S3F84K4

8-BIT CMOS
MICROCONTROLLERS
USER'S MANUAL

Revision 1.0

SAMSUNG
ELECTRONICS

Important Notice

S3F84K4 8-Bit CMOS Microcontrollers
User's Manual, Revision 1.0
Publication Number: 21.0-S3-F84K4-102006

Samsung Electronics' microcontroller business has been awarded full ISO-14001 certification (BSI Certificate No. FM24653). All semiconductor products are designed and manufactured in accordance with the highest quality standards and objectives.

Samsung Electronics Co., Ltd.
San #24 Nongseo-Dong, Giheung-Gu
Yongin-City, Gyeonggi-Do, Korea
C.P.O. Box #37, Suwon 446-711

TEL: (82)-(31)-209-5238
FAX: (82)-(31)-209-6494

Home-Page URL: Http://www.samsungsemi.com
Printed in the Republic of Korea

NOTIFICATION OF REVISIONS

ORIGINATOR: Samsung Electronics, LSI Development Group, Ki-Heung, South Korea

PRODUCT NAME: S3F84K4 8-bit CMOS Microcontroller

DOCUMENT NAME: S3F84K4 User's Manual, Revision 1.0

DOCUMENT NUMBER: 21.0-S3-F84K4-102006

EFFECTIVE DATE: October, 2006

SUMMARY: As a result of additional product testing and evaluation, some specifications published in S3F84K4 User's Manual, Revision 0, have been changed.
These changes for S3F84K4 microcontroller, which are described in detail in the *Revision Descriptions* section below, are related to the followings:

— Chapter 1. Overview
— Chapter 2. Address Spaces
— Chapter 7. Clock Circuit
— Chapter 8. RESET and Power Down
— Chapter 10. Basic Timer and Timer 0
— Chapter 12. A/D Converter
— Chapter 13. Electrical Data

DIRECTIONS: Please note the changes in your copy (copies) of the S3F84K4 User’s Manual, Revision 0. Or, simply attach the *Revision Descriptions* of the next page to S3F84K4 User’s Manual, Revision 0.

REVISION HISTORY

Revision	Date	Remark
0	July, 2005	Preliminary Spec for internal release only.
1	October, 2006	First edition.

REVISION DESCRIPTIONS

1. CHAPTER 1 OVERVIEW

Page 1-2

Oscillation Frequency

- 1 MHz to 8 MHz external crystal oscillator
- Typical 4MHz external RC oscillator
- Internal RC: 8 MHz (typ.), 1 MHz (typ.) at V_{DD} = 5 V
- Maximum 8 MHz CPU clock

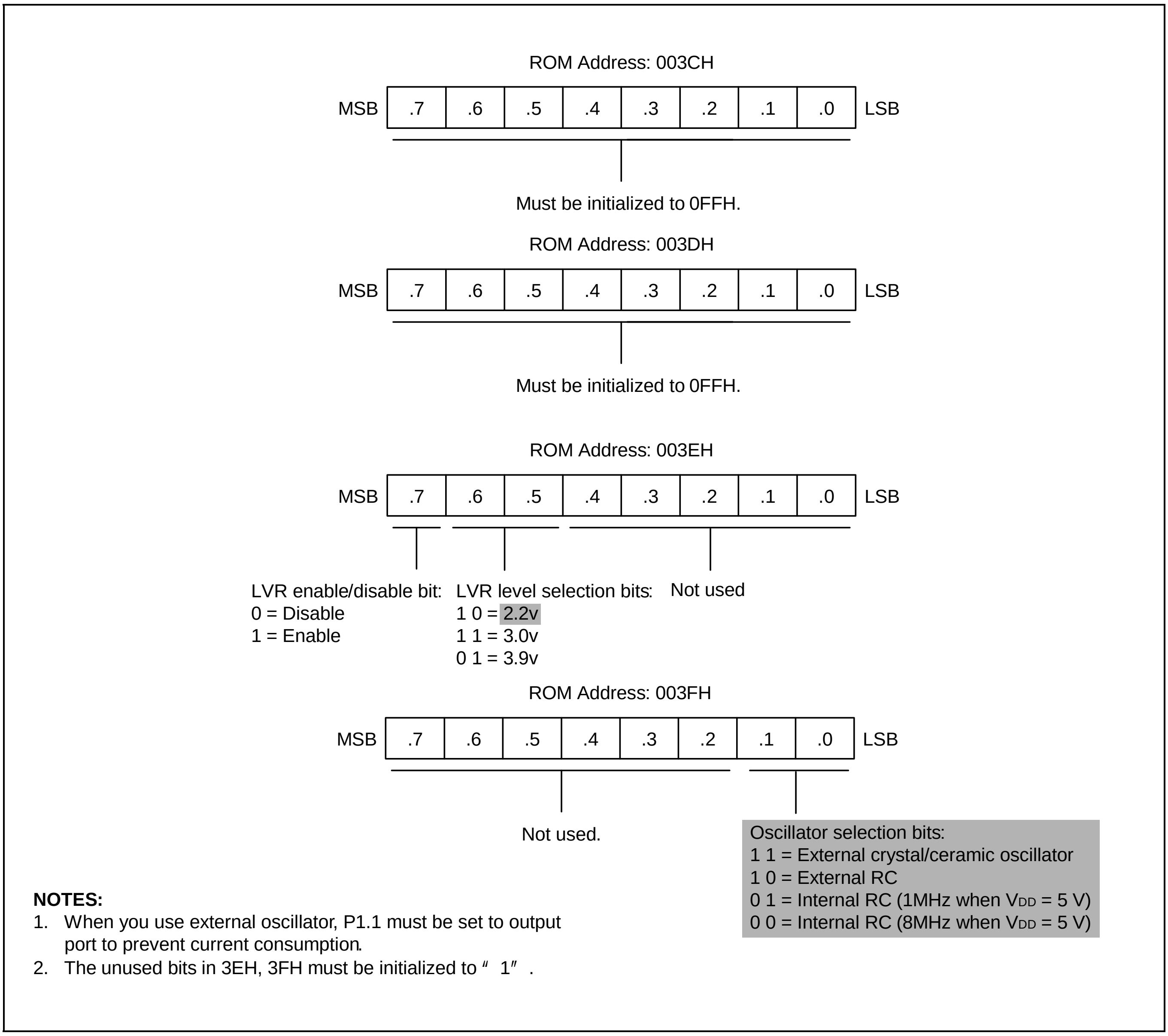
Page 1-7

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

Main Chip	During Programming			
Pin Name	**Pin Name**	**Pin No.**	**I/O**	**Function**
P0.1	SDA	18 (20-pin) 14 (16-pin)	I/O	Serial data pin (output when reading, Input when writing) Input and push-pull output port can be assigned
P0.0	SCL	19 (20-pin) 15 (16-pin)	I	Serial clock pin (input only pin)
RESET, P1.0	V_{PP}	4	I	Power supply pin for flash ROM cell writing (indicates that MTP enters into the writing mode). When 12 V is applied, MTP is in writing mode and when 5 V is applied, MTP is in reading mode. (Option) **Note:** **A 100pF capacitor must be connected between Vpp and Vss when data in the flash ROM are read or written by a tool (SPW2+, Gang Writer).**
V_{DD}/V_{SS}	V_{DD}/V_{SS}	20 (20-pin), 16 (16-pin) 1 (20-pin), 1 (16-pin)	I	Logic power supply pin.

2. CHAPTER 2 ADRESS SPACES

Page 2-3



NOTES:
1. When you use external oscillator, P1.1 must be set to output port to prevent current consumption.
2. The unused bits in 3EH, 3FH must be initialized to " 1" .

Figure 2-2. Smart Option

3. CHAPTER 7. CLOCK CIRCUIT

Page 7-1

OVERVIEW

By smart option (3FH.0 in ROM), user can select internal oscillator or external oscillator. When using external RC oscillator or internal RC oscillator, XOUT (P1.1) can be used as normal I/O pins. An internal RC oscillator source provides a typical 8 MHz or 1 MHz (at V_{DD} = 5 V) depending on smart option.

4. CHAPTER 8. RESET AND POWER DOWN

Page 8-1

The nRESET pin must be held to Low level for a minimum time interval after the power supply comes within tolerance in order to allow time for internal CPU clock oscillation to stabilize. The minimum required oscillation stabilization time for a reset is approximately 65.5 ms (@ $2^{16}/f_{OSC}$, f_{OSC} = 8 MHz).

……

The on-chip Low Voltage Reset, features static Reset when supply voltage is below a reference value (Typ. 2.2, 2.9, 3.9V). Thanks to this feature, external reset circuit can be removed while keeping the application safety. As long as the supply voltage is below the reference value, there is an internal and static RESET. The MCU can start only when the supply voltage rises over the reference value.

Page 8-3

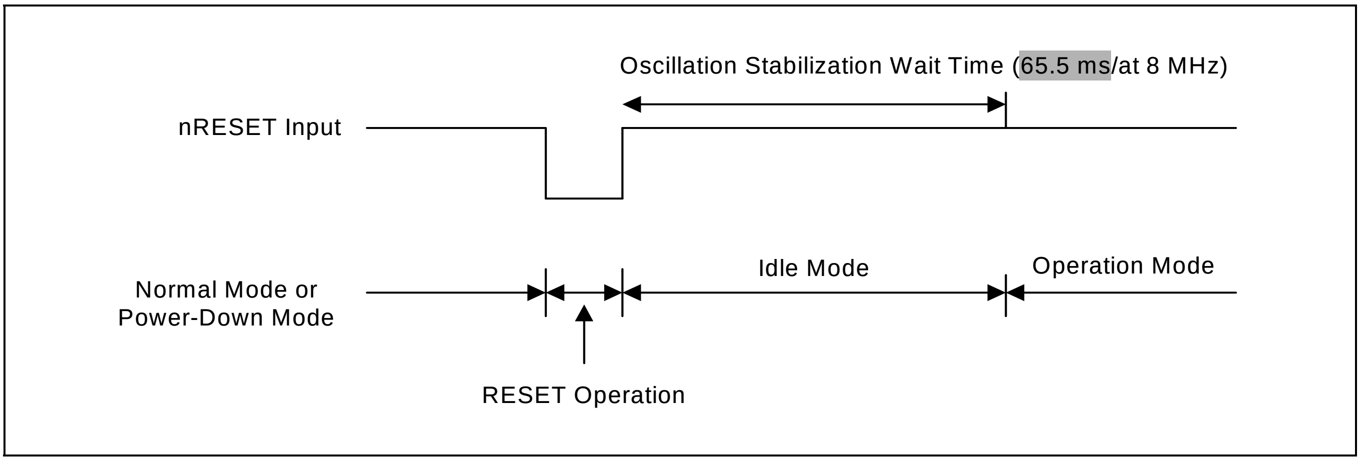


Figure 8-3. Timing for S3F84K4 after RESET

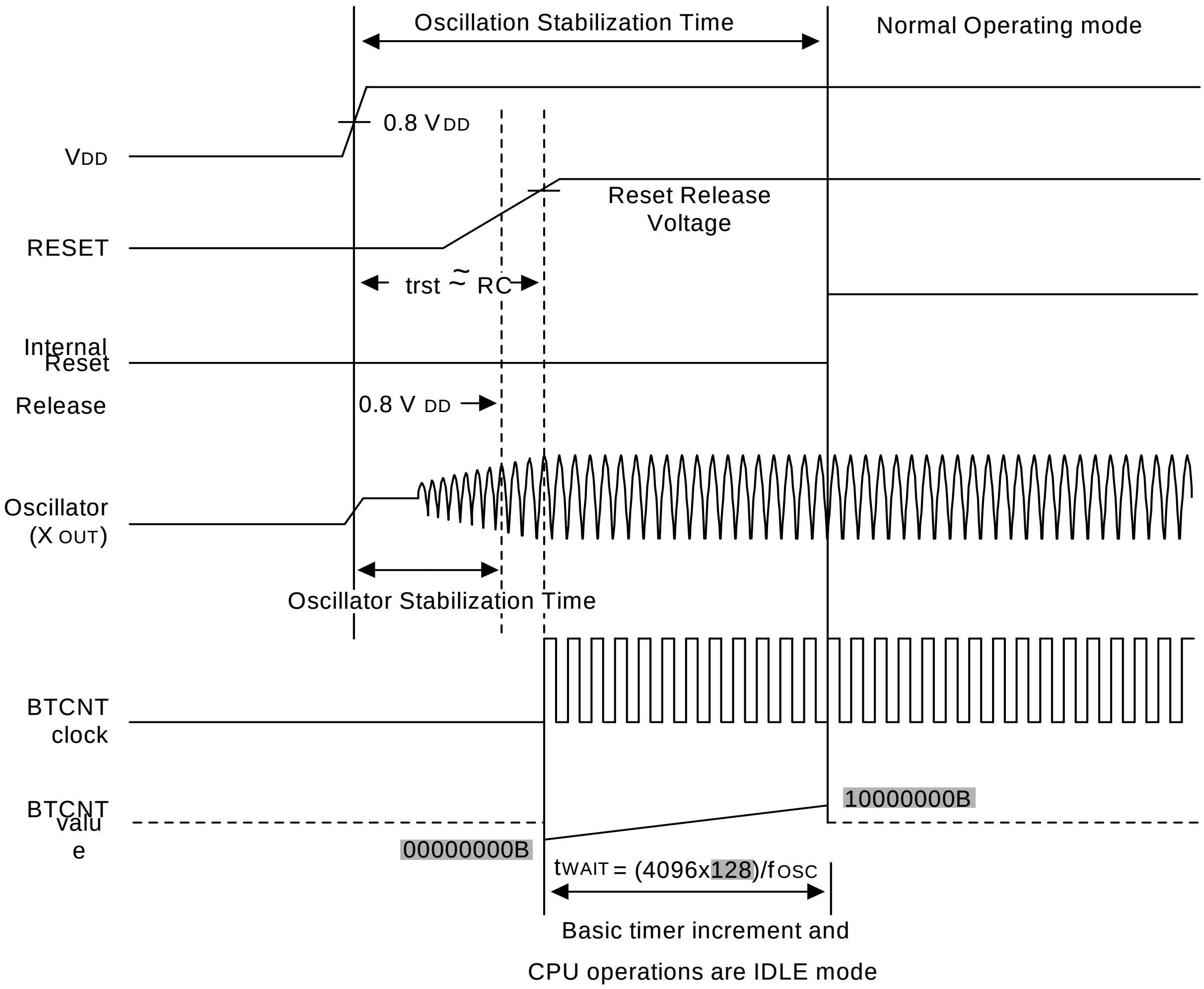
5. CHAPTER 10. BASIC TIMER AND TIMER 0

Page 10-3

When BTCNT.**7** is set, a signal is generated to indicate that the stabilization interval has elapsed and to gate the clock signal off to the CPU so that it can resume normal operation.

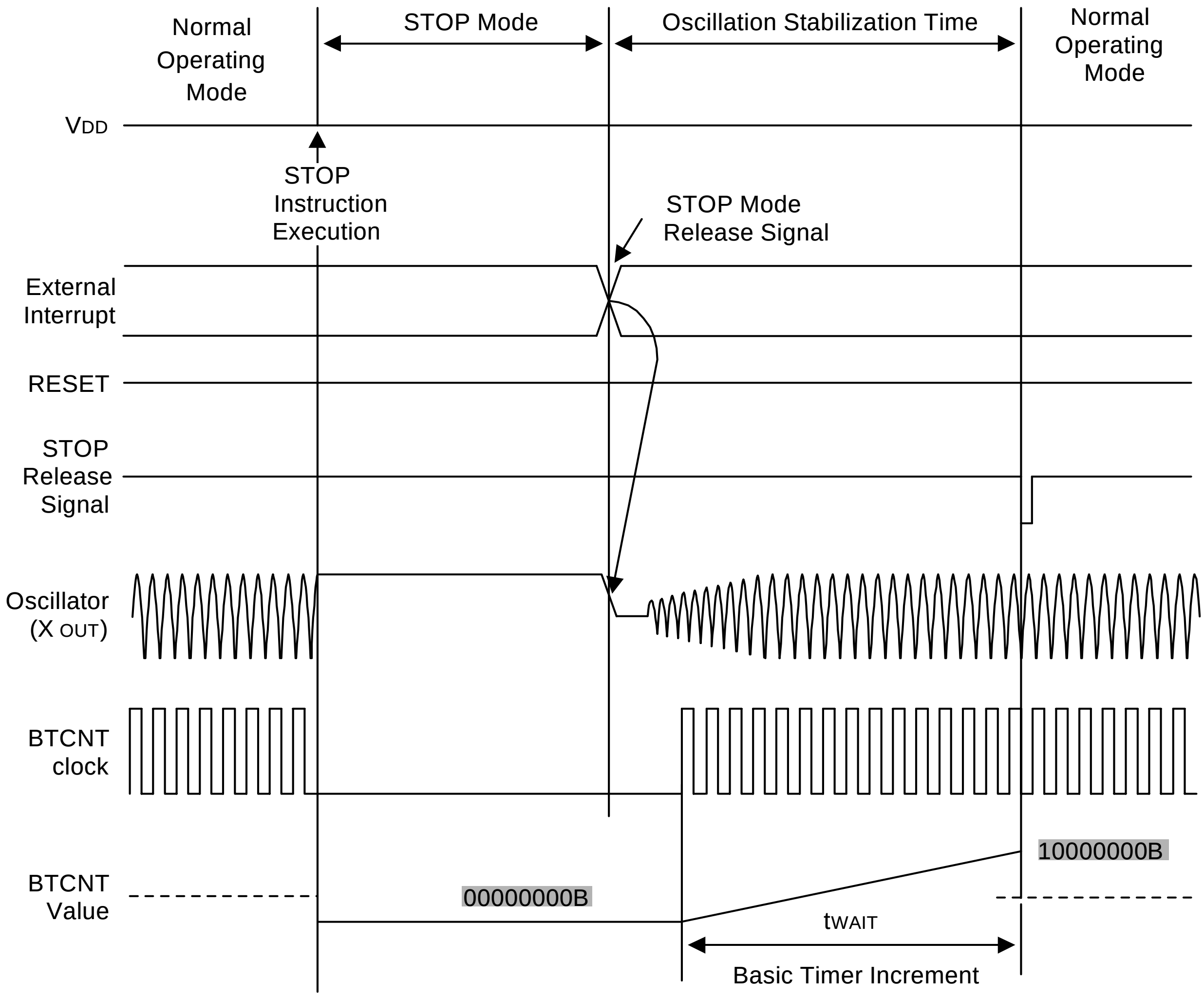
In summary, the following events occur when Stop mode is released:

1. During Stop mode, an external power-on Reset or an external interrupt occurs to trigger the Stop mode release and oscillation starts.
2. If an external power-on Reset occurred, the basic timer counter will increase at the rate of f_{OSC}/4096. If an external interrupt is used to release Stop mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until bit **7** of the basic timer counter is set.
4. When a BTCNT.**7** is set, normal CPU operation resumes.



NOTE: Duration of the oscillator stabilization wait time, t_{WAIT}, when it is released by a Power-on-reset is 4096 x 128/f_{OSC}.
$t_{RST} \approx$ RC (R and C are value of external power on Reset)

Figure 10-2. Oscillation Stabilization Time on RESET



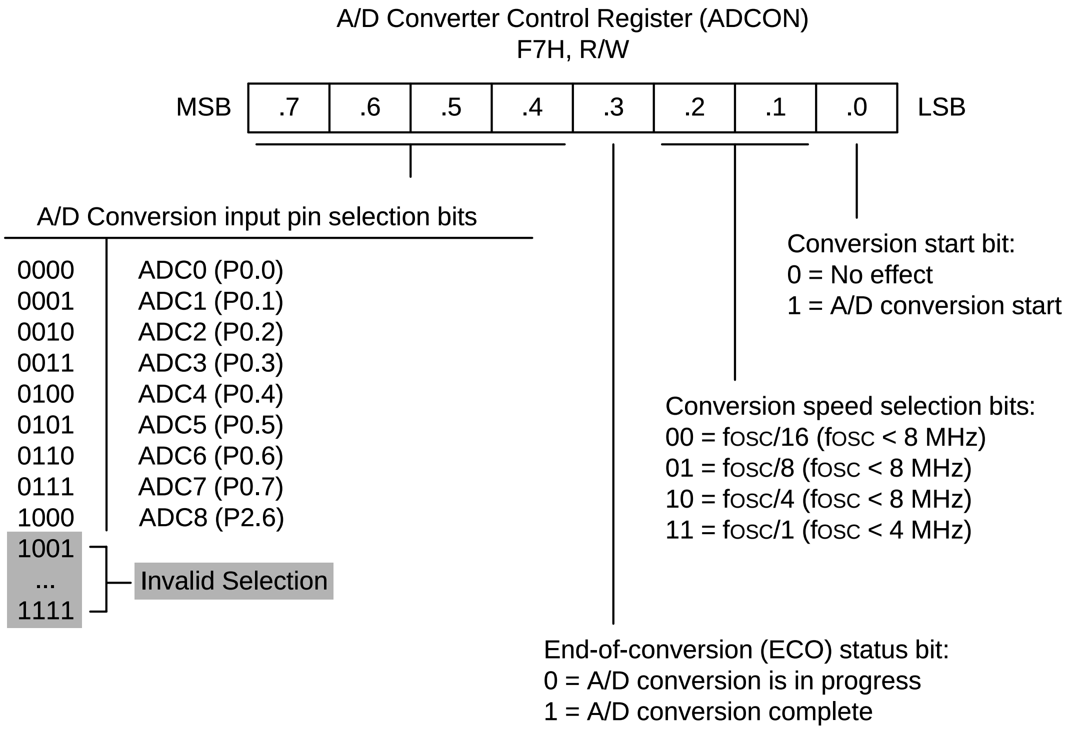
NOTE: Duration of the oscillator stabilzation wait time, t WAIT , it is released by an interrupt is determined by the setting in basic timer control register, BTCON.

BTCON.3	BTCON.2	t WAIT	t WAIT (When fosc is 8 MHz)
0	0	(4096 x 128)/fosc	65.5 ms
0	1	(1024 x 128)/fosc	16.4 ms
1	0	(128 x 128)/fosc	2.04 ms
1	1	Invalid setting	—

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

6. CHAPTER 12. A/D CONVERTER

Page 12-2



NOTE:
1. Maximum ADC clock input = 4 MHz
2. Do NOT set ADCON.7-4 to any value of the Invalid Selection.

Figure 12-1. A/D Converter Control Register (ADCON)

7. CHAPTER 13. ELECTRICAL DATA

Page 13-3

Table 13-2. DC Electrical Characteristics

(T_A = – 25 °C to + 85 °C, V_{DD} = 2.0 V to 5.5 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
……							
Supply current	I_{DD1}	Run mode 8 MHz CPU clock	V_{DD} = 2.0 to 5.5 V	–	3	6	mA
	I_{DD2}	Idle mode 8 MHz CPU clock	V_{DD} = 2.0 to 5.5 V	–	2	4	
	I_{DD3}	Stop mode T_A = 25°C	V_{DD} = 2.0 to 5.5 V	–	200	400	uA

Page 13-5

Table 13-4. Oscillator Characteristics

(T_A = – 25 °C to + 85 °C)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
……						
External RC oscillator	–	V_{DD} = 5 V	–	4	–	MHz
Internal RC oscillator	–	V_{DD} = 5 V	–	8		MHz
		Tolerance:20% at T_A =25°C		1		MHz

NOTE: For the resistor of External RC oscillator, we recommend using 28kΩ for 8MHz (at T_A =25°C).

Page 13-9

Table 13-8. LVR Circuit Characteristics

(T_A = 25 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Low voltage reset	V_{LVR}	–	1.9 2.6 3.4	2.2 3.0 3.9	2.5 3.4 4.4	V

Page 13-9

Table 13-9. AC Electrical Characteristics for Internal Flash ROM (Added)

(T_A = –25 °C to + 85 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Flash Erase/Write/Read Voltage	Fewrv	V_{DD}	2.0	5.0	5.5	V
Programming Time (note 1)	Ftp		32	–	60	μS
Sector Erasing Time (note 2)	Ftp1		10	–	20	mS
Chip Erasing Time (note 3)	Ftp2		50	–	100	mS
Data Access Time	Ft_{RS}	V_{DD} = 2.0 V	–	250	–	nS
Number of Writing/Erasing	FNwe	–	10,000	–	–	Times
Data Retention	Ftdr	–	10	–	–	Years

NOTES:
1. The programming time is the time during which one byte (8-bit) is programmed.
2. The Sector erasing time is the time during which all 128-bytes of one sector block is erased.
3. In the case of S3F84K4, the chip erasing is available in Tool Program Mode only.

Preface

The S3F84K4 *Microcontroller User's Manual* is designed for application designers and programmers who are using the S3F84K4 microcontroller for application development. It is organized in two main parts:

Part I Programming Model	Part II Hardware Descriptions

Part I contains software-related information to familiarize you with the microcontroller's architecture, programming model, instruction set, and interrupt structure. It has six chapters:

Chapter 1	Product Overview	Chapter 4	Control Registers
Chapter 2	Address Spaces	Chapter 5	Interrupt Structure
Chapter 3	Addressing Modes	Chapter 6	Instruction Set

Chapter 1, "Product Overview," is a high-level introduction to S3F84K4 with general product descriptions, as well as detailed information about individual pin characteristics and pin circuit types.

Chapter 2, "Address Spaces," describes program and data memory spaces, the internal register file, and register addressing. Chapter 2 also describes working register addressing, as well as system stack and user-defined stack operations.

Chapter 3, "Addressing Modes," contains detailed descriptions of the addressing modes that are supported by the S3F8-series CPU.

Chapter 4, "Control Registers," contains overview tables for all mapped system and peripheral control register values, as well as detailed one-page descriptions in a standardized format. You can use these easy-to-read, alphabetically organized, register descriptions as a quick-reference source when writing programs.

Chapter 5, "Interrupt Structure," describes the S3F84K4 interrupt structure in detail and further prepares you for additional information presented in the individual hardware module descriptions in Part II.

Chapter 6, "Instruction Set," describes the features and conventions of the instruction set used for all S3F8-series microcontrollers. Several summary tables are presented for orientation and reference. Detailed descriptions of each instruction are presented in a standard format. Each instruction description includes one or more practical examples of how to use the instruction when writing an application program.

A basic familiarity with the information in Part I will help you to understand the hardware module descriptions in Part II. If you are not yet familiar with the S3F8-series microcontroller family and are reading this manual for the first time, we recommend that you first read Chapters 1-3 carefully. Then, briefly look over the detailed information in Chapters 4, 5, and 6. Later, you can reference the information in Part I as necessary.

Part II "hardware Descriptions," has detailed information about specific hardware components of the S3F84K4 microcontroller. Also included in Part II are electrical, mechanical, flash, and development tools data. It has 9 chapters:

Chapter 7	Clock Circuit	Chapter 12	A/D Converter
Chapter 8	RESET and Power-Down	Chapter 13	Electrical Data
Chapter 9	I/O Ports	Chapter 14	Mechanical Data
Chapter 10	Basic Timer and Timer 0	Chapter 15	Development Tools
Chapter 11	12-bit PWM		

Table of Contents

Part I — Programming Model

Chapter 1 Product Overview

Chapter 2 Address Spaces

Chapter 3 Addressing Modes

Table of Contents (Continued)

Chapter 4 Control Registers

Chapter 5 Interrupt Structure

Chapter 6 Instruction Set

Table of Contents (Continued)

Part II Hardware Descriptions

Chapter 7 Clock Circuit

Chapter 8 RESET and Power-Down

Chapter 9 I/O Ports

Chapter 10 Basic Timer and Timer 0

Table of Contents (Continued)

Chapter 11 12-bit PWM

Chapter 12 A/D Converter

Chapter 13 Electrical Data

Chapter 14 Mechanical Data

Chapter 15 Development Tools

List of Figures

List of Figures (Continued)

Figure Number	Title	Page Number

List of Figures (Concluded)

List of Tables

List of Programming Tips

Description **Page Number**

List of Register Descriptions

List of Instruction Descriptions

List of Instruction Descriptions (Continued)

1 PRODUCT OVERVIEW

S3F8-SERIES MICROCONTROLLERS

Samsung's S3F8 series of 8-bit single-chip CMOS microcontrollers offers a fast and efficient CPU, a wide range of integrated peripherals, and various mask-programmable ROM sizes. Important CPU features include:

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

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

S3F84K4 MICROCONTROLLER

The S3F84K4 single-chip CMOS micro-controller is fabricated using a highly advanced CMOS process and is based on Samsung's newest CPU architecture. Its design is based on the powerful SAM8RC CPU core. Stop and idle (power-down) modes were implemented to reduce power consumption.

The S3F84K4 is a micro-controller with a 4k-byte multi-time-programmable Flash ROM embedded.

Using the SAM8RC design approach, the following peripherals were integrated with the SAM8RC core:

- Three configurable I/O ports (17 pins)
- Five interrupt sources with five vectors and three interrupt levels
- A 16-bit timer 0 with one 16-bit timer or two 8-bit timers mode.
- Analog to digital converter with nine input channels (MAX) and 10-bit resolution
- One 12-bit PWM output

The S3F84K4 microcontroller is ideal for use in a wide range of electronic applications requiring simple timer/counter, PWM, ADC. S3F84K4 is available in a 20/16-pin DIP and a 20/16-pin SOP and a 20/16-pin SSOP package.

FEATURES

CPU

- SAM8RC CPU core

Memory

- 4-Kbyte internal program memory
- 208-byte general-purpose register area

Instruction Set

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

Instruction Execution Time

- 500 ns at 8 MHz f_{OSC} (minimum)

Interrupts

- 3 interrupt levels and 5 interrupt sources (2 external interrupt and 3 internal interrupt)
- Fast interrupt processing feature

General I/O

- Three I/O ports (Max 17 pins)
- Bit programmable ports

12-bit High-speed PWM

- 12-bit PWM 1-ch
- 6-bit base + 6-bit extension

Built-in Reset Circuit

- Low voltage detector for safe Reset

Timer/Counters

- One 8-bit basic timer for watchdog function
- One 16-bit timer or two 8-bit timers A/B with time interval mode

A/D Converter

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

Oscillation Frequency

- 1 MHz to 8 MHz external crystal oscillator
- Typical 4MHz external RC oscillator
- Internal RC: 8 MHz (typ.), 1 MHz (typ.) at V_{DD} = 5 V
- Maximum 8 MHz CPU clock

Operating Temperature Range

- – 25°C to + 85°C

Operating Voltage Range

- 2.0 V to 5.5 V (LVR disable)
- LVR to 5.5V (LVR enable)

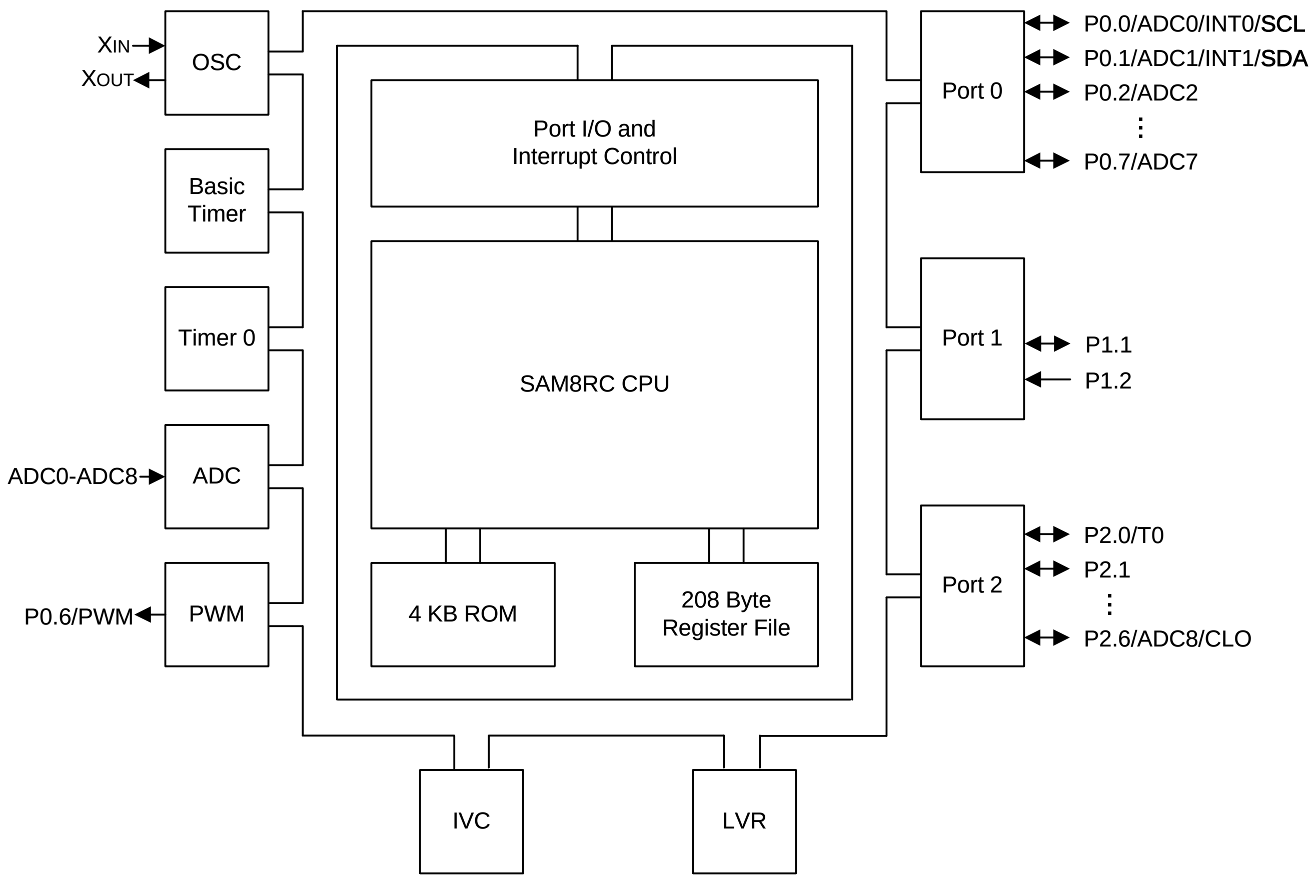
Smart Option

- LVR enable/disable
- Oscillator selection

Package Types

– 20-SSOP-225
– 20-DIP-300A
– 20-SOP-375
– 16-SOP-BD300-SG
– 16-DIP-300A
– 16-SSOP-BD44

BLOCK DIAGRAM



NOTE: 1. P1.2 is used as input only. P1.0 is reserved.
2. IVC (Internal Voltage Converter) is for S3F84K4's 0.35µm process. It's not configurable.

Figure 1-1. Block Diagram

PIN ASSIGNMENTS

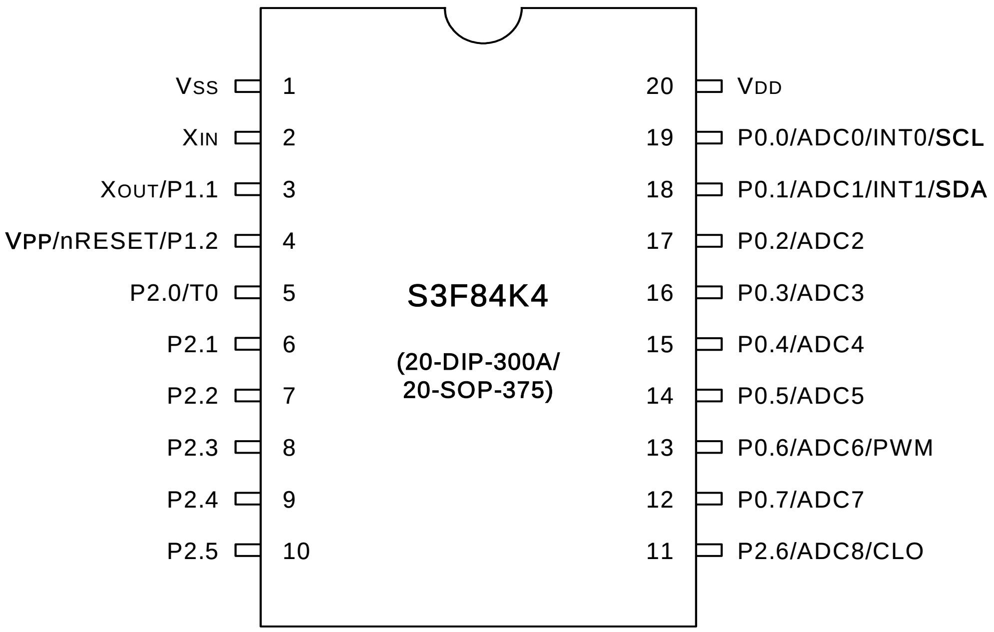


Figure 1-2. Pin Assignment Diagram (20-Pin DIP/SOP Package)

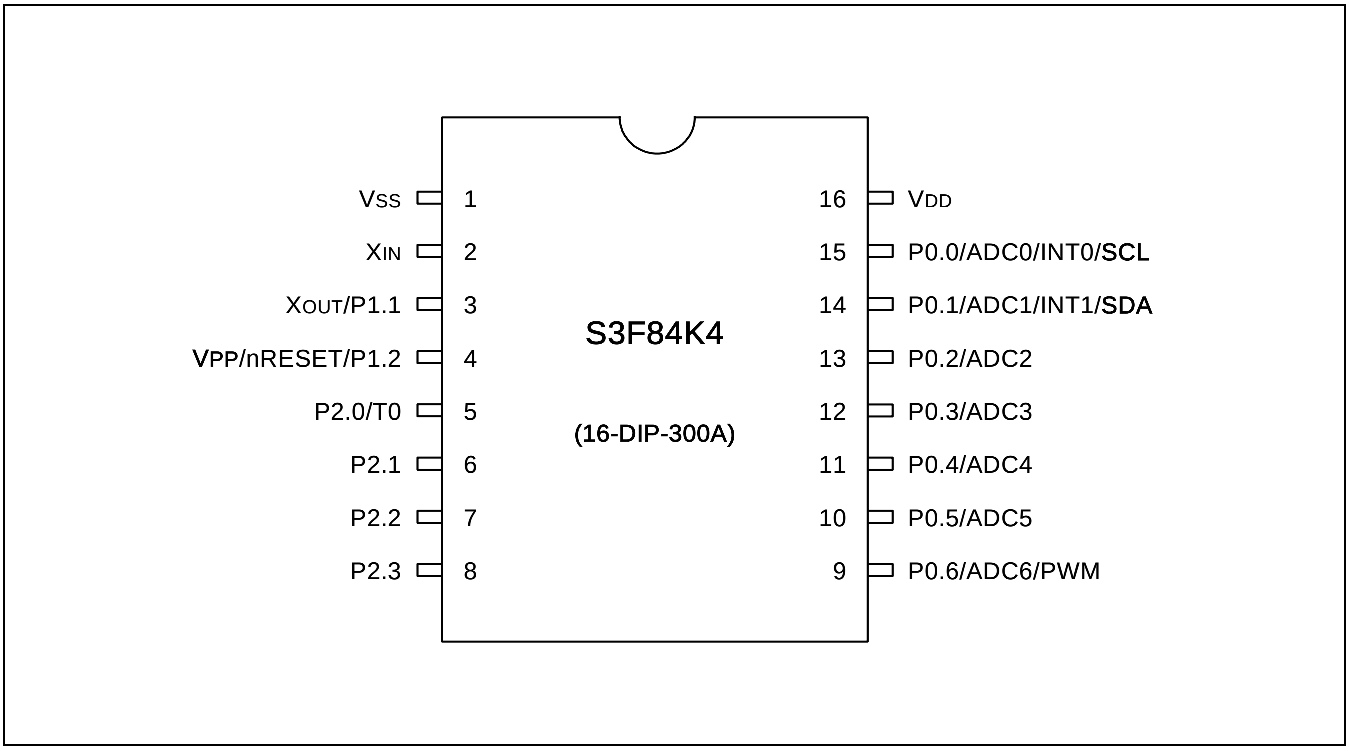


Figure 1-3. Pin Assignment Diagram (16-Pin DIP Package)

PIN DESCRIPTIONS

Table 1-1. S3F84K4 Pin Descriptions

Pin Name	Input/ Output	Pin Description	Pin Type	Share Pins
P0.0–P0.7	I/O	Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are assignable by software. Port0 pins can also be used as A/D converter input, PWM output or external interrupt input.	E-1	ADC0–ADC7 INT0/INT1 PWM
P1.1	I/O	Bit-programmable I/O port for Schmitt trigger input or push-pull, open-drain output. Pull-up resistors or pull-down resistors are assignable by software.	E-2	X_{OUT}
P1.2	I	Schmitt trigger input port	B	RESET
P2.0–P2.6	I/O	Bit-programmable I/O port for Schmitt trigger input or push-pull, open-drain output. Pull-up resistors are assignable by software.	E	– ADC8/CLO T0
X_{IN}, X_{OUT}	–	Crystal/Ceramic, or RC oscillator signal for system clock.		X_{OUT} is shared with P1.1
nRESET	I	Internal LVR or external RESET	B	P1.2
V_{DD}, V_{SS}	–	Voltage input pin and ground		–
CLO	O	System clock output port	E	P2.6
INT0–INT1	I	External interrupt input port	E-1	P0.0, P0.1
PWM	O	12-Bit high speed PWM output	E-1	P0.6
T0	O	Timer0/A match output	E	P2.0
ADC0–ADC8	I	A/D converter input	E-1 E	P0.0–P0.7 P2.6

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

Main Chip	During Programming			
Pin Name	**Pin Name**	**Pin No.**	**I/O**	**Function**
P0.1	SDA	18 (20-pin) 14 (16-pin)	I/O	Serial data pin (output when reading, Input when writing) Input and push-pull output port can be assigned
P0.0	SCL	19 (20-pin) 15 (16-pin)	I	Serial clock pin (input only pin)
RESET, P1.0	V_{PP}	4	I	Power supply pin for flash ROM cell writing (indicates that MTP enters into the writing mode). When 12 V is applied, MTP is in writing mode and when 5 V is applied, MTP is in reading mode. (Option) **Note:** **A 100pF capacitor must be connected between Vpp and Vss when data in the flash ROM are read or written by a tool (SPW2+, Gang Writer).**
V_{DD}/V_{SS}	V_{DD}/V_{SS}	20 (20-pin), 16 (16-pin) 1 (20-pin), 1 (16-pin)	I	Logic power supply pin.

