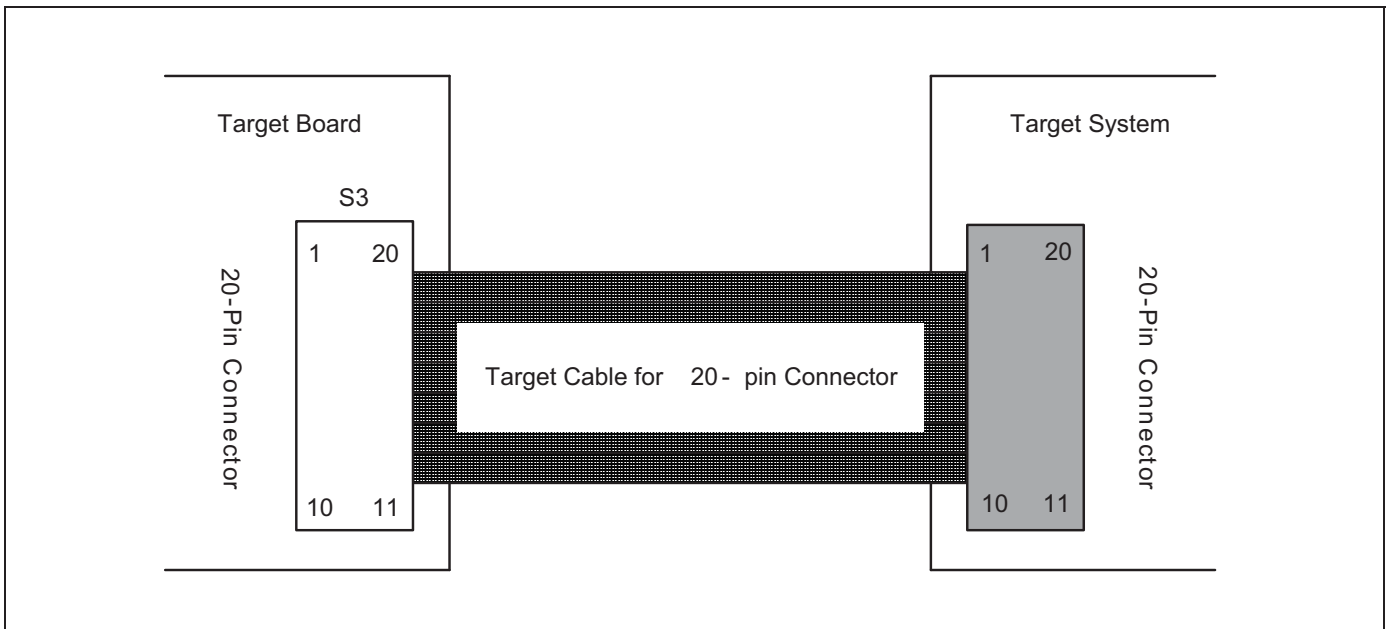


Figure 22-5 S3F84B8 Probe Adapter for 20-DIP Package

22.4 THIRD PARTIES FOR DEVELOPMENT TOOLS

Zilog provides a complete line of development tools that support the S3 Family of microcontroller. With wide experience in developing MCU systems, these third party firms are bonafide leaders in MCU development tool technology.

In-Circuit Emulators

- OPENice-i500/2000
- SK-1200 SmartKit

OTP/MTP Programmer

- GW-Uni2
- AS-Pro2
- Elnec programmers

To obtain the S3 Family development tools that will satisfy your S3F84B8 development objectives, contact your local [Zilog Sales Office](#), or visit Zilog's [Third Pary Tools](#) page to review our list of third party tool suppliers.

23 MECHANICAL DATA

23.1 OVERVIEW OF MECHANICAL DATA

S3F84B8 is available in a 20-pin DIP package (Zilog: 20-DIP-300A) and a 20-pin SOP package (Zilog: 20-SOP-375).

[Figure 23-1](#) and [Figure 23-2](#) show the 20-DIP-300A and 20-SOP-375 package dimensions, respectively.

