



## **S3 Family 8-Bit Microcontrollers**

# **S3F8S28/S3F8S24**

## **Product Specification**

PS031305-1017





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

---

## LIFE SUPPORT POLICY

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

### As used herein

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

### Document Disclaimer

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concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded.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

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

# Revision History

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

Date	Revision Level	Description	Page
Oct 2017	05	Added Zilog Library-based Development Platform and updated to most current 3rd party tools. Removed 'Preliminary' from footer.	CH 21 All
Jan 2015	04	Modified P0.0 and P0.1 descriptions in Figures 1-1 through 1-3 and Table 1-2 to include SCLK and SDAT values, respectively.	1-5, 1-6, 1-7, 1-9
Jan 2015	03	Updated the Third Parties for Development Tools section.	22-9
Apr 2014	02	Modified language in nRESET pin statement, 2nd paragraph of 8.1 System Reset section; corrected misspelling in Figure 8-1; corrected superscript error in Table 19-5; corrected erroneous note, Figure 19-4.	8-1, 8-2, 19-5 19-6
Aug 2013	01	Original Zilog issue.	n/a
May 2012	1.1	Deleted P2.5 and P2.4 open-drain output functions.	n/a
May 2011	1.0	Released version V1.0.	n/a

# Table of Contents

Table of Contents.....	1
List of Figures .....	8
List of Tables.....	12
List of Examples.....	14
1 Product Overview.....	1-1
1.1 S3C8/S3F8 Series Microcontrollers .....	1-1
1.2 S3F8S28/S3F8S24 Microcontroller .....	1-1
1.3 Features .....	1-2
1.4 Block Diagram .....	1-5
1.5 Pin Assignments .....	1-6
1.6 Pin Descriptions .....	1-8
1.7 Pin Circuits .....	1-10
2 Address Spaces.....	2-1
2.1 Overview .....	2-1
2.2 Program Memory (ROM).....	2-2
2.2.1 Normal Operating Mode .....	2-2
2.2.2 Smart Option.....	2-3
2.3 Register Architecture.....	2-5
2.3.1 Register Page Pointer (PP) .....	2-7
2.3.2 Register Set 1 .....	2-7
2.3.3 Register Set 2 .....	2-8
2.3.4 Prime Register Space .....	2-8
2.3.5 Working Registers.....	2-9
2.3.6 Using the Register Pointers .....	2-10
2.4 Register Addressing.....	2-12
2.4.1 Common Working Register Area (C0H to CFH).....	2-14
2.4.2 4-Bit Working Register Addressing.....	2-15
2.4.3 8-Bit Working Register Addressing.....	2-17
2.5 System and User Stacks.....	2-19
2.5.1 Stack Operations .....	2-19
2.5.2 User-Defined Stacks.....	2-19
2.5.3 Stack Pointers (SPL) .....	2-19
3 Addressing Modes .....	3-1
3.1 Overview .....	3-1
3.2 Register Addressing Mode (R).....	3-2
3.3 Indirect Register Addressing Mode (IR).....	3-3
3.4 Indexed Addressing Mode (X).....	3-7
3.5 Direct Address Mode (DA) .....	3-10
3.6 Indirect Address Mode (IA) .....	3-12
3.7 Relative Address Mode (RA).....	3-13
3.8 Immediate Mode (IM) .....	3-14