PIN CIRCUITS

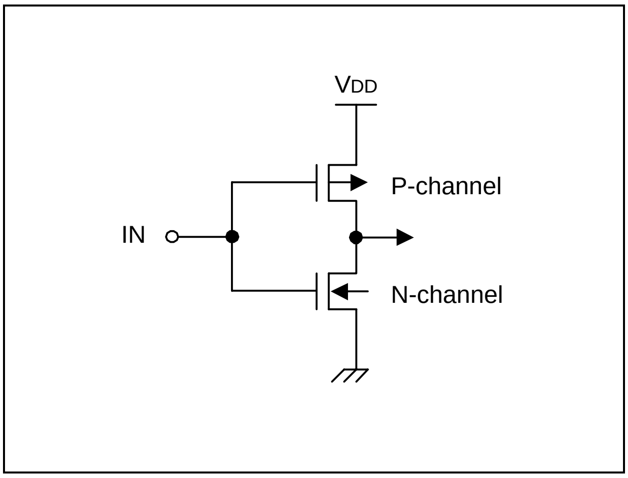


Figure 1-5. Pin Circuit Type A

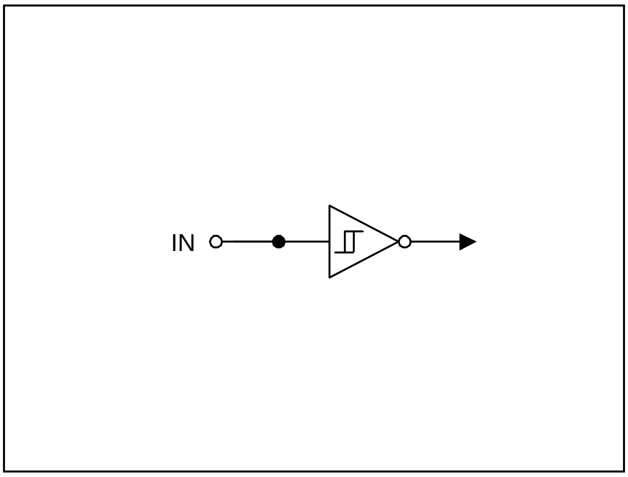


Figure 1-6. Pin Circuit Type B

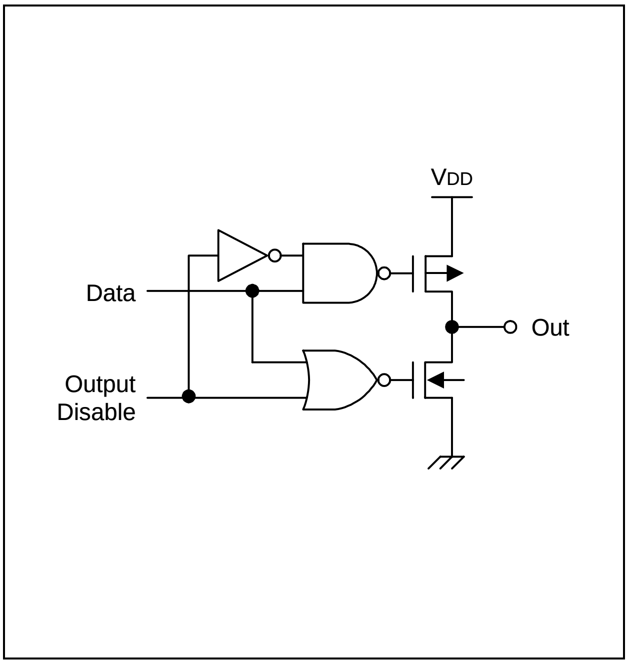


Figure 1-7. Pin Circuit Type C

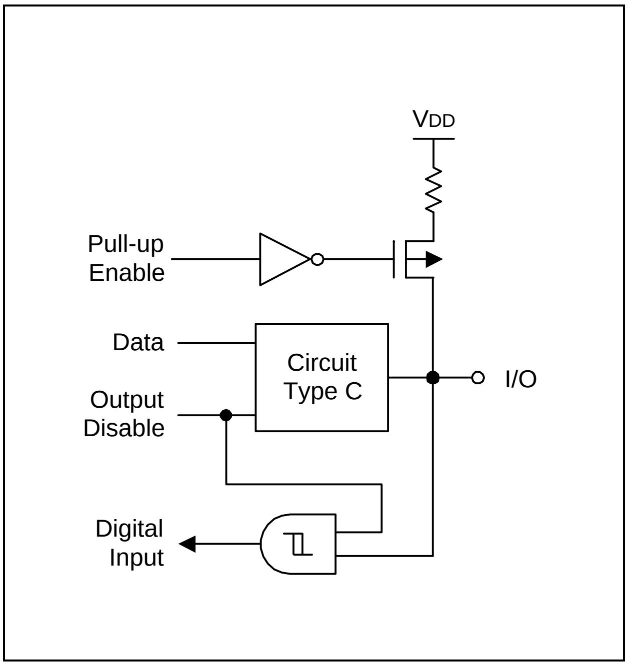


Figure 1-8. Pin Circuit Type D

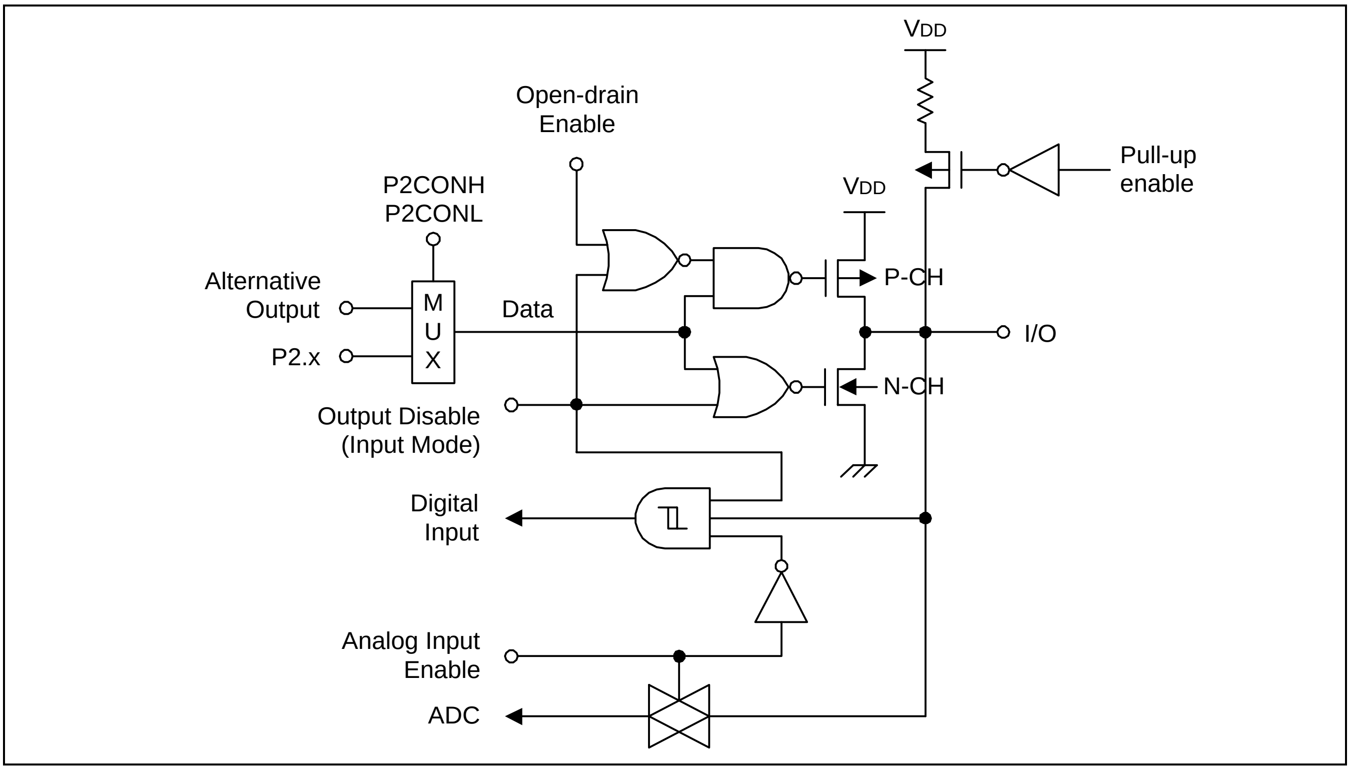


Figure 1-9. Pin Circuit Type E

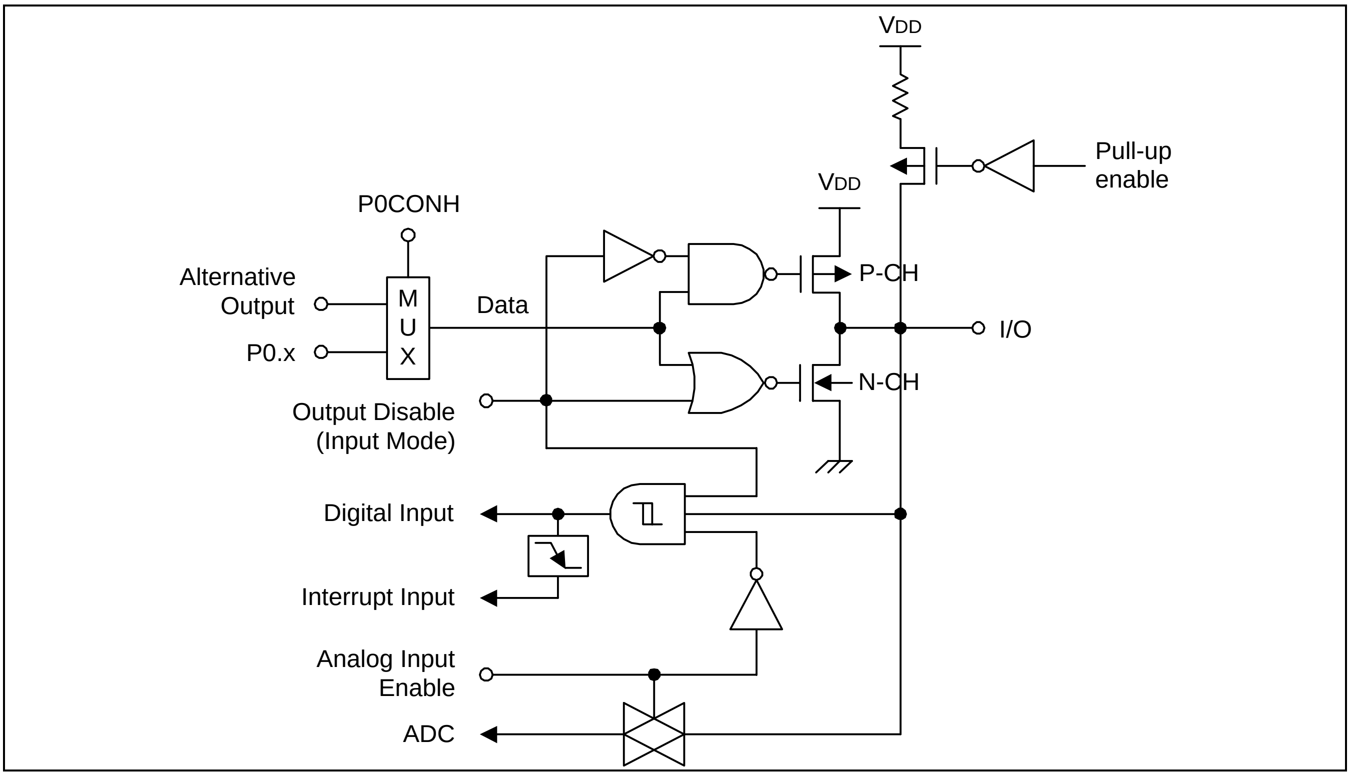


Figure 1-10. Pin Circuit Type E-1

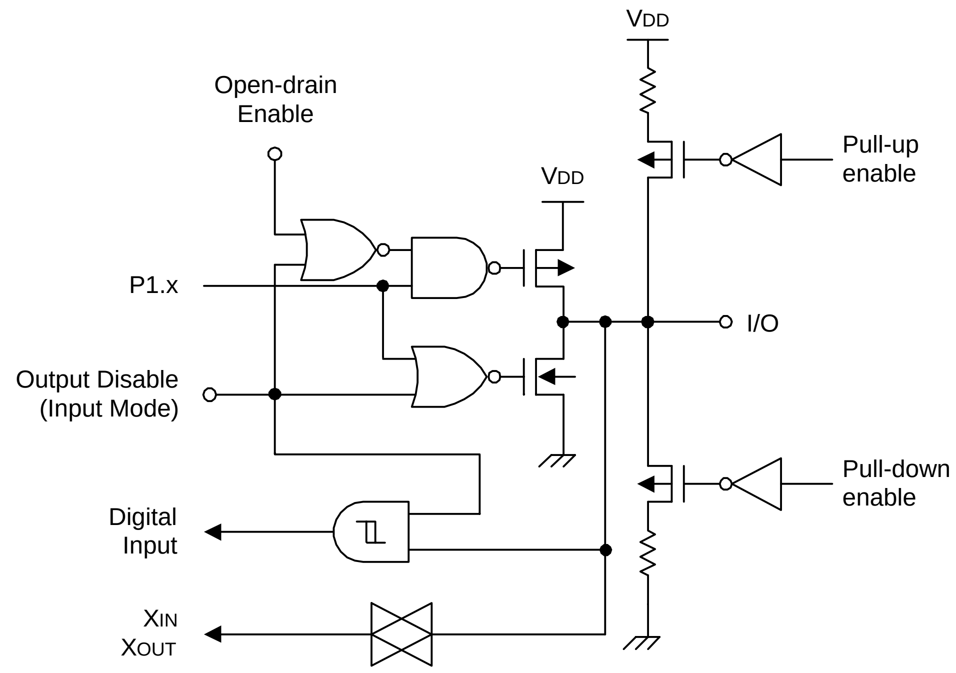


Figure 1-11. Pin Circuit Type E-2

2 ADDRESS SPACES

OVERVIEW

The S3F84K4 microcontroller has two kinds of address space:

— Internal program memory (ROM)

— Internal register file

A 12-bit address bus supports program memory operations. A separate 8-bit register bus carries addresses and data between the CPU and the internal register file.

The S3F84K4 have 4-Kbytes of multi-time-programmable Flash program memory: which is configured as the Internal ROM mode, all of the 4-Kbyte internal program memory is used.

The S3F84K4 microcontroller has 208 general-purpose registers in its internal register file. 37 bytes in the register file are mapped for system and peripheral control functions.

PROGRAM MEMORY (ROM)

Normal Operating Mode

The S3F84K4 have 4-Kbytes (locations 0H–0FFFH) of internal multi-time-programmable Flash program memory.

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

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

The program Reset address in the ROM is 0100H.

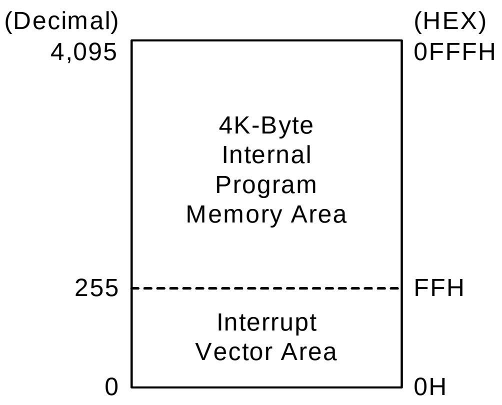
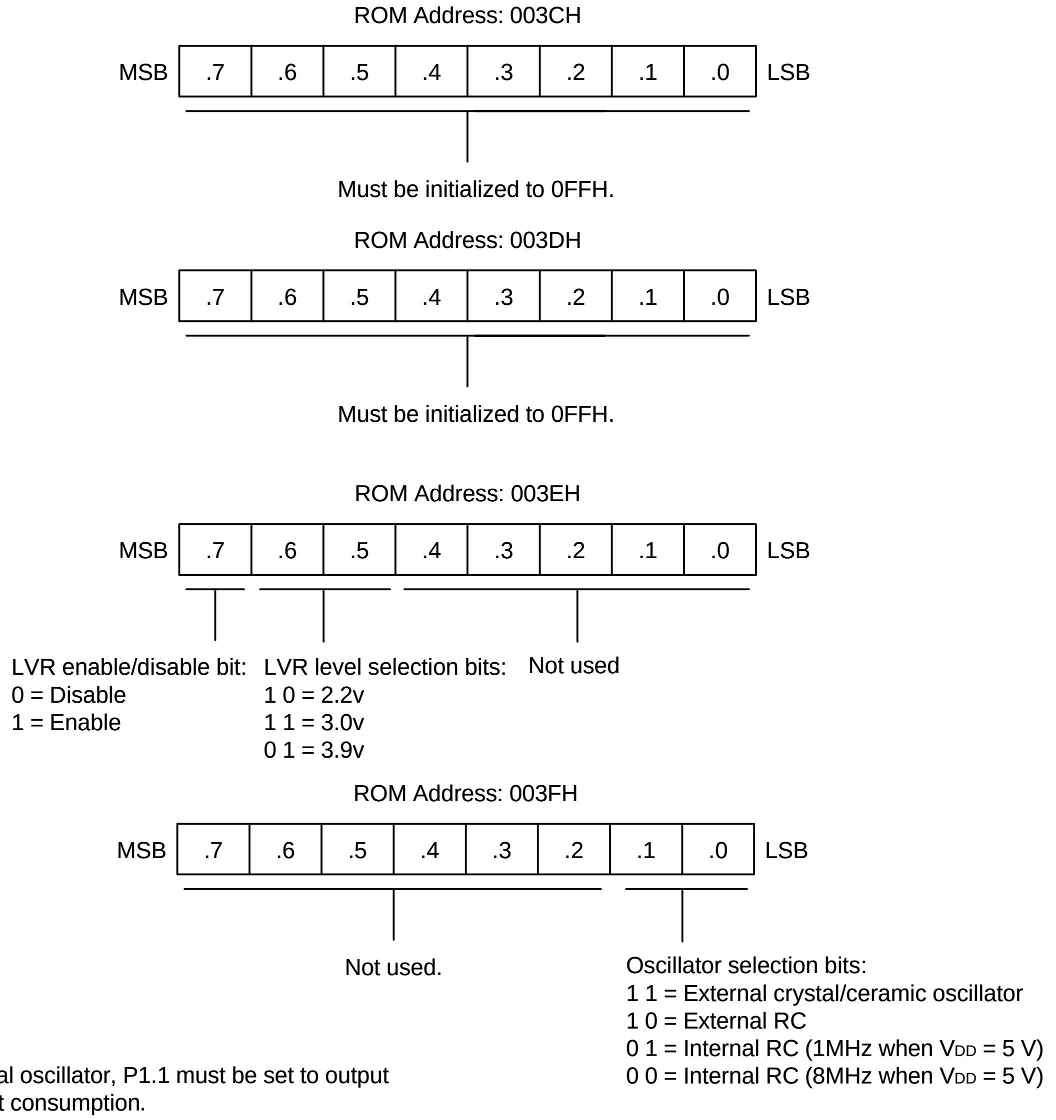


Figure 2-1. Program Memory Address Space

Smart Option

Smart option is the ROM option for starting condition of the chip.
The ROM addresses used by smart option are from 003CH to 003FH. The S3F84K4 only uses 003EH and 003FH. Not used ROM address 003CH, 003DH should be initialized to 0FFH. The default value of ROM is FFH (LVR enable, internal RC oscillator).



NOTES:
1. When you use external oscillator, P1.1 must be set to output port to prevent current consumption.
2. The unused bits in 3EH, 3FH must be initialized to “1”.

Figure 2-2. Smart Option

☞ PROGRAMMING TIP — Smart Option Setting

```
                  ORG        0000H
;---------------- << Smart Option Setting >>

                  ORG        003CH
                  DB         0FFH                ;  003CH, must be initialized to 0FFH.
                  DB         0FFH                ;  003DH, must be initialized to 0FFH.
                  DB         0FFH                ;  003EH, enable LVR (3.0v)
                  DB         0FEH                ;  003FH, External RC oscillator

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

                  VECTOR     0F6H, INT_TIMER0    ;  Timer 0 interrupt

;                 << Reset >>

                  ORG        0100H
                  RESET:     DI
                             •
                             •
                             •
```

REGISTER ARCHITECTURE

The upper 64-bytes of the S3F84K4's internal register file are addressed as working registers, system control registers and peripheral control registers. The lower 192-bytes of internal register file(00H–BFH) is called the *general purpose register space*. 245 registers in this space can be accessed; 208 are available for general-purpose use.

For many SAM8RC microcontrollers, the addressable area of the internal register file is further expanded by additional register pages at the general-purpose register space (00H–BFH: page0). This register file expansion is not implemented in the S3F84K4, however.

The specific register types and the area (in bytes) that they occupy in the internal register file are summarized in Table 2-1.

Table 2-1. Register Type Summary

Register Type	Number of Bytes
CPU and system control registers, peripherals, I/O, and clock control and data registers	37
General-purpose registers (including the 16-bit common working register area)	208
Total Addressable Bytes	245

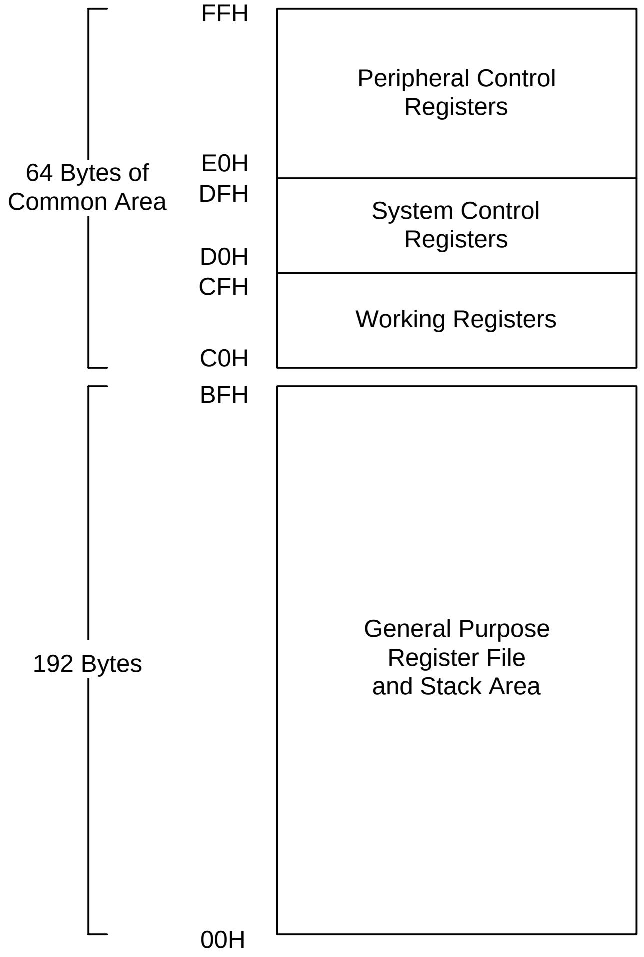


Figure 2-3. Internal Register File Organization

COMMON WORKING REGISTER AREA (C0H–CFH)

The SAM8RC register architecture provides an efficient method of working register addressing that takes full advantage of shorter instruction formats to reduce execution time.

This 16-byte address range is called common area. That is, locations in this area can be used as working registers by operations that address any location on any page in the register file. Typically, these working registers serve as temporary buffers for data operations between different pages. However, because the S3F84K4 uses only page 0, you can use the common area for any internal data operation.

The Register (R) addressing mode can be used to access this area

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

Figure 2-4. 16-Bit Register Pairs

☞ PROGRAMMING TIP — Addressing the Common Working Register Area

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

Examples:

1. LD 0C2H,40H ; Invalid addressing mode!

   Use working register addressing instead:

   LD R2,40H ; R2 (C2H) ← the value in location 40H

2. ADD 0C3H,#45H ; Invalid addressing mode!

   Use working register addressing instead:

   ADD R3,#45H ; R3 (C3H) ← R3 + 45H

SYSTEM STACK

S3F-series microcontrollers use the system stack for subroutine calls and returns and to store data. The PUSH and POP instructions are used to control system stack operations. The S3F84K4 architecture supports stack operations in the internal register file.

Stack Operations

Return addresses for procedure calls and interrupts and data are stored on the stack. The contents of the PC are saved to stack by a CALL instruction and restored by the RET instruction. When an interrupt occurs, the contents of the PC and the FLAGS register are pushed to the stack. The IRET instruction then pops these values back to their original locations. The stack address is always decremented *before* a push operation and incremented *after* a pop operation. The stack pointer (SPL) always points to the stack frame stored on the top of the stack, as shown in Figure 2-5.

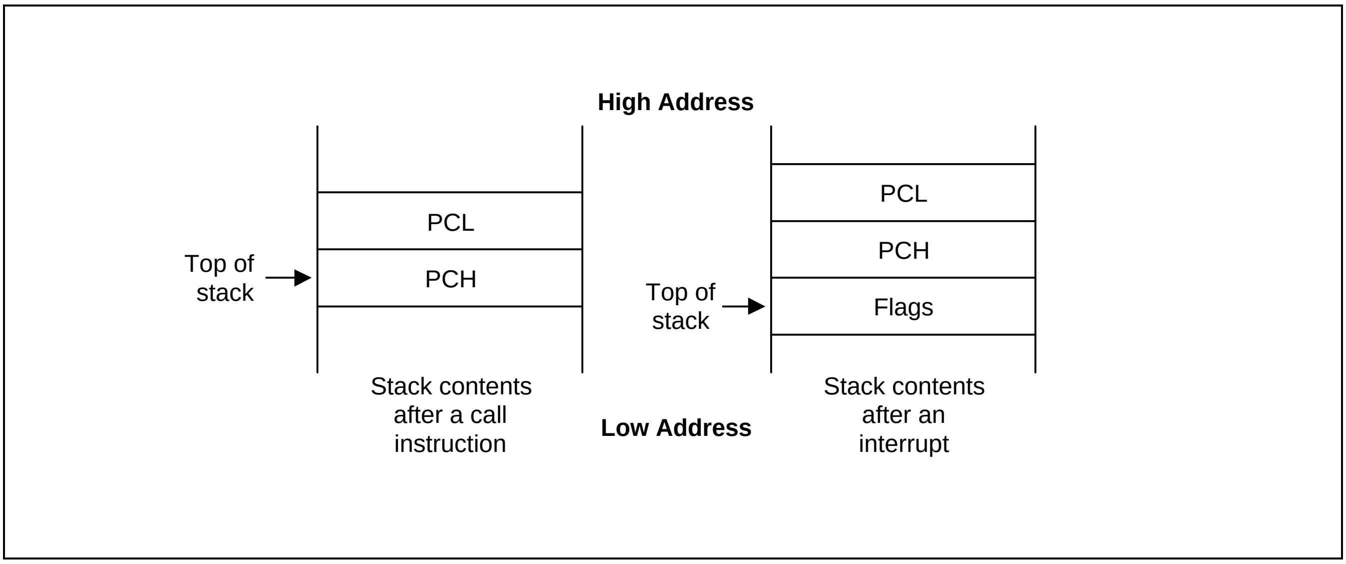


Figure 2-5. Stack Operations

Stack Pointer (SP)

Register location D9H contains the 8-bit stack pointer (SPL) that is used for system stack operations. After a reset, the SPL value is undetermined.

Because only internal memory space is implemented in the S3F84K4, the SPL must be initialized to an 8-bit value in the range 00H–0C0H.

NOTE

In case a Stack Pointer is initialized to 00H, it is decreased to FFH when stack operation starts. This means that a Stack Pointer access invalid stack area. We recommend that a stack pointer is initialized to C0H to set upper address of stack to BFH.

☞ PROGRAMMING TIP — Standard Stack Operations Using PUSH and POP

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

```
LD      SPL,#0C0H       ; SP ← C0H (Normally, the SP is set to C0H by the
                        ; initialization routine)
•
•
•
PUSH    SYM             ; Stack address 0BFH ← SYM
PUSH    R15             ; Stack address 0BEH ← R15
PUSH    20H             ; Stack address 0BDH ← 20H
PUSH    R3              ; Stack address 0BCH ← R3
•
•
•
POP     R3              ; R3 ← Stack address 0BCH
POP     20H             ; 20H ← Stack address 0BDH
POP     R15             ; R15 ← Stack address 0BEH
POP     SYM             ; SYM ← Stack address 0BFH
```

3 ADDRESSING MODES

OVERVIEW

Instructions that are stored in program memory are fetched for execution using the program counter. Instructions indicate the operation to be performed and the data to be operated on. Addressing mode is the method used to determine the location of the data operand. The operands specified in SAM88RC instructions may be condition codes, immediate data, or a location in the register file, program memory, or data memory.

The S3F-series instruction set supports seven explicit addressing modes. Not all of these addressing modes are available for each instruction. The seven addressing modes and their symbols are:

— Register (R)

— Indirect Register (IR)

— Indexed (X)

— Direct Address (DA)

— Indirect Address (IA)

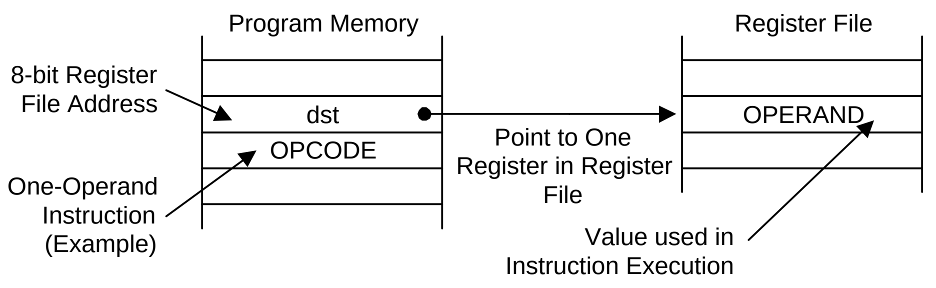
— Relative Address (RA)

— Immediate (IM)

REGISTER ADDRESSING MODE (R)

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

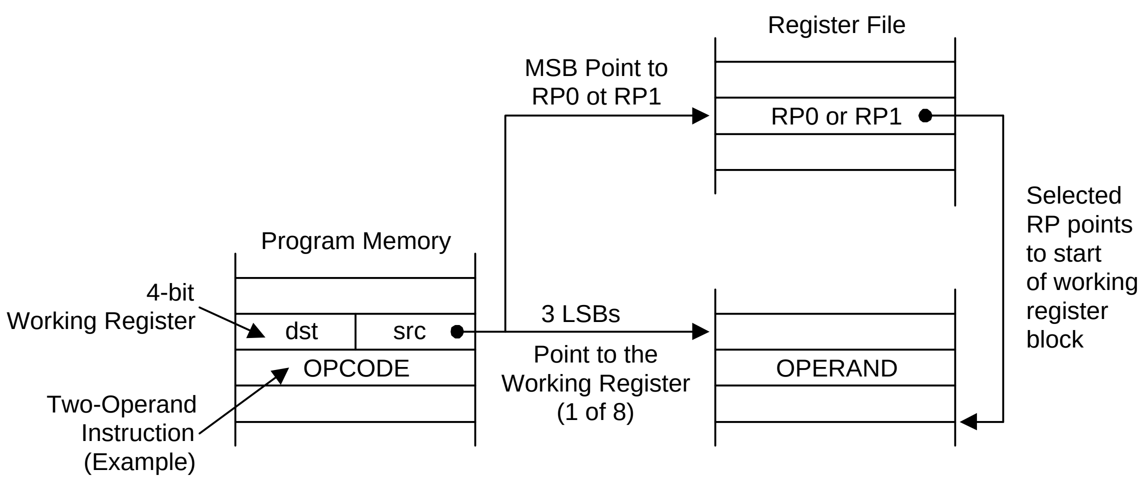
Working register addressing differs from Register addressing in that it uses a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space (see Figure 3-2).



Sample Instruction:

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

Figure 3-1. Register Addressing



Sample Instruction:

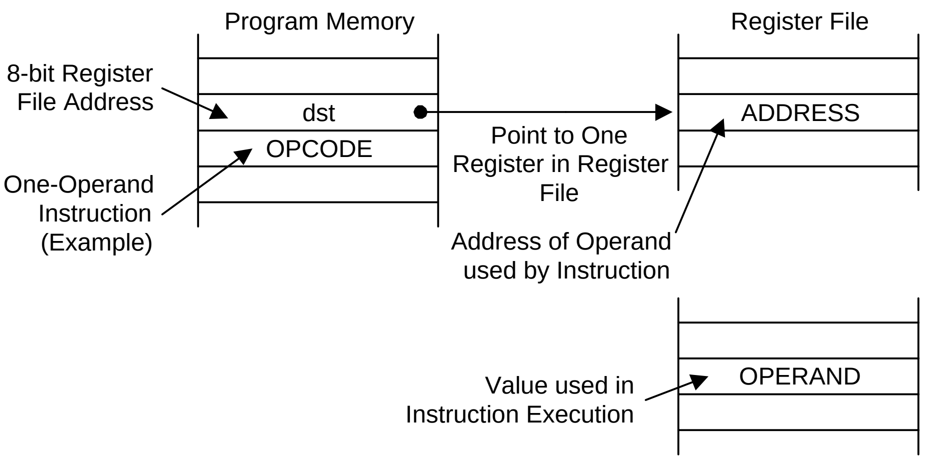
ADD R1, R2 ; Where R1 and R2 are registers in the currently selected working register area.

Figure 3-2. Working Register Addressing

INDIRECT REGISTER ADDRESSING MODE (IR)

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

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



Sample Instruction:

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

Figure 3-3. Indirect Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Continued)

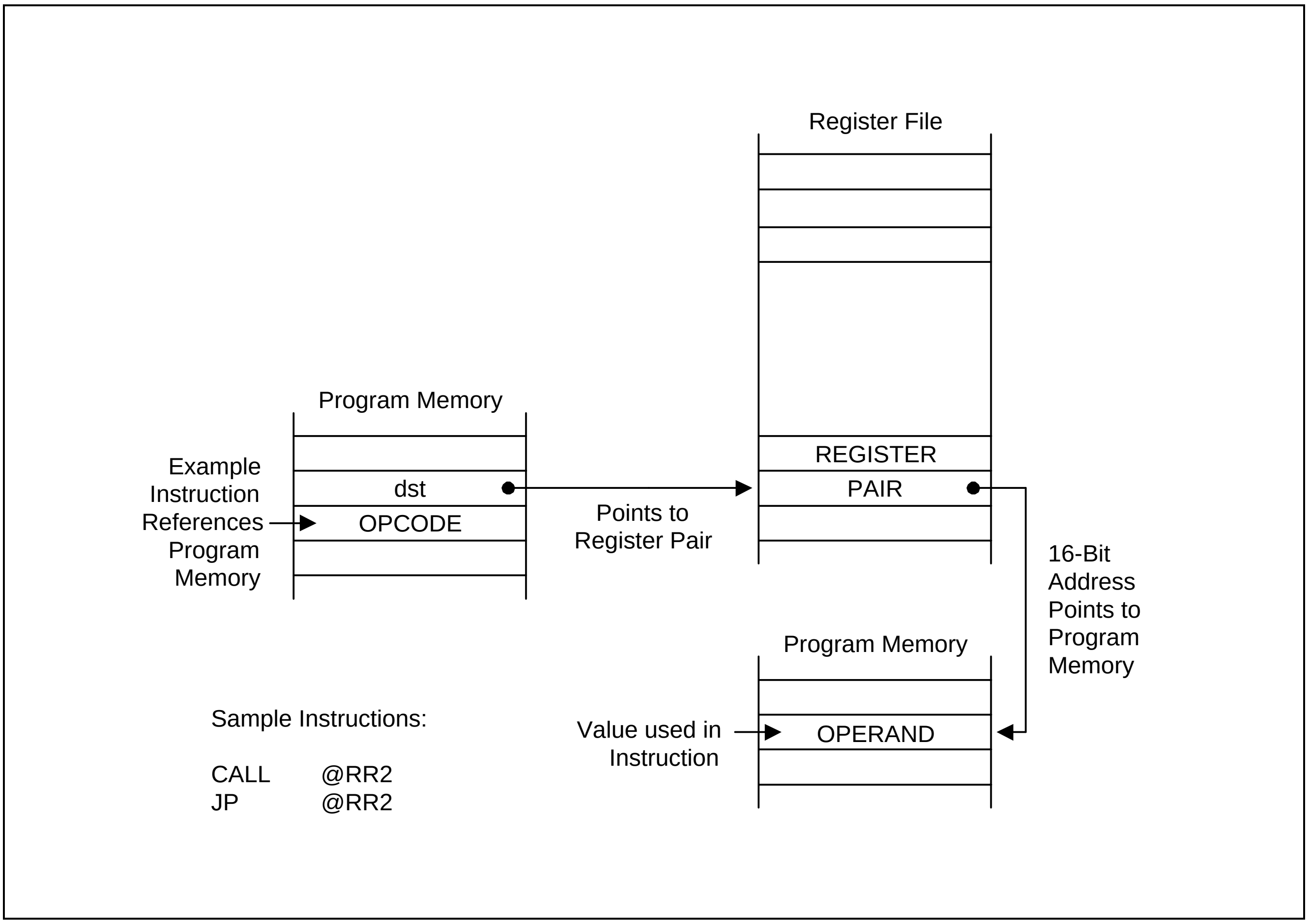


Figure 3-4. Indirect Register Addressing to Program Memory

INDIRECT REGISTER ADDRESSING MODE (Continued)

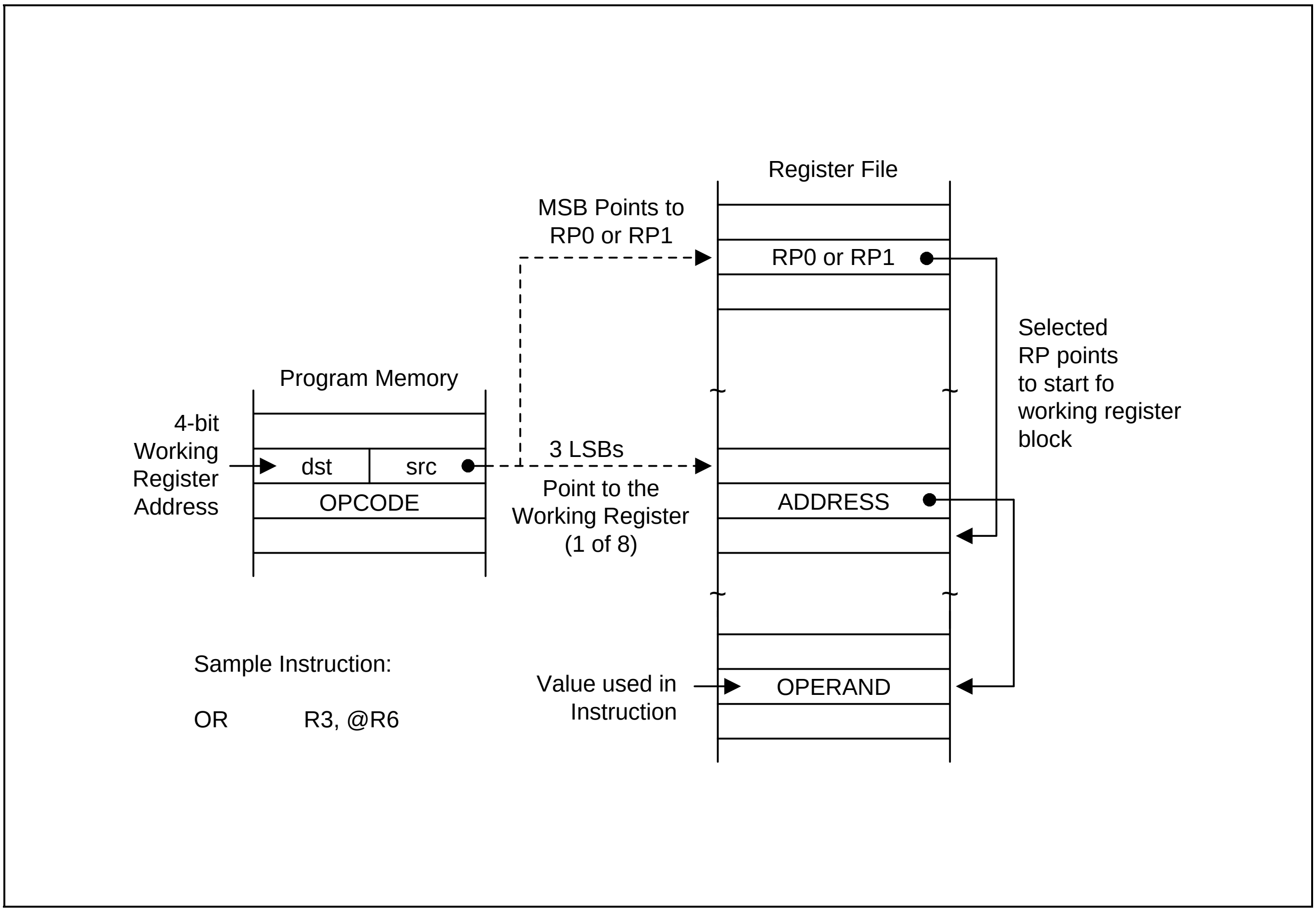
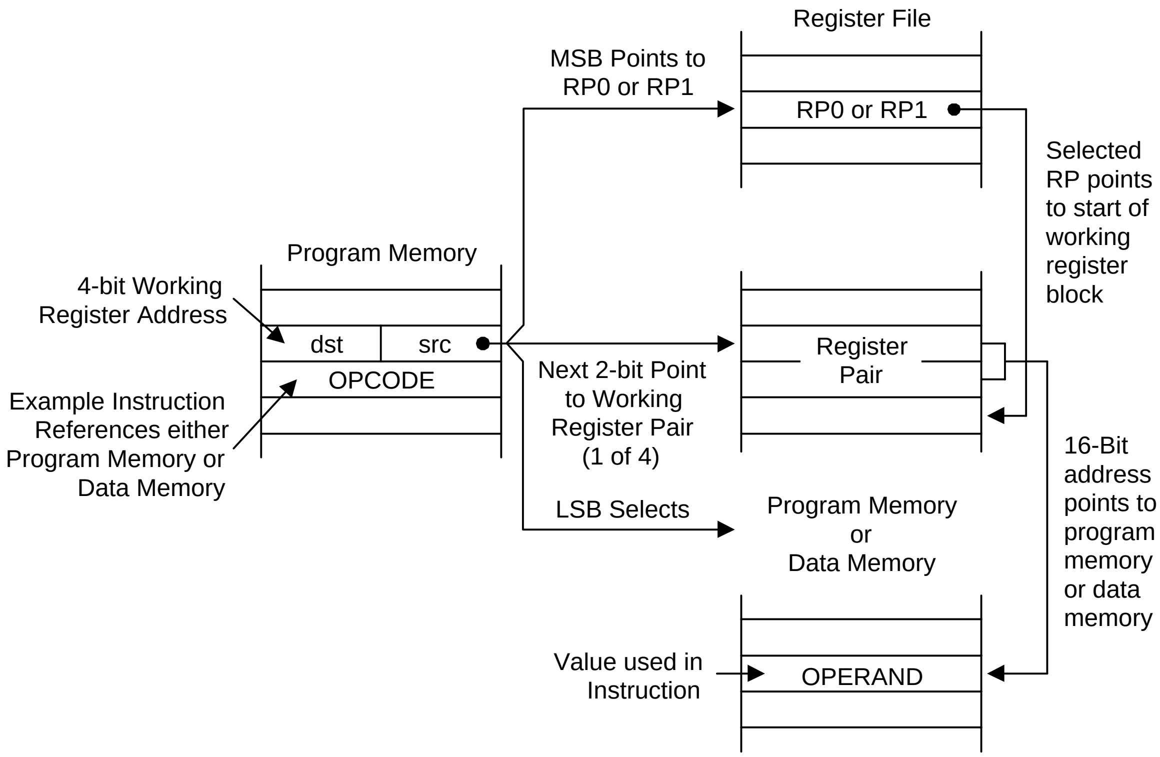


Figure 3-5. Indirect Working Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Concluded)



Sample Instructions:

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

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

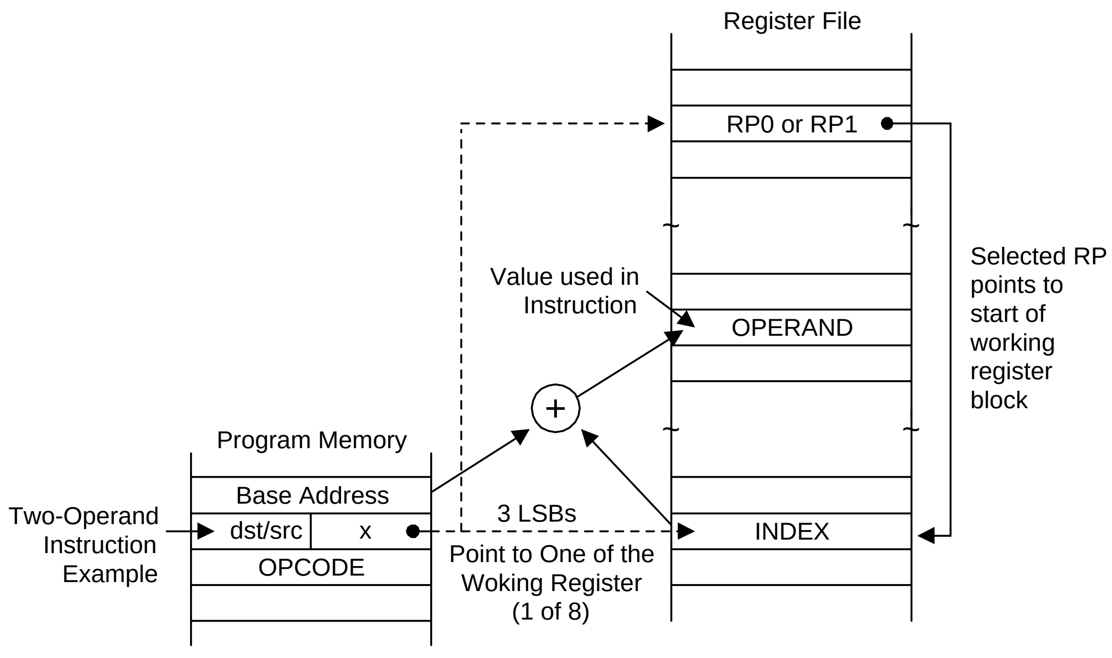
INDEXED ADDRESSING MODE (X)

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

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

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

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

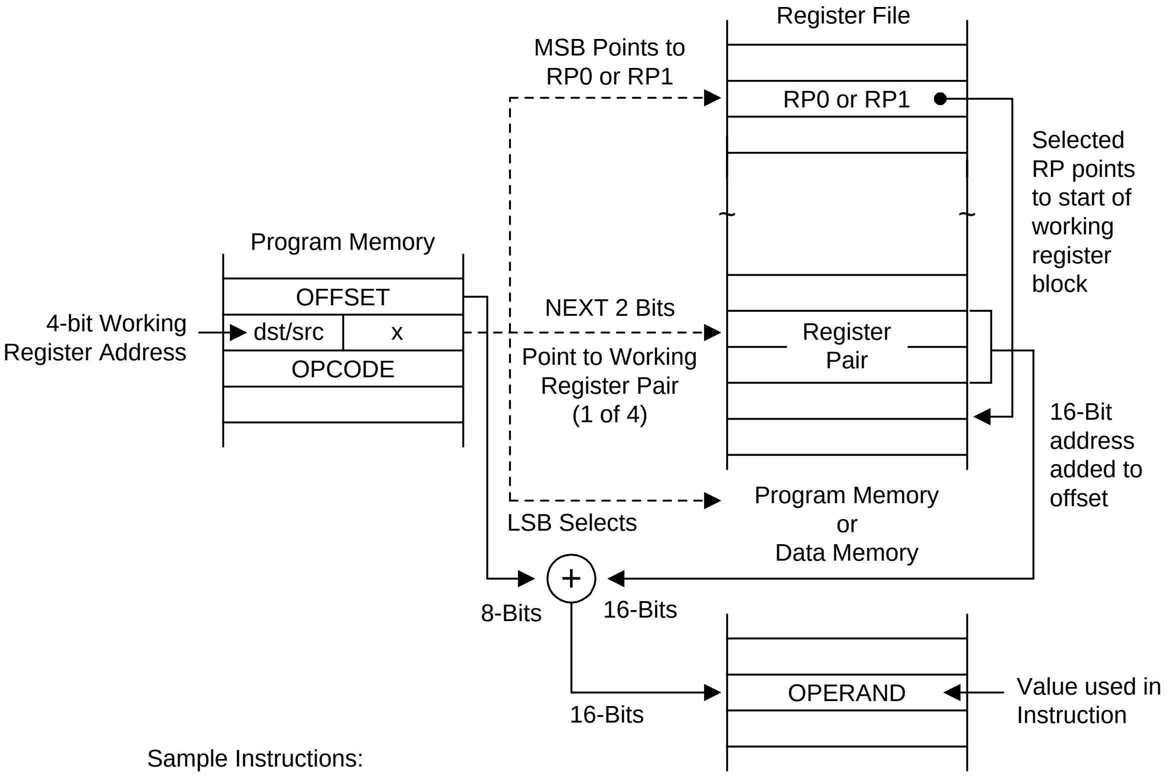


Sample Instruction:

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

Figure 3-7. Indexed Addressing to Register File

INDEXED ADDRESSING MODE (Continued)

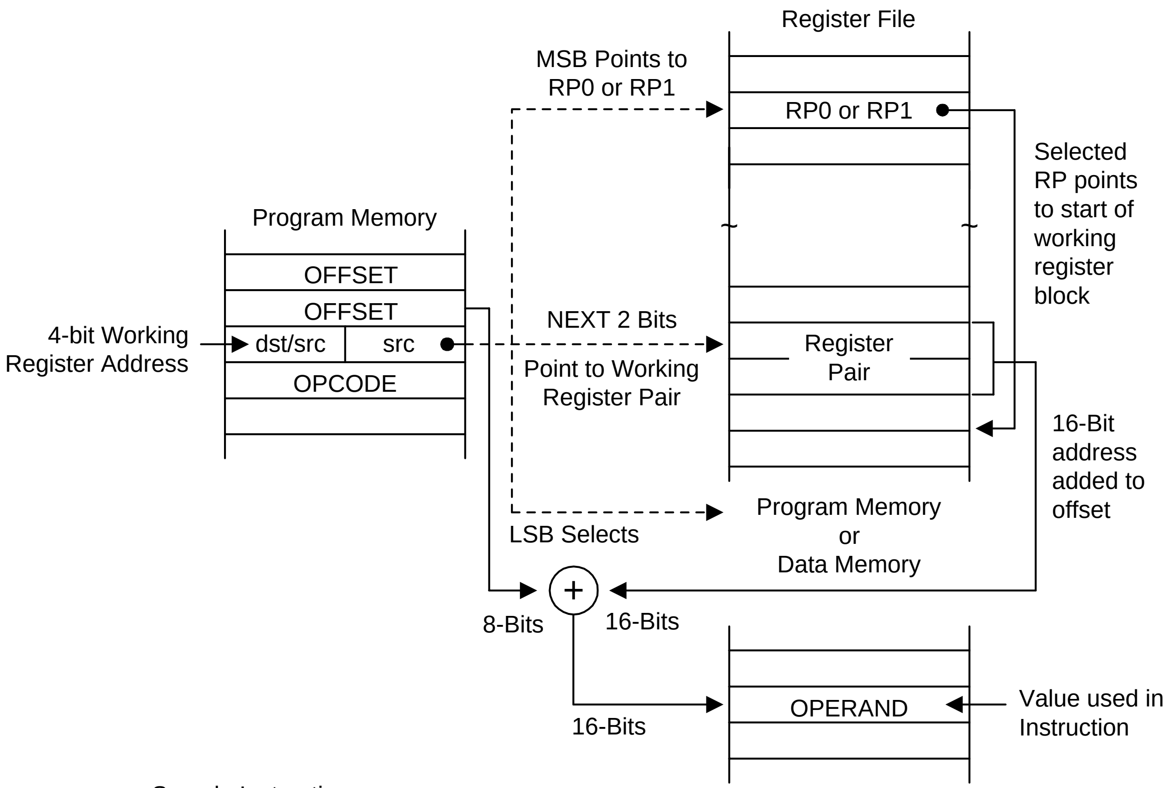


Sample Instructions:

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

Figure 3-8. Indexed Addressing to Program or Data Memory with Short Offset

INDEXED ADDRESSING MODE (Concluded)



Sample Instructions:

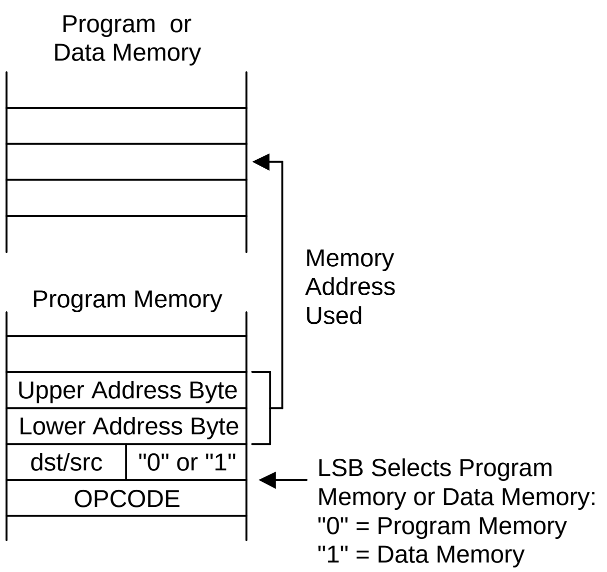
LDC	R4, #1000H[RR2]	; The values in the program address (RR2 + 1000H) are loaded into register R4.
LDE	R4,#1000H[RR2]	; Identical operation to LDC example, except that external program memory is accessed.