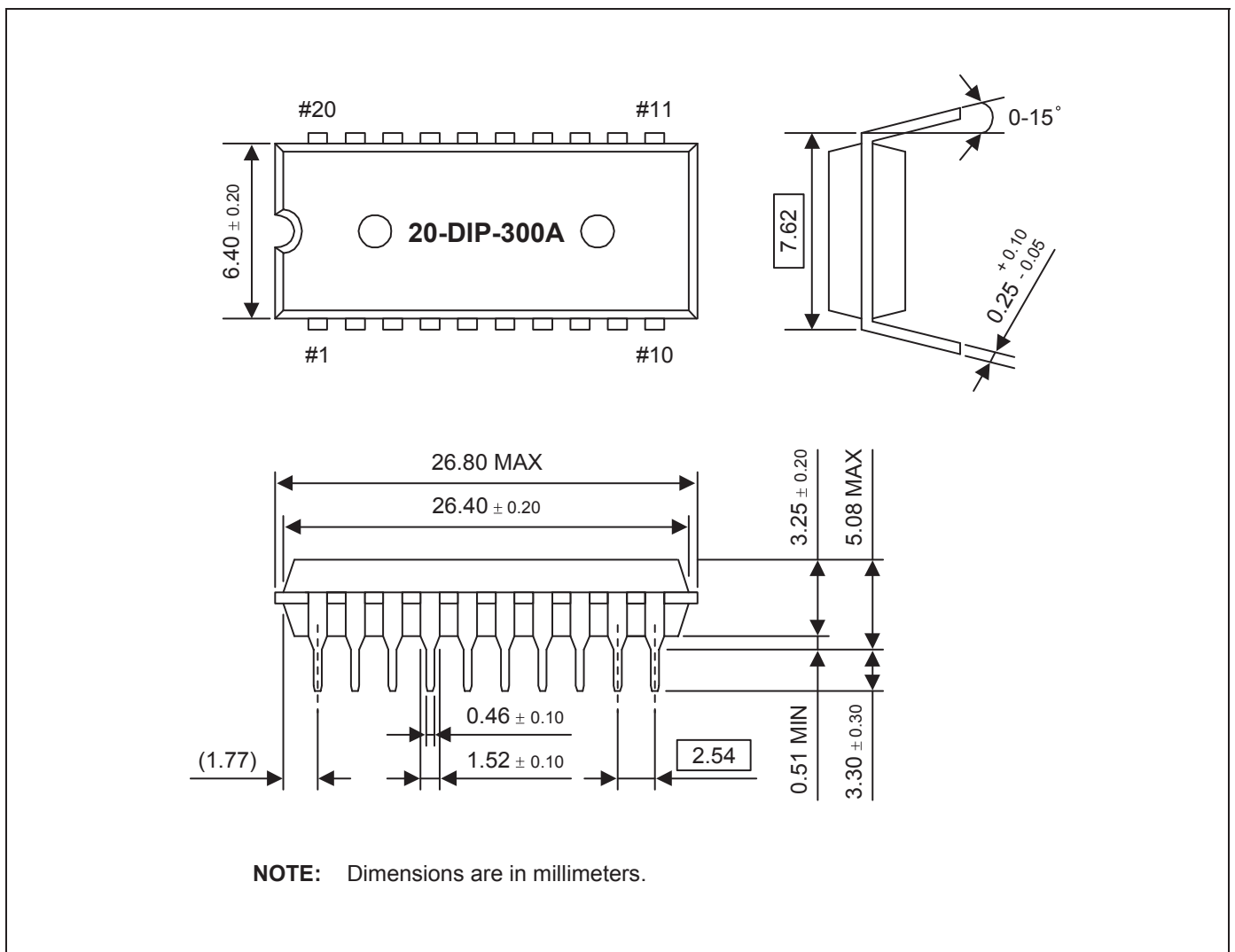
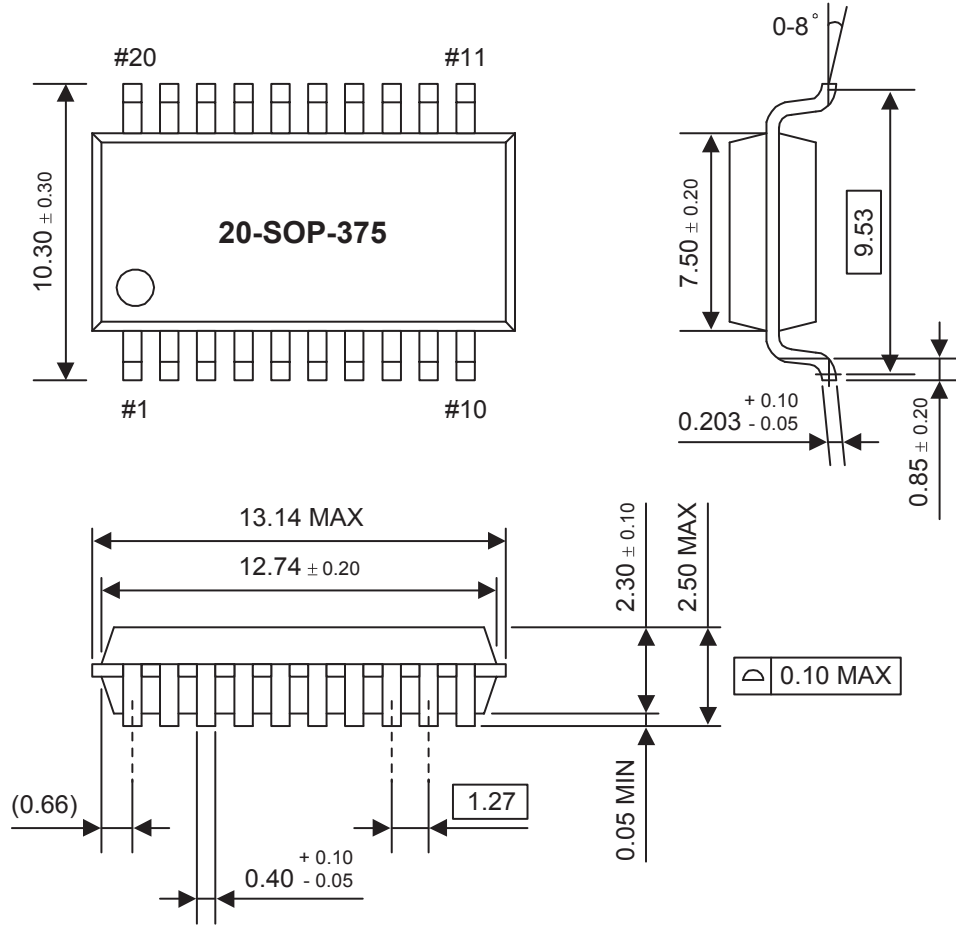


Figure 23-1 20-DIP-300A Package Dimensions



NOTE: Dimensions are in millimeters.

Figure 23-2 20-SOP-375 Package Dimensions