4 Control Registers .....	4-1
4.1 Overview .....	4-1
4.1.1 ADCON .....	4-6
4.1.2 BTCON .....	4-7
4.1.3 CLKCON .....	4-8
4.1.4 EMT .....	4-9
4.1.5 FLAGS .....	4-10
4.1.6 FMCON .....	4-11
4.1.7 FMSECH .....	4-11
4.1.8 FMSECL .....	4-12
4.1.9 FMUSR .....	4-12
4.1.10 ICCR .....	4-13
4.1.11 ICSR .....	4-14
4.1.12 IMR .....	4-15
4.1.13 IPH .....	4-16
4.1.14 IPL .....	4-16
4.1.15 IPR .....	4-17
4.1.16 IRQ .....	4-18
4.1.17 LVDCON .....	4-19
4.1.18 P0CONH .....	4-20
4.1.19 P0CONL .....	4-21
4.1.20 P0PND .....	4-22
4.1.21 P0PUR .....	4-24
4.1.22 P1CON .....	4-25
4.1.23 P2CONH .....	4-26
4.1.24 P2CONL .....	4-27
4.1.25 P2PUR .....	4-28
4.1.26 P3CON .....	4-29
4.1.27 P3PND .....	4-30
4.1.28 PP .....	4-31
4.1.29 PWM0CON .....	4-32
4.1.30 PWM1CON .....	4-33
4.1.31 PWM0EX .....	4-34
4.1.32 PWM1EX .....	4-34
4.1.33 RESETID .....	4-35
4.1.34 ROSCCON .....	4-36
4.1.35 RP0 .....	4-36
4.1.36 RP1 .....	4-37
4.1.37 SPL .....	4-37
4.1.38 STOPCON .....	4-37
4.1.39 SYM .....	4-38
4.1.40 T1CON .....	4-39
4.1.41 T1PS .....	4-40
4.1.42 TACON .....	4-40
4.1.43 TBCON .....	4-42
4.1.44 UARTCON .....	4-43
4.1.45 UARTPND .....	4-44
4.1.46 WDTCON .....	4-45
5 Interrupt Structure .....	5-1
5.1 Overview .....	5-1
5.1.1 Levels .....	5-1
5.1.2 Vectors .....	5-1
5.1.3 Sources .....	5-1

5.2	Interrupt Types .....	5-2
5.3	S3F8S28/S3F8S24 Interrupt Structure .....	5-3
5.3.1	Interrupt Vector Addresses .....	5-4
5.3.2	Enable/Disable Interrupt Instructions (EI, DI) .....	5-4
5.4	System-Level Interrupt Control Registers .....	5-5
5.5	Interrupt Processing Control Points .....	5-6
5.6	Peripheral Interrupt Control Registers .....	5-7
5.7	System Mode Register (SYM).....	5-8
5.8	Interrupt Mask Register (IMR).....	5-9
5.9	Interrupt Priority Register (IPR).....	5-10
5.10	Interrupt Request Register (IRQ) .....	5-12
5.11	Interrupt Pending Function Types .....	5-13
5.11.1	Overview .....	5-13
5.11.2	Pending Bits Cleared Automatically by Hardware .....	5-13
5.11.3	Pending Bits Cleared by the Service Routine.....	5-13
5.12	Interrupt Source Polling Sequence .....	5-14
5.13	Interrupt Service Routines.....	5-14
5.14	Generating Interrupt Vector Addresses .....	5-15
5.15	Nesting of Vectored Interrupts .....	5-15
5.16	Instruction Pointer (IP) .....	5-15
5.17	Fast Interrupt Processing .....	5-16
5.18	Procedure for Initiating Fast Interrupts.....	5-16
5.19	Fast Interrupt Service Routine .....	5-16
5.20	Relationship to Interrupt Pending Bit Types.....	5-17
5.21	Programming Guidelines.....	5-17
6	Instruction Set.....	6-1
6.1	Overview .....	6-1
6.1.1	Data Types.....	6-1
6.1.2	Register Addressing .....	6-1
6.1.3	Addressing Modes .....	6-1
6.2	Flags Register (FLAGS).....	6-5
6.2.1	Flag Descriptions .....	6-6
6.3	Instruction Set Notation.....	6-7
6.4	Condition Codes.....	6-11
6.5	Instruction Descriptions.....	6-12
6.5.1	ADC (Add with Carry) .....	6-13
6.5.2	ADD (Add).....	6-14
6.5.3	AND (Logical AND) .....	6-15
6.5.4	BAND (Bit AND).....	6-16
6.5.5	BCP (Bit Compare) .....	6-17
6.5.6	BITC (Bit Complement).....	6-18
6.5.7	BITR (Bit Reset).....	6-19
6.5.8	BITS (Bit Set) .....	6-20
6.5.9	BOR (Bit OR) .....	6-21
6.5.10	BTJRF (Bit Test, Jump Relative on False) .....	6-22
6.5.11	BTJRT (Bit Test, Jump Relative on True).....	6-23
6.5.12	BXOR (Bit XOR) .....	6-24
6.5.13	CALL (Call Procedure).....	6-25
6.5.14	CCF (Complement Carry Flag).....	6-26
6.5.15	CLR (Clear).....	6-27
6.5.16	COM (Complement).....	6-28
6.5.17	CP (Compare).....	6-29
6.5.18	CPIJE (Compare, Increment, and Jump on Equal) .....	6-30