Figure 3-9. Indexed Addressing to Program or Data Memory

DIRECT ADDRESS MODE (DA)

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

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

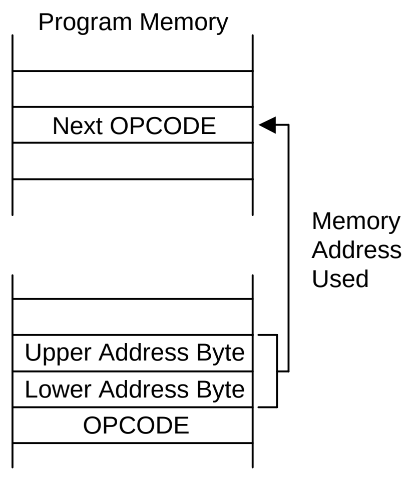


Sample Instructions:

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

Figure 3-10. Direct Addressing for Load Instructions

DIRECT ADDRESS MODE (Continued)



Sample Instructions:

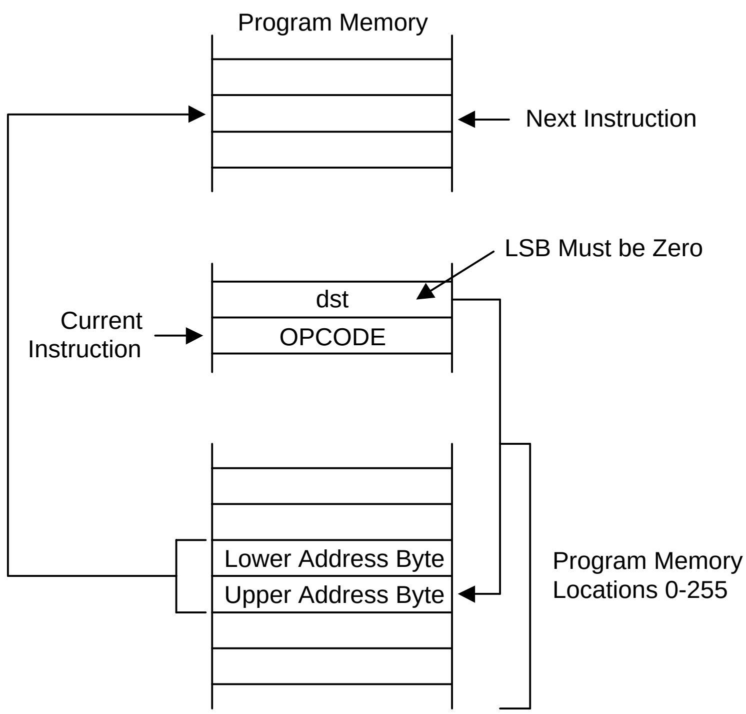
```
JP      C,JOB1       ;  Where JOB1 is a 16-bit immediate address
CALL    DISPLAY      ;  Where DISPLAY is a 16-bit immediate address
```

Figure 3-11. Direct Addressing for Call and Jump Instructions

INDIRECT ADDRESS MODE (IA)

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

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



Sample Instruction:

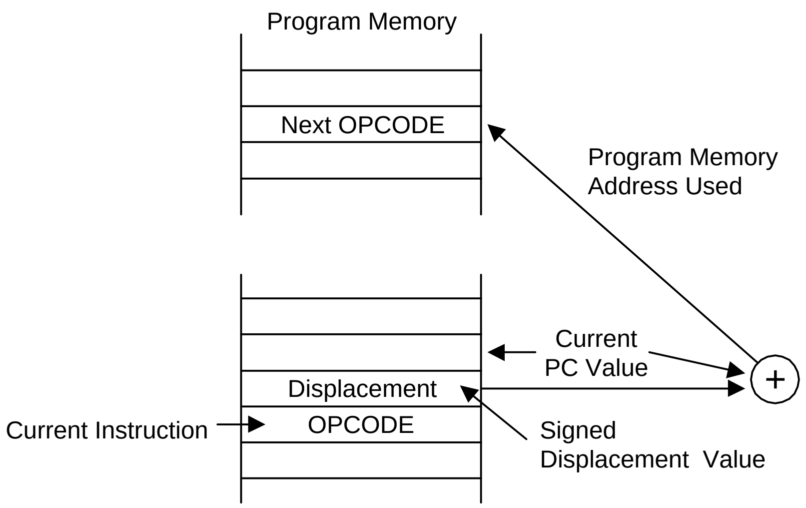
CALL #40H ; The 16-bit value in program memory addresses 40H and 41H is the subroutine start address.

Figure 3-12. Indirect Addressing

RELATIVE ADDRESS MODE (RA)

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

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



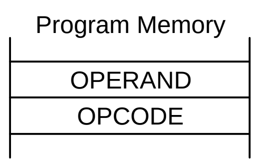
Sample Instructions:

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

Figure 3-13. Relative Addressing

IMMEDIATE MODE (IM)

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



(The Operand value is in the instruction)

Sample Instruction:

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

Figure 3-14. Immediate Addressing

4 CONTROL REGISTERS

OVERVIEW

In this section, detailed descriptions of the S3F84K4 control registers are presented in an easy-to-read format. These descriptions will help familiarize you with the mapped locations in the register file. You can also use them as a quick-reference source when writing application programs.

System and peripheral registers are summarized in Table 4-1. Figure 4-1 illustrates the important features of the standard register description format.

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

Table 4-1. System and Peripheral Control Registers

Register name	Mnemonic	Address & Location		RESET value (Bit)							
		Address	R/W	7	6	5	4	3	2	1	0
Timer A counter register	TACNT	D0H	R	0	0	0	0	0	0	0	0
Timer A data register	TADATA	D1H	R/W	1	1	1	1	1	1	1	1
Timer 0/A control register	TACON	D2H	R/W	0	0	0	0	0	0	0	0
Basic timer control register	BTCON	D3H	R/W	0	0	0	0	0	0	0	0
Clock control register	CLKCON	D4H	R/W	0	–	–	0	0	–	–	–
System flags register	FLAGS	D5H	R/W	x	x	x	x	x	x	0	0
Register Pointer 0	RP0	D6H	R/W	1	1	0	0	0	–	–	–
Register Pointer 1	RP1	D7H	R/W	1	1	0	0	1	–	–	–
Location D8H is not mapped											
Stack Pointer register	SPL	D9H	R/W	x	x	x	x	x	x	x	x
Instruction Pointer (High Byte)	IPH	DAH	R/W	x	x	x	x	x	x	x	x
Instruction Pointer (Low Byte)	IPL	DBH	R/W	x	x	x	x	x	x	x	x
Interrupt Request register	IRQ	DCH	R	0	0	0	0	0	0	0	0
Interrupt Mask Register	IMR	DDH	R/W	0	0	0	0	0	0	0	0
System Mode Register	SYM	DEH	R/W	0	–	–	x	x	x	0	0
Register Page Pointer	PP	DFH	R/W	0	0	0	0	0	0	0	0

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

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

Register Name	Mnemonic	Address	R/W	Bit Values After RESET							
		Hex		7	6	5	4	3	2	1	0
Port 0 data register	P0	E0H	R/W	0	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	R/W	–	–	–	–	–	0	0	–
Port 2 data register	P2	E2H	R/W	–	0	0	0	0	0	0	0
Locations E3H–E5H are not mapped											
Port 0 control register (High byte)	P0CONH	E6H	R/W	0	0	0	0	0	0	0	0
Port 0 control register (Low byte)	P0CONL	E7H	R/W	0	0	0	0	0	0	0	0
Port 0 interrupt pending register	P0PND	E8H	R/W	–	–	–	–	0	0	0	0
Port 1 control register	P1CON	E9H	R/W	0	–	–	–	0	0	–	–
Port 2 control register (High byte)	P2CONH	EAH	R/W	–	0	0	0	0	0	0	0
Port 2 control register (Low byte)	P2CONL	EBH	R/W	0	0	0	0	0	0	0	0
Timer B counter register	TBCNT	ECH	R	0	0	0	0	0	0	0	0
Timer B data register	TBDATA	EDH	R/W	1	1	1	1	1	1	1	1
Timer B control register	TBCON	EEH	R/W	–	–	0	0	0	0	0	0
Locations EFH–F0H are not mapped											
PWM extension data register	PWMEX	F1H	R/W	0	0	0	0	0	0	–	–
PWM data register	PWMDATA	F2H	R/W	–	–	0	0	0	0	0	0
PWM control register	PWMCON	F3H	R/W	0	0	–	0	0	0	0	0
STOP control register	STOPCON	F4H	R/W	0	0	0	0	0	0	0	0
Locations F5H–F6H are not mapped											
A/D control register	ADCON	F7H	R/W	0	0	0	0	0	0	0	0
A/D converter data register (High)	ADDATAH	F8H	R	x	x	x	x	x	x	x	x
A/D converter data register (Low)	ADDATAL	F9H	R	0	0	0	0	0	0	x	x
Locations FAH–FCH are not mapped											
Basic timer counter	BTCNT	FDH	R	0	0	0	0	0	0	0	0
External memory timing register	EMT	FEH	R/W	0	1	1	1	1	1	0	–
Interrupt priority register	IPR	FFH	R/W	x	x	x	x	x	x	x	x

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

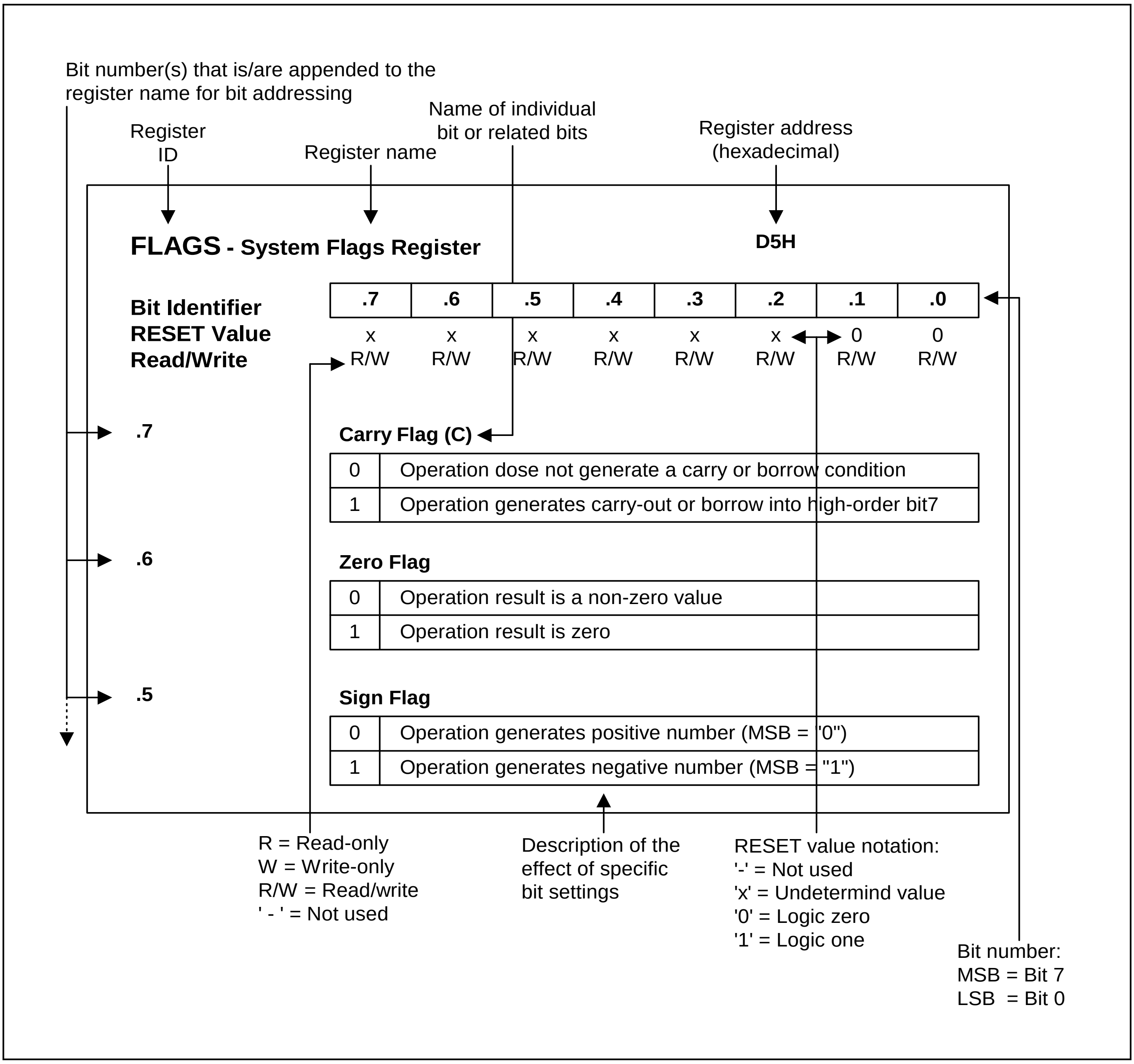


Figure 4-1. Register Description Format

ADCON — A/D Converter Control Register F7H

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 **A/D Converter Input Pin Selection Bits**

0	0	0	0	ADC0 (P0.0)
0	0	0	1	ADC1 (P0.1)
0	0	1	0	ADC2 (P0.2)
0	0	1	1	ADC3 (P0.3)
0	1	0	0	ADC4 (P0.4)
0	1	0	1	ADC5 (P0.5)
0	1	1	0	ADC6 (P0.6)
0	1	1	1	ADC7 (P0.7)
1	0	0	0	ADC8 (P2.6)
1	0	0	1	Invalid Selection
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

.3 **End-of-Conversion Status Bit**

0	A/D conversion is in progress
1	A/D conversion complete

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

0	0	$f_{OSC}/16$ ($f_{OSC} \leq 8$ MHz)
0	1	$f_{OSC}/8$ ($f_{OSC} \leq 8$ MHz)
1	0	$f_{OSC}/4$ ($f_{OSC} \leq 8$ MHz)
1	1	$f_{OSC}/1$ ($f_{OSC} \leq 4$ MHz)

.0 **Conversion Start Bit**

0	No meaning
1	A/D conversion start

NOTE: Maximum ADC clock input = 4 MHz.

BTCON — Basic Timer Control Register D3H

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

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

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

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

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

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

0	No effect
1	Clear the basic timer counter value

.0 **Basic Timer Divider Clear Bit**

0	No effect
1	Clear both dividers

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

CLKCON — Clock Control Register D4H

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

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

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

.6–.5 Not used for S3F84K4

.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 S3F84K4

EMT — External Memory Timing Register FEH

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

.7 **External wait Input Function Enable Bit**

0	Disable **wait** input function for external device
1	Disable **wait** input function for external device

.6 **Slow Memory Timing Enable Bit**

0	Disable slow memory timing
1	Enable slow memory timing

.5 and .4 **Program Memory Automatic Wait Control Bits**

0	0	No wait
0	1	Wait one cycle
1	0	Wait two cycles
1	1	Wait three cycles

.3 and .2 **Data Memory Automatic Wait Control Bits**

0	0	No wait
0	1	Wait one cycle
1	0	Wait two cycles
1	1	Wait three cycles

.1 **Stack Area Selection Bit**

0	Select internal register file area
1	Select external register file area

.0 Not used for the S3F84K4

NOTE: The EMT register is not used, because an external peripheral interface is not implemented. The program initialization routine should clear the EMT register to '00H' following a reset. Modification of EMT values during normal operation may cause a system malfunction

FLAGS — System Flags Register D5H

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

.7 **Carry Flag (C)**

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

.6 **Zero Flag (Z)**

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

.5 **Sign Flag (S)**

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

.4 **Overflow Flag (V)**

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

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

0	Add operation completed
1	Subtraction operation completed

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

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

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

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

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

0	Bank 0 is selected
1	Bank 1 is selected

IMR — Interrupt Mask Register DDH

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

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

0	Disable (mask)
1	Enable (unmask)

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

0	Disable (mask)
1	Enable (unmask)

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

0	Disable (mask)
1	Enable (unmask)

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

0	Disable (mask)
1	Enable (unmask)

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

0	Disable (mask)
1	Enable (unmask)

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

0	Disable (mask)
1	Enable (unmask)

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

0	Disable (mask)
1	Enable (unmask)

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

0	Disable (mask)
1	Enable (unmask)

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

IPH — Instruction Pointer (High Byte) DAH

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

.7–.0 **Instruction Pointer Address (High Byte)**

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

IPL — Instruction Pointer (Low Byte) DBH

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

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

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

IPR — Interrupt Priority Register FFH

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

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

0	0	0	Group priority undefined
0	0	1	B > C > A
0	1	0	A > B > C
0	1	1	B > A > C
1	0	0	C > A > B
1	0	1	C > B > A
1	1	0	A > C > B
1	1	1	Group priority undefined

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

0	IRQ6 > IRQ7
1	IRQ7 > IRQ6

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

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

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

0	IRQ3 > IRQ4
1	IRQ4 > IRQ3

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

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

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

0	IRQ0 > IRQ1
1	IRQ1 > IRQ0

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

IRQ — Interrupt Request Register DCH

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

P0CONH — Port 0 Control Register (High Byte) E6H

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 **Port 0, P0.7/INT7 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	A/D converter input (ADC7); Schmitt trigger input off

.5–.4 **Port 0, P0.6/ADC6/PWM Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Alternative function (PWM output)
1	0	Push-pull output
1	1	A/D converter input (ADC6); Schmitt trigger input off

.3–.2 **Port 0, P0.5/ADC5 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	A/D converter input (ADC5); Schmitt trigger input off

.1–.0 **Port 0, P0.4/ADC4 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	A/D converter input (ADC4); Schmitt trigger input off

P0CONL — Port 0 Control Register (Low Byte) E7H

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 **Port 0, P0.3/INT3 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; pull-up enable
1	0	Push-pull output
1	1	A/D converter input (ADC3); Schmitt trigger input off

.5–.4 **Port 0, P0.2/ADC2 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; pull-up enable
1	0	Push-pull output
1	1	A/D converter input (ADC2); Schmitt trigger input off

.3–.2 **Port 0, P0.1/ADC1/INT1 Configuration Bits**

0	0	Schmitt trigger input/falling edge interrupt input
0	1	Schmitt trigger input; pull-up enable/falling edge interrupt input
1	0	Push-pull output
1	1	A/D converter input (ADC1); Schmitt trigger input off

.1–.0 **Port 0, P0.0/ADC0/INT0 Configuration Bits**

0	0	Schmitt trigger input/falling edge interrupt input
0	1	Schmitt trigger input; pull-up enable/falling edge interrupt input
1	0	Push-pull output
1	1	A/D converter input (ADC0); Schmitt trigger input off

P0PND — Port 0 Interrupt Pending Register E8H

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 the S3F84K4

.3 **Port 0.1/ADC1/INT1 Interrupt Enable Bit**

0	INT1 falling edge interrupt disable
1	INT1 falling edge interrupt enable

.2 **Port 0.1/ADC1/INT1 Interrupt Pending Bit**

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

.1 **Port 0.0/ADC0/INT0 Interrupt Enable Bit**

0	INT0 falling edge interrupt disable
1	INT0 falling edge interrupt enable

.0 **Port 0.0/ADC0/INT0 Interrupt Pending Bit**

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

P1CON — Port 1 Control Register E9H

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	R/W	R/W

.7 **Part 1.1 N-channel open-drain Enable Bit**

0	Configure P1.1 as a push-pull output
1	Configure P1.1 as a n-channel open-drain output

.6–.4 Not used for S3F84K4

.3–.2 **Port 1, P1.1 Interrupt Pending Bits**

0	0	Schmitt trigger input;
0	1	Schmitt trigger input; pull-up enable
1	0	Output
1	1	Schmitt trigger input; pull-down enable

.1–.0 Not used for S3F84K4

NOTE: When you use external oscillator, P1.1 must be set to output port to prevent current consumption.

P2CONH — Port 2 Control Register (High Byte) EAH

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

.7 Not used for the S3F84K4

.6–.4 **Port 2, P2.6/ADC8/CLO Configuration Bits**

0	0	0	Schmitt trigger input; pull-up enable
0	0	1	Schmitt trigger input
0	1	x	ADC input
1	0	0	Push-pull output
1	0	1	Open-drain output; pull-up enable
1	1	0	Open-drain output
1	1	1	Alternative function; CLO output

.3–.2 **Port 2, 2.5 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	Open-drain output

.1–.0 **Port 2, 2.4 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	Open-drain output

NOTE: When noise problem is important issue, you had better not use CLO output.

P2CONL — Port 2 Control Register (Low Byte) EBH

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 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	Open-drain output

.5–.4 **Port 2, P2.2 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	Open-drain output

.3–.2 **Port 2, P2.1 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	Open-drain output

.1–.0 **Port 2, P2.0 Configuration Bits**

0	0	Schmitt trigger input; pull-up enable
0	1	Schmitt trigger input
1	0	Push-pull output
1	1	T0 match output

PWMCON — PWM Control Register F3H

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

.7–.6 **PWM Input Clock Selection Bits**

0	0	$f_{OSC}/256$
0	1	$f_{OSC}/64$
1	0	$f_{OSC}/8$
1	1	$f_{OSC}/1$

.5–.4

Not used for S3F84K4

.3 **PWM Counter Clear Bit**

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

.2 **PWM Counter Enable Bit**

0	Stop counter
1	Start (Resume countering)

.1 **PWM Overflow Interrupt Enable Bit (12-Bit Overflow)**

0	Disable interrupt
1	Enable interrupt

.0 **PWM Overflow Interrupt Pending Bit**

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

NOTE: PWMCON.0 is not auto-cleared. You must pay attention when clear pending bit. (Refer to page 11-7).

PP — Register Page Pointer DFH

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

.7–.0	Not used for the S3F84K4.

NOTE: In S3F84K4, 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.

RP0 — Register Pointer 0 D6H

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

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

Register pointer 0 can independently point to one of the 208-byte working register areas in the register file. Using the register pointers RP0 and RP1, you can select two 8-byte register slices at one time as active working register space. After a reset, RP0 points to address C0H, selecting the 8-byte working register slice C0H–C7H.

.2–.0 Not used for the S3F84K4

RP1 — Register Pointer 1 D7H

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

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

Register pointer 1 can independently point to one of the 208-byte working register areas in the register file. Using the register pointers RP0 and RP1, you can select two 8-byte register slices at one time as active working register space. After a reset, RP1 points to address C8H, selecting the 8-byte working register slice C8H–CFH.

.2–.0 Not used for the S3F84K4

SPL — Stack Pointer D9H

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

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

The SP value is undefined following a reset.

STOPCON — STOP Mode Control Register F4H

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

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

10100101	Enable STOP instruction
Other value	Disable STOP instruction

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

SYM — System Mode Register DEH

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

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

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

.6–.5

Not used for the S3F84K4

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

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

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

0	Disable fast interrupt processing
1	Enable fast interrupt processing

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

0	Disable all interrupt processing
1	Enable all interrupt processing

NOTES:
1. Because an external interface is not implemented, SYM.7 must always be ‘0’.
2. You can select only one interrupt level at a time for fast interrupt processing.
3. Setting SYM.1 to "1" enables fast interrupt processing for the interrupt level currently selected by SYM.2-SYM.4.
4. Following a reset, you must enable global interrupt processing by executing an EI instruction (not by writing a "1" to SYM.0).

TACON — Timer 0/A Control Register D2H

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

.7 **Timer 0 Operating Mode Selection Bit**

0	Two 8-bit timers mode (Timer A/B)
1	One 16-bit timer mode (Timer 0)

.6 Must be always “0”

.5–.4 **Timer 0/A Clock Selection Bits**

0	0	fxx/256
0	1	fxx/64
1	0	fxx/8
1	1	fxx

.3 **Timer 0/A Counter Clear Bit (NOTE)**

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

.2 **Timer 0/A Counter Run Enable Bit**

0	Disable Counter Running
1	Enable Counter Running

.1 **Timer 0/A Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.0 **Timer 0/A Interrupt Pending Bit**

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

NOTES:
1. When you write "1" to TACON.3, the Timer 0/A counter value is cleared to "00H". Immediately following the write operation, the TACON.3 value is automatically cleared to "0".
2. TACON.6 must be always "0" during normal operation.

TBCON — Timer B Control Register EEH

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 and .6

Not used for the S3F84K4

.5 and .4 **Timer B Clock Selection Bits**

0	0	fxx/256
0	1	fxx/64
1	0	fxx/8
1	1	fxx

.3 **Timer B Counter Clear Bit (NOTE)**

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

.2 **Timer B Counter Run Enable Bit**

0	Disable Counter Running
1	Enable Counter Running

.1 **Timer B Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.0 **Timer B Interrupt Pending Bit**

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

NOTE: When you write a "1" to TBCON.3, the Timer B counter value is cleared to "00H". Immediately following the write operation, the TBCON.3 value is automatically cleared to "0".

5 INTERRUPT STRUCTURE

OVERVIEW

The S3F8-series interrupt structure has three basic components: levels, vectors, and sources. The SAM8RC CPU recognizes up to eight interrupt levels and supports up to 128 interrupt vectors. When a specific interrupt level has more than one vector address, the vector priorities are established in hardware. A vector address can be assigned to one or more sources.

Levels

Interrupt levels are the main unit for interrupt priority assignment and recognition. All peripherals and I/O blocks can issue interrupt requests. In other words, peripheral and I/O operations are interrupt-driven. There are eight possible interrupt levels: IRQ0–IRQ7, also called level 0–level 7. Each interrupt level directly corresponds to an interrupt request number (IRQn). The total number of interrupt levels used in the interrupt structure varies from device to device. The S3F84K4 interrupt structure recognizes three interrupt levels.

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

Vectors

Each interrupt level can have one or more interrupt vectors, or it may have no vector address assigned at all. The maximum number of vectors that can be supported for a given level is 128 (The actual number of vectors used for S3F8-series devices is always much smaller). If an interrupt level has more than one vector address, the vector priorities are set in hardware. S3F84K4 uses 5 vectors.

Sources

A source is any peripheral that generates an interrupt. A source can be an external pin or a counter overflow. Each vector can have several interrupt sources. In the S3F84K4 interrupt structure, there are 5 possible interrupt sources.

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

INTERRUPT TYPES

The three components of the S3F8 interrupt structure described before — levels, vectors, and sources — are combined to determine the interrupt structure of an individual device and to make full use of its available interrupt logic. There are three possible combinations of interrupt structure components, called interrupt types 1, 2, and 3. The types differ in the number of vectors and interrupt sources assigned to each level (see Figure 5-1):

Type 1: One level (IRQn) + one vector (V_1) + one source (S_1)

Type 2: One level (IRQn) + one vector (V_1) + multiple sources ($S_1 – S_n$)

Type 3: One level (IRQn) + multiple vectors ($V_1 – V_n$) + multiple sources ($S_1 – S_n$, $S_{n+1} – S_{n+m}$)

In the S3F84K4 microcontroller, two interrupt types are implemented.

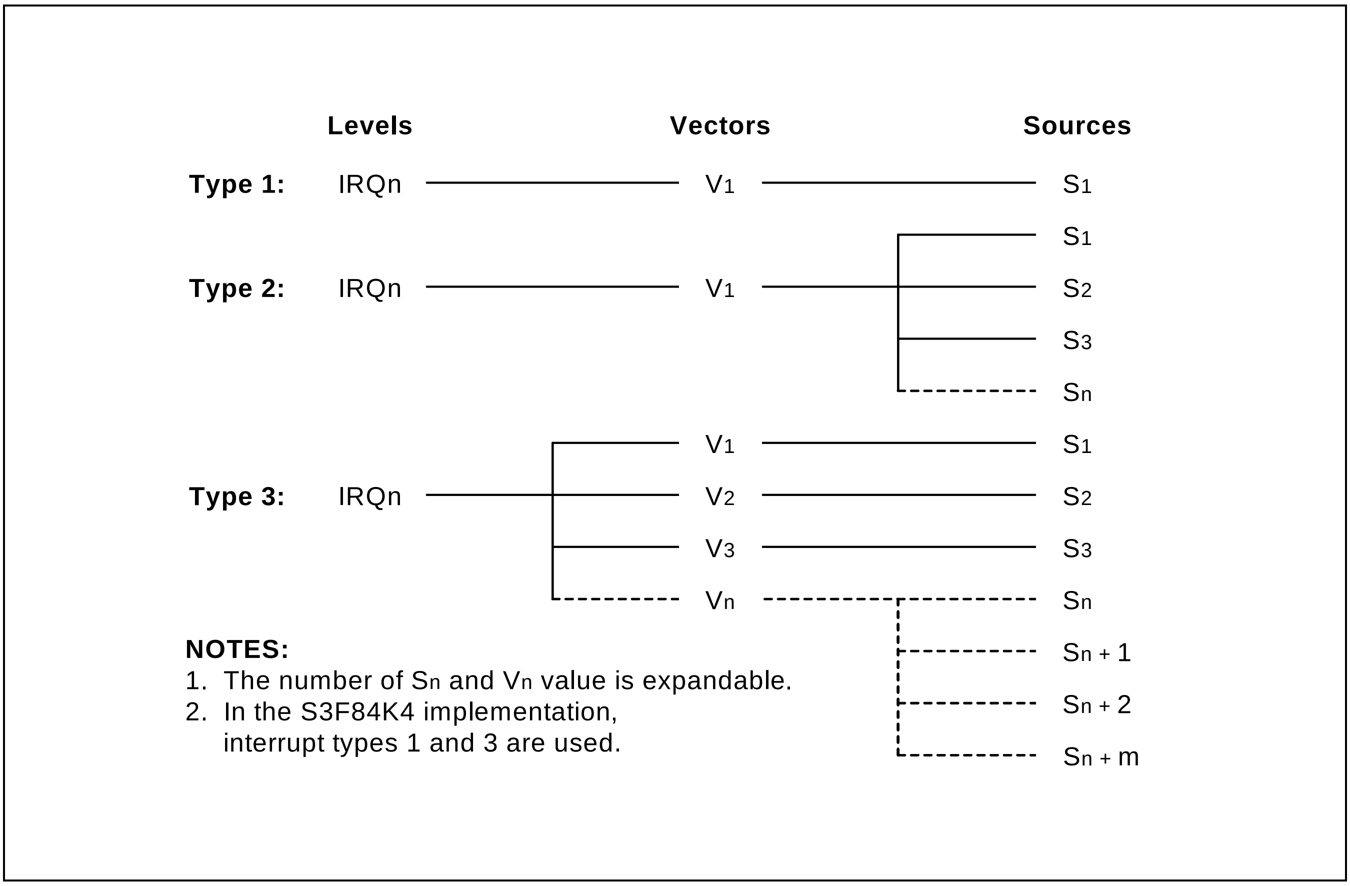


Figure 5-1. S3F8-Series Interrupt Types

S3F84K4 INTERRUPT STRUCTURE

The S3F84K4 microcontroller supports 5 interrupt sources. Every interrupt source has a corresponding interrupt address. Three interrupt levels are recognized by the CPU in this device-specific interrupt structure, as shown in Figure 5-2.

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

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

Levels	Vectors	Sources	Reset/Clear
RESET	100H	Basic timer overflow	H/W
IRQ0	FCH	External interrupt 0	S/W
	FAH	External interrupt 1	S/W
IRQ1	F6H	Timer 0/A match interrupt	S/W
	F4H	Timer B match interrupt	S/W
IRQ2	F2H	PWM overflow interrupt	S/W

NOTE: External interrupts are triggered by a falling edge.

Figure 5-2. S3F84K4 Interrupt Structure

Interrupt Vector Addresses

All interrupt vector addresses for the S3F84K4 interrupt structure is stored in the vector address area of the first 256 bytes of the program memory (ROM).

You can allocate unused locations in the vector address area as normal program memory. If you do so, please be careful not to overwrite any of the stored vector addresses.

The program reset address in the ROM is 0100H.

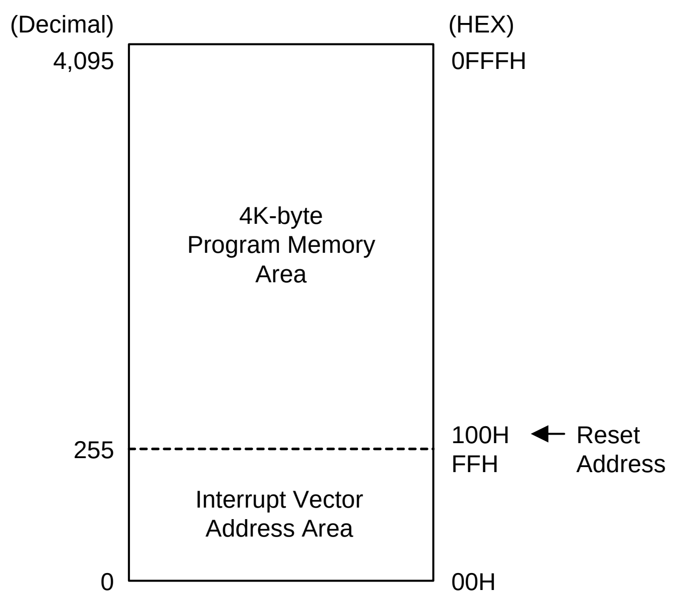


Figure 5-3. ROM Vector Address Area

Enable/Disable Interrupt Instructions (EI, DI)

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

NOTE

The system initialization routine executed after a reset must always contain an EI instruction to globally enable the interrupt structure.

During the normal operation, you can execute the DI (Disable Interrupt) instruction at any time to globally disable interrupt processing. The EI and DI instructions change the value of bit 0 in the SYM register.

SYSTEM-LEVEL INTERRUPT CONTROL REGISTERS

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

— The interrupt mask register, IMR, enables (un-masks) or disables (masks) interrupt levels.

— The interrupt priority register, IPR, controls the relative priorities of interrupt levels.

— The interrupt request register, IRQ, contains interrupt pending flags for each interrupt level (as opposed to each interrupt source).

— The system mode register, SYM, enables or disables global interrupt processing (SYM settings also enable fast interrupts and control the activity of external interface, if implemented).

Table 5-1. Interrupt Control Register Overview

Control Register	ID	R/W	Function Description
Interrupt mask register	IMR	R/W	Bit settings in the IMR register enable or disable interrupt processing for each of the eight interrupt levels: IRQ0–IRQ7.
Interrupt priority register	IPR	R/W	Controls the relative processing priorities of the interrupt levels. The eight levels of S3F84K4 are organized into three groups: A, B, and C. Group A is IRQ0 and IRQ1, group B is IRQ2, IRQ3 and IRQ4, and group C is IRQ5, IRQ6, and IRQ7.
Interrupt request register	IRQ	R	This register contains a request pending bit for each interrupt level.
System mode register	SYM	R/W	This register enables/disables fast interrupt processing, and dynamic global interrupt processing.

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

INTERRUPT PROCESSING CONTROL POINTS

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

— Global interrupt enable and disable (by EI and DI instructions or by direct manipulation of SYM.0)

— Interrupt level enable/disable settings (IMR register)

— Interrupt level priority settings (IPR register)

— Interrupt source enable/disable settings in the corresponding peripheral control registers

NOTE

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

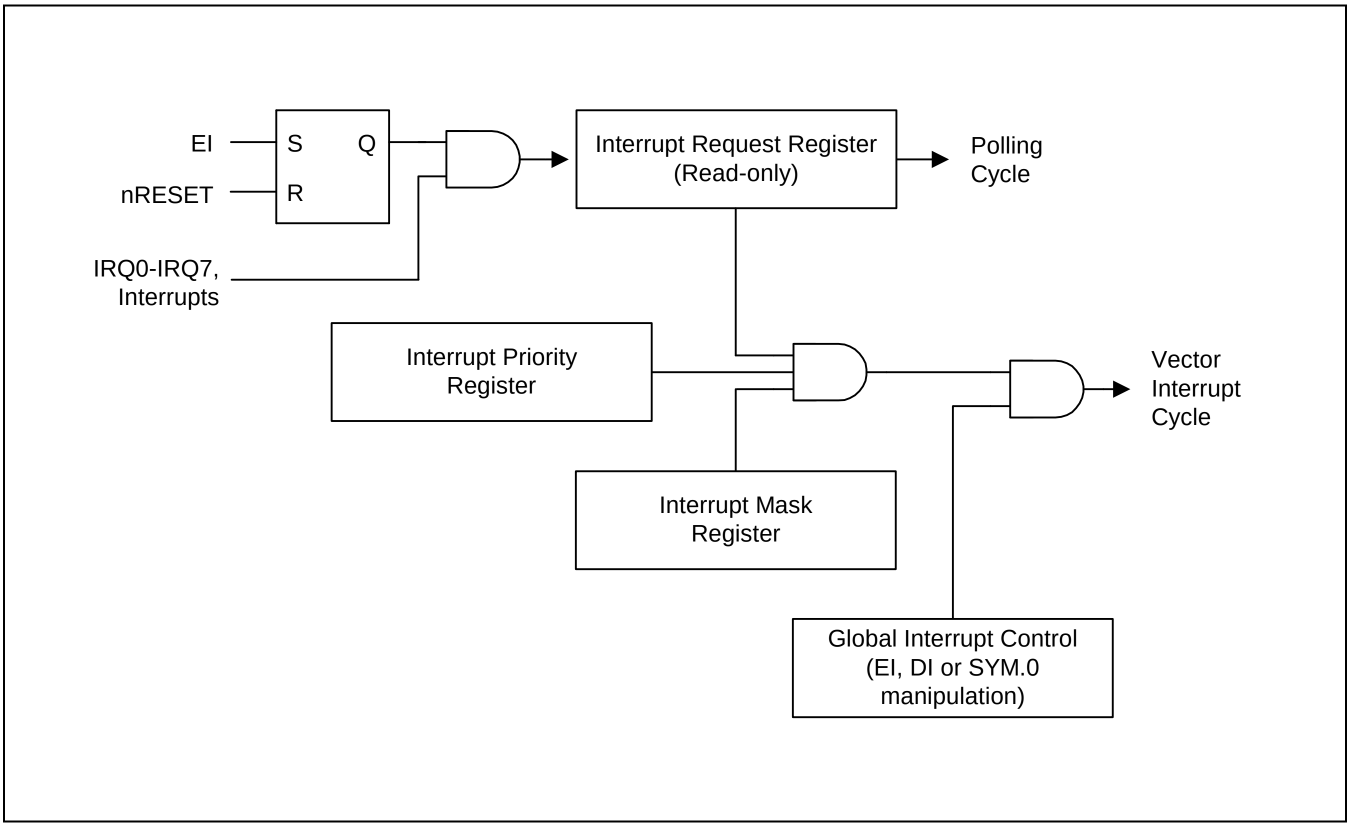


Figure 5-4. Interrupt Function Diagram

PERIPHERAL INTERRUPT CONTROL REGISTERS

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

Table 5-2. Interrupt Source Control and Data Registers

Interrupt Source	Interrupt Level	Register(s)	Location(s)
P0.0 external interrupt P0.1 external interrupt	IRQ0	P0CONL P0PND	E7H E8H
Timer 0/A match interrupt Timer B match interrupt	IRQ1	TACON P2CONL TBCON	D0H EBH D1H
PWM overflow interrupt	IRQ2	PWMCON P0CONH	F3H E6H

NOTE: If a interrupt is un-mask(Enable interrupt level) in the IMR register, the pending bit and enable bit of the interrupt should be written after a DI instruction is executed.

SYSTEM MODE REGISTER (SYM)

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

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

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

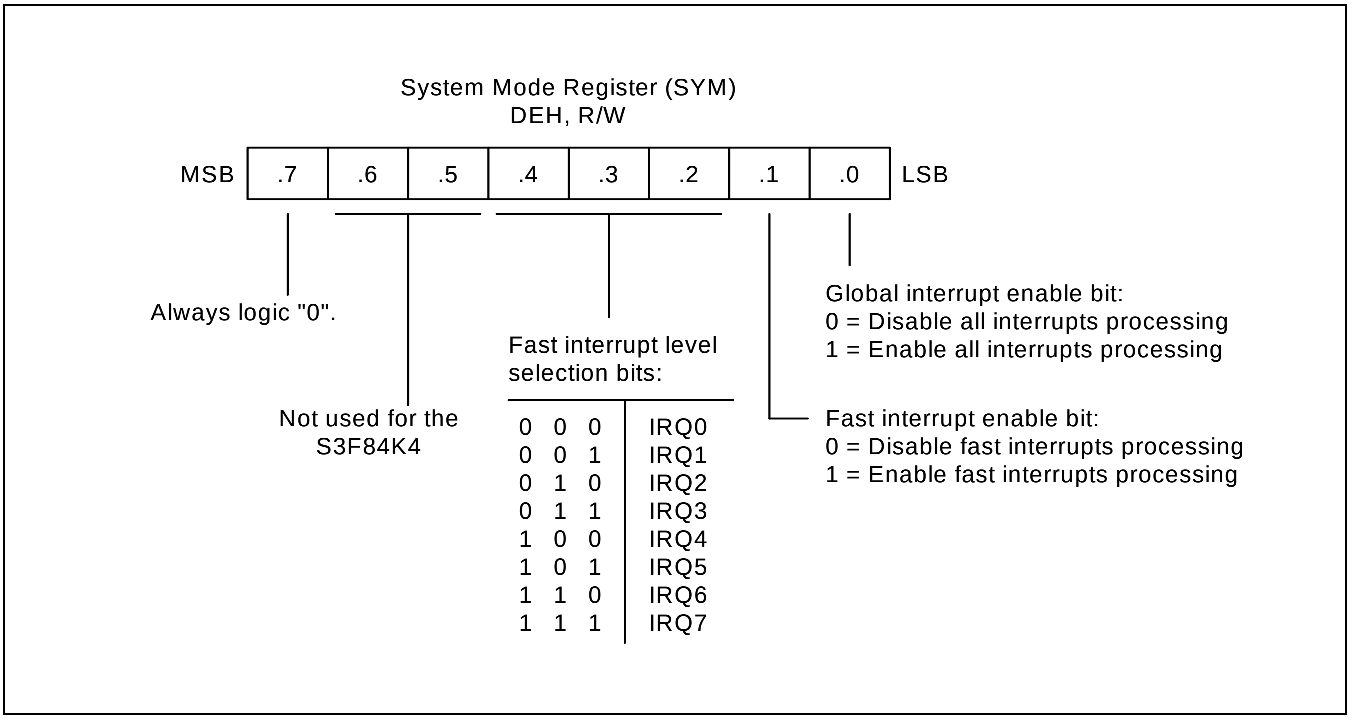


Figure 5-5. System Mode Register (SYM)

INTERRUPT MASK REGISTER (IMR)

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

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

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

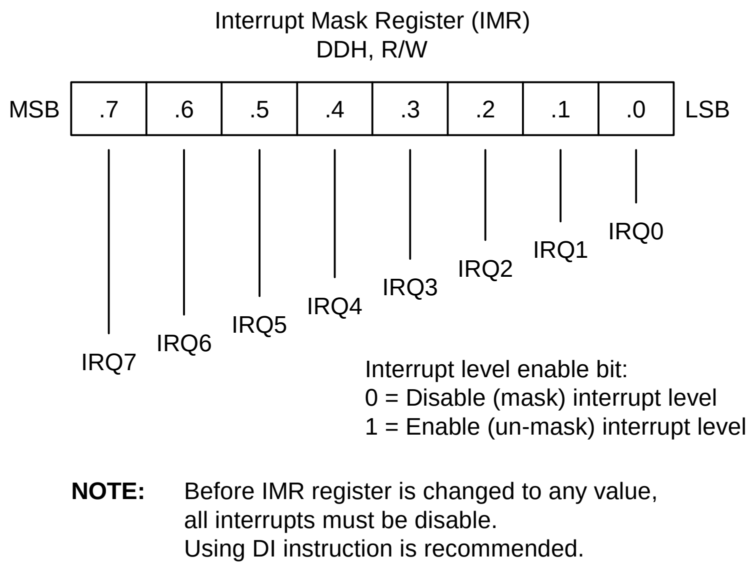


Figure 5-6. Interrupt Mask Register (IMR)

INTERRUPT PRIORITY REGISTER (IPR)

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

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

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

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

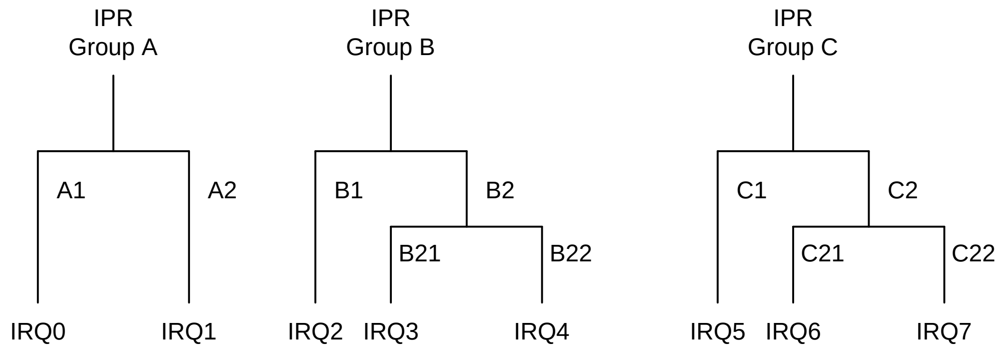


Figure 5-7. Interrupt Request Priority Groups

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

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

- IPR.5 controls the relative priorities of group C interrupts.
- Interrupt group C includes a subgroup that has an additional priority relationship among the interrupt levels 5, 6, and 7. IPR.6 defines the subgroup C relationship. IPR.5 controls the interrupt group C.
- IPR.0 controls the relative priority setting of IRQ0 and IRQ1 interrupts.

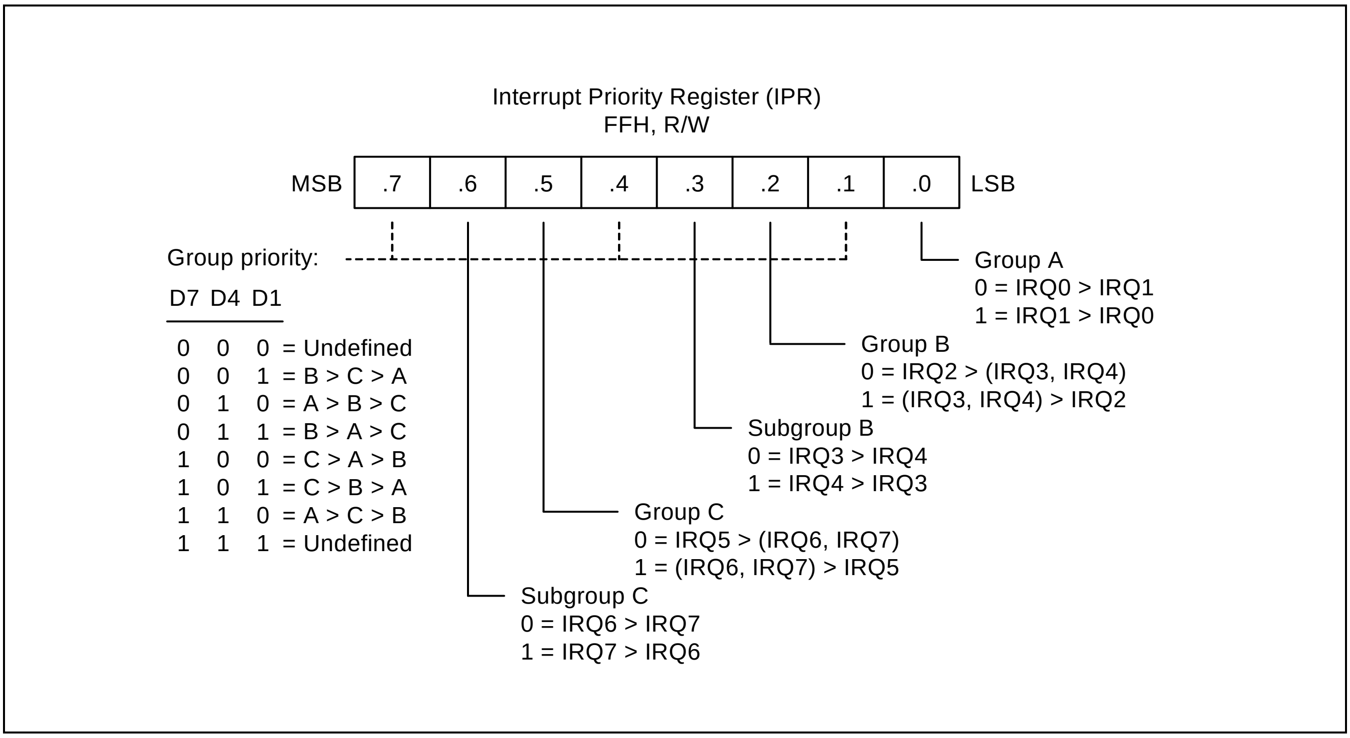


Figure 5-8. Interrupt Priority Register (IPR)

INTERRUPT REQUEST REGISTER (IRQ)

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

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

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

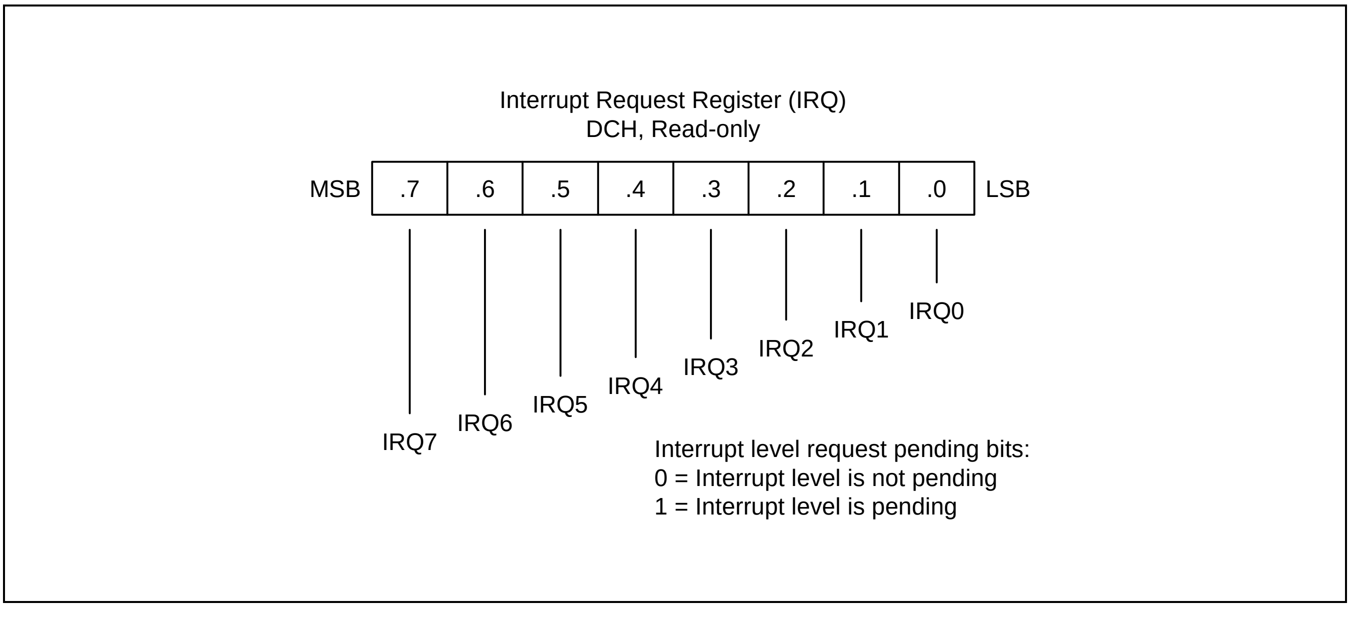


Figure 5-9. Interrupt Request Register (IRQ)

INTERRUPT PENDING FUNCTION TYPES

Overview

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

Pending Bits Cleared Automatically by Hardware

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

In the S3F84K4 interrupt structure, no interrupt belongs to this category of interrupts in which pending condition is cleared automatically by hardware.

Pending Bits Cleared by the Service Routine

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

INTERRUPT SOURCE POLLING SEQUENCE

The interrupt request polling and servicing sequence is as follows:

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

INTERRUPT SERVICE ROUTINES

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

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

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

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

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

GENERATING INTERRUPT VECTOR ADDRESSES

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

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

NOTE

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

NESTING OF VECTORED INTERRUPTS

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

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

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

INSTRUCTION POINTER (IP)

The instruction pointer (IP) is adopted by all the S3C8/S3F8-series microcontrollers to control the optional high-speed interrupt processing feature called *fast interrupts*. The IP consists of register pair DAH and DBH. The names of IP registers are IPH (high byte, IP15–IP8) and IPL (low byte, IP7–IP0).

FAST INTERRUPT PROCESSING

The feature called *fast interrupt processing* allows an interrupt within a given level to be completed in approximately 6 clock cycles rather than the usual 16 clock cycles. To select a specific interrupt level for fast interrupt processing, you write the appropriate 3-bit value to SYM.4–SYM.2. Then, to enable fast interrupt processing for the selected level, you set SYM.1 to “1”.

FAST INTERRUPT PROCESSING (Continued)

Two other system registers support fast interrupt processing:

— The instruction pointer (IP) contains the starting address of the service routine (and is later used to swap the program counter values), and

— When a fast interrupt occurs, the contents of the FLAGS register are stored in an unmapped, dedicated register called FLAGS' ("FLAGS prime").

NOTE

For the S3F84K4 microcontroller, the service routine for any one of the eight interrupt levels: IRQ0–IRQ7, can be selected for fast interrupt processing.

PROCEDURE FOR INITIATING FAST INTERRUPTS

To initiate fast interrupt processing, follow these steps:

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

FAST INTERRUPT SERVICE ROUTINE

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

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

RELATIONSHIP TO INTERRUPT PENDING BIT TYPES

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

PROGRAMMING GUIDELINES

Remember that the only way to enable/disable a fast interrupt is to set/clear the fast interrupt enable bit in the SYM register, SYM.1. Executing an EI or DI instruction globally enables or disables all interrupt processing, including fast interrupts. If you use fast interrupts, remember to load the IP with a new start address when the fast interrupt service routine ends.

6 INSTRUCTION SET

OVERVIEW

The SAM8RC instruction set is specifically designed to support the large register files that are typical of most SAM8 microcontrollers. There are 78 instructions. The powerful data manipulation capabilities and features of the instruction set include:

— A full complement of 8-bit arithmetic and logic operations, including multiply and divide

— No special I/O instructions (I/O control/data registers are mapped directly into the register file)

— Decimal adjustment included in binary-coded decimal (BCD) operations

— 16-bit (word) data can be incremented and decremented

— Flexible instructions for bit addressing, rotate, and shift operations

DATA TYPES

The SAM8 CPU performs operations on bits, bytes, BCD digits, and two-byte words. Bits in the register file can be set, cleared, complemented, and tested. Bits within a byte are numbered from 7 to 0, where bit 0 is the least significant (right-most) bit.

REGISTER ADDRESSING

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

ADDRESSING MODES

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

Table 6-1. Instruction Group Summary

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

Table 6-1. Instruction Group Summary (Continued)

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

Table 6-1. Instruction Group Summary (Continued)

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

Table 6-1. Instruction Group Summary (Concluded)

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

FLAGS REGISTER (FLAGS)

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

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

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

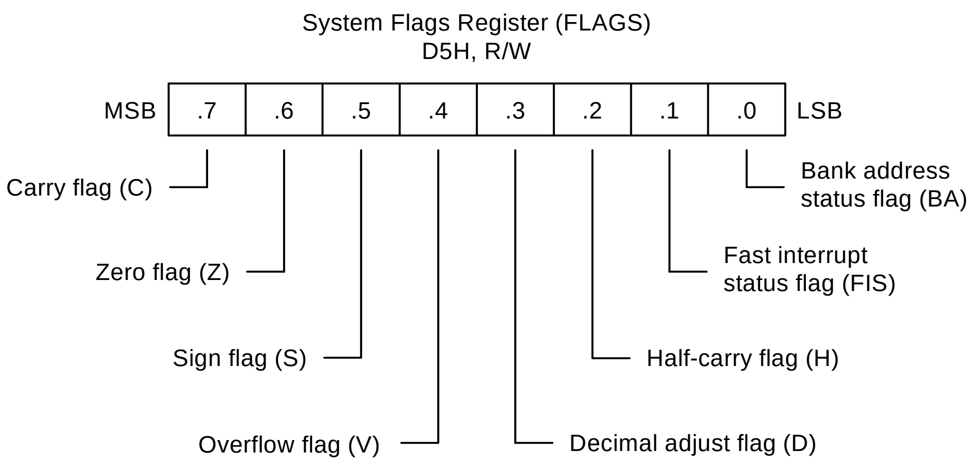


Figure 6-1. System Flags Register (FLAGS)

FLAG DESCRIPTIONS

C Carry Flag (FLAGS.7)

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

Z Zero Flag (FLAGS.6)

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

S Sign Flag (FLAGS.5)

Following arithmetic, logic, rotate, or shift operations, the sign bit identifies the state of the MSB of the result. A logic zero indicates a positive number and a logic one indicates a negative number.

V Overflow Flag (FLAGS.4)

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

D Decimal Adjust Flag (FLAGS.3)

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

H Half-Carry Flag (FLAGS.2)

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

FIS Fast Interrupt Status Flag (FLAGS.1)

The FIS bit is set during a fast interrupt cycle and reset during the IRET following interrupt servicing. When set, it inhibits all interrupts and causes the fast interrupt return to be executed when the IRET instruction is executed.

BA Bank Address Flag (FLAGS.0)

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

INSTRUCTION SET NOTATION

Table 6-2. Flag Notation Conventions

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

Table 6-3. Instruction Set Symbols

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

Table 6-4. Instruction Notation Conventions

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

Table 6-5. Opcode Quick Reference

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

Table 6-5. Opcode Quick Reference (Continued)

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

CONDITION CODES

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

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

Table 6-6. Condition Codes

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

NOTES:

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

INSTRUCTION DESCRIPTIONS

This section contains detailed information and programming examples for each instruction in the SAM8 instruction set. Information is arranged in a consistent format for improved readability and for fast referencing. The following information is included in each instruction description:

— Instruction name (mnemonic)

— Full instruction name

— Source/destination format of the instruction operand

— Shorthand notation of the instruction's operation

— Textual description of the instruction's effect

— Specific flag settings affected by the instruction

— Detailed description of the instruction's format, execution time, and addressing mode(s)

— Programming example(s) explaining how to use the instruction

ADC — Add with carry

ADC dst,src

Operation: dst ← dst + src + c

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

Flags:

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

Format:

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

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

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

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

ADD — Add

ADD dst,src

Operation: dst ← dst + src

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

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

Format:

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

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

ADD	R1,R2	→	R1 = 15H, R2 = 03H
ADD	R1,@R2	→	R1 = 1CH, R2 = 03H
ADD	01H,02H	→	Register 01H = 24H, register 02H = 03H
ADD	01H,@02H	→	Register 01H = 2BH, register 02H = 03H
ADD	01H,#25H	→	Register 01H = 46H

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

AND — Logical AND

AND dst,src

Operation: dst ← dst AND src

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

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

Format:

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

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

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

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

BAND — Bit AND

BAND dst,src.b

BAND dst.b,src

Operation: dst(0) ← dst(0) AND src(b)

or

dst(b) ← dst(b) AND src(0)

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

Flags:

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

Format:

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

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

Examples: Given: R1 = 07H and register 01H = 05H:

BAND R1,01H.1 → R1 = 06H, register 01H = 05H
BAND 01H.1,R1 → Register 01H = 05H, R1 = 07H

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

BCP — Bit Compare

BCP dst,src.b

Operation: dst(0) – src(b)

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

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

Format:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | b | 0 | src	3	6	17	r0	Rb

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

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

BCP R1,01H.1 → R1 = 07H, register 01H = 01H

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

BITC — Bit Complement

BITC dst.b

Operation: dst(b) ← NOT dst(b)

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

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

Format:

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

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

Example: Given: R1 = 07H

BITC R1.1 → R1 = 05H

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

BITR — Bit Reset

BITR dst.b

Operation: dst(b) ← 0

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

Flags: No flags are affected.

Format:

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

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

Example: Given: R1 = 07H:

BITR R1.1 → R1 = 05H

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

BITS — Bit Set

BITS dst.b

Operation: dst(b) ← 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 the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BITS R1.3 → R1 = 0FH

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

BOR — Bit OR

BOR dst,src.b

BOR dst.b,src

Operation: dst(0) ← dst(0) OR src(b)

or

dst(b) ← dst(b) OR src(0)

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

Flags:

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

Format:

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

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

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

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

BOR 01H.2, R1 → Register 01H = 07H, R1 = 07H

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

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

BTJRF — Bit Test, Jump Relative on False

BTJRF dst,src.b

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

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

Flags: No flags are affected.

Format:

(Note 1)

opc	src | b | 0	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
3	10	37	RA	rb

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

Example: Given: R1 = 07H:

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

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

BTJRT — Bit Test, Jump Relative on True

BTJRT dst,src.b

Operation: If src(b) is a "1", then PC ← PC + dst

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

Flags: No flags are affected.

Format:

(Note 1)

opc	src | b | 1	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
3	10	37	RA	rb

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

Example: Given: R1 = 07H:

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

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

BXOR — Bit XOR

BXOR dst,src.b

BXOR dst.b,src

Operation: dst(0) ← dst(0) XOR src(b)

or

dst(b) ← dst(b) XOR src(0)

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

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

Format:

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

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

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

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

BXOR 01H.2,R1 → Register 01H = 07H, R1 = 07H

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

CALL — Call Procedure

CALL dst

Operation:

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

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

Flags: No flags are affected.

Format:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	3	14	F6	DA
opc | dst	2	12	F4	IRR
opc | dst	2	14	D4	IA

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

CALL 3521H	→	SP = 0000H (Memory locations 0000H = 1AH, 0001H = 4AH, where 4AH is the address that follows the instruction.)
CALL @RR0	→	SP = 0000H (0000H = 1AH, 0001H = 49H)
CALL #40H	→	SP = 0000H (0000H = 1AH, 0001H = 49H)

In the first example, if the program counter value is 1A47H and the stack pointer contains the value 0002H, the statement "CALL 3521H" pushes the current PC value onto the top of the stack. The stack pointer now points to memory location 0000H. The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed.
If the contents of the program counter and stack pointer are the same as in the first example, the statement "CALL @RR0" produces the same result except that the 49H is stored in stack location 0001H (because the two-byte instruction format was used). The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed.
Assuming that the contents of the program counter and stack pointer are the same as in the first example, if program address 0040H contains 35H and program address 0041H contains 21H, the statement "CALL #40H" produces the same result as in the second example.

CCF — Complement Carry Flag

CCF

Operation: C ← NOT C

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

Flags: **C:** Complemented.

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	EF

Example: Given: The carry flag = "0":

CCF

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

CLR — Clear

CLR dst

Operation: dst ← "0"

The destination location is cleared to "0".

Flags: No flags are affected.

Format:

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

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

CLR 00H → Register 00H = 00H

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

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

COM — Complement

COM dst

Operation: dst ← NOT dst

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

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

Format:

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

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

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

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

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

CP — Compare

CP dst,src

Operation: dst – src

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

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

Format:

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

Examples:

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

```
CP      R1,R2  →      Set the C and S flags
```

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

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

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

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

CPIJE — Compare, Increment, and Jump on Equal

CPIJE dst,src,RA

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

Ir ← Ir + 1

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

Flags: No flags are affected.

Format:

opc	src	dst	RA

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

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

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

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

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

CPIJNE — Compare, Increment, and Jump on Non-Equal

CPIJNE dst,src,RA

Operation: If dst – src "0", PC ← PC + RA

Ir ← Ir + 1

The source operand is compared to (subtracted from) the destination operand. If the result is not "0", the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise the instruction following the CPIJNE instruction is executed. In either case the source pointer is incremented by one before the next instruction.

Flags: No flags are affected.

Format:

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

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

Example: Given: R1 = 02H, R2 = 03H, and register 03H = 04H:

CPIJNER1,@R2,SKIP → R2 = 04H, PC jumps to SKIP location

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

DA — Decimal Adjust

DA dst

Operation: dst ← DA dst

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

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

Flags:

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

Format:

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

DA — Decimal Adjust

DA (Continued)

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

```
ADD   R1,R0  ;    C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = C, R1 ← 3CH
DA    R1     ;    R1 ← 3CH + 06
```

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

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

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

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

Assuming the same values given above, the statements

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

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

DEC — Decrement

DEC dst

Operation: dst ← dst – 1

The contents of the destination operand are decremented by one.

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

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode 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.

DECW — Decrement Word

DECW dst

Operation: dst ← dst – 1

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

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

Format:

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

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

DECW RR0 → R0 = 12H, R1 = 33H

DECW @R2 → Register 30H = 0FH, register 31H = 20H

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

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

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

DI — Disable Interrupts

DI

Operation: SYM (0) ← 0

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

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	8F

Example: Given: SYM = 01H:

DI

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

Before changing IMR, interrupt pending and interrupt source control register, be sure DI state.

DIV — Divide (Unsigned)

DIV dst,src

Operation: dst ÷ src

dst (UPPER) ← REMAINDER

dst (LOWER) ← QUOTIENT

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

Flags:
C: Set if the V flag is set and quotient is between 2^8 and 2^9 –1; cleared otherwise.
Z: Set if divisor or quotient = "0"; cleared otherwise.
S: Set if MSB of quotient = "1"; cleared otherwise.
V: Set if quotient is ≥ 2^8 or if divisor = "0"; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

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

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

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

DIV RR0,R2 → R0 = 03H, R1 = 40H
DIV RR0,@R2 → R0 = 03H, R1 = 20H
DIV RR0,#20H → R0 = 03H, R1 = 80H

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

DJNZ — Decrement and Jump if Non-Zero

DJNZ r,dst

Operation: r ← r – 1

If r ≠ 0, PC ← PC + dst

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

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

Flags: No flags are affected.

Format:

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

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

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

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

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

EI — Enable Interrupts

EI

Operation: SYM (0) ← 1

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

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	9F

Example: Given: SYM = 00H:

EI

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

ENTER — Enter

ENTER

Operation:

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

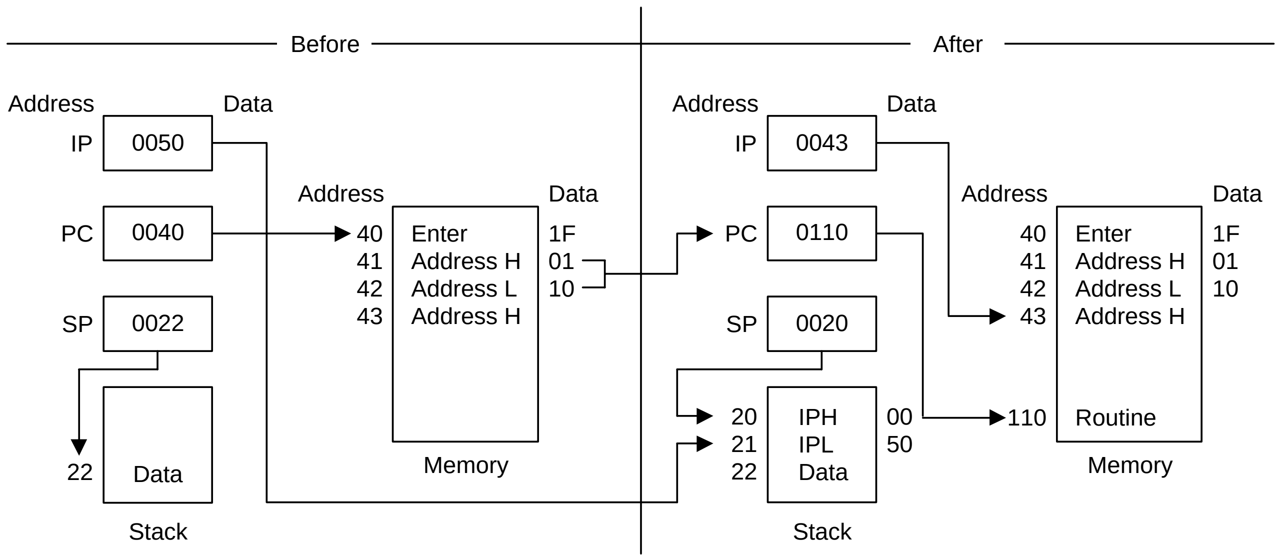
This instruction is useful when implementing threaded-code languages. The contents of the instruction pointer are pushed to the stack. The program counter (PC) value is then written to the instruction pointer. The program memory word that is pointed to by the instruction pointer is loaded into the PC,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

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	14	1F

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



EXIT — Exit

EXIT

Operation:

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

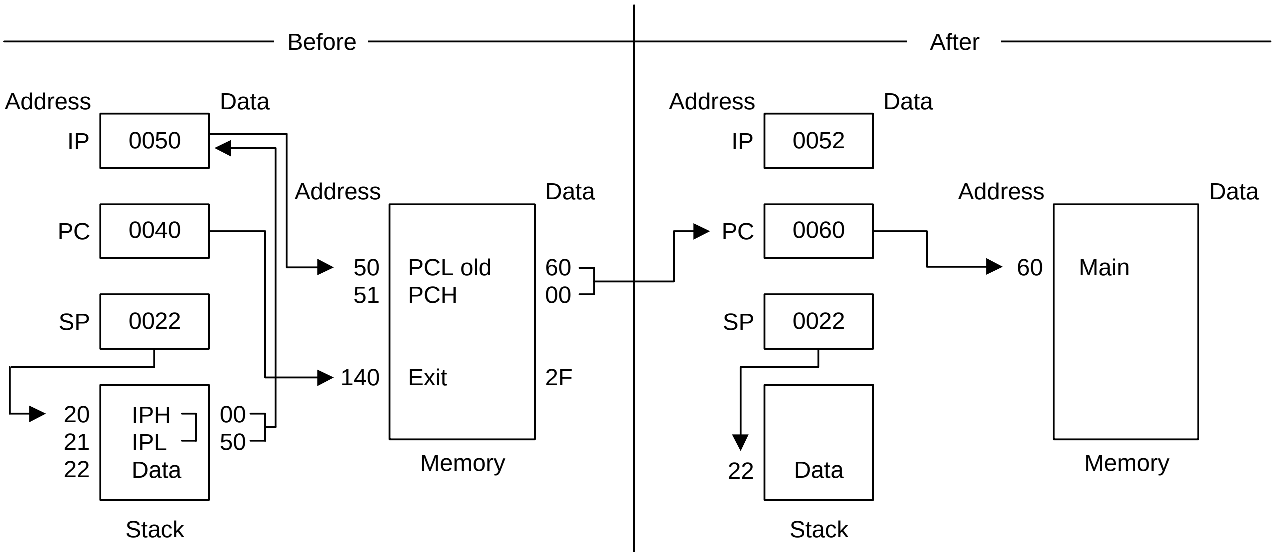
This instruction is useful when implementing threaded-code languages. The stack value is popped and loaded into the instruction pointer. The program memory word that is pointed to by the instruction pointer is then loaded into the program counter, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

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

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



IDLE — Idle Operation

IDLE

Operation:

The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue. Idle mode can be released by an interrupt request (IRQ) or an external reset operation.
In application programs, a IDLE instruction must be immediately followed by at least three NOP instructions. This ensures an adeguate time interval for the clock to stabilize before the next instruction is executed. If three or more NOP instructons are not used after IDLE instruction, leakage current could be flown because of the floating state in the internal bus.

Flags: No flags are affected.

Format:

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

Example: The instruction

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

INC — Increment

INC dst

Operation: dst ← dst + 1

The contents of the destination operand are incremented by one.

Flags:

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

Format:

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

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

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

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

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

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

INCW — Increment Word

INCW dst

Operation: dst ← dst + 1

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

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

Format:

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

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

INCW RR0 → R0 = 1AH, R1 = 03H
INCW @R1 → Register 02H = 10H, register 03H = 00H

In the first example, the working register pair RR0 contains the value 1AH in register R0 and 02H in register R1. The statement "INCW RR0" increments the 16-bit destination by one, leaving the value 03H in register R1. In the second example, the statement "INCW @R1" uses Indirect Register (IR) addressing mode to increment the contents of general register 03H from 0FFH to 00H and register 02H from 0FH to 10H.

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

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

IRET — Interrupt Return

IRET	IRET (Normal)	IRET (Fast)
Operation:	FLAGS ← @SP	PC ↔ IP
	SP ← SP + 1	FLAGS ← FLAGS'
	PC ← @SP	FIS ← 0
	SP ← SP + 2	
	SYM(0) ← 1	

This instruction is used at the end of an interrupt service routine. It restores the flag register and the program counter. It also re-enables global interrupts. A "normal IRET" is executed only if the fast interrupt status bit (FIS, bit one of the FLAGS register, 0D5H) is cleared (= "0"). If a fast interrupt occurred, IRET clears the FIS bit that was set at the beginning of the service routine.

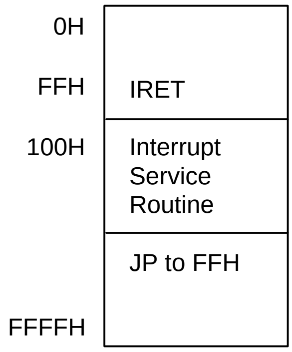
Flags: All flags are restored to their original settings (that is, the settings before the interrupt occurred).

Format:

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

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

Example: In the figure below, the instruction pointer is initially loaded with 100H in the main program before interrupts are enabled. When an interrupt occurs, the program counter and instruction pointer are swapped. This causes the PC to jump to address 100H and the IP to keep the return address. The last instruction in the service routine normally is a jump to IRET at address FFH. This causes the instruction pointer to be loaded with 100H "again" and the program counter to jump back to the main program. Now, the next interrupt can occur and the IP is still correct at 100H.



NOTE: In the fast interrupt example above, if the last instruction is not a jump to IRET, you must pay attention to the order of the last two instructions. The IRET cannot be immediately proceeded by a clearing of the interrupt status (as with a reset of the IPR register).

JP — Jump

JP cc,dst (Conditional)

JP dst (Unconditional)

Operation: If cc is true, PC ← dst

The conditional JUMP instruction transfers program control to the destination address if the condition specified by the condition code (cc) is true; otherwise, the instruction following the JP instruction is executed. The unconditional JP simply replaces the contents of the PC with the contents of the specified register pair. Control then passes to the statement addressed by the PC.

Flags: No flags are affected.

Format: (1)

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

NOTES:
1. The 3-byte format is used for a conditional jump and the 2-byte format for an unconditional jump.
2. In the first byte of the three-byte instruction format (conditional jump), the condition code and the opcode are both four bits.

Examples: Given: The carry flag (C) = "1", register 00 = 01H, and register 01 = 20H:

```
JP    C,LABEL_W    →   LABEL_W = 1000H, PC = 1000H
JP      @00H       →   PC = 0120H
```

The first example shows a conditional JP. Assuming that the carry flag is set to "1", the statement "JP C,LABEL_W" replaces the contents of the PC with the value 1000H and transfers control to that location. Had the carry flag not been set, control would then have passed to the statement immediately following the JP instruction.

The second example shows an unconditional JP. The statement "JP @00" replaces the contents of the PC with the contents of the register pair 00H and 01H, leaving the value 0120H.

JR — Jump Relative

JR cc,dst

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

If the condition specified by the condition code (cc) is true, the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise, the instruction following the JR instruction is executed. (See list of condition codes).

The range of the relative address is +127, –128, and the original value of the program counter is taken to be the address of the first instruction byte following the JR statement.

Flags: No flags are affected.

Format:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
(1) cc | opc | dst	2	6	ccB	RA
			cc = 0 to F	

NOTE: In the first byte of the two-byte instruction format, the condition code and the opcode are each four bits.

Example: Given: The carry flag = "1" and LABEL_X = 1FF7H:

JR C,LABEL_X → PC = 1FF7H

If the carry flag is set (that is, if the condition code is true), the statement "JR C,LABEL_X" will pass control to the statement whose address is now in the PC. Otherwise, the program instruction following the JR would be executed.

LD — Load

LD dst,src

Operation: dst ← src

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

Flags: No flags are affected.

Format:

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

LD — Load

LD (Continued)

Examples: Given: R0 = 01H, R1 = 0AH, register 00H = 01H, register 01H = 20H, register 02H = 02H, LOOP = 30H, and register 3AH = 0FFH:

LD	R0,#10H	→	R0 = 10H
LD	R0,01H	→	R0 = 20H, register 01H = 20H
LD	01H,R0	→	Register 01H = 01H, R0 = 01H
LD	R1,@R0	→	R1 = 20H, R0 = 01H
LD	@R0,R1	→	R0 = 01H, R1 = 0AH, register 01H = 0AH
LD	00H,01H	→	Register 00H = 20H, register 01H = 20H
LD	02H,@00H	→	Register 02H = 20H, register 00H = 01H
LD	00H,#0AH	→	Register 00H = 0AH
LD	@00H,#10H	→	Register 00H = 01H, register 01H = 10H
LD	@00H,02H	→	Register 00H = 01H, register 01H = 02, register 02H = 02H
LD	R0,#LOOP[R1]	→	R0 = 0FFH, R1 = 0AH
LD	#LOOP[R0],R1	→	Register 31H = 0AH, R0 = 01H, R1 = 0AH

LDB — Load Bit

LDB dst,src.b

LDB dst.b,src

Operation: dst(0) ← src(b)

or

dst(b) ← src(0)

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

Flags: No flags are affected.

Format:

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

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

Examples: Given: R0 = 06H and general register 00H = 05H:

LDB R0,00H.2 → R0 = 07H, register 00H = 05H

LDB 00H.0,R0 → R0 = 06H, register 00H = 04H

In the first example, destination working register R0 contains the value 06H and the source general register 00H the value 05H. The statement "LD R0,00H.2" loads the bit two value of the 00H register into bit zero of the R0 register, leaving the value 07H in register R0.

In the second example, 00H is the destination register. The statement "LD 00H.0,R0" loads bit zero of register R0 to the specified bit (bit zero) of the destination register, leaving 04H in general register 00H.

LDC/LDE — Load Memory

LDC/LDE dst,src

Operation: dst ← src

This instruction loads a byte from program or data memory into a working register or vice-versa. The source values are unaffected. LDC refers to program memory and LDE to data memory. The assembler makes 'Irr' or 'rr' values an even number for program memory and odd an odd number for data memory.

Flags: No flags are affected.

Format:

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

NOTES:
1. The source (src) or working register pair [rr] for formats 5 and 6 cannot use register pair 0–1.
2. For formats 3 and 4, the destination address 'XS [rr]' and the source address 'XS [rr]' are each one byte.
3. For formats 5 and 6, the destination address 'XL [rr] and the source address 'XL [rr]' are each two bytes.
4. The DA and r source values for formats 7 and 8 are used to address program memory; the second set of values, used in formats 9 and 10, are used to address data memory.

LDC/LDE — Load Memory

LDC/LDE (Continued)

Examples: Given: R0 = 11H, R1 = 34H, R2 = 01H, R3 = 04H; Program memory locations 0103H = 4FH, 0104H = 1A, 0105H = 6DH, and 1104H = 88H. External data memory locations 0103H = 5FH, 0104H = 2AH, 0105H = 7DH, and 1104H = 98H:

```
LDC      R0,@RR2            ; R0 ← contents of program memory location 0104H
                            ; R0 = 1AH, R2 = 01H, R3 = 04H
LDE      R0,@RR2            ; R0 ← contents of external data memory location 0104H
                            ; R0 = 2AH, R2 = 01H, R3 = 04H
LDC (note)@RR2,R0           ; 11H (contents of R0) is loaded into program memory
                            ; location 0104H (RR2),
                            ; working registers R0, R2, R3 → no change
LDE      @RR2,R0            ; 11H (contents of R0) is loaded into external data memory
                            ; location 0104H (RR2),
                            ; working registers R0, R2, R3 → no change
LDC      R0,#01H[RR2]       ; R0 ← contents of program memory location 0105H
                            ; (01H + RR2),
                            ; R0 = 6DH, R2 = 01H, R3 = 04H
LDE      R0,#01H[RR2]       ; R0 ← contents of external data memory location 0105H
                            ; (01H + RR2), R0 = 7DH, R2 = 01H, R3 = 04H
LDC (note) #01H[RR2],R0     ; 11H (contents of R0) is loaded into program memory location
                            ; 0105H (01H + 0104H)
LDE      #01H[RR2],R0       ; 11H (contents of R0) is loaded into external data memory
                            ; location 0105H (01H + 0104H)
LDC      R0,#1000H[RR2]     ; R0 ← contents of program memory location 1104H
                            ; (1000H + 0104H), R0 = 88H, R2 = 01H, R3 = 04H
LDE      R0,#1000H[RR2]     ; R0 ← contents of external data memory location 1104H
                            ; (1000H + 0104H), R0 = 98H, R2 = 01H, R3 = 04H
LDC      R0,1104H           ; R0 ← contents of program memory location 1104H,
                            ; R0 = 88H
LDE      R0,1104H           ; R0 ← contents of external data memory location 1104H,
                            ; R0 = 98H
LDC (note)1105H,R0          ; 11H (contents of R0) is loaded into program memory location
                            ; 1105H, (1105H) ← 11H
LDE      1105H,R0           ; 11H (contents of R0) is loaded into external data memory
                            ; location 1105H, (1105H) ← 11H
```

NOTE: These instructions are not supported by masked ROM type devices.

LDCD/LDED — Load Memory and Decrement

LDCD/LDED dst,src

Operation: dst ← src

rr ← rr – 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then decremented. The contents of the source are unaffected.

LDCD references program memory and LDED references external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

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

Examples: Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory location 1033H = 0CDH, and external data memory location 1033H = 0DDH:

```
LDCD    R8,@RR6    ;  0CDH (contents of program memory location 1033H) is loaded
                   ;  into R8 and RR6 is decremented by one
                   ;  R8 = 0CDH, R6 = 10H, R7 = 32H (RR6 ← RR6 – 1)

LDED    R8,@RR6    ;  0DDH (contents of data memory location 1033H) is loaded
                   ;  into R8 and RR6 is decremented by one (RR6 ← RR6 – 1)
                   ;  R8 = 0DDH, R6 = 10H, R7 = 32H
```

LDCI/LDEI — Load Memory and Increment

LDCI/LDEI dst,src

Operation: dst ← src

rr ← rr + 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then incremented automatically. The contents of the source are unaffected.

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

Flags: No flags are affected.

Format:

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

Examples: Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory locations 1033H = 0CDH and 1034H = 0C5H; external data memory locations 1033H = 0DDH and 1034H = 0D5H:

```
LDCI    R8,@RR6    ; 0CDH (contents of program memory location 1033H) is loaded
                   ; into R8 and RR6 is incremented by one (RR6 ← RR6 + 1)
                   ; R8 = 0CDH, R6 = 10H, R7 = 34H

LDEI    R8,@RR6    ; 0DDH (contents of data memory location 1033H) is loaded
                   ; into R8 and RR6 is incremented by one (RR6 ← RR6 + 1)
                   ; R8 = 0DDH, R6 = 10H, R7 = 34H
```

LDCPD/LDEPD — Load Memory with Pre-Decrement

LDCPD/
LDEPD dst,src

Operation: rr ← rr – 1

dst ← src

These instructions are used for block transfers of data from program or data memory from the register file. The address of the memory location is specified by a working register pair and is first decremented. The contents of the source location are then loaded into the destination location. The contents of the source are unaffected.

LDCPD refers to program memory and LDEPD refers to external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for external data memory.

Flags: No flags are affected.

Format:

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

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

```
LDCPD @RR6,R0     ; (RR6 ← RR6 – 1)
                  ; 77H (contents of R0) is loaded into program memory location
                  ; 2FFFH (3000H – 1H)
                  ; R0 = 77H, R6 = 2FH, R7 = 0FFH
LDEPD @RR6,R0     ; (RR6 ← RR6 – 1)
                  ; 77H (contents of R0) is loaded into external data memory
                  ; location 2FFFH (3000H – 1H)
                  ; R0 = 77H, R6 = 2FH, R7 = 0FFH
```

LDCPI/LDEPI — Load Memory with Pre-Increment

LDCPI/
LDEPI dst,src

Operation: rr ← rr + 1

dst ← src

These instructions are used for block transfers of data from program or data memory from the register file. The address of the memory location is specified by a working register pair and is first incremented. The contents of the source location are loaded into the destination location. The contents of the source are unaffected.

LDCPI refers to program memory and LDEPI refers to external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

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

Examples: Given: R0 = 7FH, R6 = 21H, and R7 = 0FFH:

```
LDCPI  @RR6,R0     ; (RR6 ← RR6 + 1)
                   ; 7FH (contents of R0) is loaded into program memory
                   ; location 2200H (21FFH + 1H)
                   ; R0 = 7FH, R6 = 22H, R7 = 00H
LDEPI  @RR6,R0     ; (RR6 ← RR6 + 1)
                   ; 7FH (contents of R0) is loaded into external data memory
                   ; location 2200H (21FFH + 1H)
                   ; R0 = 7FH, R6 = 22H, R7 = 00H
```

LDW — Load Word

LDW dst,src

Operation: dst ← src

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

Flags: No flags are affected.

Format:

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

Examples: Given: R4 = 06H, R5 = 1CH, R6 = 05H, R7 = 02H, register 00H = 1AH, register 01H = 02H, register 02H = 03H, and register 03H = 0FH:

LDW	RR6,RR4	→	R6 = 06H, R7 = 1CH, R4 = 06H, R5 = 1CH
LDW	00H,02H	→	Register 00H = 03H, register 01H = 0FH, register 02H = 03H, register 03H = 0FH
LDW	RR2,@R7	→	R2 = 03H, R3 = 0FH,
LDW	04H,@01H	→	Register 04H = 03H, register 05H = 0FH
LDW	RR6,#1234H	→	R6 = 12H, R7 = 34H
LDW	02H,#0FEDH	→	Register 02H = 0FH, register 03H = 0EDH

In the second example, please note that the statement "LDW 00H,02H" loads the contents of the source word 02H, 03H into the destination word 00H, 01H. This leaves the value 03H in general register 00H and the value 0FH in register 01H.

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

MULT — Multiply (Unsigned)

MULT dst,src

Operation: dst ← dst × src

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

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

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode 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, register 03H = 06H:

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

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

NEXT — Next

NEXT

Operation: PC ← @ IP

IP ← IP + 2

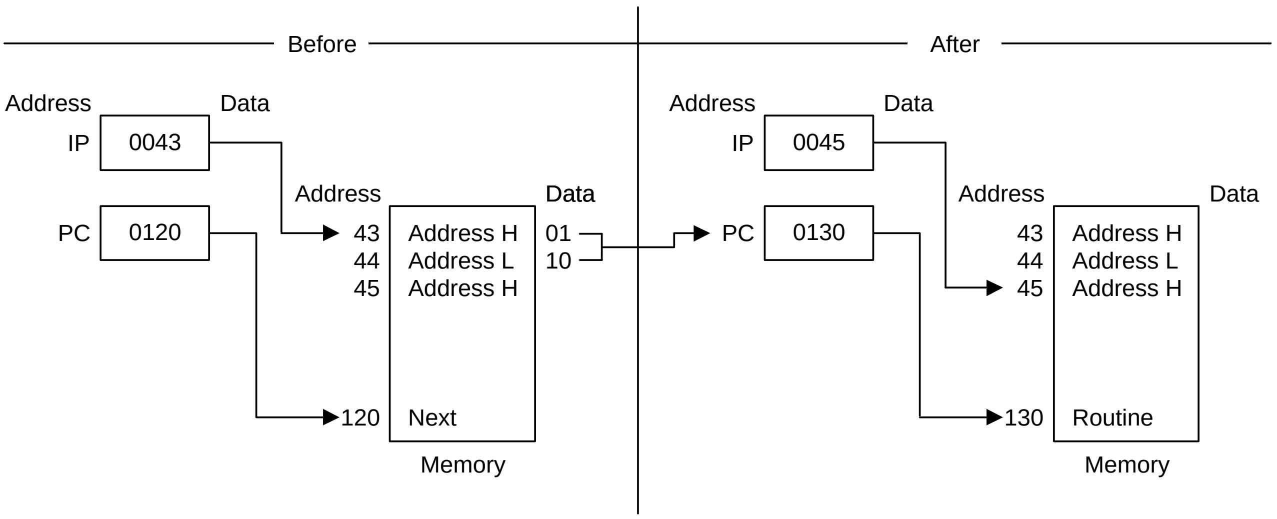
The NEXT instruction is useful when implementing threaded-code languages. The program memory word that is pointed to by the instruction pointer is loaded into the program counter. The instruction pointer is then incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	10	0F

Example: The following diagram shows one example of how to use the NEXT instruction.



SAMSUNG ELECTRONICS

NOP — No Operation

NOP

Operation: No action is performed when the CPU executes this instruction. Typically, one or more NOPs are executed in sequence in order to effect a timing delay of variable duration.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	FF

Example: When the instruction

NOP

is encountered in a program, no operation occurs. Instead, there is a delay in instruction execution time.

OR — Logical OR

OR dst,src

Operation: dst ← dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are unaffected. The OR operation results in a "1" being stored whenever either of the corresponding bits in the two operands is a "1"; otherwise a "0" is stored.

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

Format:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	4	42	r	r
		6	43	r	Ir
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 the value 2AH, the statement "OR R0,R1" logical-ORs the R0 and R1 register contents and stores the result (3FH) in destination register R0.

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

POP — Pop From Stack

POP dst

Operation: dst ← @SP

SP ← SP + 1

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

Flags: No flags affected.

Format:

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

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

POP 00H → Register 00H = 55H, SP = 00FCH

POP @00H → Register 00H = 01H, register 01H = 55H, SP = 00FCH

In the first example, general register 00H contains the value 01H. The statement "POP 00H" loads the contents of location 00FBH (55H) into destination register 00H and then increments the stack pointer by one. Register 00H then contains the value 55H and the SP points to location 00FCH.

POPUD — Pop User Stack (Decrementing)

POPUD dst,src

Operation: dst ← src

IR ← IR – 1

This instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack pointer is then decremented.

Flags: No flags are affected.

Format:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | src | dst	3	8	92	R	IR

Example: Given: Register 00H = 42H (user stack pointer register), register 42H = 6FH, and register 02H = 70H:

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

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

POPUI — Pop User Stack (Incrementing)

POPUI dst,src

Operation: dst ← src

IR ← IR + 1

The POPUI instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack pointer is then incremented.

Flags: No flags are affected.

Format:

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

Example: Given: Register 00H = 01H and register 01H = 70H:

POPUI 02H,@00H → Register 00H = 02H, register 01H = 70H, register 02H = 70H

If general register 00H contains the value 01H and register 01H the value 70H, the statement "POPUI 02H,@00H" loads the value 70H into the destination general register 02H. The user stack pointer (register 00H) is then incremented by one, changing its value from 01H to 02H.

PUSH — Push To Stack

PUSH src

Operation: SP ← SP – 1

@SP ← src

A PUSH instruction decrements the stack pointer value and loads the contents of the source (src) into the location addressed by the decremented stack pointer. The operation then adds the new value to the top of the stack.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
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 the value 4FH, the statement "PUSH 40H" decrements the stack pointer from 0000 to 0FFFFH. It then loads the contents of register 40H into location 0FFFFH and adds this new value to the top of the stack.

PUSHUD — Push User Stack (Decrementing)

PUSHUD dst,src

Operation: IR ← IR – 1

dst ← src

This instruction is used to address user-defined stacks in the register file. PUSHUD decrements the user stack pointer and loads the contents of the source into the register addressed by the decremented stack pointer.

Flags: No flags are affected.

Format:

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

Example: Given: Register 00H = 03H, register 01H = 05H, and register 02H = 1AH:

PUSHUD @00H,01H → Register 00H = 02H, register 01H = 05H, register 02H = 05H

If the user stack pointer (register 00H, for example) contains the value 03H, the statement "PUSHUD @00H,01H" decrements the user stack pointer by one, leaving the value 02H. The 01H register value, 05H, is then loaded into the register addressed by the decremented user stack pointer.