6.5.19 CPIJNE (Compare, Increment, and Jump on Non-Equal).....	6-31
6.5.20 DA (Decimal Adjust) .....	6-32
6.5.21 DEC (Decrement) .....	6-34
6.5.22 DECW (Decrement Word) .....	6-35
6.5.23 DI (Disable Interrupts).....	6-36
6.5.24 DIV (Divide-Unsigned) .....	6-37
6.5.25 DJNZ (Decrement and Jump if Non-Zero).....	6-38
6.5.26 EI (Enable Interrupts).....	6-39
6.5.27 ENTER (Enter).....	6-40
6.5.28 EXIT (Exit).....	6-41
6.5.29 IDLE (Idle Operation).....	6-42
6.5.30 INC (Increment) .....	6-43
6.5.31 INCW (Increment Word) .....	6-44
6.5.32 IRET (Interrupt Return) .....	6-45
6.5.33 JP (Jump).....	6-46
6.5.34 JR (Jump Relative) .....	6-47
6.5.35 LD (Load) .....	6-48
6.5.36 LDB (Load Bit) .....	6-49
6.5.37 LDC/LDE (Load Memory) .....	6-50
6.5.38 LDCD/LDED (Load Memory and Decrement) .....	6-52
6.5.39 LDCI/LDEI (Load Memory and Increment) .....	6-53
6.5.40 LDCPD/LDEPD (Load Memory with Pre-Decrement) .....	6-54
6.5.41 LDCPI/LDEPI (Load Memory with Pre-Increment) .....	6-55
6.5.42 LDW (Load Word).....	6-56
6.5.43 MULT (Multiply-Unsigned) .....	6-57
6.5.44 NEXT (Next).....	6-58
6.5.45 NOP (No Operation) .....	6-59
6.5.46 OR (Logical OR) .....	6-60
6.5.47 POP (Pop from Stack) .....	6-61
6.5.48 POPUD (Pop User Stack-Decrementing).....	6-62
6.5.49 POPUI (Pop User Stack-Incrementing) .....	6-63
6.5.50 PUSH (Push to Stack) .....	6-64
6.5.51 PUSHUD (Push User Stack-Decrementing).....	6-65
6.5.52 PUSHUI (Push User Stack-Incrementing) .....	6-66
6.5.53 RCF (Reset Carry Flag).....	6-67
6.5.54 RET (Return).....	6-68
6.5.55 RL (Rotate Left) .....	6-69
6.5.56 RLC (Rotate Left Through Carry) .....	6-70
6.5.57 RR (Rotate Right) .....	6-71
6.5.58 RRC (Rotate Right Through Carry) .....	6-72
6.5.59 SB0 (Select Bank 0) .....	6-73
6.5.60 SB1 (Select Bank 1) .....	6-74
6.5.61 SBC (Subtract with Carry) .....	6-75
6.5.62 SCF (Set Carry Flag) .....	6-76
6.5.63 SRA (Shift Right Arithmetic) .....	6-77
6.5.64 SRP/SRP0/SRP1 (Set Register Pointer) .....	6-78
6.5.65 STOP (Stop Operation) .....	6-79
6.5.66 SUB (Subtract).....	6-80
6.5.67 SWAP (Swap Nibbles).....	6-81
6.5.68 TCM (Test Complement under Mask) .....	6-82
6.5.69 TM (Test under Mask) .....	6-83
6.5.70 WFI (Wait for Interrupt).....	6-84
6.5.71 XOR (Logical Exclusive OR).....	6-85