PUSHUI — Push User Stack (Incrementing)

PUSHUI dst,src

Operation: IR ← IR + 1

dst ← src

This instruction is used for user-defined stacks in the register file. PUSHUI increments the user stack pointer and then loads the contents of the source into the register location addressed by the incremented user stack pointer.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	3	8	83	IR	R

Example: Given: Register 00H = 03H, register 01H = 05H, and register 04H = 2AH:

PUSHUI @00H,01H → Register 00H = 04H, register 01H = 05H, register 04H = 05H

If the user stack pointer (register 00H, for example) contains the value 03H, the statement "PUSHUI @00H,01H" increments the user stack pointer by one, leaving the value 04H. The 01H register value, 05H, is then loaded into the location addressed by the incremented user stack pointer.

RCF — Reset Carry Flag

RCF RCF

Operation: C ← 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.

RET — Return

RET

Operation: PC ← @SP

SP ← SP + 2

The RET instruction is normally used to return to the previously executing procedure at the end of a procedure entered by a CALL instruction. The contents of the location addressed by the stack pointer are popped into the program counter. The next statement that is executed is the one that is addressed by the new program counter value.

Flags: No flags are affected.

Format:

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

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

RET → PC = 101AH, SP = 00FEH

The statement "RET" pops the contents of stack pointer location 00FCH (10H) into the high byte of the program counter. The stack pointer then pops the value in location 00FEH (1AH) into the PC's low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location 00FEH.

RL — Rotate Left

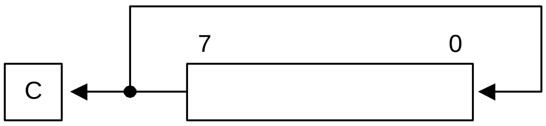
RL dst

Operation: C ← dst (7)

dst (0) ← dst (7)

dst (n + 1) ← dst (n), n = 0–6

The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit zero (LSB) position and also replaces the carry flag.



Flags:
C: Set if the bit rotated from the most significant bit position (bit 7) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

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.

RLC — Rotate Left Through Carry

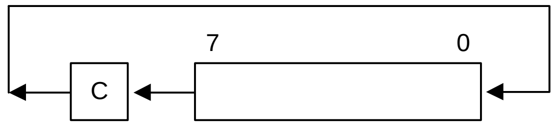
RLC dst

Operation: dst (0) ← C

C ← dst (7)

dst (n + 1) ← dst (n), n = 0–6

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



Flags:

C: Set if the bit rotated from the most significant bit position (bit 7) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

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

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

RLC 00H → Register 00H = 54H, C = "1"

RLC @01H → Register 01H = 02H, register 02H = 2EH, C = "0"

In the first example, if general register 00H has the value 0AAH (10101010B), the statement "RLC 00H" rotates 0AAH one bit position to the left. The initial value of bit 7 sets the carry flag and the initial value of the C flag replaces bit zero of register 00H, leaving the value 55H (01010101B). The MSB of register 00H resets the carry flag to "1" and sets the overflow flag.

RR — Rotate Right

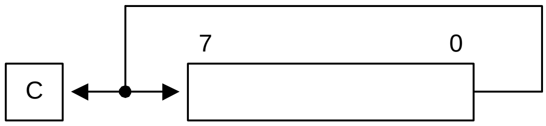
RR dst

Operation: C ← dst (0)

dst (7) ← dst (0)

dst (n) ← dst (n + 1), n = 0–6

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



Flags:

C: Set if the bit rotated from the least significant bit position (bit zero) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

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

Examples: Given: Register 00H = 31H, register 01H = 02H, and register 02H = 17H:

RR 00H → Register 00H = 98H, C = "1"

RR @01H → Register 01H = 02H, register 02H = 8BH, C = "1"

In the first example, if general register 00H contains the value 31H (00110001B), the statement "RR 00H" rotates this value one bit position to the right. The initial value of bit zero is moved to bit 7, leaving the new value 98H (10011000B) in the destination register. The initial bit zero also resets the C flag to "1" and the sign flag and overflow flag are also set to "1".

RRC — Rotate Right Through Carry

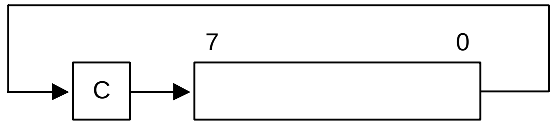
RRC dst

Operation: dst (7) ← C

C ← dst (0)

dst (n) ← dst (n + 1), n = 0–6

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



Flags:
C: Set if the bit rotated from the least significant bit position (bit zero) was "1".
Z: Set if the result is "0" cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

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

Examples: Given: Register 00H = 55H, register 01H = 02H, register 02H = 17H, and C = "0":

RRC 00H → Register 00H = 2AH, C = "1"

RRC @01H → Register 01H = 02H, register 02H = 0BH, C = "1"

In the first example, if general register 00H contains the value 55H (01010101B), the statement "RRC 00H" rotates this value one bit position to the right. The initial value of bit zero ("1") replaces the carry flag and the initial value of the C flag ("1") replaces bit 7. This leaves the new value 2AH (00101010B) in destination register 00H. The sign flag and overflow flag are both cleared to "0".

SB0 — Select Bank 0

SB0

Operation: BANK ← 0

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

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	4F

Example: The statement

SB0

clears FLAGS.0 to "0", selecting bank 0 register addressing.

SB1 — Select Bank 1

SB1

Operation: BANK ← 1

The SB1 instruction sets the bank address flag in the FLAGS register (FLAGS.0) to logic one, selecting bank 1 register addressing in the set 1 area of the register file. (Bank 1 is not implemented in some S3C8-series microcontrollers.)

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	5F

Example: The statement

SB1

sets FLAGS.0 to "1", selecting bank 1 register addressing, if implemented.

SBC — Subtract with Carry

SBC dst,src

Operation: dst ← dst − src − c

The source operand, along with the current value of the carry flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's-complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of the low-order operands to be subtracted from the subtraction of high-order operands.

Flags:

C: Set if a borrow occurred (src > dst); cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the operands were of opposite sign and the sign of the result is the same as the sign of the source; cleared otherwise.
D: Always set to "1".
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise, indicating a "borrow".

Format:

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

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

```
SBC   R1,R2     →  R1 = 0CH, R2 = 03H
SBC   R1,@R2    →  R1 = 05H, R2 = 03H, register 03H = 0AH
SBC   01H,02H   →  Register 01H = 1CH, register 02H = 03H
SBC   01H,@02H  →  Register 01H = 15H,register 02H = 03H, register 03H = 0AH
SBC   01H,#8AH  →  Register 01H = 95H; C, S, and V = "1"
```

In the first example, if working register R1 contains the value 10H and register R2 the value 03H, the statement "SBC R1,R2" subtracts the source value (03H) and the C flag value ("1") from the destination (10H) and then stores the result (0CH) in register R1.

SCF — Set Carry Flag

SCF

Operation: C ← 1

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

Flags: C: Set to "1".

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	DF

Example: The statement

SCF

sets the carry flag to logic one.

SRA — Shift Right Arithmetic

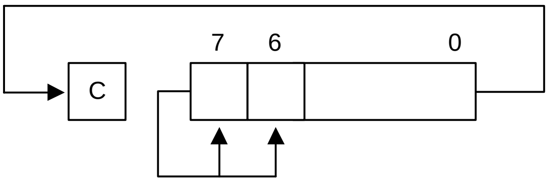
SRA dst

Operation: dst (7) ← dst (7)

C ← dst (0)

dst (n) ← dst (n + 1), n = 0–6

An arithmetic shift-right of one bit position is performed on the destination operand. Bit zero (the LSB) replaces the carry flag. The value of bit 7 (the sign bit) is unchanged and is shifted into bit position 6.



Flags:
C: Set if the bit shifted from the LSB position (bit zero) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Always cleared to "0".
D: Unaffected.
H: Unaffected.

Format:

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

Examples: Given: Register 00H = 9AH, register 02H = 03H, register 03H = 0BCH, and C = "1":

SRA 00H → Register 00H = 0CD, C = "0"

SRA @02H → Register 02H = 03H, register 03H = 0DEH, C = "0"

In the first example, if general register 00H contains the value 9AH (10011010B), the statement "SRA 00H" shifts the bit values in register 00H right one bit position. Bit zero ("0") clears the C flag and bit 7 ("1") is then shifted into the bit 6 position (bit 7 remains unchanged). This leaves the value 0CDH (11001101B) in destination register 00H.

SRP/SRP0/SRP1 — Set Register Pointer

SRP src

SRP0 src

SRP1 src

Operation:

If src (1) = 1 and src (0) = 0 then:	RP0 (3–7)	←	src (3–7)
If src (1) = 0 and src (0) = 1 then:	RP1 (3–7)	←	src (3–7)
If src (1) = 0 and src (0) = 0 then:	RP0 (4–7)	←	src (4–7),
	RP0 (3)	←	0
	RP1 (4–7)	←	src (4–7),
	RP1 (3)	←	1

The source data bits one and zero (LSB) determine whether to write one or both of the register pointers, RP0 and RP1. Bits 3–7 of the selected register pointer are written unless both register pointers are selected. RP0.3 is then cleared to logic zero and RP1.3 is set to logic one.

Flags: No flags are affected.

Format:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode src
opc | src	2	4	31	IM

Examples: The statement

SRP #40H

sets register pointer 0 (RP0) at location 0D6H to 40H and register pointer 1 (RP1) at location 0D7H to 48H.

The statement "SRP0 #50H" sets RP0 to 50H, and the statement "SRP1 #68H" sets RP1 to 68H.

STOP — Stop Operation

STOP

Operation:

The STOP instruction stops the both the CPU clock and system clock and causes the microcontroller to enter Stop mode. During Stop mode, the contents of on-chip CPU registers, peripheral registers, and I/O port control and data registers are retained. Stop mode can be released by an external reset operation or by external interrupts. For the reset operation, the nRESET pin must be held to Low level until the required oscillation stabilization interval has elapsed.

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

Flags: No flags are affected.

Format:

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

Example: The statement

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

SUB — Subtract

SUB dst,src

Operation: dst ← dst – src

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

Flags:

C: Set if a "borrow" occurred; cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is of the same as the sign of the source operand; cleared otherwise.
D: Always set to "1".
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow".

Format:

Format	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode 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"

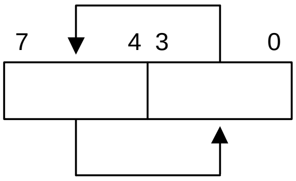
In the first example, if working register R1 contains the value 12H and if register R2 contains the value 03H, the statement "SUB R1,R2" subtracts the source value (03H) from the destination value (12H) and stores the result (0FH) in destination register R1.

SWAP — Swap Nibbles

SWAP dst

Operation: dst (0 – 3) ↔ dst (4 – 7)

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



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

Format:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode 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).

TCM — Test Complement Under Mask

TCM dst,src

Operation: (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logic one value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

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

Format:

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

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 12H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

TCM	R0,R1	→	R0 = 0C7H, R1 = 02H, Z = "1"
TCM	R0,@R1	→	R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
TCM	00H,01H	→	Register 00H = 2BH, register 01H = 02H, Z = "1"
TCM	00H,@01H	→	Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "1"
TCM	00H,#34	→	Register 00H = 2BH, Z = "0"

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TCM R0,R1" tests bit one in the destination register for a "1" value. Because the mask value corresponds to the test bit, the Z flag is set to logic one and can be tested to determine the result of the TCM operation.

TM — Test Under Mask

TM dst,src

Operation: dst AND src

This instruction tests selected bits in the destination operand for a logic zero value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

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

Format:

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

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

TM	R0,R1	→	R0 = 0C7H, R1 = 02H, Z = "0"
TM	R0,@R1	→	R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
TM	00H,01H	→	Register 00H = 2BH, register 01H = 02H, Z = "0"
TM	00H,@01H	→	Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "0"
TM	00H,#54H	→	Register 00H = 2BH, Z = "1"

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TM R0,R1" tests bit one in the destination register for a "0" value. Because the mask value does not match the test bit, the Z flag is cleared to logic zero and can be tested to determine the result of the TM operation.

WFI — Wait for Interrupt

WFI

Operation:

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

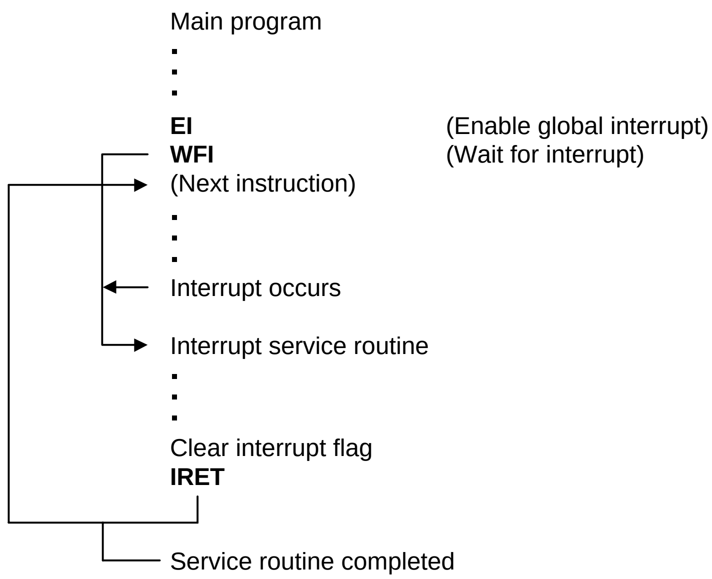
Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	4n	3F

(n = 1, 2, 3, …)

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



XOR — Logical Exclusive OR

XOR dst,src

Operation: dst ← dst XOR src

The source operand is logically exclusive-ORed with the destination operand and the result is stored in the destination. The exclusive-OR operation results in a "1" bit being stored whenever the corresponding bits in the operands are different; otherwise, a "0" bit is stored.

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

Format:

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

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

XOR	R0,R1	→	R0 = 0C5H, R1 = 02H
XOR	R0,@R1	→	R0 = 0E4H, R1 = 02H, register 02H = 23H
XOR	00H,01H	→	Register 00H = 29H, register 01H = 02H
XOR	00H,@01H	→	Register 00H = 08H, register 01H = 02H, register 02H = 23H
XOR	00H,#54H	→	Register 00H = 7FH

In the first example, if working register R0 contains the value 0C7H and if register R1 contains the value 02H, the statement "XOR R0,R1" logically exclusive-ORs the R1 value with the R0 value and stores the result (0C5H) in the destination register R0.

7 CLOCK CIRCUIT

OVERVIEW

By smart option (3FH.0 in ROM), user can select internal oscillator or external oscillator. When using external RC oscillator or internal RC oscillator, XOUT (P1.1) can be used as normal I/O pins. An internal RC oscillator source provides a typical 8 MHz or 1 MHz (at V_{DD} = 5 V) depending on smart option.

An external RC oscillation source provides a typical 4 MHz clock for S3F84K4. An external crystal or ceramic oscillation source provides a maximum 8 MHz clock. The X_{IN} and X_{OUT} pins connect the oscillation source to the on-chip clock circuit. Simplified external RC oscillator and crystal/ceramic oscillator circuits are shown in Figures 7-1 and 7-2. When you use external oscillator, P1.1 must be set to output port to prevent current consumption

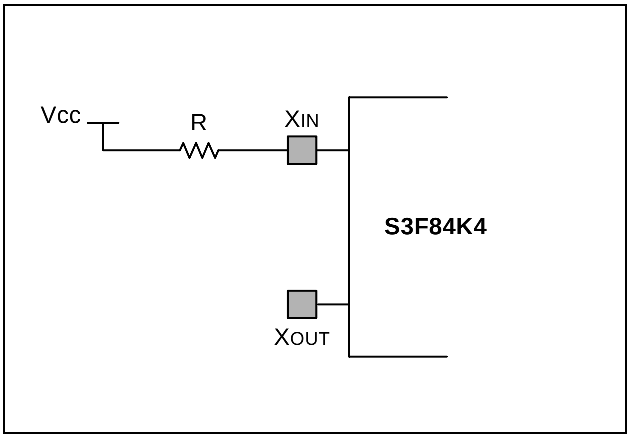


Figure 7-1. Main Oscillator Circuit (RC Oscillator with Internal Capacitor)

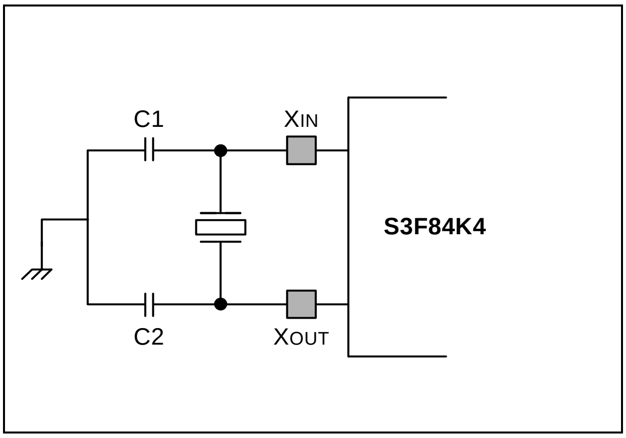


Figure 7-2. Main Oscillator Circuit (Crystal/Ceramic Oscillator)

MAIN OSCILLATOR LOGIC

To increase processing speed and to reduce clock noise, non-divided logic is implemented for the main oscillator circuit. For this reason, very high resolution waveforms (square signal edges) must be generated in order for the CPU to efficiently process logic operations.

CLOCK STATUS DURING POWER-DOWN MODES

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

— In Stop mode, the main oscillator "freezes", halting the CPU and peripherals. The contents of the register file and current system register values are retained. Stop mode is released, and the oscillator started, by a reset operation or by an external interrupt with RC-delay noise filter (for S3F84K4, INT0–INT1).

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

SYSTEM CLOCK CONTROL REGISTER (CLKCON)

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

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

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

The CLKCON register controls whether or not an external interrupt can be used to trigger a Stop mode release (This is called the "IRQ wake-up" function). The IRQ wake-up enable bit is CLKCON.7.

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

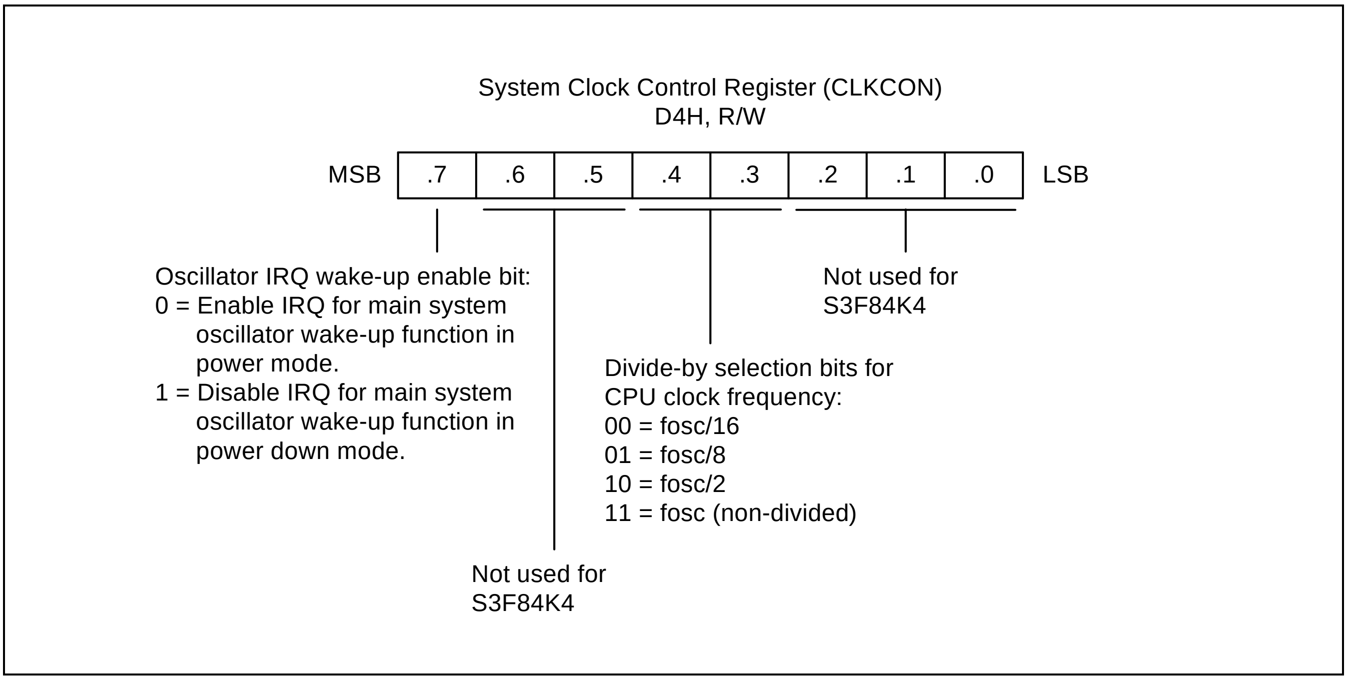
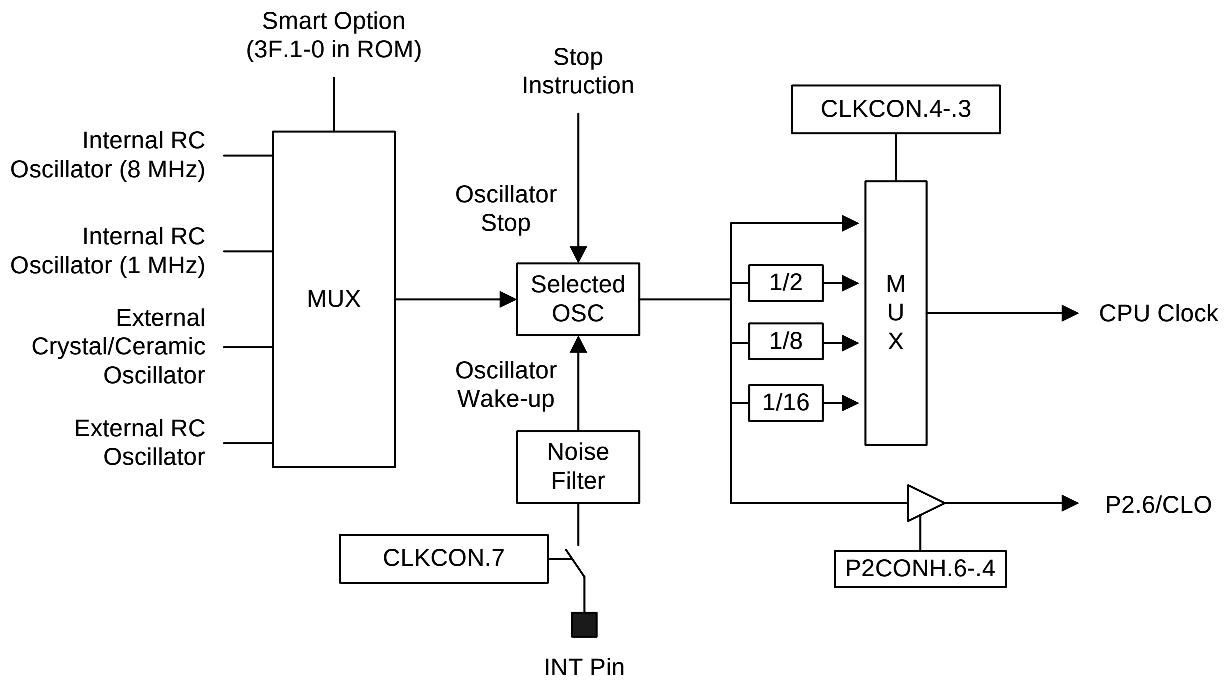


Figure 7-3. System Clock Control Register (CLKCON)



NOTE: An external interrupt (with RC-delay noise filter) can be used to release stop mode and "wake-up" the main oscillator.
In the S3F84K4, the INT0-INT1 external interrupts are of this type.

Figure 7-4. System Clock Circuit Diagram

8 RESET and POWER-DOWN

SYSTEM RESET

OVERVIEW

By smart option (3EH.7 in ROM), user can select internal RESET (LVR) or external RESET. When using internal RESET (LVR), nRESET pin (P1.2) can be used by normal I/O pin.

The S3F84K4 can be RESET in four ways:

— by external power-on-reset
— by the external nRESET input pin pulled low
— by the digital watchdog peripheral timing out
— by Low Voltage Reset (LVR)

During an external power-on reset, the voltage at V_{DD} is High level and the nRESET pin is forced to Low level. The nRESET signal is an input through a Schmitt trigger circuit where it is then synchronized with the CPU clock. This brings the S3F84K4 into a known operating status. To ensure correct start-up, the user should take care that nRESET signal is not released before the V_{DD} level is sufficient to allow MCU operation at the chosen frequency.

The nRESET pin must be held to Low level for a minimum time interval after the power supply comes within tolerance in order to allow time for internal CPU clock oscillation to stabilize. The minimum required oscillation stabilization time for a reset is approximately 65.5 ms (@ $2^{16}/f_{OSC}$, f_{OSC} = 8 MHz).

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

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

The on-chip Low Voltage Reset, features static Reset when supply voltage is below a reference value (Typ. 2.2, 3.0, 3.9V). Thanks to this feature, external reset circuit can be removed while keeping the application safety. As long as the supply voltage is below the reference value, there is an internal and static RESET. The MCU can start only when the supply voltage rises over the reference value.

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

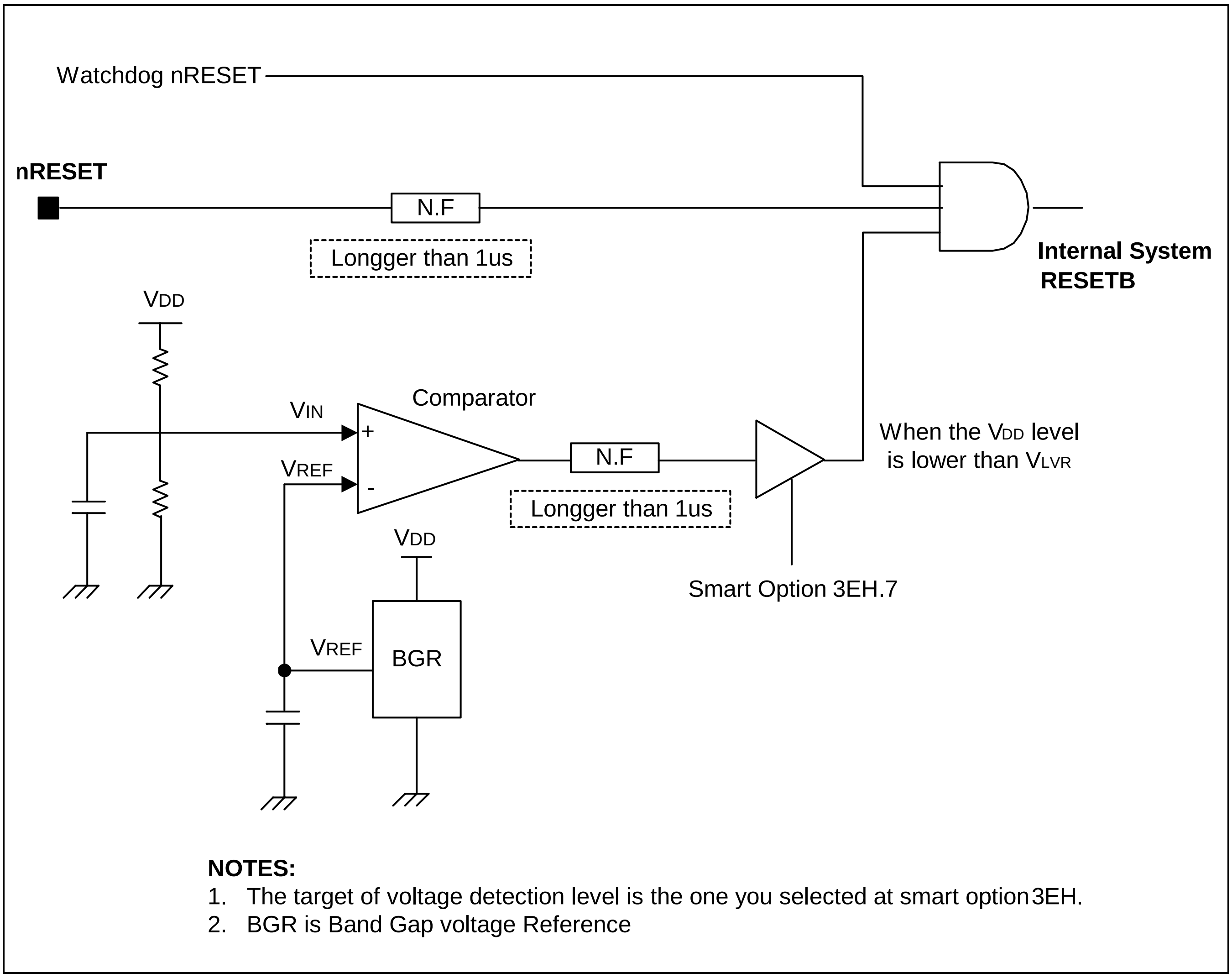


Figure 8-1. Low Voltage Reset Circuit

NOTE

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

MCU Initialization Sequence

The following sequence of events occurs during a Reset operation:

— All interrupts are disabled.
— The watchdog function (basic timer) is enabled.
— Ports 0–2 are set to input mode
— Peripheral control and data registers are disabled and reset to their initial values (see Table 8-1).
— The program counter is loaded with the ROM reset address, 0100H.
— When the programmed oscillation stabilization time interval has elapsed, the address stored in ROM location 0100H (and 0101H) is fetched and executed.

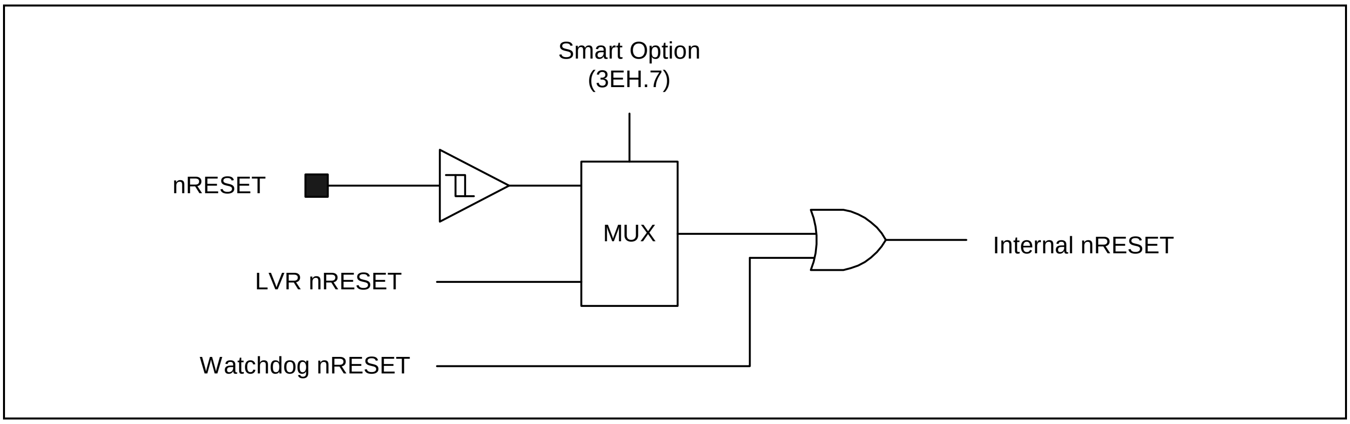


Figure 8-2. Reset Block Diagram

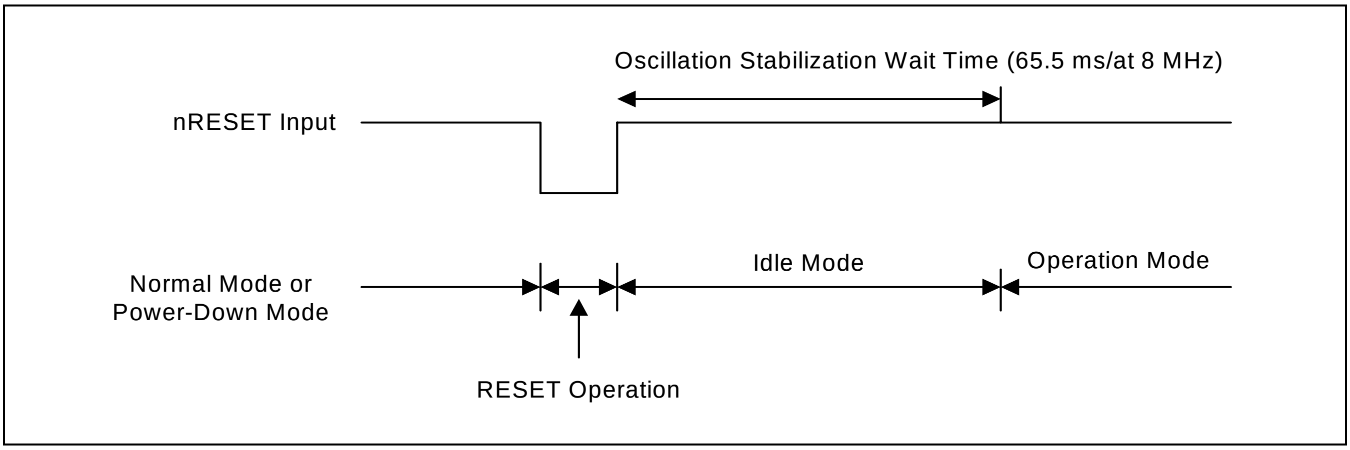


Figure 8-3. Timing for S3F84K4 after RESET

POWER-DOWN MODES

STOP MODE

Stop mode is invoked by the instruction STOP (opcode 7FH). In Stop mode, the operation of the CPU and all peripherals is halted. That is, the on-chip main oscillator stops and the supply current is reduced to less than 200 μA except that the LVR (Low Voltage Reset) is enable. All system functions are halted when the clock "freezes", but data stored in the internal register file is retained. Stop mode can be released in one of two ways: by an nRESET signal or by an external interrupt.

Using RESET to Release Stop Mode

Stop mode is released when the nRESET signal is released and returns to High level. All system and peripheral control registers are then reset to their default values and the contents of all data registers are retained. A Reset operation automatically selects a slow clock (f_{OSC}/16) because CLKCON.3 and CLKCON.4 are cleared to "00B". After the oscillation stabilization interval has elapsed, the CPU executes the system initialization routine by fetching the 16-bit address stored in ROM locations 0100H and 0101H.

Using an External Interrupt to Release Stop Mode

External interrupts with an RC-delay noise filter circuit can be used to release Stop mode (Clock-related external interrupts cannot be used). External interrupts INT0-INT1 in the S3F84K4 interrupt structure meet this criterion.

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

The external interrupt is serviced when the Stop mode release occurs. Following the IRET from the service routine, the instruction immediately following the one that initiated Stop mode is executed.

IDLE MODE

Idle mode is invoked by the instruction IDLE (opcode 6FH). In Idle mode, CPU operations are halted while select peripherals remain active. During Idle mode, the internal clock signal is gated off to the CPU, but not to interrupt logic and timer/counters. Port pins retain the mode (input or output) they had at the time Idle mode was entered.

There are two ways to release Idle mode:

1. Execute a Reset. All system and peripheral control registers are reset to their default values and the contents of all data registers are retained. The Reset automatically selects a slow clock (f_{OSC}/16) because CLKCON.3 and CLKCON.4 are cleared to "00B". If interrupts are masked, a Reset is the only way to release Idle mode.

2. Activate any enabled interrupt, causing Idle mode to be released. When you use an interrupt to release Idle mode, the CLKCON.3 and CLKCON.4 register values remain unchanged, and the currently selected clock value is used. The interrupt is then serviced. Following the IRET from the service routine, the instruction immediately following the one that initiated Idle mode is executed.

NOTES

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

HARDWARE RESET VALUES

Table 8-1 lists the values for CPU and system registers, peripheral control registers, and peripheral data registers following a Reset operation in normal operating mode.

— 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 following a reset.

— A dash ("–") means that the bit is either not used or not mapped.

Table 8-1. Register Values After a Reset

Register name	Mnemonic	Address & Location		RESET value (Bit)							
		Address	R/W	7	6	5	4	3	2	1	0
Timer A counter register	TACNT	D0H	R	0	0	0	0	0	0	0	0
Timer A data register	TADATA	D1H	R/W	1	1	1	1	1	1	1	1
Timer 0/A control register	TACON	D2H	R/W	0	0	0	0	0	0	0	0
Location D2H is not mapped											
Basic timer control register	BTCON	D3H	R/W	0	0	0	0	0	0	0	0
Clock control register	CLKCON	D4H	R/W	0	–	–	0	0	–	–	–
System flags register	FLAGS	D5H	R/W	x	x	x	x	x	x	0	0
Register Pointer 0	RP0	D6H	R/W	1	1	0	0	0	–	–	–
Register Pointer 1	RP1	D7H	R/W	1	1	0	0	1	–	–	–
Location D8H is not mapped											
Stack pointer register	SPL	D9H	R/W	x	x	x	x	x	x	x	x
Instruction Pointer (High Byte)	IPH	DAH	R/W	x	x	x	x	x	x	x	x
Instruction Pointer (Low Byte)	IPL	DBH	R/W	x	x	x	x	x	x	x	x
Interrupt Request Register	IRQ	DCH	R	0	0	0	0	0	0	0	0
Interrupt Mask Register	IMR	DDH	R/W	0	0	0	0	0	0	0	0
System Mode Register	SYM	DEH	R/W	0	–	–	x	x	x	0	0
Register Page Pointer	PP	DFH	R/W	0	0	0	0	0	0	0	0

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

Table 8-1. Register Values After a Reset (Continued)

Register Name	Mnemonic	Address	R/W	Bit Values After RESET							
		Hex		7	6	5	4	3	2	1	0
Port 0 data register	P0	E0H	R/W	0	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	R/W	–	–	–	–	–	0	0	–
Port 2 data register	P2	E2H	R/W	–	0	0	0	0	0	0	0
Locations E3H–E5H are not mapped											
Port 0 control register (High byte)	P0CONH	E6H	R/W	0	0	0	0	0	0	0	0
Port 0 control register (Low byte)	P0CONL	E7H	R/W	0	0	0	0	0	0	0	0
Port 0 interrupt pending register	P0PND	E8H	R/W	–	–	–	–	0	0	0	0
Port 1 control register	P1CON	E9H	R/W	0	–	–	–	0	0	–	–
Port 2 control register (High byte)	P2CONH	EAH	R/W	–	0	0	0	0	0	0	0
Port 2 control register (Low byte)	P2CONL	EBH	R/W	0	0	0	0	0	0	0	0
Timer B counter register	TBCNT	ECH	R	0	0	0	0	0	0	0	0
Timer B data register	TBDATA	EDH	R/W	1	1	1	1	1	1	1	1
Timer B control register	TBCON	EEH	R/W	–	–	0	0	0	0	0	0
Location F0H is not mapped											
PWM extension data register	PWMEX	F1H	R/W	0	0	0	0	0	0	–	–
PWM data register	PWMDATA	F2H	R/W	–	–	0	0	0	0	0	0
PWM control register	PWMCON	F3H	R/W	0	0	–	0	0	0	0	0
STOP control register	STOPCON	F4H	R/W	0	0	0	0	0	0	0	0
Locations F5H–F6H are not mapped											
A/D control register	ADCON	F7H	R/W	0	0	0	0	0	0	0	0
A/D converter data register (High)	ADDATAH	F8H	R	x	x	x	x	x	x	x	x
A/D converter data register (Low)	ADDATAL	F9H	R	0	0	0	0	0	0	x	x
Locations FAH–FCH are not mapped											
Basic timer counter	BTCNT	FDH	R	0	0	0	0	0	0	0	0
External memory timing register	EMT	FEH	R/W	0	1	1	1	1	1	0	–
Interrupt priority register	IPR	FFH	R/W	x	x	x	x	x	x	x	x

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

SAMSUNG ELECTRONICS

☞ PROGRAMMING TIP — Sample S3F84K4 Initialization Routine

```
                ORG         0000H
;--------------<< Smart Option >>
                ORG         003CH
                DB          0FFH                ;   003CH, must be initialized to 0
                DB          0FFH                ;   003DH, must be initialized to 0
                DB          0FFH                ;   003EH, enable LVR (3.0v)
                DB          0FEH                ;   003FH, External RC oscillator

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

                VECTOR      0F2H, PWMOVF_INT    ;
                VECTOR      0F4H, INT_TIMERB    ;
                VECTOR      0F6H, INT_TIMERA    ;
                VECTOR      0FAH, INT_EXT1      ;
                VECTOR      0FCH, INT_EXT0      ;

;--------------<< Initialize System and Peripherals >>
                ORG         0100H
RESET:          DI                              ;   disable interrupt
                LD          BTCON,#10100011B    ;   Watch-dog disable
                LD          CLKCON,#00011000B   ;   Select non-divided CPU clock
                LD          SPL,#0C0H           ;   Stack pointer must be set

                LD          P0CONH,#10101010B   ;
                LD          P0CONL,#10101010B   ;   P0.0–P0.7 push-pull output
                LD          P0PND,#00001010B    ;   P0.0, P0.1 interrupt enable
                LD          P1CON,#00001000B    ;   P1.1 push-pull output
                LD          P2CONH,#01001010B   ;
                LD          P2CONL,#10101010B   ;   P2.0–P2.6 push-pull output

                LD          IMR,#00000111B      ;   Enable IRQ0, IRQ1, IRQ2 interrupt
                LD          IPR,#00010011B      ;   IRQ2>IRQ1>IRQ0

;--------------<< Timer 0 settings >>

                LD          TADATA,#50H         ;
                LD          TBDATA,#50H
                LD          TACON,#00000110B    ;   fOSC/256, Timer A interrupt enable
                LD          TBCON,#00000110B    ;   fOSC/256, Timer B interrupt enable



;--------------<< Initialize other registers >>
                •
                •
                EI                              ;   Enable interrupt
```

☞ PROGRAMMING TIP — Sample S3F84K4 Initialization Routine (Continued)

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

MAIN:       NOP                           ;   Start main loop
            LD          BTCON,#02H        ;   Enable watchdog function
                                          ;   Basic counter (BTCNT) clear
            •
            •
            CALL        KEY_SCAN          ;
            •
            •
            •
            CALL        LED_DISPLAY       ;
            •
            •
            •
            CALL        JOB               ;
            •
            •
            •
            JR          T,MAIN            ;

;--------------<< Subroutines >>

KEY_SCAN: NOP                             ;
            •
            •
            •
            RET

LED_DISPLAY: NOP                          ;
            •
            •
            •
            RET

JOB:        NOP                           ;
            •
            •
            •
            RET
```

☞ PROGRAMMING TIP — Sample S3F84K4 Initialization Routine (Continued)

```
;--------------< Timer A interrupt service routine >

INT_TIMERA:

                •                                ;
                •
                AND       TACON,#11111110B       ;  Pending bit clear
                IRET                             ;  Interrupt return

;--------------< Timer B interrupt service routine >

INT_TIMERB:

                •                                ;
                •
                AND       TBCON,#11111110B       ;  Pending bit clear
                IRET

;--------------< PWM overflow interrupt service routine >

PWMOVF_INT:

                •
                •
                AND       PWMCON,#11111110B      ;  Pending bit clear
                IRET                             ;  Interrupt return

;--------------< External interrupt0 service routine >

INT_EXT0:       •
                •
                AND       P0PND,#11111110B       ;  EXT0 Pending bit clear
                IRET                             ;  Interrupt return

;--------------< External interrupt1 service routine >

INT_EXT1:       •
                •
                AND       P0PND,#11111011B       ;  EXT1 Pending bit clear
                IRET                             ;  Interrupt return
                •
                •
                .END                             ;
```

9 I/O PORTS

OVERVIEW

The S3F84K4 has three I/O ports: with 17 pins total. You access these ports directly by writing or reading port data register addresses.

All ports can be configured as LED drive. (High current output: typical 10 mA)

Table 9-1. S3F84K4 Port Configuration Overview

Port	Function Description	Programmability
0	Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are assignable by software. Port 0 pins can also be used as alternative function. (ADC input, external interrupt input).	Bit
1	Bit-programmable I/O port for Schmitt trigger input or push-pull, open-drain output. Pull-up or pull-down resistors are assignable by software. Port 1 pins can also be oscillator input/output or reset input by smart option. P1.0 is input only.	Bit
2	Bit-programmable I/O port for Schmitt trigger input or push-pull, open-drain output. Pull-up resistors are assignable by software. Port 2 can also be used as alternative function (ADC input, CLO, T0 clock output)	Bit

PORT DATA REGISTERS

Table 9-2 gives you an overview of the port data register names, locations, and addressing characteristics. Data registers for ports 0-2 have the structure shown in Figure 9-1.

Table 9-2. Port Data Register Summary

Register Name	Mnemonic	Hex	R/W
Port 0 data register	P0	E0H	R/W
Port 1 data register	P1	E1H	R/W
Port 2 data register	P2	E2H	R/W

NOTE: A reset operation clears the P0–P2 data register to "00H".

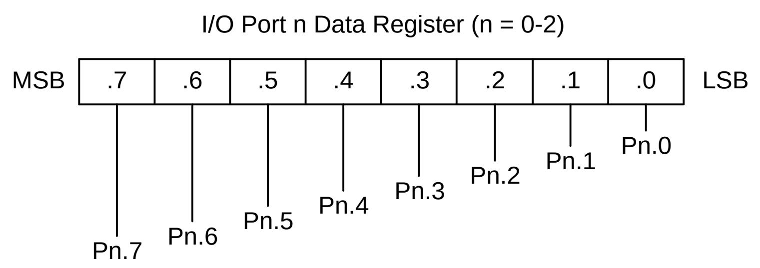


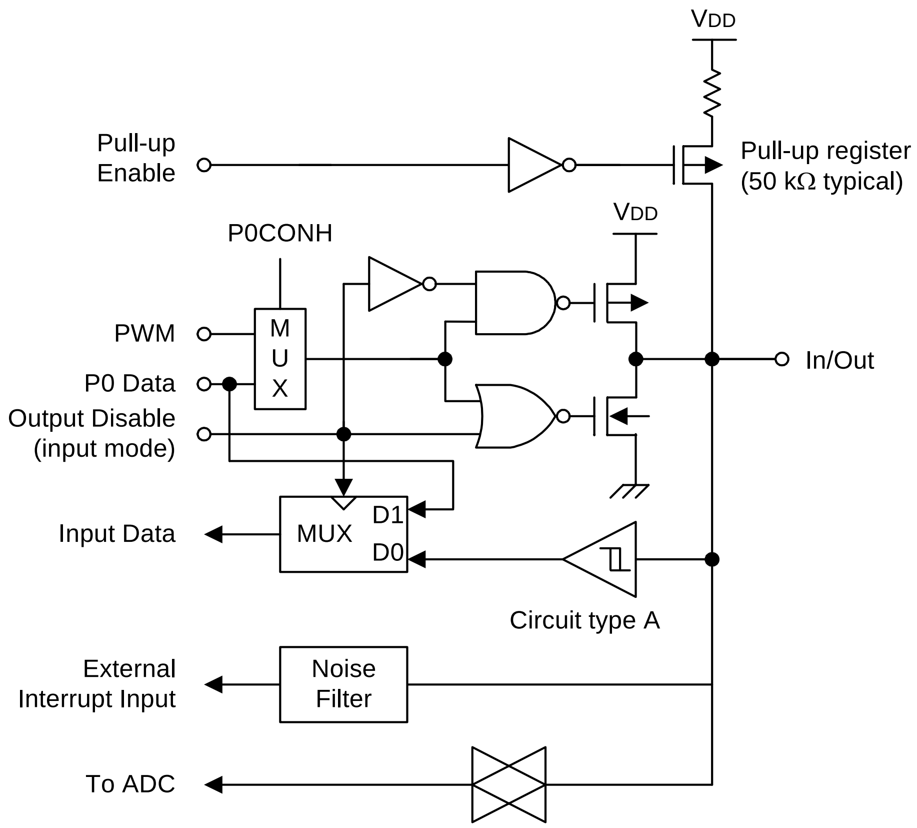
Figure 9-1. Port Data Register Format

PORT 0

Port 0 is a bit-programmable, general-purpose, I/O ports. You can select normal input or push-pull output mode. In addition, you can configure a pull-up resistor to individual pins using control register settings.
It is designed for high-current functions such as LED direct drive. Part 0 pins can also be used as alternative functions (ADC input, external interrupt input and PWM output).

Three control registers are used to control Port 0: P0CONH (E6H), P0CONL (E7H) and P0PND (E8H).

You access port 0 directly by writing or reading the corresponding port data register, P0 (E0H).



NOTE: I/O pins have protection diodes through V_{DD} and V_{SS}.

Mode	Input Data
Output	D0
Input	D1

Figure 9-2. Port 0 Circuit Diagram

Port 0 Control Register (High Byte)
E6H, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] Port, P0.7/ADC7 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = A/D converter input (ADC7); schmitt trigger input off

[.5-.4] Port 0, P0.6/ADC6/PWM Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Alternative function (PWM output)
1 0 = Push-pull output
1 1 = A/D converter input (ADC6); schmitt trigger input off

[.3-.2] Port 0, P0.5/ADC5 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = A/D converter input (ADC5); schmitt trigger input off

[.1-.0] Port 0, P0.4/ADC4 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = A/D converter input (ADC4); schmitt trigger input off

Figure 9-3. Port 0 Control Register (P0CONH, High Byte)

Port 0 Control Register (Low Byte)
E7H, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] Port 0, P0.3/ADC3 Configuration Bits
0 0 = Schmitt trigger input
0 1 = Schmitt trigger input; pull-up enable
1 0 = Push-pull output
1 1 = A/D converter input (ADC3); Schmitt trigger input off

[.5-.4] Port 0, P0.2/ADC2 Configuration Bits
0 0 = Schmitt trigger input
0 1 = Schmitt trigger input; pull-up enable
1 0 = Push-pull output
1 1 = A/D converter input (ADC2); Schmitt trigger input off

[.3-.2] Port 0, P0.1/ADC1/INT1 Configuration Bits
0 0 = Schmitt trigger input/falling edge interrupt input
0 1 = Schmitt trigger input; pull-up enable/falling edge interrupt input
1 0 = Push-pull output
1 1 = A/D converter input (ADC1); Schmitt trigger input off

[.1-.0] Port 0, P0.0/ADC0/INT0 Configuration Bits
0 0 = Schmitt trigger input/falling edge interrupt input
0 1 = Schmitt trigger input; pull-up enable/falling edge interrupt input
1 0 = Push-pull output
1 1 = A/D converter input (ADC0); Schmitt trigger input off

Figure 9-4. Port 0 Control Register (P0CONL, Low Byte)

Port 0 Interrupt Pending Register
E8H, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.4] Not used for S3F84K4

[.3] Port 0.1/ADC1/INT1, Interrupt Enable Bit
0 = INT1 falling edge interrupt disable
1 = INT1 falling edge interrupt enable

[.2] Port 0.1/ADC1/INT1, Interrupt Pending Bit
0 = No interrupt pending (when read)
0 = Pending bit clear (when write)
1 = Interrupt is pending (when read)
1 = No effect (when write)

[.1] Port 0.0/ADC0/INT0, Interrupt Enable Bit
0 = INT0 falling edge interrupt disable
1 = INT0 falling edge interrupt enable

[.0] Port 0.0/ADC0/INT0, Interrupt Pending Bit
0 = No interrupt pending (when read)
0 = Pending bit clear (when write)
1 = Interrupt is pending (when read)
1 = No effect (when write)

Figure 9-5. Port 0 Interrupt Pending Registers (P0PND)

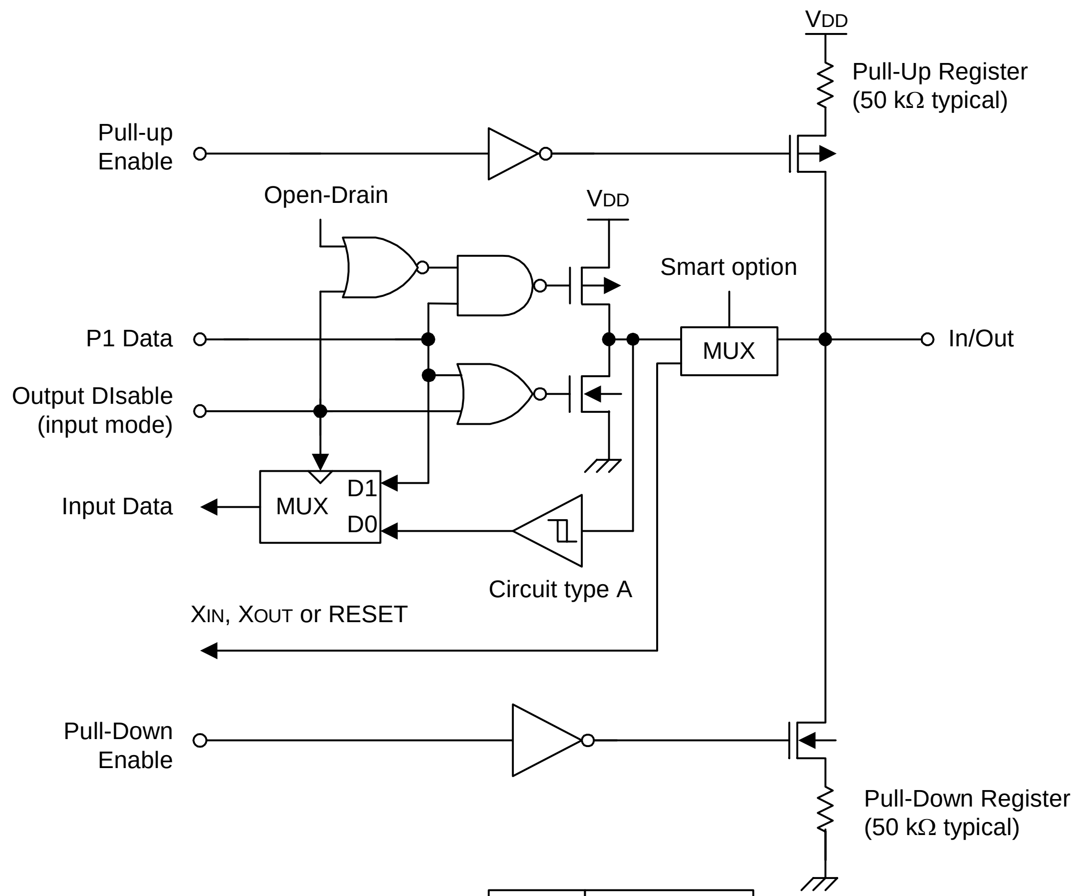
PORT 1

Port 1, is a 2-bit I/O port with individually configurable pins. It can be used for general I/O port (Schmitt trigger input mode, push-pull output mode or n-channel open-drain output mode). In addition, you can configure a pull-up and pull-down resistor to individual pin using control register settings. It is designed for high-current functions such as LED direct drive.

P1.1 is used for oscillator output by smart option. Also, P1.2 is used for RESET pin by smart option.

NOTE: When P1.2 is configured as a general I/O port, it can be used only for Schmitt trigger input.

One control register is used to control port 1: P1CON (E9H).
You address port 1 bits directly by writing or reading the port 1 data register, P1 (E1H). When you use external oscillator, P1.1 must be set to output port to prevent current consumption.



NOTE: I/O pins have protection diodes through VDD and VSS.

Mode	Input Data
Output	D0
Input	D1

Figure 9-6. Port 1 Circuit Diagram

Port 1 Control Register
E9H, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7] Port 1.1 N-Channel Open-Drain Enable Bit
0 = Configure P1.1 as a push-pull output
1 = Configure P1.1 as a n-channel open-drain output

[.6-.4] Not used for S3F84K4

[.3-.2] Port 1, P1.1 Configuration Bits
0 0 = Schmitt trigger input;
0 1 = Schmitt trigger input; pull-up enable
1 0 = Output
1 1 = Schmitt trigger input; pull-down enable

[.1-.0] Not used for S3F84K4

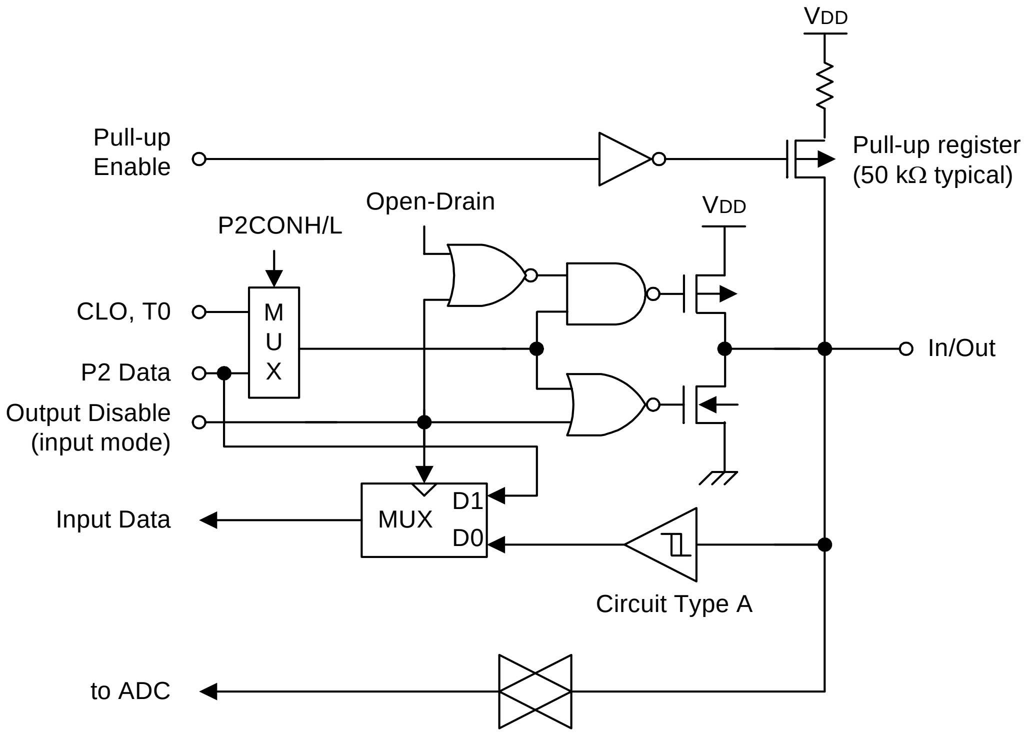
NOTE: When you use external oscillator, P1.1 must be set to output port to prevent current consumption.

Figure 9-7. Port 1 Control Register (P1CON)

PORT 2

Port 2 is a 7-bit I/O port with individually configurable pins. It can be used for general I/O port (Schmitt trigger input mode, push-pull output mode or N-channel open-drain output mode). You can also use some pins of port 2 as ADC input, CLO output and T0 match output. In addition, you can configure a pull-up resistor to individual pins using control register settings. It is designed for high-current functions such as LED direct drive.

You address port 2 bits directly by writing or reading the port 2 data register, P2 (E2H). The port 2 control registers, P2CONH and P2CONL is located at addresses EAH, EBH respectively.



NOTE: I/O pins have protection diodes through VDD and VSS.

Mode	Input Data
Output	D0
Input	D1

Figure 9-8. Port 2 Circuit Diagram

Port 2 Control Register (High Byte)
EAH, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7] Not used for S3F84K4

[.6-.4] Port 2, P2.6/ADC8/CLO Configuration Bits
0 0 0 = Schmitt trigger input; pull-up enable
0 0 1 = Schmitt trigger input
0 1 x = ADC input
1 0 0 = Push-pull output
1 0 1 = Open-drain output; pull-up enable
1 1 0 = Open-drain output
1 1 1 = Alternative function; CLO output

[.3-.2] Port 2, P2.5 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = Open-drain output

[.1-.0] Port 2, P2.4 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = Open-drain output

NOTE: When noise problem is important issue, you had better not use CLO output

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

Port 2 Control Register (Low Byte)
EBH, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] Port 2, P2.3 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = Open-drain output

[.5-.4] Port 2, P2.2 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = Open-drain output

[.3-.2] Port 2, P2.1 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = Open-drain output

[.1-.0] Port 2, P2.0 Configuration Bits
0 0 = Schmitt trigger input; pull-up enable
0 1 = Schmitt trigger input
1 0 = Push-pull output
1 1 = T0 match output

Figure 9-10. Port 2 Control Register (P2CONL, Low Byte)

10 BASIC TIMER and TIMER 0

MODULE OVERVIEW

The S3F84K4 has two default timers: an 8-bit basic timer, and a 16-bit general-purpose timer, called timer 0.

Basic Timer (BT)

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

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

— To signal the end of the required oscillation stabilization interval after a Reset or a Stop mode release.

The functional components of the basic timer block are:

— Clock frequency divider (f_{OSC} divided by 4096, 1024, or 128) with multiplexer

— 8-bit basic timer counter, BTCNT (FDH, read-only)

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

Timer 0

The 16-bit timer 0 is used in one 16-bit timer or two 8-bit timers mode. When TACON.7 is set to "1", it is in one 16-bit timer mode. When TACON.7 is set to "0", the timer 0 is used as two 8-bit timers.

— One 16-bit timer mode (Timer 0)

— Two 8-bit timers mode (Timer A and B)

BASIC TIMER (BT)

BASIC TIMER CONTROL REGISTER (BTCON)

The basic timer control register, BTCON, is used to select the input clock frequency, to clear the basic timer counter and frequency dividers, and to enable or disable the watchdog timer function.

A Reset clears BTCON to "00H". This enables the watchdog function and selects a basic timer clock frequency of f_{OSC}/4096. To disable the watchdog function, you must write the signature code "1010B" to the basic timer register control bits BTCON.7–BTCON.4.

The 8-bit basic timer counter, BTCNT, can be cleared during normal operation by writing a "1" to BTCON.1. To clear the frequency dividers for both the basic timer input clock and the timer 0 clock, you write a "1" to BTCON.0.

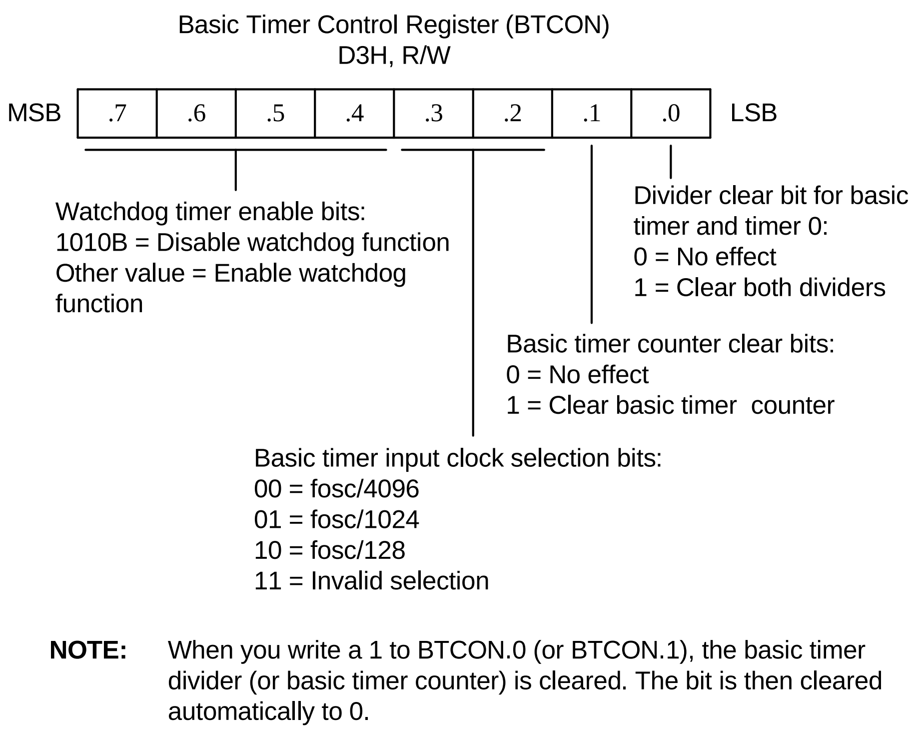


Figure 10-1. Basic Timer Control Register (BTCON)

BASIC TIMER FUNCTION DESCRIPTION

Watchdog Timer Function

You can program the basic timer overflow signal (BTOVF) to generate a Reset by setting BTCON.7–BTCON.4 to any value other than "1010B" (The "1010B" value disables the watchdog function). A Reset clears BTCON to "00H", automatically enabling the watchdog timer function. A Reset also selects the oscillator clock divided by 4096 as the BT clock.

A Reset whenever a basic timer counter overflow occurs. During normal operation, the application program must prevent the overflow, and the accompanying reset operation, from occurring. To do this, the BTCNT value must be cleared (by writing a "1" to BTCON.1) at regular intervals.

If a system malfunction occurs due to circuit noise or some other error condition, the BT counter clear operation will not be executed and a basic timer overflow will occur, initiating a Reset. In other words, during normal operation, the basic timer overflow loop (a bit 7 overflow of the 8-bit basic timer counter, BTCNT) is always broken by a BTCNT clear instruction. If a malfunction does occur, a Reset is triggered automatically.

Oscillation Stabilization Interval Timer Function

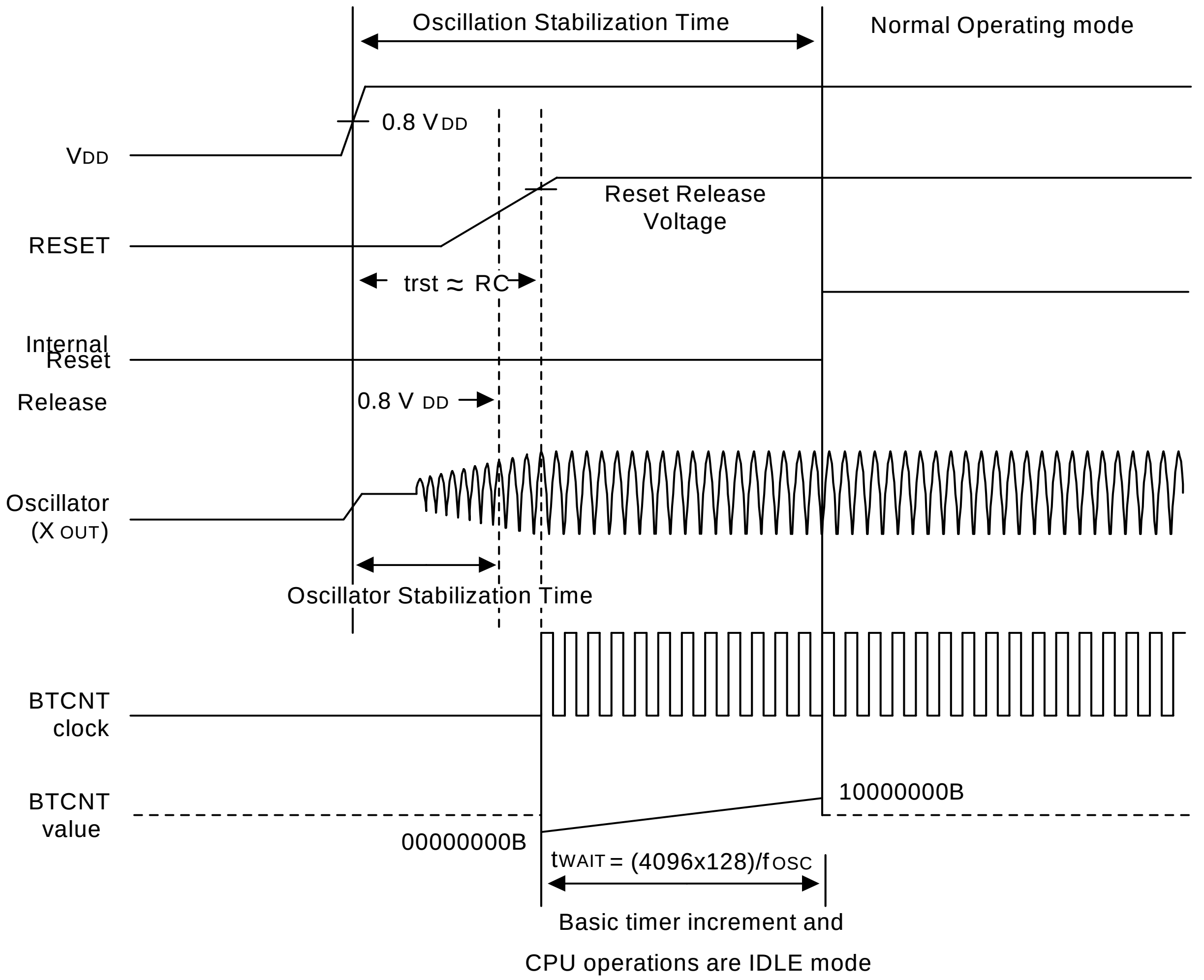
You can also use the basic timer to program a specific oscillation stabilization interval following a Reset or when Stop mode has been released by an external interrupt.

In Stop mode, whenever a Reset or an external interrupt occurs, the oscillator starts. The BTCNT value then starts increasing at the rate of f_{OSC}/4096 (for Reset), or at the rate of the preset clock source (for an external interrupt). When BTCNT.7 is set, a signal is generated to indicate that the stabilization interval has elapsed and to gate the clock signal off to the CPU so that it can resume normal operation.

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

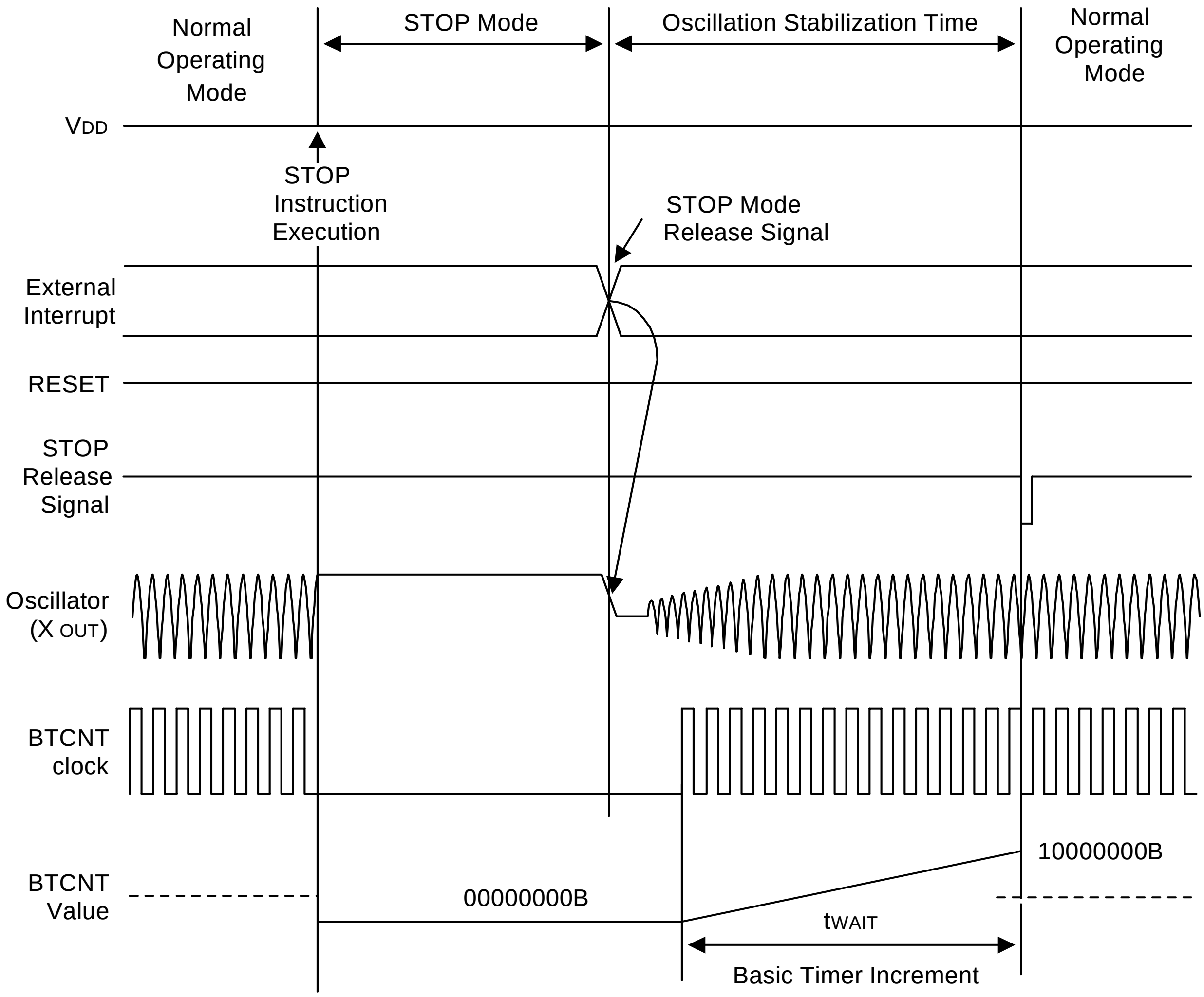
1. During Stop mode, an external power-on Reset or an external interrupt occurs to trigger the Stop mode release and oscillation starts.
2. If an external power-on Reset occurred, the basic timer counter will increase at the rate of f_{OSC}/4096. If an external interrupt is used to release Stop mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until bit 7 of the basic timer counter is set.
4. When a BTCNT.7 is set, normal CPU operation resumes.

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



NOTE: Duration of the oscillator stabilization wait time, t_{WAIT}, when it is released by a Power-on-reset is 4096 x 128/f_{OSC}.
$t_{RST} \approx$ RC (R and C are value of external power on Reset)

Figure 10-2. Oscillation Stabilization Time on RESET



NOTE: Duration of the oscillator stabilzation wait time, 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 x 128)/fosc	65.5 ms
0	1	(1024 x 128)/fosc	16.4 ms
1	0	(128 x 128)/fosc	2.04 ms
1	1	Invalid setting	—

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

☞ PROGRAMMING TIP — Configuring the Basic Timer

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

```
        ORG     0000H

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

        ORG     003CH
        DB      0FFH                    ;  003CH, must be initialized to 1
        DB      0FFH                    ;  003DH, must be initialized to 1
        DB      0FFH                    ;  003EH, enable LVR (3.0v)
        DB      0FEH                    ;  003FH, External RC oscillator

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

        ORG     0100H

RESET:  DI                              ;  Disable interrupt
        LD      CLKCON,#00011000B       ;  Select non-divided CPU clock
        LD      SP,#0C0H                ;  Stack pointer must be set
        •
        •
        LD      BTCON,#02H              ;  Enable watchdog function
                                        ;  Basic timer clock: fOSC/4096
                                        ;  Basic counter (BTCNT) clear
        •
        •
        •
        EI                              ;  Enable interrupt

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

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

ONE 16-BIT TIMER MODE (TIMER 0)

The 16-bit timer 0 is used in one 16-bit timer or two 8-bit timers mode. When TACON.7 is set to "1", it is in one 16-bit timer mode. When TACON.7 is set to "0", the timer 0 is used as two 8-bit timers.

— One 16-bit timer mode (Timer 0)

— Two 8-bit timers mode (Timer A and B)

Overview

The 16-bit timer 0 is a 16-bit general-purpose timer. Timer 0 includes interval timer mode using appropriate TACON setting.

Timer 0 has the following functional components:

— Clock frequency divider (fxx divided by 256, 64, 8, or 1) with multiplexer

— 16-bit counter (TACNT, TBCNT), 16-bit comparator, and 16-bit reference data register (TADATA, TBDATA)

— Timer 0 match interrupt (IRQ1, vector F6H) generation

— Timer 0 control register, TACON (D2H, read/write)

Function Description

Interval Timer Function

The timer 0 module can generate an interrupt, the timer 0 match interrupt (T0INT). T0INT belongs to the interrupt level IRQ1, and is assigned a separate vector address, F6H.

The T0INT pending condition should be cleared by software after IRQ1 is serviced. The T0INT pending bit must be cleared by the application sub-routine by writing a "0" to the TACON.0 pending bit.

In interval timer mode, a match signal is generated when the counter value is identical to the values written to the T0 reference data registers, TADATA and TBDATA. The match signal generates a timer 0 match interrupt (T1INT, vector F6H) and clears the counter.

If, for example, you write the value 10H and 32H to TADATA and TBDATA, respectively, and 8EH to TACON, the counter will increment until it reaches 3210H. At this point, the T0 interrupt request is generated, the counter value is reset, and counting resumes.

Timer 0 Control Register (TACON)

You use the timer 0 control register, TACON, to

— Enable the timer 0 operating (interval timer)

— Select the timer 0 input clock frequency

— Clear the timer 0 counter, TACNT and TBCNT

— Enable the timer 0 interrupt

— Clear timer 0 interrupt pending condition

TACON is located at address D0H, and is read/write addressable using register addressing mode.

A reset clears TACON to "00H". This sets timer 0 to disable interval timer mode, selects an input clock frequency of fxx/256, and disables timer 0 interrupt. You can clear the timer 0 counter at any time during the normal operation by writing a "1" to TACON.3.

To enable the timer 0 interrupt (IRQ1, vector F6H), you must write TACON.7, TACON.2, and TACON.1 to "1". To generate the exact time interval, you should set TACON.3 and TACON.0 to “10B”, which clear counter and interrupt pending bit. When the T0INT sub-routine is serviced, the pending condition must be cleared by software by writing a "0" to the timer 0 interrupt pending bit, TACON.0.

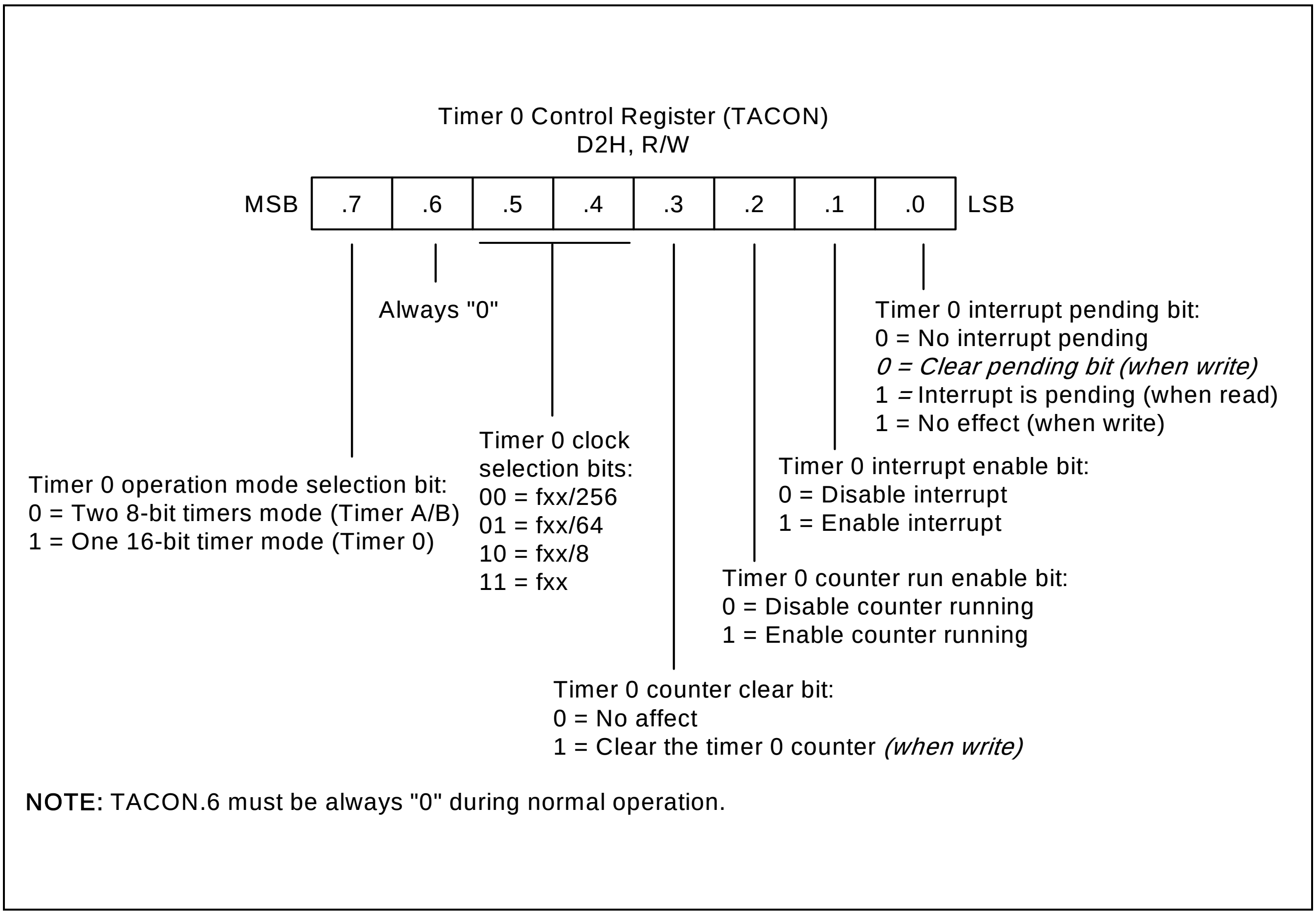


Figure 10-4. Timer 0 Control Register (TACON)

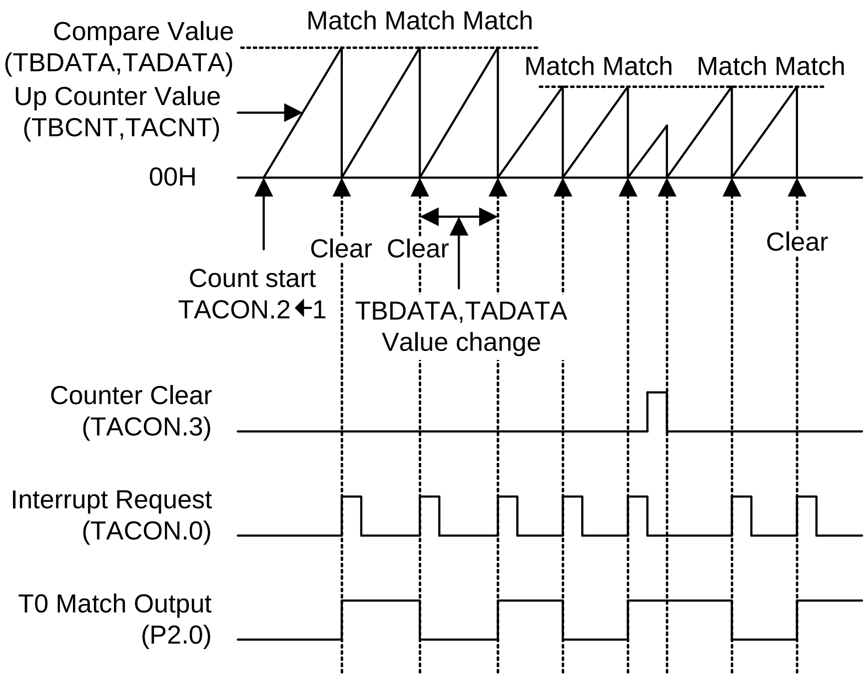
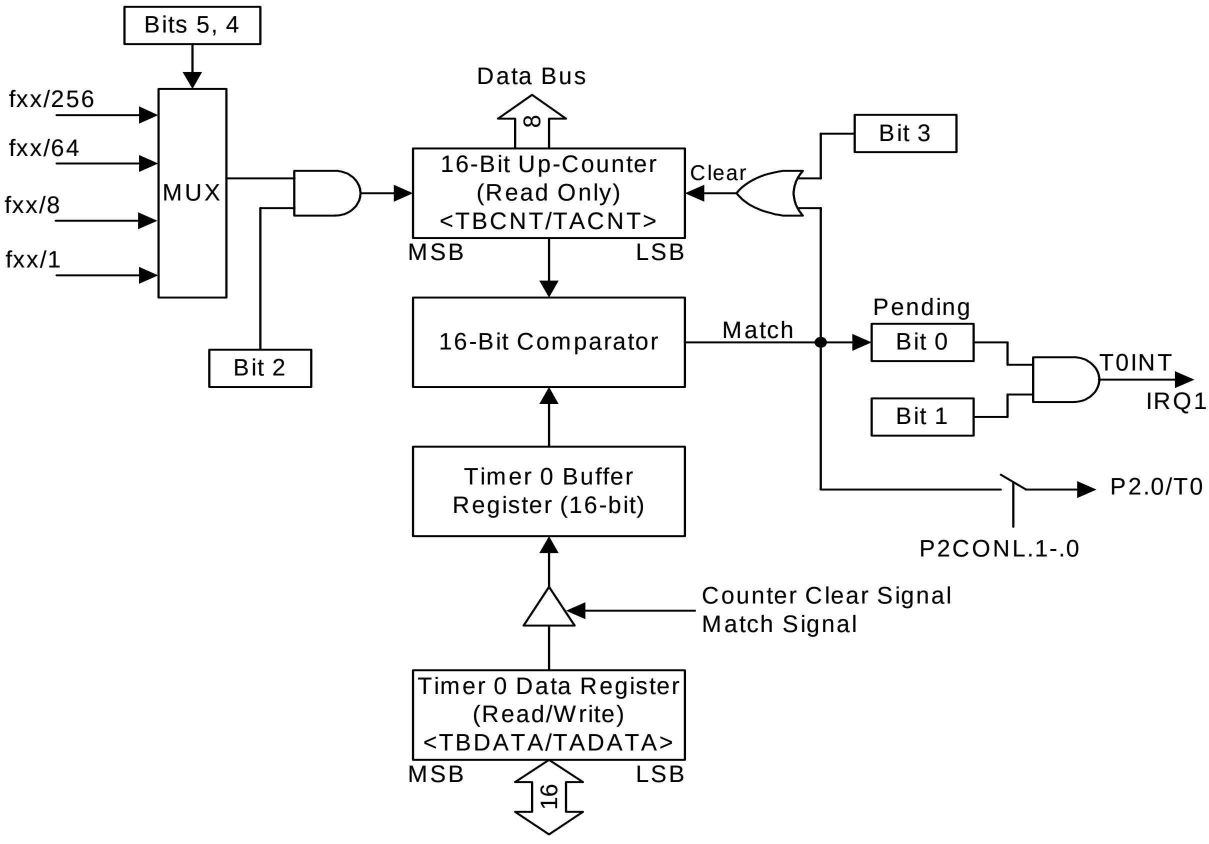


Figure 10-5. Timer 0 Timing Diagram

BLOCK DIAGRAM



NOTE: When TACON.7 is "1", 16-bit timer 0.

Figure 10-6. Timer 0 Functional Block Diagram

TWO 8-BIT TIMERS MODE (TIMER A and B)

OVERVIEW

The 8-bit timer A and B are the 8-bit general-purpose timers. Timer A and B support interval timer mode using appropriate TACON and TBCON setting, respectively.

Timer A and B have the following functional components:

- Clock frequency divider with multiplexer
  - fxx divided by 256, 64, 8, or 1 for timer A
  - fxx divided by 256, 64, 8, or 1 for timer B
- 8-bit counter (TACNT, TBCNT), 8-bit comparator, and 8-bit reference data register (TADATA, TBDATA)
- Timer A match interrupt (IRQ1, vector F6H) generation
- Timer A control register, TACON (D2H, read/write)
- Timer B match interrupt (IRQ1, vector F4H) generation
- Timer B control register, TBCON (EEH, read/write)

Function Description

Interval Timer Function

The timer A and B module can generate an interrupt: the timer A match interrupt (TAINT) and the timer B match interrupt (TBINT). TAINT belongs to the interrupt level IRQ1, and is assigned a separate vector address, F6H. TBINT belongs to the interrupt level IRQ1 and is assigned a separate vector address, F4H.

The TAINT and TBINT pending condition should be cleared by software after they are serviced.

In interval timer mode, a match signal is generated when the counter value is identical to the values written to the TA or TB reference data registers, TADATA or TBDATA. The match signal generates corresponding match interrupt (TAINT, vector F6H; TBINT, vector F4H) and clears the counter.

If, for example, you write the value 10H to TBDATA, "0" to TACON.7, and 0EH to TBCON, the counter will increment until it reaches 10H. At this point, the TB interrupt request is generated, the counter value is reset, and counting resumes.

Timer A and B Control Register (TACON, TBCON)

You use the timer A and B control register, TACON and TBCON, to

- Enable the timer A and B operating (interval timer)
- Select the timer A and B input clock frequency
- Clear the timer A and B counter, TACNT and TBCNT
- Enable the timer A and B interrupts
- Clear timer A and B interrupt pending conditions

TACON and TBCON are located at address D2H and EEH, and is read/write addressable using register addressing mode.

A reset clears TACON and TBCON to "00H". This sets timer A and B to disable interval timer mode, selects an input clock frequency of fxx/256, and disables timer A and B interrupt. You can clear the timer A and B counter at any time during normal operation by writing a "1" to TACON.3 and TBCON.3.

To enable the timer A and B interrupt (IRQ1, vector F6H, F4H), you must write TACON.7 to "0", TACON.2 (TBCON.2) and TACON.1 (TBCON.1) to "1". To generate the exact time interval, you should set TACON.3 (TBCON.3) and TACON.0 (TBCON.0) to “10B”, which clear counter and interrupt pending bit, respectively. When the TAINT or TBINT sub-routine is serviced, the pending condition must be cleared by software by writing a "0" to the timer A or B interrupt pending bits, TACON.0 or TBCON.0.