7 Clock Circuit.....	7-1
7.1 Overview .....	7-1
7.2 Main Oscillator Logic.....	7-2
7.3 Clock Status During Power-Down Modes.....	7-2
7.4 System Clock Control Register (CLKCON).....	7-3
7.5 Ring Oscillator Control Register (ROSCCON).....	7-4
8 RESET and Power-Down .....	8-1
8.1 System Reset.....	8-1
8.1.1 Overview.....	8-1
8.1.2 External RESET Pin .....	8-3
8.1.3 MCU Initialization Sequence.....	8-4
8.2 Power-Down Modes.....	8-5
8.2.1 Stop Mode.....	8-5
8.2.2 Sources to Release Stop Mode.....	8-5
8.2.3 Idle Mode .....	8-6
8.3 Hardware Reset Values .....	8-7
9 I/O Ports.....	9-1
9.1 Overview .....	9-1
9.2 Port Data Registers.....	9-2
9.2.1 Port 0 .....	9-3
9.2.2 Port 1 .....	9-8
9.2.3 Port 2 .....	9-10
9.2.4 Port 3 .....	9-14
10 Basic Timer and Timer 0.....	10-1
10.1 Module Overview .....	10-1
10.1.1 Basic Timer (BT).....	10-1
10.1.2 Timer 0.....	10-1
10.2 Basic Timer (BT) .....	10-2
10.2.1 Basic Timer Control Register (BTCON).....	10-2
10.2.2 Basic Timer Function Description .....	10-3
10.3 One 16-Bit Timer Mode (Timer 0).....	10-7
10.3.1 Overview .....	10-7
10.3.2 Function Description .....	10-7
10.3.3 Block Diagram.....	10-10
10.4 Two 8-Bit Timers Mode (Timer A and B) .....	10-11
10.4.1 Overview .....	10-11
10.4.2 Function Description .....	10-11
11 16-Bit Timer 1 .....	11-1
11.1 Overview .....	11-1
11.2 Function Description .....	11-2
11.2.1 Timer 1 Interrupts .....	11-2
11.2.2 Timer 1 Overflow Interrupt.....	11-2
11.2.3 Interval Mode (Match).....	11-2
11.2.4 Capture Mode .....	11-3
11.3 Timer 1 Control Register (T1CON) .....	11-4
12 Watchdog Timer.....	12-1
12.1 Overview .....	12-1
12.2 Function Description .....	12-2