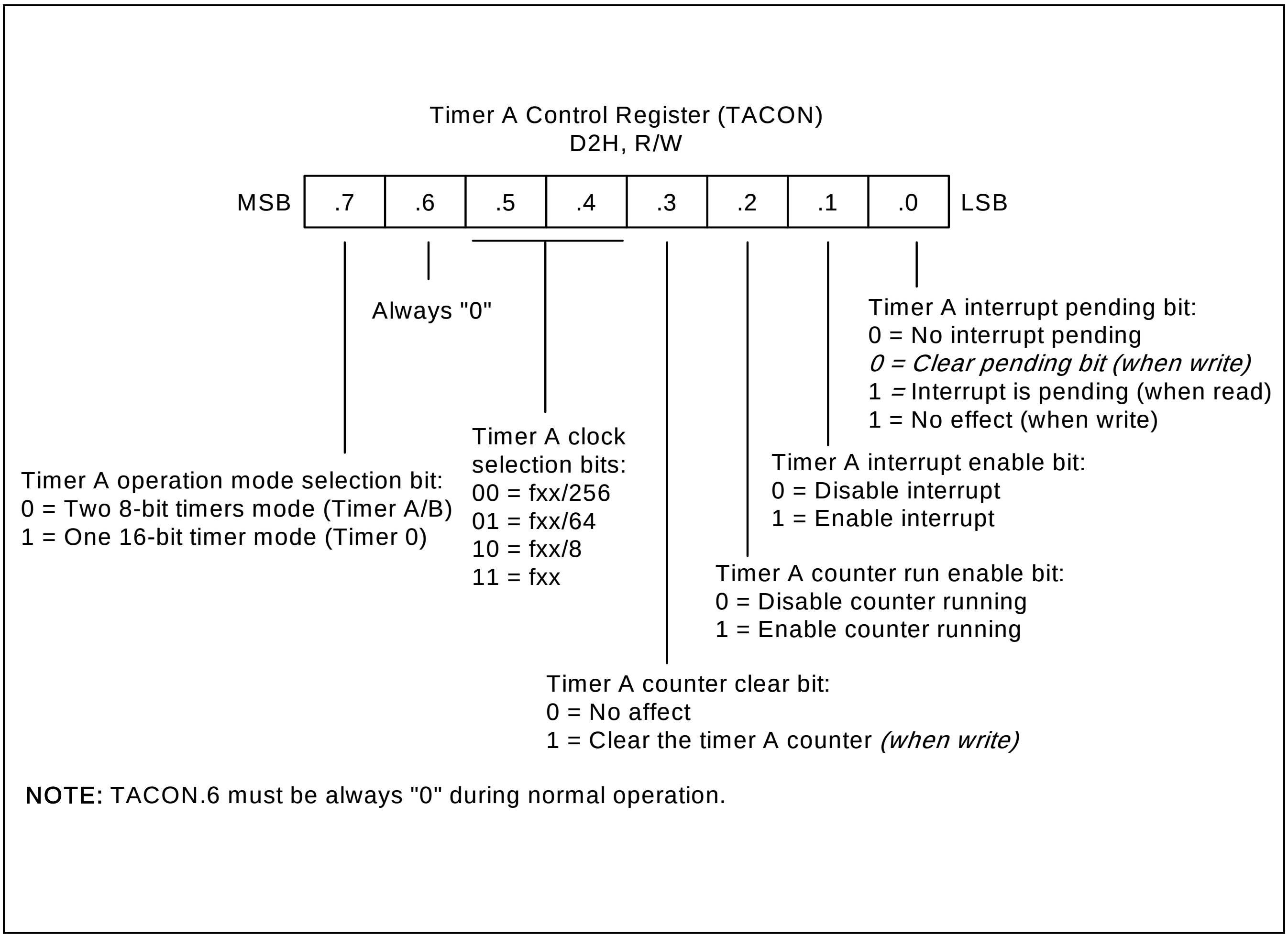


Figure 10-7. Timer A Control Register (TACON)

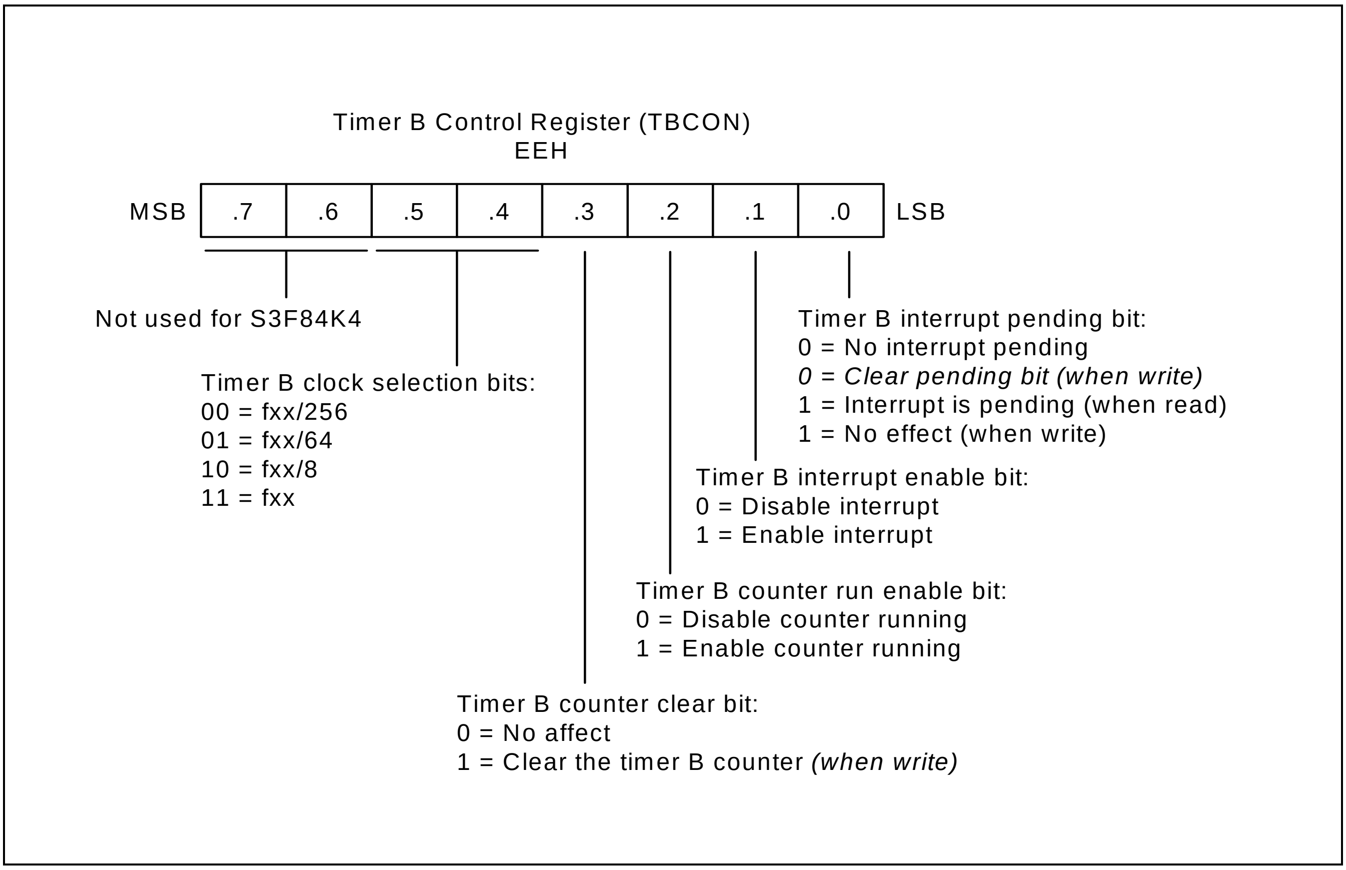


Figure 10-8. Timer B Control Register (TBCON)

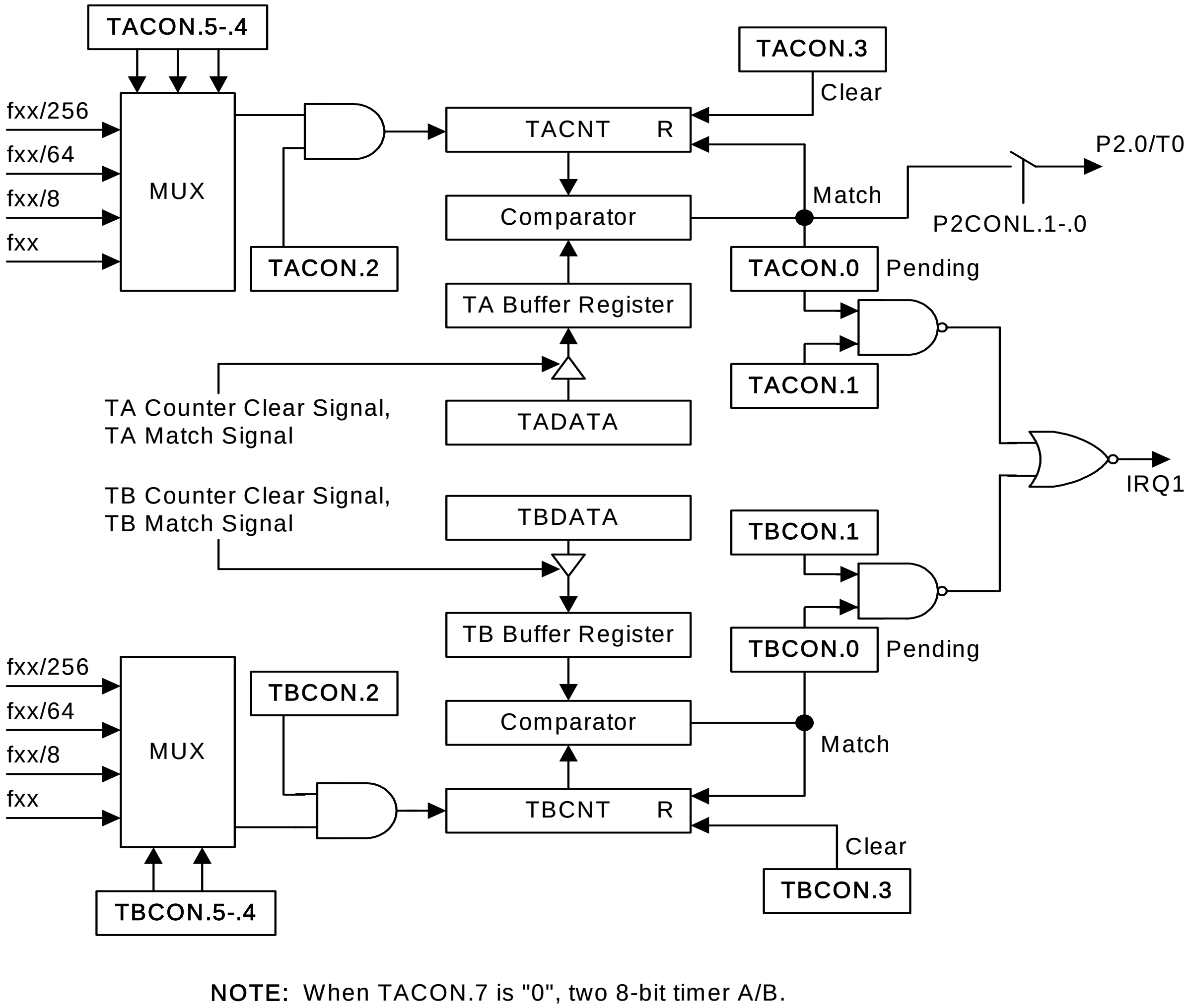


Figure 10-9. Timer A and B Function Block Diagram

11 12-BIT PWM (PULSE WIDTH MODULATION)

OVERVIEW

This microcontroller has the 12-bit PWM circuit. The operation of all PWM circuit is controlled by a single control register, PWMCON.

The PWM counter is a 12-bit incrementing counter. It is used by the 12-bit PWM circuits. To start the counter and enable the PWM circuits, you set PWMCON.2 to "1". If the counter is stopped, it retains its current count value; when re-started, it resumes counting from the retained count value. When there is a need to clear the counter you set PWMCON.3 to "1".

You can select a clock for the PWM counter by set PWMCON.6-.7. Clocks which you can select are Fosc/256, Fosc/64, Fosc/8, Fosc/1.

FUNCTION DESCRIPTION

PWM

The 12-bit PWM circuits have the following components:

- 6-bit comparator and extension cycle circuit
- 6-bit reference data registers (PWMDATA)
- 6-bit extension data registers (PWMEX)
- PWM output pins (P0.6/PWM)

PWM counter

The PWM counter is a 12-bit incrementing counter comprised of a lower 6-bit counter and an upper 6-bit counter.

To determine the PWM module's base operating frequency, the lower byte counter is compared to the PWM data register value. In order to achieve higher resolutions, the six bits of the upper counter can be used to modulate the "stretch" cycle. To control the "stretching" of the PWM output duty cycle at specific intervals, the 6-bit extended counter value is compared with the 6-bit value (bits 7-2) that you write to the module's extension register.

PWM Data and Extension Registers

PWM (duty) data registers, located in F1H and F2H, determine the output value generated by the 12-bit PWM circuit.

— 8-bit data register PWMDATA (F2H), of which only bits 5-0 are used.

— 8-bit extension registers PWMEX (F1H), of which only bits 7-2 are used

To program the required PWM output, you load the appropriate initialization values into the 6-bit data registers (PWMDATA) and the 6-bit extension registers (PWMEX). To start the PWM counter, or to resume counting, you set PWMCON.2 to "1".

A reset operation disables all PWM output. The current counter value is retained when the counter stops. When the counter starts, counting resumes at the retained value.

PWM Clock Rate

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

Table 11-1. PWM Control and Data Registers

Register Name	Mnemonic	Address	Function
PWM data registers	PWMDATA	F2H	6-bit PWM basic cycle frame value
	PWMEX	F1H	6-bit extension ("stretch") value
PWM control registers	PWMCON	F3H	PWM counter stop/start (resume), and Fosc clock settings

PWM Function Description

The PWM output signal toggles to Low level whenever the lower 6-bit counter matches the reference value stored in the module's data register (PWMDATA). If the value in the PWMDATA register is not zero, an overflow of the lower counter causes the PWM output to toggle to High level. In this way, the reference value written to the data register determines the module's base duty cycle.

The value in the 6-bit extension counter is compared with the extension settings in the 6-bit extension data register (PWMEX). This 6-bit extension counter value, together with extension logic and the PWM module's extension register, is then used to "stretch" the duty cycle of the PWM output. The "stretch" value is one extra clock period at specific intervals, or cycles (see Table 16-2).

If, for example, the value in the extension register is '04H', the 32nd cycle will be one pulse longer than the other 63 cycles. If the base duty cycle is 50 %, the duty of the 32nd cycle will therefore be "stretched" to approximately 51% duty. For example, if you write 80H to the extension register, all odd-numbered pulses will be one cycle longer. If you write FCH to the extension register, all pulses will be stretched by one cycle except the 64th pulse. PWM output goes to an output buffer and then to the corresponding PWM output pin. In this way, you can obtain high output resolution at high frequencies.

Table 11-2. PWM output "stretch" Values for Extension Registers PWMEX

PWMEX Bit	"Stretched" Cycle Number
7	1, 3, 5, 7, 9, . . . , 55, 57, 59, 61, 63
6	2, 6, 10, 14, . . . , 50, 54, 58, 62
5	4, 12, 20, . . . , 44, 52, 60
4	8, 24, 40, 56
3	16, 48
2	32
1	Not used
0	Not used

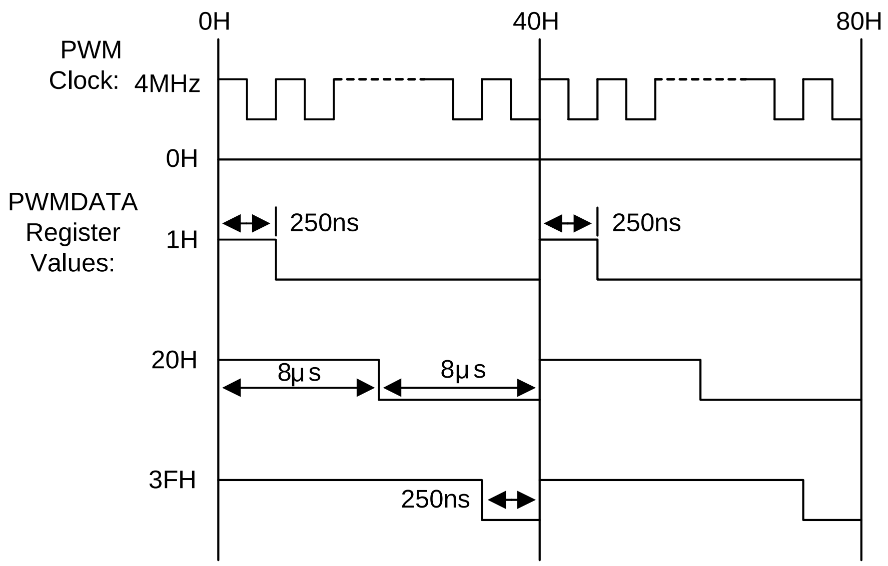


Figure 11-1. 12-Bit PWM Basic Waveform

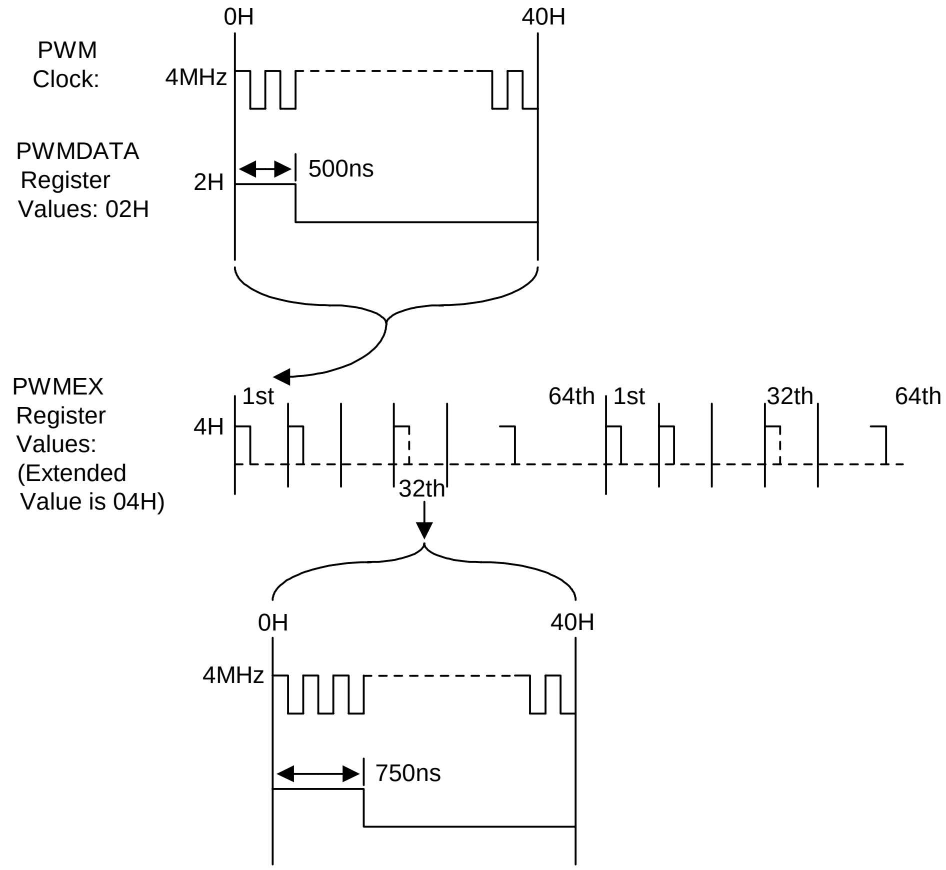


Figure 11-2. 12-Bit Extended PWM Waveform

PWM CONTROL REGISTER (PWMCON)

The control register for the PWM module, PWMCON, is located at register address F3H. PWMCON is used the 12-bit PWM modules. Bit settings in the PWMCON register control the following functions:

— PWM counter clock selection
— PWM data reload interval selection
— PWM counter clear
— PWM counter stop/start (or resume) operation
— PWM counter overflow (upper 6-bit counter overflow) interrupt control

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

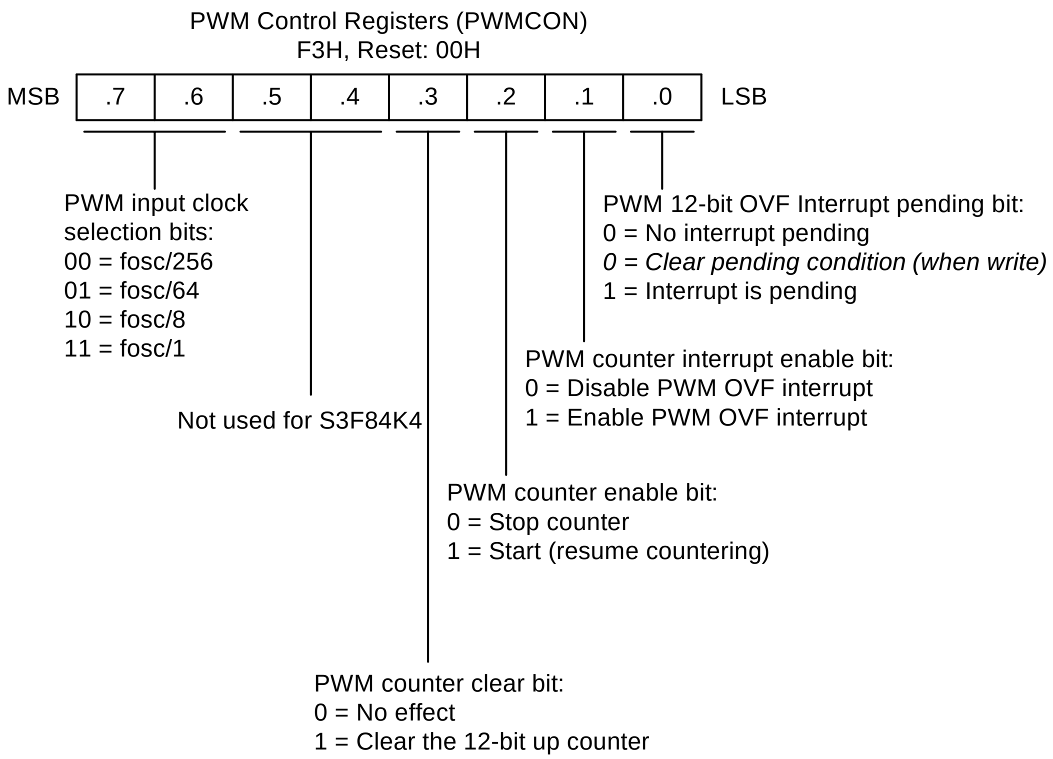


Figure 11-3. PWM/Capture Module Control Register (PWMCON)

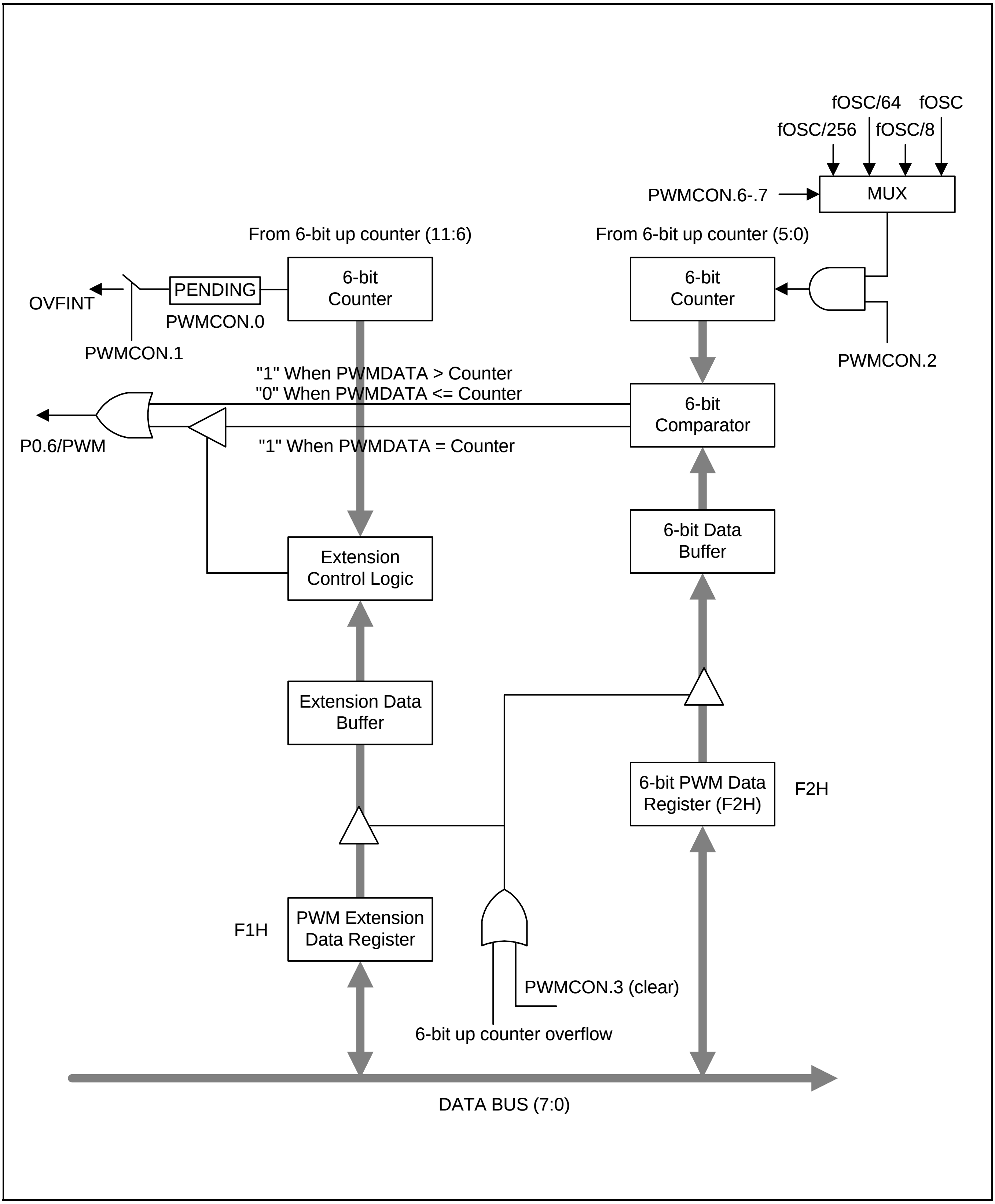


Figure 11-4. PWM/Capture Module Functional Block Diagram

☞ PROGRAMMING TIP — Programming the PWM Module to Sample Specifications

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

ORG         0000H
            VECTOR    0F2H,INT_PWM          ;  S3F84K4 PWM interrupt vector
;--------------<< Smart Option >>

ORG         003CH
            DB        0FFH                  ;  003CH, must be initialized to 1.
            DB        0FFH                  ;  003DH, must be initialized to 1.
            DB        0FFH                  ;  003EH, enable LVR (3.0)
            DB        0FEH                  ;  003FH, External RC oscillator

;--------------<< Initialize System and Peripherals >>
              ORG     0100H
RESET:        DI                            ;  disable interrupt
              LD      BTCON,#10100011B      ;  Watchdog disable
              •
              •
              LD      P0CONH,#10011010B     ;  Configure P0.6 PWM output
              LD      PWMCON,#01000110B     ;  fOSC/64, counter/interrupt enable
              LD      PWMEX,#00H
              LD      PWMDATA,#80H          ;
              •
              •
              EI                            ;  Enable interrupt

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

MAIN:                                       ;
              •                             ;
              •                             ;
              •                             ;
              •                             ;
              JR      t,MAIN                ;

INT_PWM:                                    ;  PWM interrupt service routine
              •
              •
              •
              AND     PWMCON,#11111110B     ;  pending bit clear
              IRET                          ;
              •
              •
              END
```

12 A/D CONVERTER

OVERVIEW

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

— Analog comparator with successive approximation logic

— D/A converter logic

— ADC control register (ADCON)

— Nine multiplexed analog data input pins (ADC0–ADC8)

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

To initiate an analog-to-digital conversion procedure, you write the channel selection data in the A/D converter control register ADCON to select one of the nine analog input pins (ADCn, n = 0–8) and set the conversion start or enable bit, ADCON.0. The read-write ADCON register is located at address F7H.

During a normal conversion, ADC logic initially sets the successive approximation register to 200H (the approximate half-way point of an 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.7–4) in the ADCON register. To start the A/D conversion, you should set a the enable bit, ADCON.0. When a conversion is completed, ACON.3, the end-of-conversion (EOC) bit is automatically set to 1 and the result is dumped into the ADDATA register where it can be read. The A/D converter then enters an idle state. Remember to read the contents of ADDATA before another conversion starts. Otherwise, the previous result will be overwritten by the next conversion result.

NOTE

Because the ADC does not use sample-and-hold circuitry, it is important that any fluctuations in the analog level at the ADC0–ADC8 input pins during a conversion procedure be kept to an absolute minimum. Any change in the input level, perhaps due to circuit noise, will invalidate the result.

USING A/D PINS FOR STANDARD DIGITAL INPUT

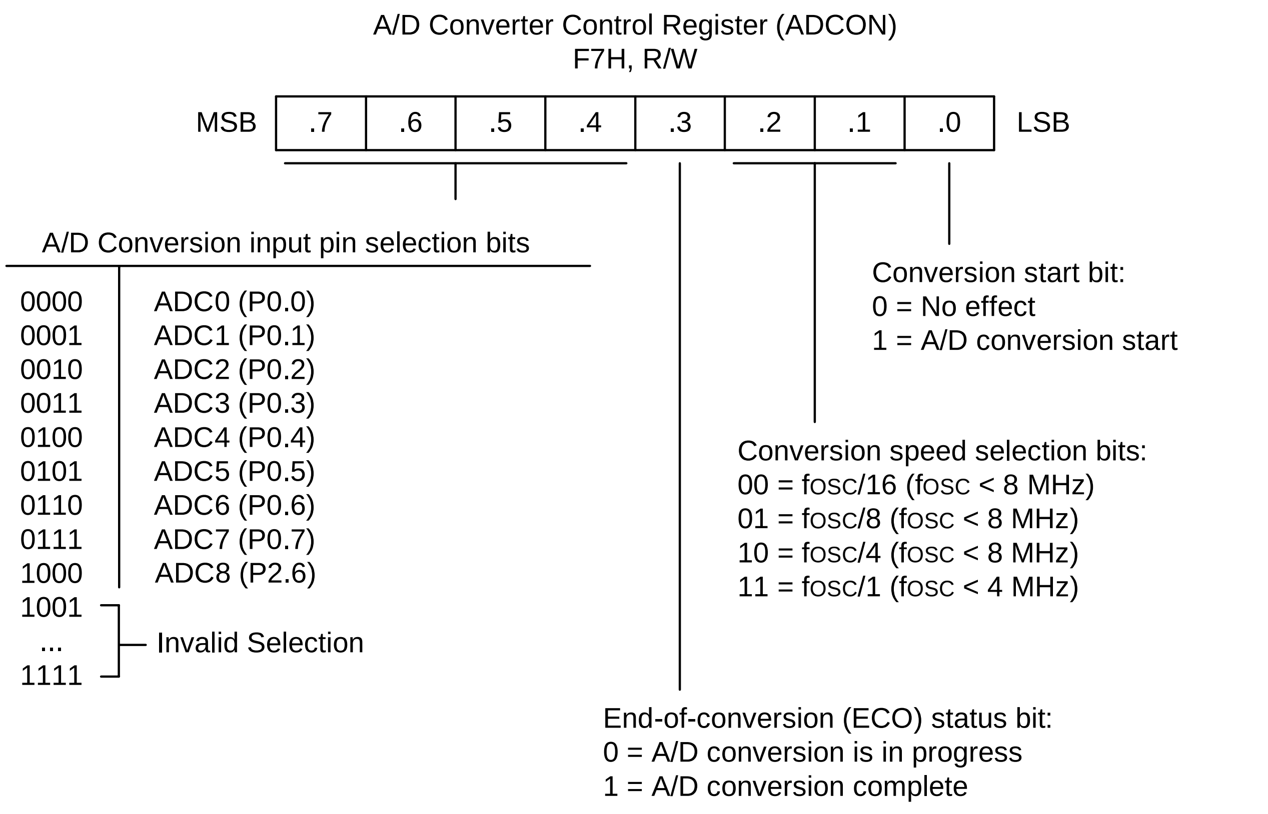
The ADC module's input pins are alternatively used as digital input in port 0 and P2.6.

A/D CONVERTER CONTROL REGISTER (ADCON)

The A/D converter control register, ADCON, is located at address F7H. ADCON has four functions:

- Bits 7-4 select an analog input pin (ADC0–ADC8).
- Bit 3 indicates the status of the A/D conversion.
- Bits 2-1 select a conversion speed.
- Bit 0 starts the A/D conversion.

Only one analog input channel can be selected at a time. You can dynamically select any one of the nine analog input pins (ADC0–ADC8) by manipulating the 4-bit value for ADCON.7–ADCON.4.



NOTES:
1. Maximum ADC clock input = 4 MHz
2. Do NOT set ADCON.7– 4 to any value of the Invalid Selection.

Figure 12-1. A/D Converter Control Register (ADCON)

INTERNAL REFERENCE VOLTAGE LEVELS

In the ADC function block, the analog input voltage level is compared to the reference voltage. The analog input level must remain within the range V_{SS} to V_{DD}.

Different reference voltage levels are generated internally along the resistor tree during the analog conversion process for each conversion step. The reference voltage level for the first bit conversion is always 1/2 V_{DD}.

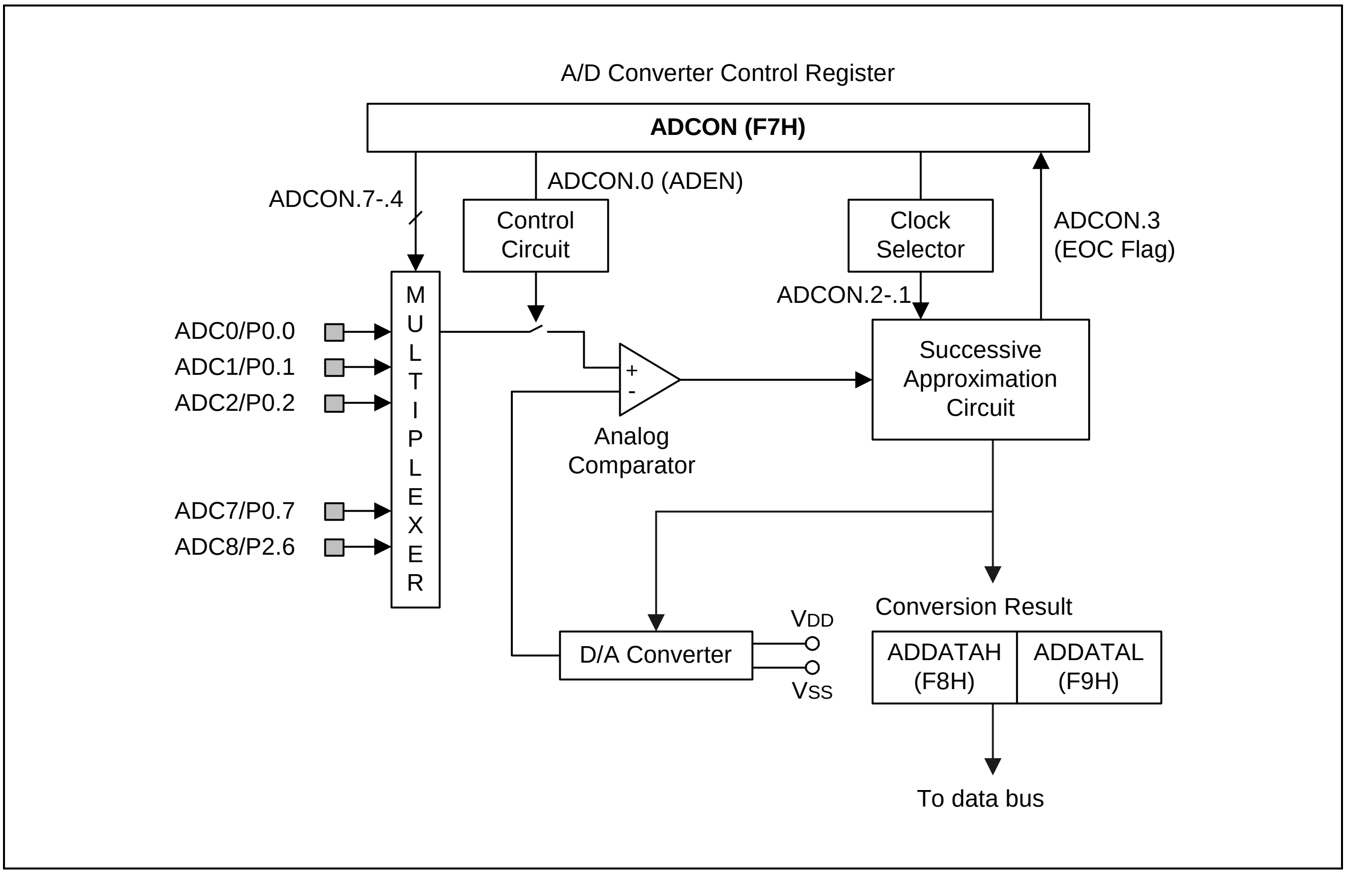


Figure 12-2. A/D Converter Circuit Diagram

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

Figure 12-3. A/D Converter Data Register (ADDATAH/L)

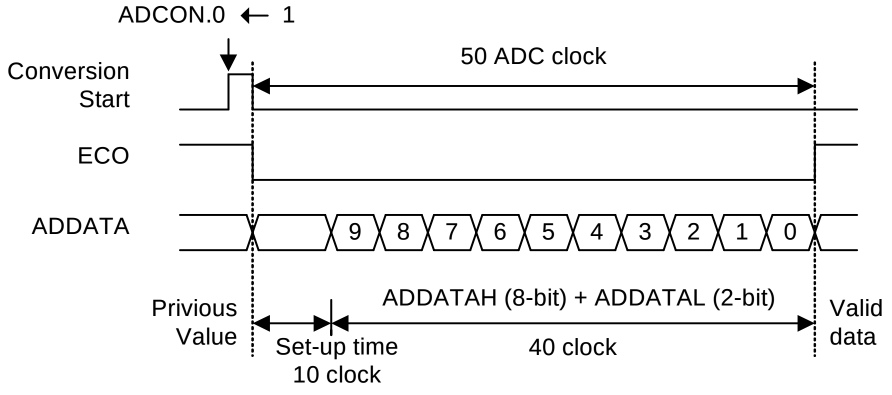


Figure 12-4. A/D Converter Timing Diagram

CONVERSION TIMING

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

4 clocks/bit $\times$ 10-bits + step-up time (10 clock) = 50 clocks
50 clock $\times$ 500 ns = 25 μs at 8 MHz, 1 clock time = 4/f_{OSC} (assuming ADCON.2–.1 = 10)

INTERNAL A/D CONVERSION PROCEDURE

1. Analog input must remain between the voltage range of V_{SS} and $V_{DD.}$
2. Configure the analog input pins to input mode by making the appropriate settings in P0CONH, P0CONL and P2CONH registers.
3. Before the conversion operation starts, you must first select one of the nine input pins (ADC0–ADC8) by writing the appropriate value to the ADCON register.
4. When conversion has been completed, (50 clocks have elapsed), the EOC flag is set to “1”, so that a 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), then the ADC module enters an idle state.
6. The digital conversion result can now be read from the ADDATAH and ADDATAL register.

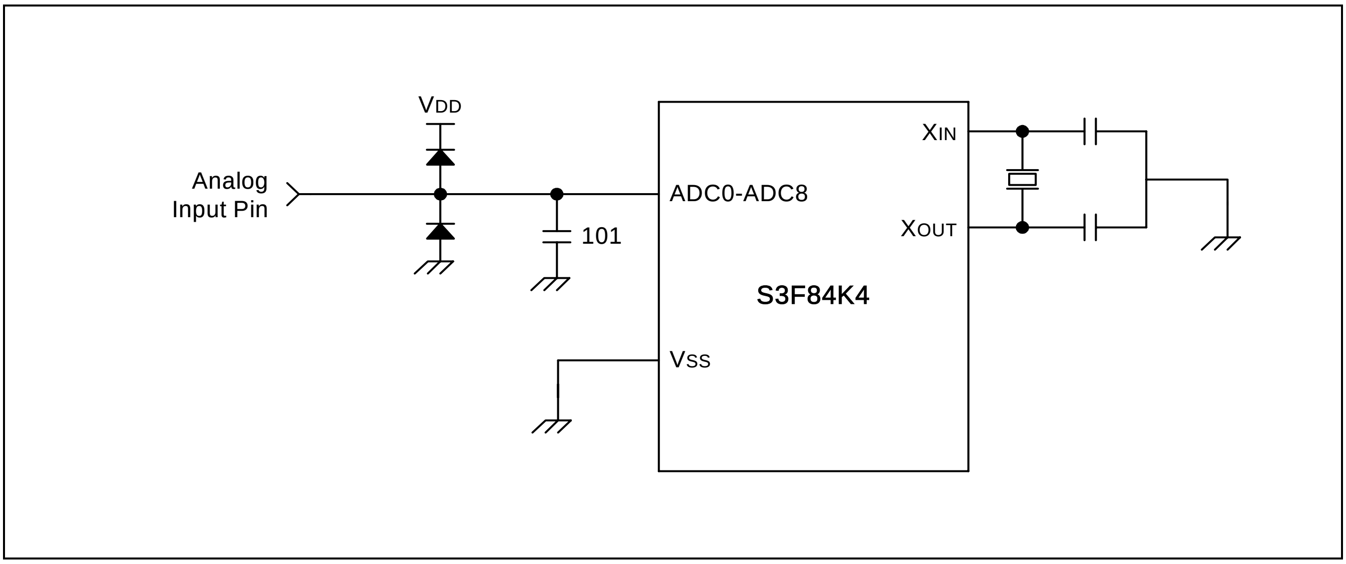


Figure 12-5. Recommended A/D Converter Circuit for Highest Absolute Accuracy

☞ PROGRAMMING TIP – Configuring A/D Converter

```
            ORG     0000H
            VECTOR  0F6H,INT_TIMER0         ;  Timer 0 interrupt vector

ORG         003CH
            DB      0FFH                    ;  003CH, must be initialized to 0
            DB      0FFH                    ;  003DH, must be initialized to 0
            DB      0FFH                    ;  003EH, enable LVR (3.0v)
            DB      0FEH                    ;  003FH, external RC oscillator

            ORG     0100H
RESET:      DI                              ;  disable interrupt
            LD      BTCON,#10100011B        ;  Watchdog disable
            •
            •
            •
            LD      P0CONH,#11111111B       ;  Configure P0.4–P0.7 AD input
            LD      P0CONL,#11111111B       ;  Configure P0.0–P0.3 AD input
            LD      P2CONH,#00100000B       ;  Configure P2.6 AD input
            EI                              ;  Enable interrupt

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

MAIN:       •
            •
            •
            CALL    AD_CONV                 ;  Subroutine for AD conversion
            •
            •
            •
            JR      t,MAIN                  ;

AD_CONV:    LD      ADCON,#00000001B        ;  Select analog input channel → P0.0
                                            ;  select conversion speed → fOSC/16
                                            ;  set conversion start bit

            NOP
            NOP                             ;  If you select conversion speed to fOSC/16
            NOP                             ;  at least three nop must be included
```


SAMSUNG ELECTRONICS

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

```
CONV_LOOP:  TM     ADCON,#00001000B     ;  Check EOC flag
            JR     Z,CONV_LOOP          ;  If EOC flag=0, jump to CONV_LOOP until EOC flag=1
            LD     R0,ADDATAH           ;  High 8 bits of conversion result are stored
                                        ;  to ADDATAH register

            LD     R1,ADDATAL           ;  Low 2 bits of conversion result are stored
                                        ;  to ADDATAL register
            LD     ADCON,#00010011B     ;  Select analog input channel → P0.1

                                        ;  Select conversion speed → fOSC/8
                                        ;  Set conversion start bit

CONV_LOOP2: TM     ADCON,#00001000B     ;  Check EOC flag
            JR     Z,CONV_LOOP2
            LD     R2,ADDATAH
            LD     R3,ADDATAL
            •
            •
            •
            RET                         ;
            •                           ;
INT_TIMER0: AND    TACON, #11111110B    ;  Pending bit clear
            IRET                        ;
            •
            •
            END
```

13 ELECTRICAL DATA

OVERVIEW

In this section, the following S3F84K4 electrical characteristics are presented in tables and graphs:

— Absolute maximum ratings
— D.C. electrical characteristics
— A.C. electrical characteristics
— Input timing measurement points
— Oscillator characteristics
— Oscillation stabilization time
— Operating voltage range
— Schmitt trigger input characteristics
— Data retention supply voltage in stop mode
— Stop mode release timing when initiated by a RESET
— A/D converter electrical characteristics
— LVR circuit characteristics
— LVR reset timing

Table 13-1. Absolute Maximum Ratings

(T_A = 25 °C)

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{DD}	–	– 0.3 to + 6.5	V
Input voltage	V_I	All ports	– 0.3 to V_{DD} + 0.3	V
Output voltage	V_O	All output ports	– 0.3 to V_{DD} + 0.3	V
Output current high	I_{OH}	One I/O pin active	– 25	mA
		All I/O pins active	– 80	
Output current low	I_{OL}	One I/O pin active	+ 30	mA
		All I/O pins active	+ 150	
Operating temperature	T_A	–	– 25 to + 85	°C
Storage temperature	T_{STG}	–	– 65 to + 150	°C

Table 13-2. DC Electrical Characteristics

(T_A = – 25 °C to + 85 °C, V_{DD} = 2.0 V to 5.5 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
Input high voltage	V_{IH1}	Ports 0, 1, 2 and RESET	V_{DD}= 2.0 to 5.5 V	0.8 V_{DD}	–	V_{DD}	V
	V_{IH2}	X_{IN} and X_{OUT}		V_{DD}- 0.1			
Input low voltage	V_{IL1}	Ports 0, 1, 2 and RESET	V_{DD}= 2.0 to 5.5 V	–	–	0.2 V_{DD}	V
	V_{IL2}	X_{IN} and X_{OUT}				0.1	
Output high voltage	V_{OH}	I_{OH} = – 10 mA ports 0, 1, 2	V_{DD}= 4.5 to 5.5 V	V_{DD}-1.5	V_{DD}- 0.4	–	V
Output low voltage	V_{OL}	I_{OL} = 25 mA port 0, 1, and 2	V_{DD}= 4.5 to 5.5 V	–	0.4	2.0	V
Input high leakage current	I_{LIH1}	All input except I_{LIH2}	V_{IN} = V_{DD}	–	–	1	uA
	I_{LIH2}	X_{IN}, X_{OUT}	V_{IN} = V_{DD}			20	
Input low leakage current	I_{LIL1}	All input except I_{LIL2}	V_{IN} = 0 V	–	–	–1	uA
	I_{LIL2}	X_{IN}, X_{OUT}	V_{IN} = 0 V			–20	
Output high leakage current	I_{LOH}	All output pins	V_{OUT} = V_{DD}	–	–	2	uA
Output low leakage current	I_{LOL}	All output pins	V_{OUT} = 0 V	–	–	–2	uA
Pull-up resistors	R_P	V_{IN} = 0 V, T_A=25°C Ports 0, 1, 2	V_{DD} = 5 V	25	50	100	kΩ
Pull-down resistors	R_P	V_{IN} = 0 V, T_A=25°C Ports 1	V_{DD} = 5 V	25	50	100	
Supply current	I_{DD1}	Run mode 8 MHz CPU clock	V_{DD} = 2.0 to 5.5 V	–	3	6	mA
	I_{DD2}	Idle mode 8 MHz CPU clock	V_{DD} = 2.0 to 5.5 V	–	2	4	
	I_{DD3}	Stop mode T_A = 25°C	V_{DD} = 2.0 to 5.5 V	–	200	400	uA

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

Table 13-3. AC Electrical Characteristics

(T_A = – 25 °C to + 85 °C, V_{DD} = 2.0 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input low width	t_{INTL}	INT0, INT1 V_{DD} = 5 V ± 10 %	500		–	ns
RESET input low width	t_{RSL}	Input V_{DD} = 5 V ± 10 %	10		–	us

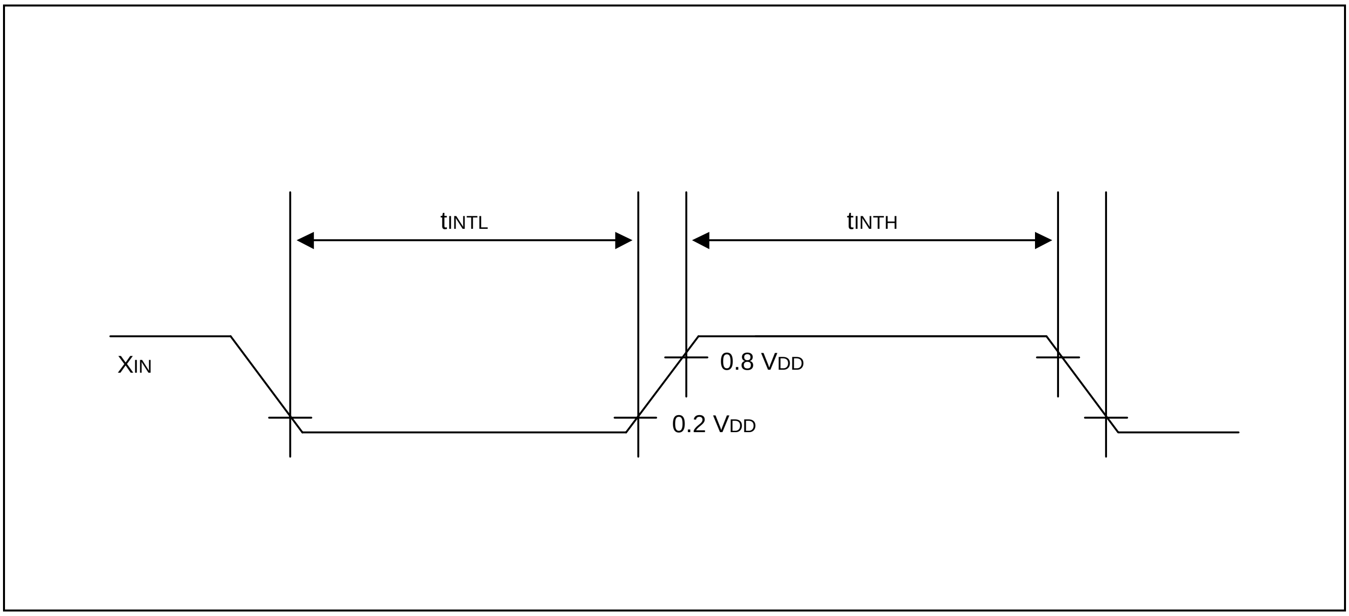
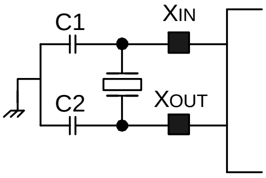
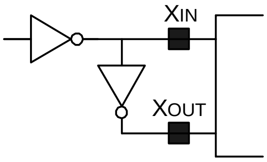


Figure 13-1. Input Timing Measurement Points

Table 13-4. Oscillator Characteristics

(T_A = – 25 °C to + 85 °C)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Main crystal or ceramic		V_{DD} = 4.5 to 5.5 V	1	–	8	MHz
		V_{DD} = 2.7 to 4.5 V	1	–	6	MHz
		V_{DD} = 2.0 to 2.7 V	1	–	3	MHz
External clock (Main System)		V_{DD} = 4.5 to 5.5 V	1	–	8	MHz
		V_{DD} = 2.7 to 4.5 V	1	–	6	MHz
		V_{DD} = 2.0 to 2.7 V	1	–	3	MHz
External RC oscillator	–	V_{DD} = 5 V	–	4	–	MHz
Internal RC oscillator	–	V_{DD} = 5 V	–	8		MHz
		Tolerance:20% at T_A =25°C		1		MHz

NOTE: For the resistor of External RC oscillator, we recommend using 28kΩ for 8MHz (at T_A =25°C).

Table 13-5. Oscillation Stabilization Time

(T_A = – 25 °C to + 85 °C, V_{DD} = 2.0 V to 5.5 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Main crystal	f_{OSC} > 1.0 MHz	–	–	20	ms
Main ceramic	Oscillation stabilization occurs when V_{DD} is equal to the minimum oscillator voltage range.	–	–	10	ms
External clock (main system)	X_{IN} input high and low width (t_{XH}, t_{XL})	25	–	500	ns
Oscillator stabilization	t_{WAIT} when released by a reset **(note 1)**	–	$2^{19}/f_{OSC}$	–	ms
Wait time	t_{WAIT} when released by an interrupt **(note 2)**	–	–	–	ms

NOTES:

1. f_{OSC} is the oscillator frequency.
2. The duration of the oscillator stabilization wait time, t_{WAIT}, when it is released by an interrupt is determined by the settings in the basic timer control register, BTCON.

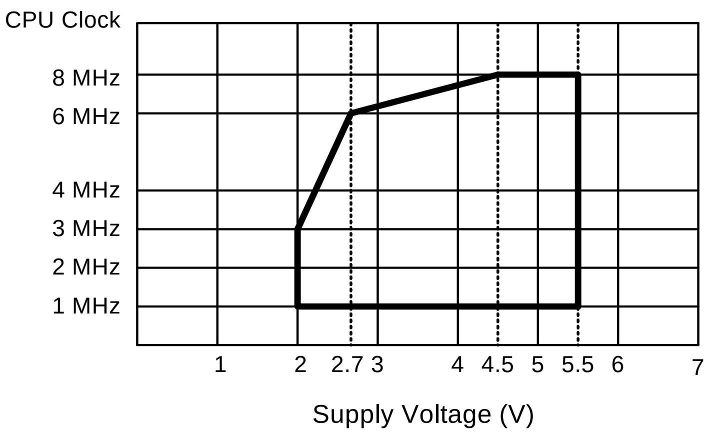


Figure 13-2. Operating Voltage Range

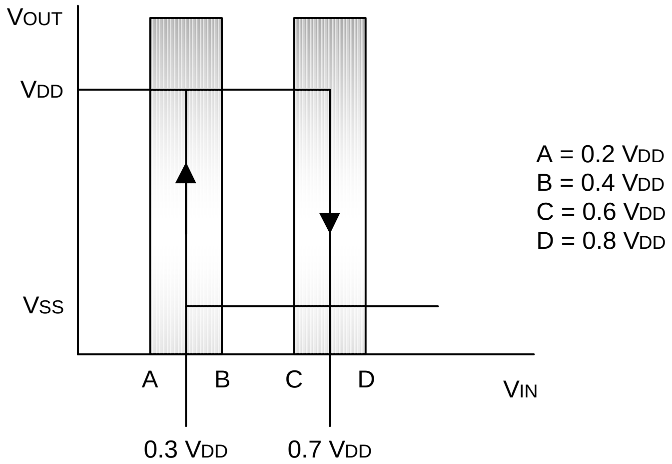


Figure 13-3. Schmitt Trigger Input Characteristics Diagram

Table 13-6. Data Retention Supply Voltage in Stop Mode

(T_A = – 25 °C to + 85 °C, V_{DD} = 2.0 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}	Stop mode	2.0	–	5.5	V
Data retention supply current	I_{DDDR}	Stop mode; V_{DDDR} = 2.0 V	–	100	200	uA

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

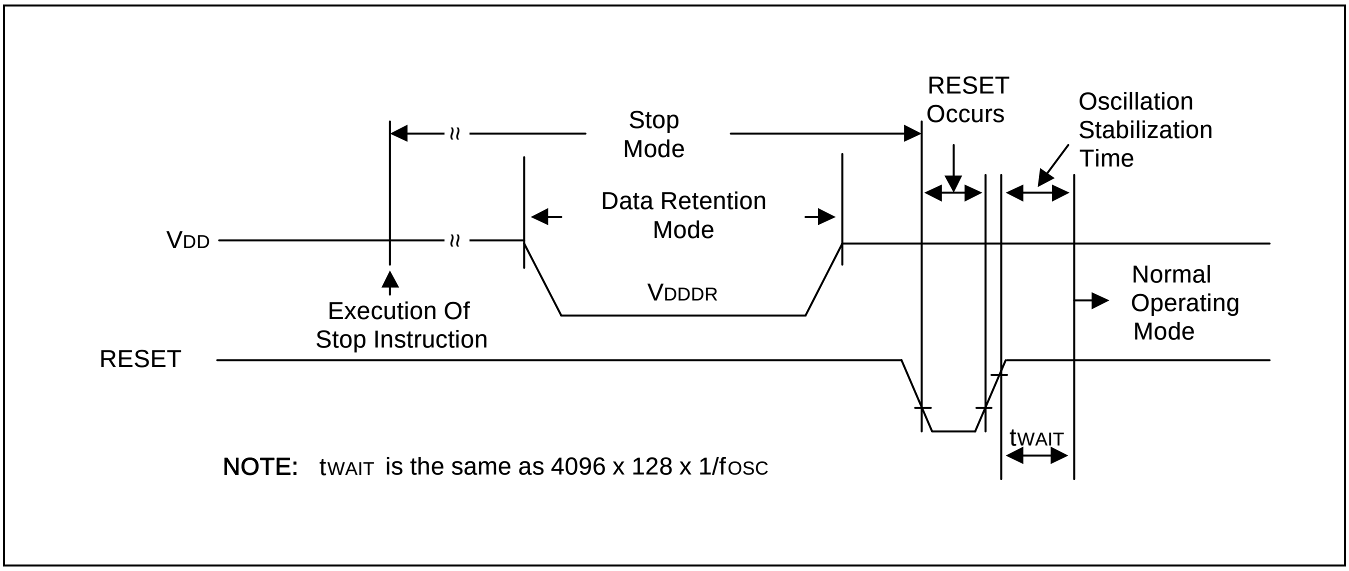


Figure 13-4. Stop Mode Release Timing When Initiated by a RESET

Table 13-7. A/D Converter Electrical Characteristics

(T_A = – 25 °C to + 85 °C, V_{DD} = 2.0 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Unit
Total accuracy	–	V_{DD} = 5.12 V CPU clock = 8 MHz V_{SS} = 0 V	–	–	± 3	LSB
Integral linearity error	ILE	″	–	–	± 2	
Differential linearity error	DLE	″	–	–	± 1	
Offset error of top	EOT	″	–	± 1	± 3	
Offset error of bottom	EOB	″	–	± 1	± 2	
Conversion time **(note 1)**	t_{CON}	f_{OSC} = 8 MHz	–	25	–	μs
Analog input voltage	V_{IAN}	–	V_{SS}	–	V_{DD}	V
Analog input impedance	R_{AN}	–	2	–	–	MΩ
Analog input current	I_{ADIN}	V_{DD} = 5 V	–	–	10	μA
Analog block current **(note 2)**	I_{ADC}	V_{DD} = 5 V	–	1	3	mA
		V_{DD} = 3 V		0.5	1.5	
		V_{DD} = 5 V power down mode	–	100	500	nA

NOTES:

1. "Conversion time" is the time required from the moment a conversion operation starts until it ends.
2. I_{ADC} is operating current during A/D conversion.

Table 13-8. LVR Circuit Characteristics

(T_A = 25 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Low voltage reset	V_{LVR}	–	1.9 2.6 3.4	2.2 3.0 3.9	2.5 3.4 4.4	V

Figure 13-5. LVR Reset Timing

Table 13-9. AC Electrical Characteristics for Internal Flash ROM

(T_A = –25 °C to + 85 °C)

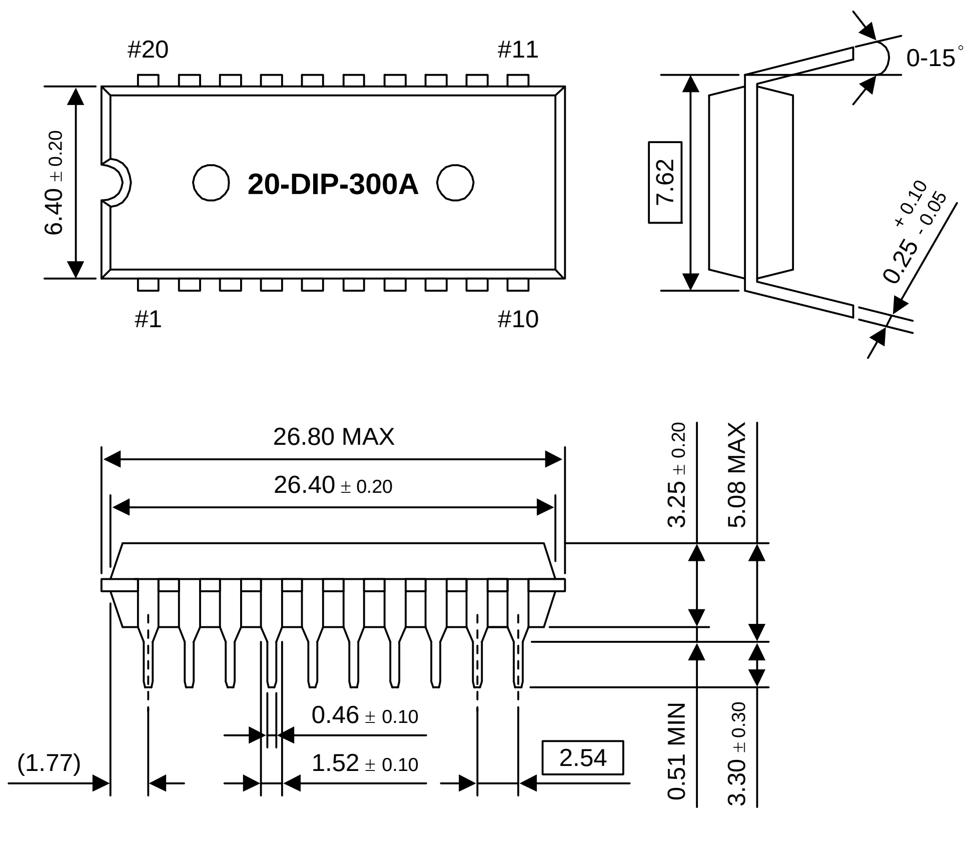
Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Flash Erase/Write/Read Voltage	Fewrv	V_{DD}	2.0	5.0	5.5	V
Programming Time (note 1)	Ftp		32	–	60	μS
Sector Erasing Time (note 2)	Ftp1		10	–	20	mS
Chip Erasing Time (note 3)	Ftp2		50	–	100	mS
Data Access Time	Ft_{RS}	V_{DD} = 2.0 V	–	250	–	nS
Number of Writing/Erasing	FNwe	–	10,000	–	–	Times
Data Retention	Ftdr	–	10	–	–	Years

NOTES:
1. The programming time is the time during which one byte (8-bit) is programmed.
2. The Sector erasing time is the time during which all 128-bytes of one sector block is erased.
3. In the case of S3F84K4, the chip erasing is available in Tool Program Mode only.

14 MECHANICAL DATA

OVERVIEW

The S3F84K4 is available in a 20-pin DIP package (Samsung: 20-DIP-300A), a 20-pin SOP package (Samsung: 20-SOP-375), a 20-pin SSOP package (Samsung: 20-SSOP-225), a 16-pin DIP package (Samsung: 16-DIP-300A), a 16-pin SOP package (Samsung: 16-SOP-BD300-SG), a 16-pin SSOP package (Samsung: 16-SSOP-BD44). Package dimensions are shown in Figure 14-1, 14-2, 14-3, 14-4, 14-5 and 14-6.



NOTE: Dimensions are in millimeters.

Figure 14-1. 20-DIP-300A Package Dimensions

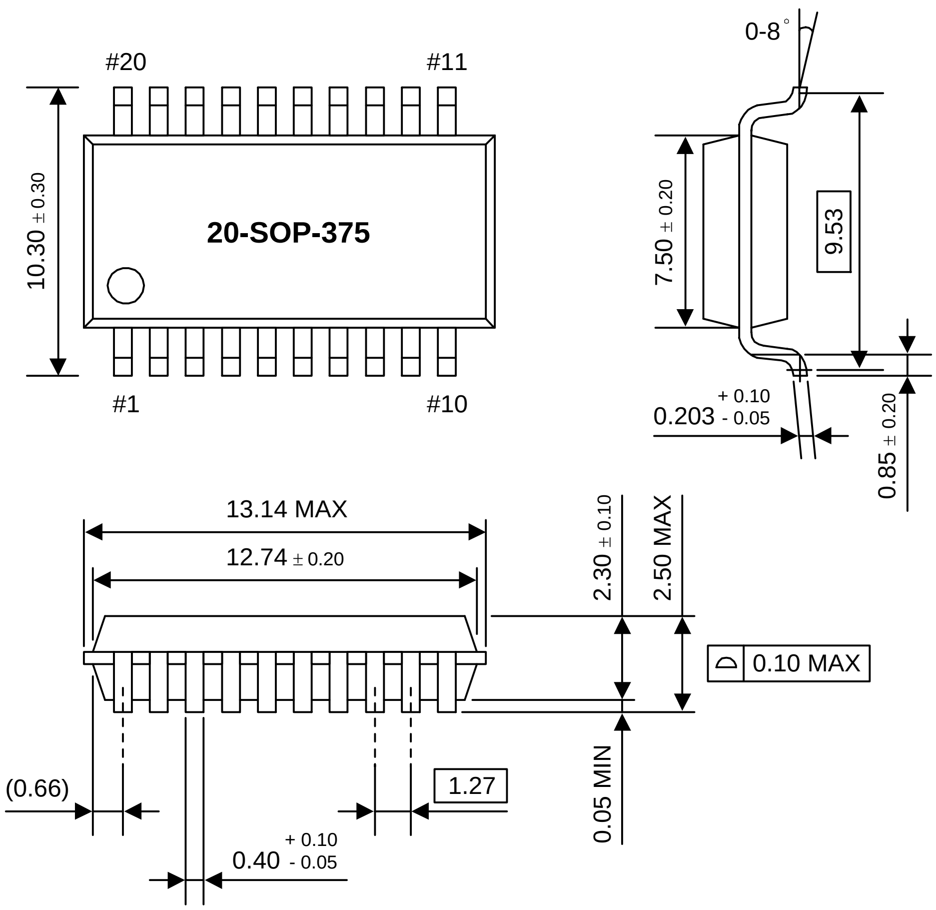
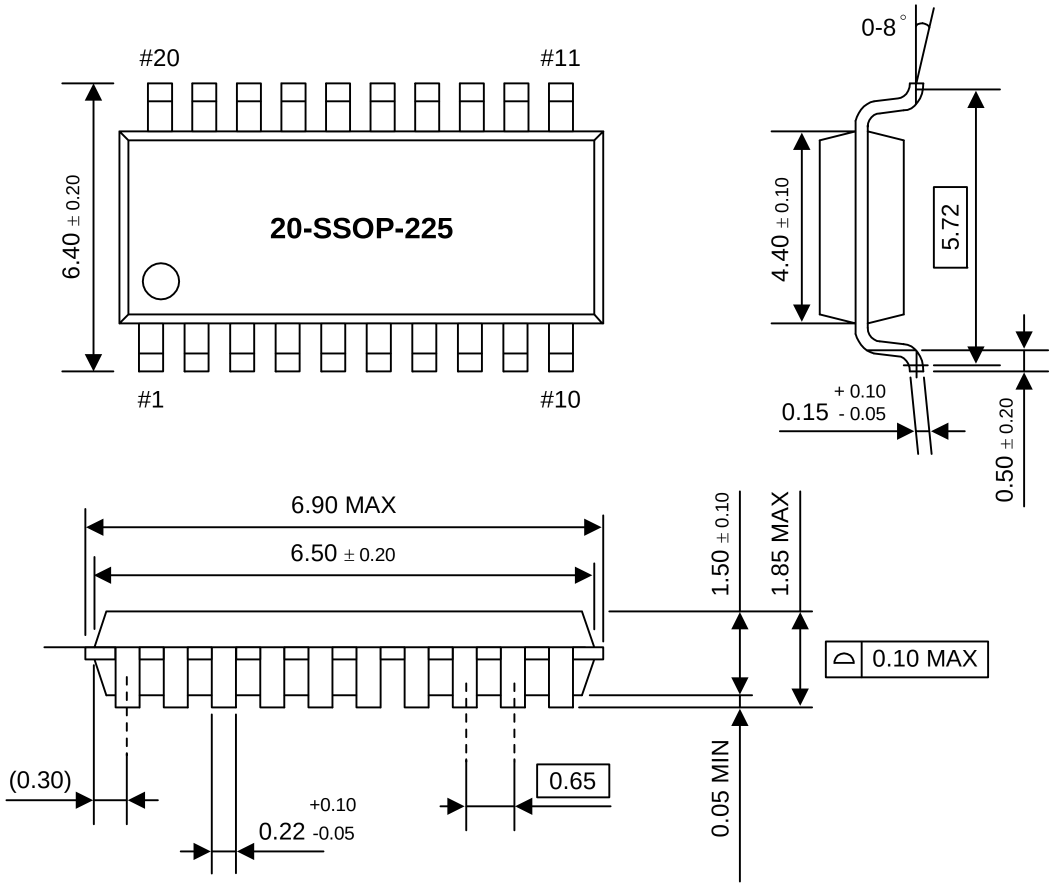
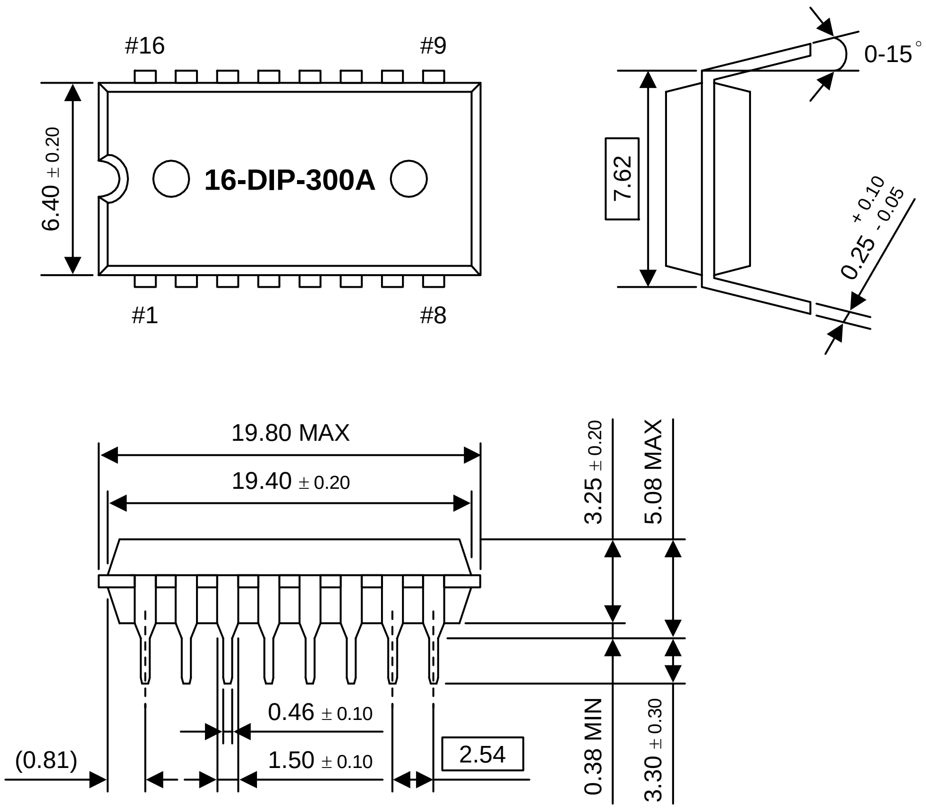


Figure 14-2. 20-SOP-375 Package Dimensions



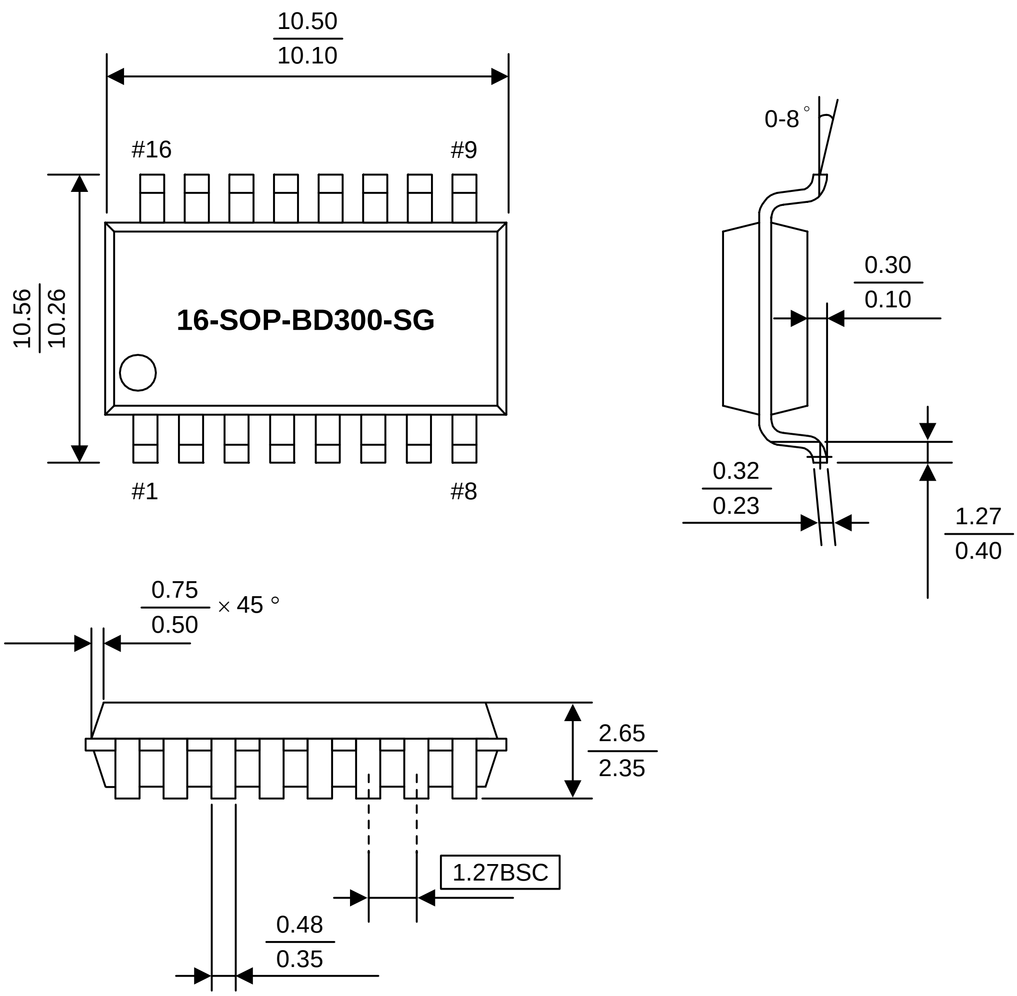
NOTE: Dimensions are in millimeters.

Figure 14-3. 20-SSOP-225 Package Dimensions



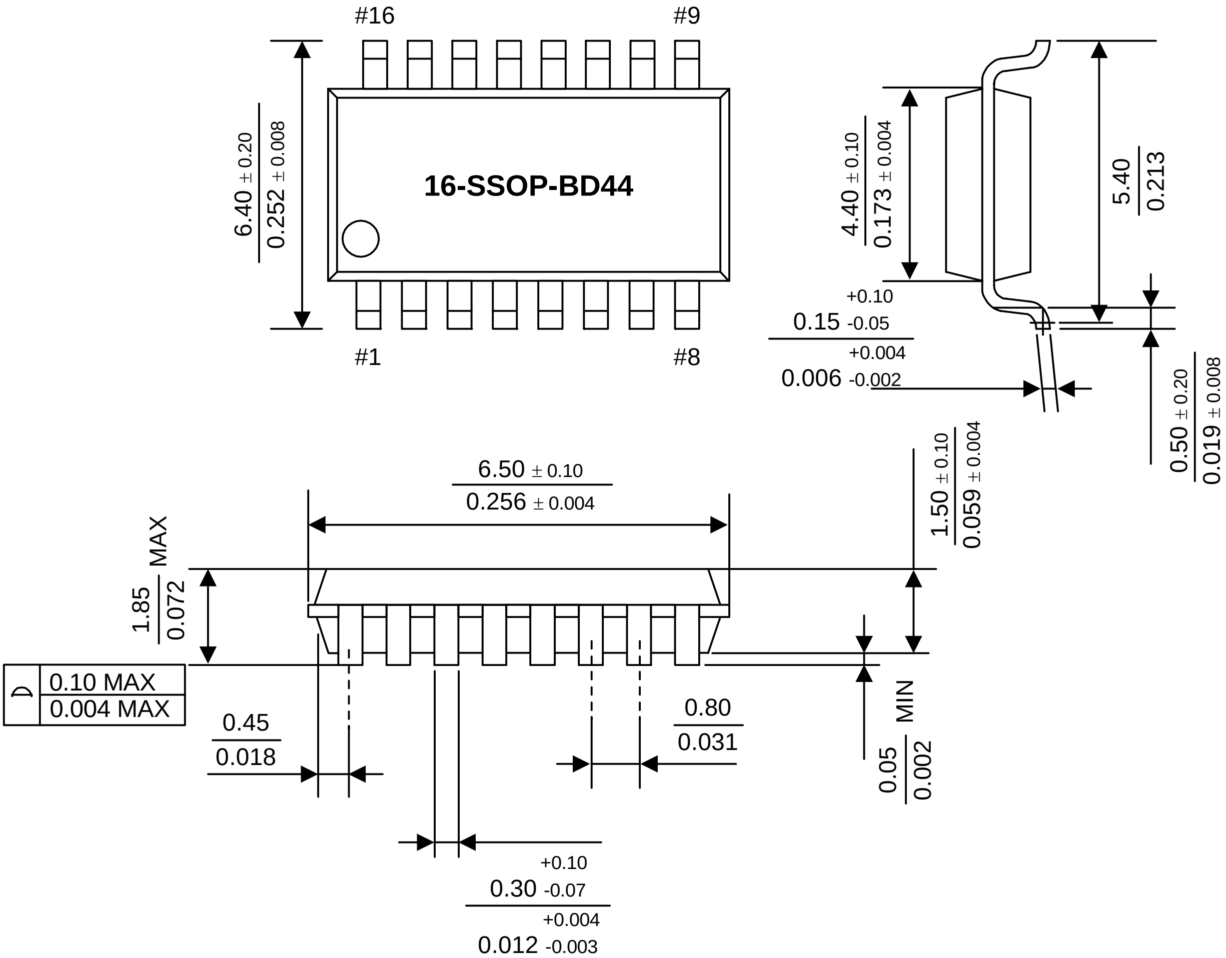
NOTE: Dimensions are in millimeters.

Figure 14-4. 16-DIP-300A Package Dimensions



NOTE: Dimensions are in millimeters.

Figure 14-5. 16-SOP-BD300-SG Package Dimensions



NOTE: Dimensions are in millimeters.

Figure 14-6. 16-SSOP-BD44 Package Dimensions

15 DEVELOPMENT TOOLS

OVERVIEW

Samsung provide a powerful and ease-to-use development support system on a turnkey basis. The development support system is composed of a host system, debugging tools, and supporting software. For a host system, any standard computer that employs Win98/2000/XP as its operating system can be used. A sophisticated debugging tool is provided both in hardware and software: the powerful in-circuit emulator, OPENice-i500, for the S3F7-, S3F9-and S3F8- microcontroller families. OPENice-i500 is supported by a third party tool vendor, AIJI System. It also offers supporting software that includes, debugger, an assembler, and a program for setting options.

SASM

The SASM takes a source file containing assembly language statements and translates them into a corresponding source code, an object code and comments. The SASM supports macros and conditional assembly. It runs on the MS-DOS operating system. As it produces the re-locatable object codes only, the user should link object files. Object files can be linked with other object files and loaded into memory. SASM requires a source file and auxiliary register file (device_name.reg) with device specific information.

SAMA ASSEMBLER

The Samsung Arrangeable Microcontorller (SAM) Assembler, SAMA, is a universal assembler, and generating an object code in the standard hexadecimal format. Assembled program codes include the object code used for ROM data and required In-circuit emulators program control data. To assemble programs, SAMA requires a source file and an auxiliary definition (device_name.def) file with device specific information.

OPENice-SLD

OPENice-SLD Host Interface for In-Circuit Emulator is a multi-window based debugger for OPENice-i500. OPENice-SLD provides pull-down and pop-up menus, mouse support, function/hot keys, and context-sensitive hyper-linked help. It has an advanced, multiple-windowed user interface that emphasizes ease of use. Each window can be easily sized, moved, scrolled, highlighted, added, or removed.

HEX2ROM

HEX2ROM file generates a ROM code from a HEX file which is produced by the assembler. A ROM code is needed to fabricate a microcontroller which has a mask ROM. When generating a ROM code(.OBJ file) by HEX2ROM, the value ”FF” is automatically filled into the unused ROM area, up to the maximum ROM size of the target device.

TARGET BOARDS

Target boards are available for all S3F8-series microcontrollers. All required target system cables and adapters are included with the device-specific target board. TB84K4 is a specific target board for S3F84K4 development.

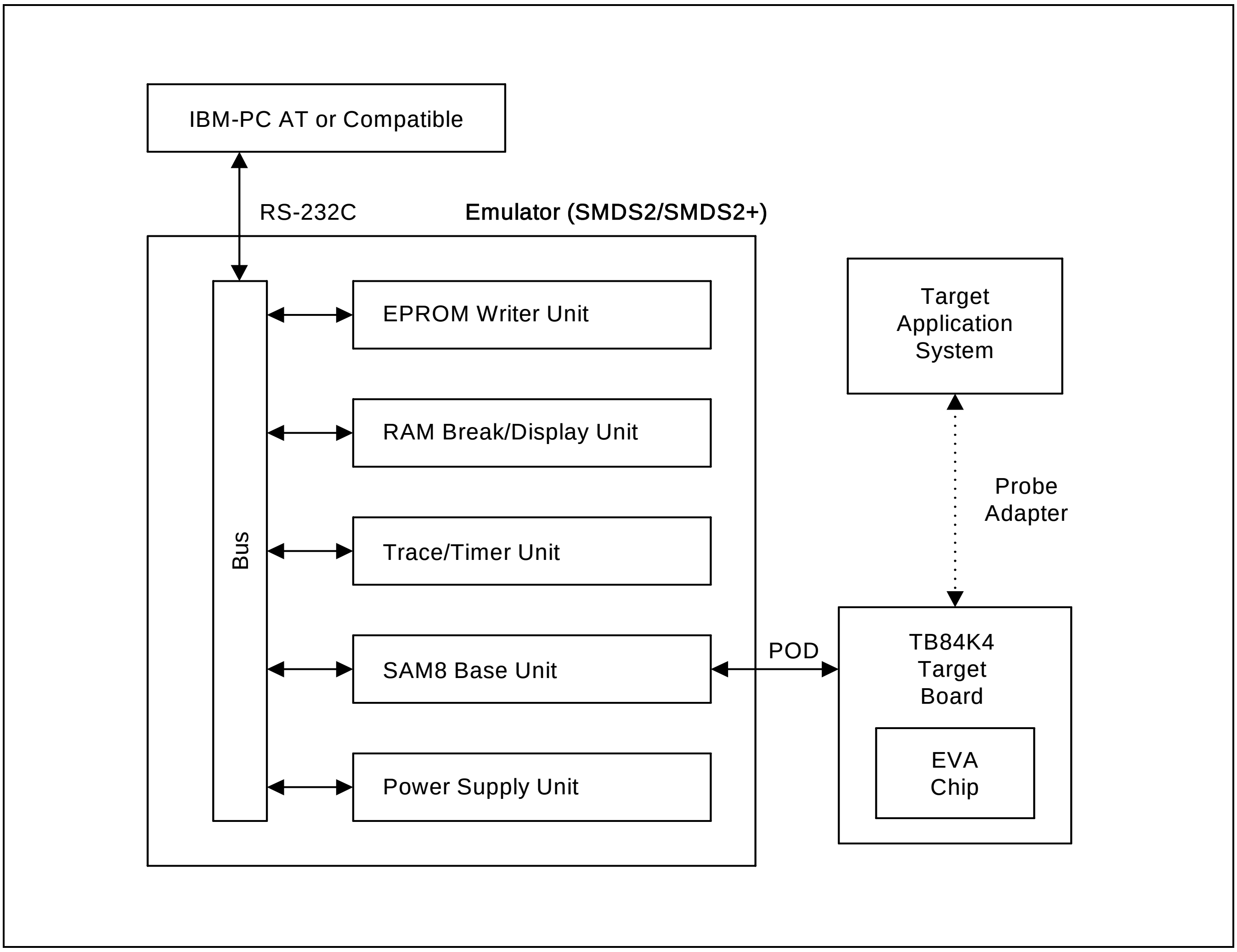


Figure 15-1. SMDS2/SMDS2+ Product Configuration

TB84K4 TARGET BOARD

The TB84K4 target board is used for the S3F84K4 microcontrollers. It is supported by the SMDS2/SMDS2+ development systems.

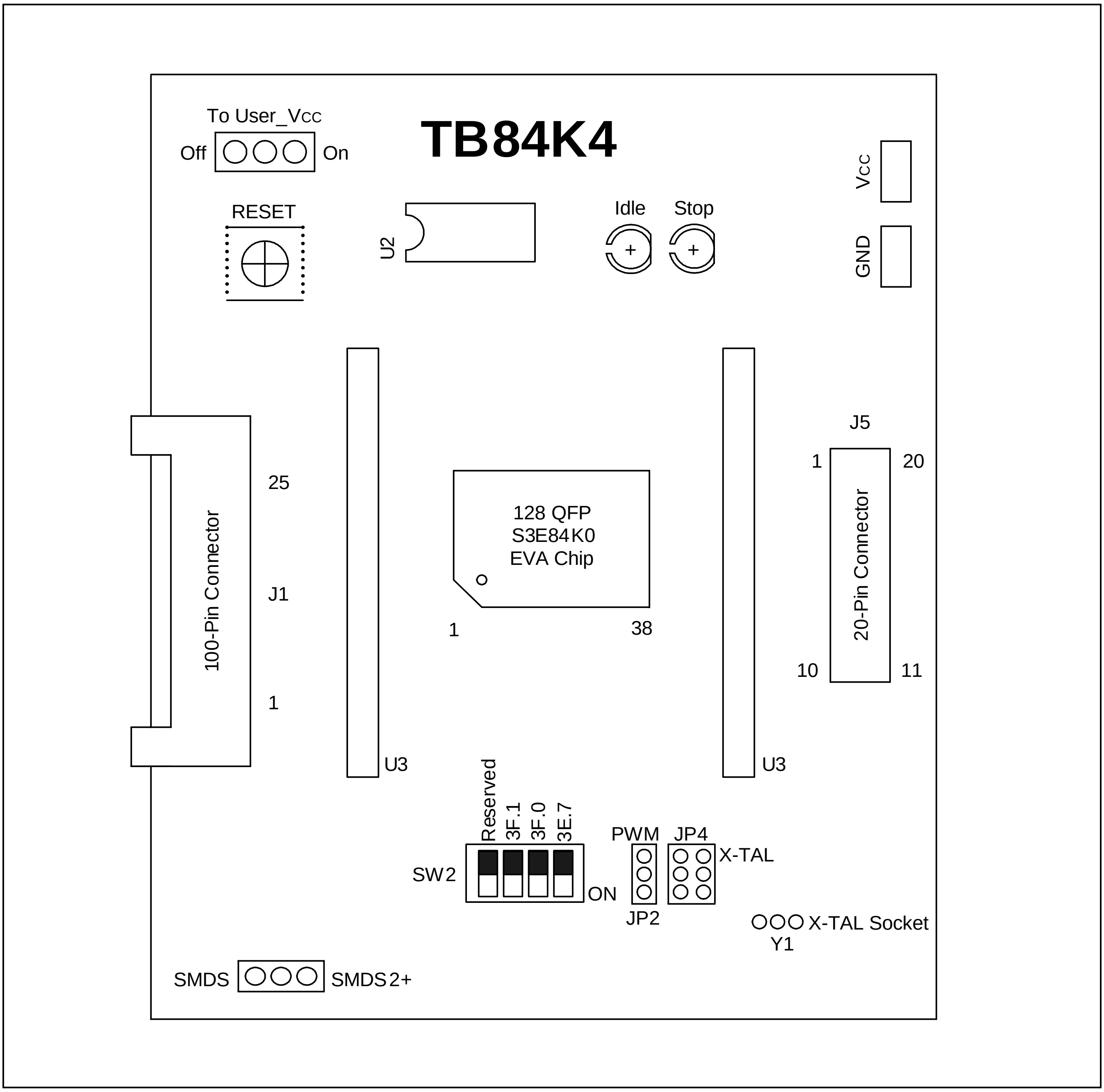
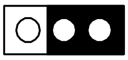
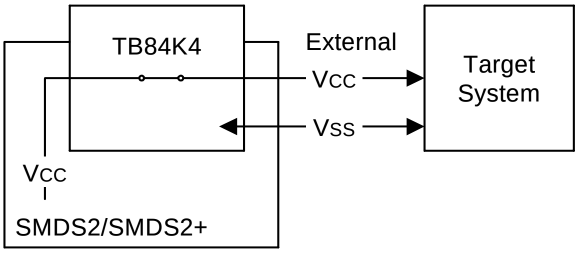
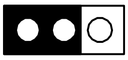
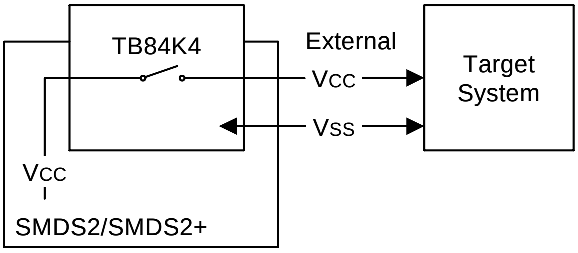


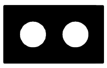
Figure 15-2. TB84K4 Target Board Configuration

NOTE: Don't care about the "U3". It's a test socket only for the chip designer.

Table 15-1. Power Selection Settings for TB84K4

"To User_Vcc" Settings	Operating Mode	Comments
To user_Vcc off  on		The SMDS2/SMDS2+ main board supplies V_{CC} to the target board (evaluation chip) and the 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:



SMDS2+ Selection (SAM8)

In order to write data into program memory that is available in SMDS2+, the target board should be selected to be for SMDS2+ through a switch as follows. Otherwise, the program memory writing function is not available.

Table 15-2. The SMDS2+ Tool Selection Setting

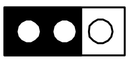
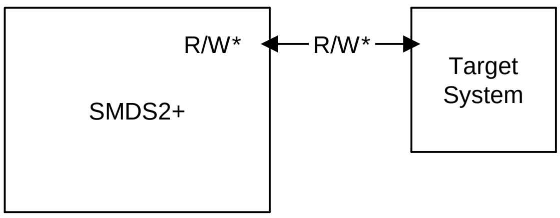
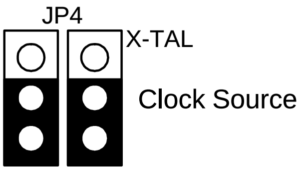
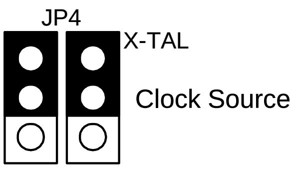
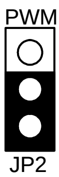
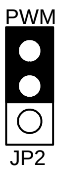
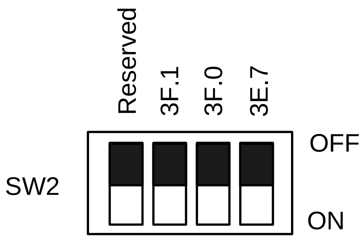
"SW1" Setting	Operating Mode
SMDS  SMDS2+	

Table 15-3. Using Single Header Pins to Select Clock Source and Enable/Disable PWM

Target Board Part	Comments
	Use SMDS2/SMDS2+ internal clock source as the system clock.
	Use external crystal or ceramic oscillator as the system clock.
	PWM function is DISABLED.
	PWM function is ENABLED.



ON	Low
OFF	High (Default)

NOTES:
1. For EVA chip, smart option is determined by DIP switch not software.
2. Please keep the reserved bits as default value (high).

Figure 15-3. DIP Switch for Smart Option

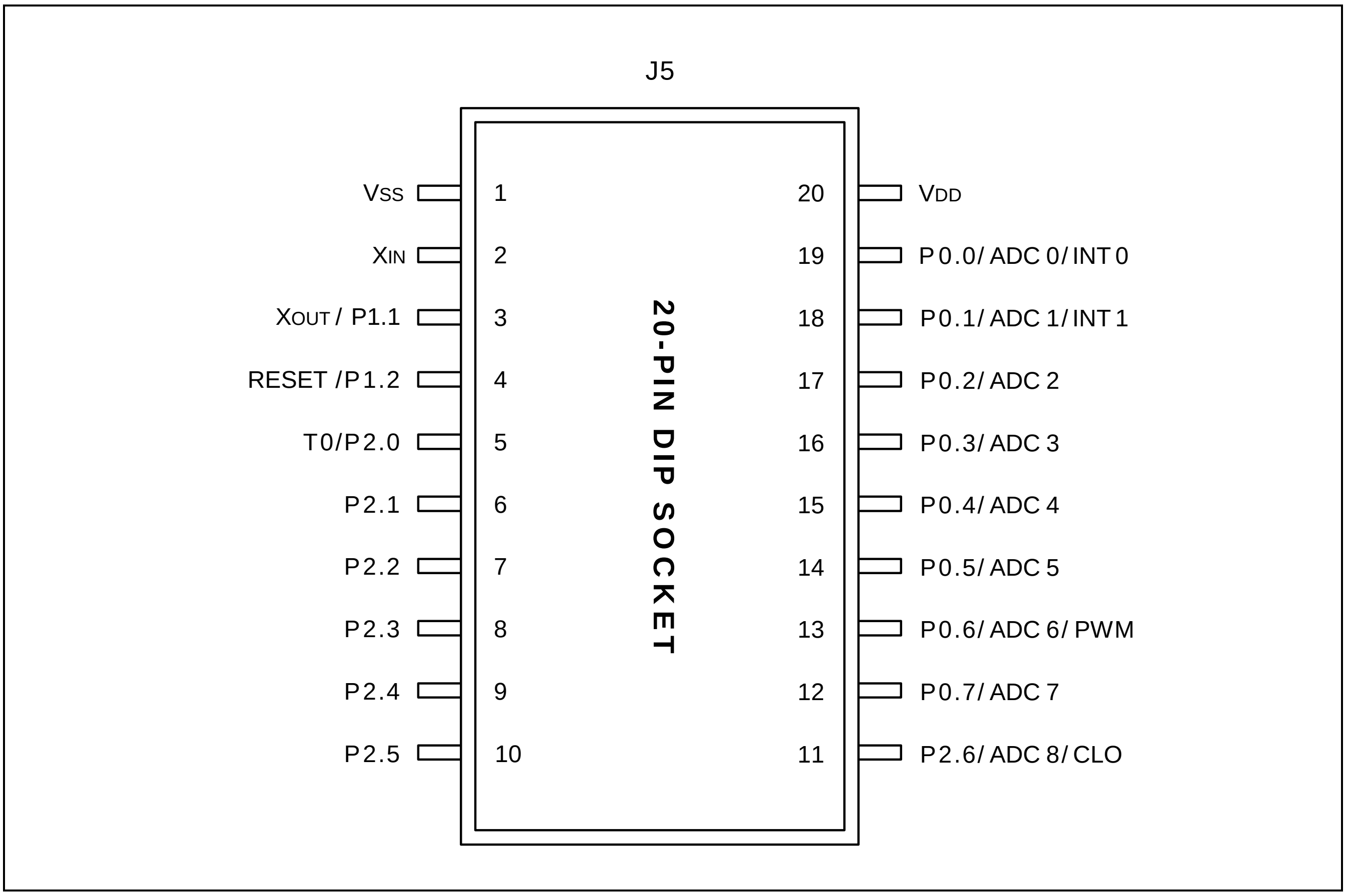


Figure 15-4. 20-Pin Connector for TB84K4

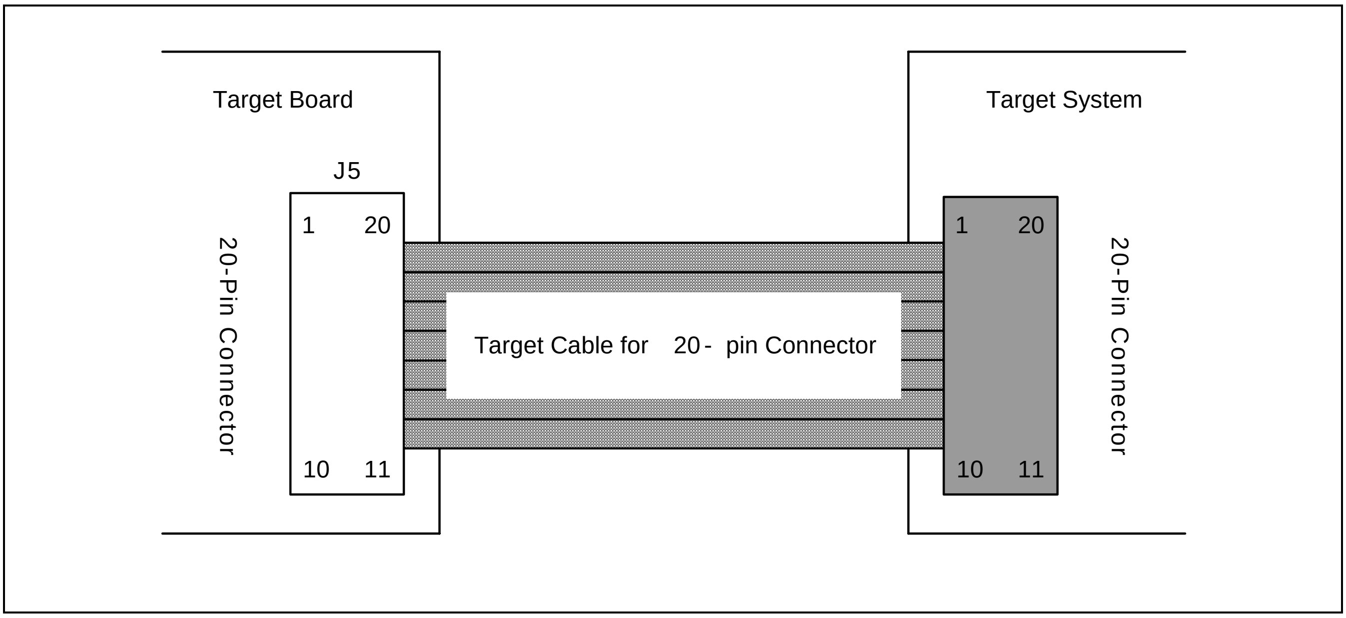


Figure 15-5. S3F84K4 Probe Adapter for 20-DIP Package