12.2.1 Watchdog Interrupt .....	12-2
12.2.2 Release Stop .....	12-2
12.2.3 System Reset.....	12-2
12.3 Watchdog Timer Control Register (WDTCON) .....	12-3
12.4 Interrupt.....	12-6
12.5 System Reset.....	12-7
12.6 Interrupt & System Reset .....	12-8
13 PWM (Pulse Width Modulation) .....	13-1
13.1 Overview .....	13-1
13.2 Function Description .....	13-2
13.2.1 PWM .....	13-2
13.2.2 PWM Counter .....	13-2
13.2.3 PWM Data and Extension Registers .....	13-2
13.2.4 PWM Clock Rate.....	13-3
13.2.5 PWM Function Description .....	13-4
13.2.6 PWM Output Waveform.....	13-5
13.3 PWM Control Register (PWM0CON/PWM1CON) .....	13-11
13.4 PWM Extension Register (PWM0EX/PWM1EX) .....	13-12
14 A/D Converter .....	14-1
14.1 Overview .....	14-1
14.2 Using A/D Pins for Standard digital Input.....	14-2
14.3 A/D Converter Control Register (ADCON) .....	14-2
14.4 Internal Reference Voltage Levels .....	14-4
14.5 Conversion timing .....	14-6
14.6 Internal A/D Conversion Procedure .....	14-7
15 UART .....	15-1
15.1 Overview .....	15-1
15.1.1 Programming Procedure.....	15-1
15.1.2 UART Control Register (UARTCON).....	15-2
15.1.3 UART Interrupt Pending Register (UARTPND) .....	15-3
15.1.4 UART Data Register (UDATA) .....	15-4
15.1.5 UART Baud Rate Data Register (BRDATA) .....	15-4
15.1.6 Baud Rate Calculations .....	15-5
15.2 Block Diagram.....	15-6
15.2.1 UART Mode 0 Function Description .....	15-7
15.2.2 UART Mode 1 Function Description .....	15-8
15.2.3 UART Mode 2 Function Description .....	15-9
15.2.4 UART Mode 3 Function Description .....	15-10
15.2.5 Serial Communication for Multiprocessor Configurations .....	15-11
16 IIC Bus Interface .....	16-1
16.1 Overview .....	16-1
16.1.1 Multi-Master IIC Bus Control Register (ICCR).....	16-2
16.1.2 Multi-Master IIC Bus Control/Status Register (ICSR).....	16-4
16.1.3 Multi-Master IIC Bus Transmit/Receive Data Shift Register (IDSR).....	16-5
16.1.4 Multi-Master IIC Bus Address Register (IAR) .....	16-6
16.2 Block Diagram.....	16-7
16.3 The IIC Bus Interface .....	16-8
16.4 Start and Stop Conditions .....	16-8
16.5 Data Transfer Formats.....	16-10
16.6 ACK Signal Transmission .....	16-11

16.7 Read/Write Operations.....	16-14
16.8 Bus Arbitration Procedures .....	16-14
16.9 Abort Conditions.....	16-14
16.10 Configuring the IIC-Bus.....	16-14
<b>17 Low Voltage Detector.....</b>	<b>17-1</b>
17.1 Overview .....	17-1
17.2 Low Voltage Detector Control Register (LVDCON) .....	17-2
17.3 Voltage (VDD) Level Detection Sequence-LVD Usage .....	17-4
<b>18 Embedded Flash Memory Interface.....</b>	<b>18-1</b>
18.1 Overview .....	18-1
18.1.1 Flash ROM Configuration .....	18-1
18.1.2 Tool Program Mode .....	18-2
18.1.3 User Program Mode .....	18-2
18.2 Flash Memory Control Registers (User Program Mode) .....	18-3
18.2.1 Flash Memory Control Register (FMCON) .....	18-3
18.2.2 Flash Memory User Programming Enable Register (FMUSR).....	18-3
18.2.3 Flash Memory Sector Address Registers .....	18-4
18.3 ISP™ (On-Board Programming) Sector.....	18-5
18.3.1 ISP Reset Vector and ISP Sector Size .....	18-6
18.4 Sector Erase.....	18-7
18.5 Programming.....	18-9
18.6 Reading .....	18-14
18.7 Hard Lock Protection.....	18-15
<b>19 Electrical Data.....</b>	<b>19-1</b>
19.1 Overview .....	19-1
<b>20 Mechanical Data .....</b>	<b>20-1</b>
20.1 Overview .....	20-1
<b>21 Flash MCU.....</b>	<b>21-1</b>
21.1 Overview .....	21-1
21.2 On Board Writing.....	21-4
<b>22 Development Tools.....</b>	<b>22-6</b>
22.1 Overview .....	22-6
22.2 Emulator-based Development System .....	22-6
22.2.1 Host Software .....	22-7
22.2.2 Target Boards .....	22-7
22.2.3 SMDS2+ Selection (SAM8) .....	22-10
22.3 Zilog Library-based Development Platform.....	22-14
22.3.1 Zilog Developer Platform Components.....	22-14
22.3.2 Compatibility with 3 <sup>rd</sup> Party Tools .....	22-16
22.3.3 Benefits and Limitations of Zilog Development Tools.....	22-16
22.3.4 Development Tools .....	22-16

## List of Figures

Figure Number	Title	Page Number
Figure 1-1	Block Diagram.....	1-5
Figure 1-2	Pin Assignment Diagram (24-Pin SOP Package).....	1-6
Figure 1-3	Pin Assignment Diagram (24-Pin DIP/SOP/SSOP Package).....	1-7
Figure 1-4	Pin Circuit Type A .....	1-10
Figure 1-5	Pin Circuit Type B .....	1-10
Figure 1-6	Pin Circuit Type C .....	1-10
Figure 1-7	Pin Circuit Type D .....	1-11
Figure 1-8	Pin Circuit Type E .....	1-11
Figure 1-9	Pin Circuit Type E-1 .....	1-12
Figure 1-10	Pin Circuit Type E-2 .....	1-12
Figure 2-1	Program Memory Address Space.....	2-2
Figure 2-2	Smart Option .....	2-3
Figure 2-3	Internal Register File Organization .....	2-6
Figure 2-4	Register Page Pointer (PP).....	2-7
Figure 2-5	Set 1, Set 2 and Prime Area Register Map.....	2-8
Figure 2-6	8-Byte Working Register Areas (Slices) .....	2-9
Figure 2-7	Contiguous 16-Byte Working Register Block.....	2-10
Figure 2-8	Non-Contiguous 16-Byte Working Register Block.....	2-11
Figure 2-9	16-Bit Register Pair .....	2-12
Figure 2-10	Register File Addressing.....	2-13
Figure 2-11	Common Working Register Area .....	2-14
Figure 2-12	4-Bit Working Register Addressing.....	2-16
Figure 2-13	4-Bit Working Register Addressing Example.....	2-16
Figure 2-14	8-Bit Working Register Addressing.....	2-17
Figure 2-15	8-Bit Working Register Addressing Example.....	2-18
Figure 2-16	Stack Operations .....	2-19
Figure 3-1	Register Addressing.....	3-2
Figure 3-2	Working Register Addressing .....	3-2
Figure 3-3	Indirect Register Addressing to Register File .....	3-3
Figure 3-4	Indirect Register Addressing to Program Memory.....	3-4
Figure 3-5	Indirect Working Register Addressing to Register File .....	3-5
Figure 3-6	Indirect Working Register Addressing to Program or Data Memory.....	3-6
Figure 3-7	Indexed Addressing to Register File .....	3-7
Figure 3-8	Indexed Addressing to Program or Data Memory with Short Offset.....	3-8
Figure 3-9	Indexed Addressing to Program or Data Memory .....	3-9
Figure 3-10	Direct Addressing for Load Instructions.....	3-10
Figure 3-11	Direct Addressing for Call and Jump Instructions.....	3-11
Figure 3-12	Indirect Addressing .....	3-12
Figure 3-13	Relative Addressing .....	3-13
Figure 3-14	Immediate Addressing .....	3-14
Figure 4-1	Register Description Format .....	4-5
Figure 5-1	S3C8/S3F8 Series Interrupt Types.....	5-2
Figure 5-2	S3F8S28/S3F8S24 Interrupt Structure.....	5-3
Figure 5-3	ROM Vector Address Area .....	5-4

Figure 5-4	Interrupt Function Diagram .....	5-6
Figure 5-5	System Mode Register (SYM) .....	5-8
Figure 5-6	Interrupt Mask Register (IMR).....	5-9
Figure 5-7	Interrupt Request Priority Groups .....	5-10
Figure 5-8	Interrupt Priority Register (IPR).....	5-11
Figure 5-9	Interrupt Request Register (IRQ) .....	5-12
Figure 6-1	System Flags Register (FLAGS).....	6-5
Figure 7-1	Main Oscillator Circuit (Crystal/Ceramic Oscillator) .....	7-1
Figure 7-2	System Clock Control Register (CLKCON) .....	7-3
Figure 7-3	Ring Oscillator Control Register (ROSCCON).....	7-4
Figure 7-4	Stop Control Register (STOPCON) .....	7-4
Figure 7-5	System Clock Circuit Diagram .....	7-5
Figure 8-1	Low Voltage Reset Circuit.....	8-2
Figure 8-2	Recommended External Reset Circuit.....	8-3
Figure 8-3	Reset Block Diagram .....	8-4
Figure 8-4	Timing for S3F8S28/S3F8S24 After Reset.....	8-4
Figure 9-1	Port Data Register Format.....	9-2
Figure 9-2	Port 0 Circuit Diagram.....	9-3
Figure 9-3	Port 0 Control Register (P0CONH, High Byte) .....	9-4
Figure 9-4	Port 0 Control Register (P0CONL, Low Byte).....	9-5
Figure 9-5	Port 0 Interrupt Pending Registers (P0PND) .....	9-6
Figure 9-6	Port 0 Pull-Up Resistor Enable Registers (P0PUR) .....	9-7
Figure 9-7	Port 1 Circuit Diagram.....	9-8
Figure 9-8	Port 1 Control Register (P1CON).....	9-9
Figure 9-9	Port 2 Circuit Diagram.....	9-10
Figure 9-10	Port 2 Control Register (P2CONH, High Byte) .....	9-11
Figure 9-11	Port 2 Control Register (P2CONL, Low Byte).....	9-12
Figure 9-12	Port 2 Open-Drain Output Mode Register (P2PUR) .....	9-13
Figure 9-13	Port 3 Circuit Diagram.....	9-14
Figure 9-14	Port 3 Control Register (P3CON).....	9-15
Figure 9-15	Port 3 Interrupt Pending Register (P3PND) .....	9-16
Figure 10-1	Basic Timer Control Register (BTCON).....	10-2
Figure 10-2	Oscillation Stabilization Time on RESET .....	10-4
Figure 10-3	Oscillation Stabilization Time on Stop Mode Release .....	10-5
Figure 10-4	Timer 0 Control Register (TACON).....	10-8
Figure 10-5	Timer 0 Timing Diagram .....	10-9
Figure 10-6	Timer 0 Functional Block Diagram.....	10-10
Figure 10-7	Timer A Control Register (TACON) .....	10-12
Figure 10-8	Timer B Control Register (TBCON) .....	10-13
Figure 10-9	Timer A and B Function Block Diagram.....	10-14
Figure 11-1	Simplified Timer 1 Function Diagram: Interval Mode.....	11-2
Figure 11-2	Simplified Timer 1 Function Diagram: Capture Mode.....	11-3
Figure 11-3	Timer 1 Control Register (T1CON) .....	11-5
Figure 11-4	Timer 1 Prescaler Register (T1PS).....	11-6
Figure 11-5	Timer 1 Data Register High (T1DATAH) .....	11-6
Figure 11-6	Timer 1 Data Register Low (T1DATAL).....	11-6
Figure 11-7	Timer 1 Functional Block Diagram.....	11-7
Figure 12-1	Watchdog Timer Control Register (WDTCON).....	12-4
Figure 12-2	Watchdog Timer Functional Block Diagram .....	12-5
Figure 12-3	Interrupt Operation Sequence.....	12-6
Figure 12-4	System Reset Operation Sequence .....	12-7
Figure 12-5	Interrupt & System Reset Operation Sequence.....	12-8
Figure 13-1	PWM Data and Extension Registers .....	13-3
Figure 13-2	PWM Basic Waveform (6-Bit Base).....	13-5
Figure 13-3	Extended PWM Waveform (6-Bit Base + 6-Bit Extension) .....	13-6

Figure 13-4	PWM Basic Waveform (6-Bit Base) .....	13-7
Figure 13-5	Extended PWM Waveform (6-Bit Base + 2-Bit Extension) .....	13-8
Figure 13-6	PWM Basic Waveform (8-Bit Base) .....	13-9
Figure 13-7	PWM Basic Waveform (8-Bit Base + 6-Bit Extension) .....	13-10
Figure 13-8	PWM Control Register (PWM0CON, PWM1CON) .....	13-11
Figure 13-9	PWM Extension Register (PWM0EX, PWM1EX) .....	13-12
Figure 13-10	PWM Data Register (PWM0DATA PWM1DATA) .....	13-12
Figure 13-11	PWM/Capture Module Functional Block Diagram .....	13-13
Figure 14-1	A/D Converter Control Register (ADCON).....	14-3
Figure 14-2	A/D Converter Circuit Diagram .....	14-4
Figure 14-3	A/D Converter Data Register (ADDATAH/L) .....	14-4
Figure 14-4	A/D Converter Timing Diagram .....	14-5
Figure 14-5	Recommended A/D Converter Circuit for Highest Absolute Accuracy .....	14-7
Figure 15-1	UART Control Register (UARTCON) .....	15-2
Figure 15-2	UART Interrupt Pending Register (UARTPND) .....	15-3
Figure 15-3	UART Data Register (UDATA).....	15-4
Figure 15-4	UART Baud Rate Data Register (BRDATA) .....	15-4
Figure 15-5	UART Functional Block Diagram .....	15-6
Figure 15-6	Timing Diagram for UART Mode 0 Operation.....	15-7
Figure 15-7	Timing Diagram for UART Mode 1 Operation.....	15-8
Figure 15-8	Timing Diagram for UART Mode 2 Operation.....	15-9
Figure 15-9	Timing Diagram for UART Mode 3 Operation.....	15-10
Figure 15-10	Connection Example for Multiprocessor Serial Data Communications .....	15-12
Figure 16-1	Multi-Master IIC Bus Control Register (ICCR) .....	16-2
Figure 16-2	Multi-Master IIC Bus Control/Status Register (ICSR) .....	16-4
Figure 16-3	Multi-Master IIC Bus Tx/Rx Data Shift Register (IDSR) .....	16-5
Figure 16-4	Multi-Master IIC Bus Address Register (IAR) .....	16-6
Figure 16-5	IIC Bus Block Diagram.....	16-7
Figure 16-6	Start and Stop Conditions .....	16-8
Figure 16-7	Input Data Protocol .....	16-9
Figure 16-8	Interrupt Pending Information .....	16-9
Figure 16-9	IIC Bus Interface Data Formats .....	16-10
Figure 16-10	Acknowledge Response from Receiver.....	16-11
Figure 16-11	Write Operation Sequence.....	16-12
Figure 16-12	Read Operation Sequence .....	16-13
Figure 17-1	LVD Control Register (LVDCON).....	17-2
Figure 17-2	Block Diagram for Low Voltage Detector.....	17-3
Figure 18-1	Flash Memory Control Register (FMCON) .....	18-3
Figure 18-2	Flash Memory User Programming Enable Register (FMUSR).....	18-3
Figure 18-3	Flash Memory Sector Address Register (FMSECH) .....	18-4
Figure 18-4	Flash Memory Sector Address Register (FMSECL) .....	18-4
Figure 18-5	Program Memory Address Space.....	18-5
Figure 18-6	Sector configurations in User Program Mode .....	18-7
Figure 18-7	Sector Erase Flowchart in User Program Mode .....	18-8
Figure 18-8	Byte Program Flowchart in a User Program Mode.....	18-10
Figure 18-9	Program Flowchart in a User Program Mode .....	18-11
Figure 19-1	Input Timing Measurement Points .....	19-4
Figure 19-2	Operating Voltage Range .....	19-5
Figure 19-3	Schmitt Trigger Input Characteristics Diagram .....	19-6
Figure 19-4	Stop Mode Release Timing When Initiated by a RESET .....	19-6
Figure 19-5	Waveform for UART Timing Characteristics .....	19-7
Figure 19-6	LVR Reset Timing.....	19-9
Figure 19-7	The Circuit Diagram to Improve EFT Characteristics .....	19-10
Figure 20-1	24-SOP-375 Package Dimensions .....	20-1
Figure 20-2	24-TSSOP-BD44 Package Dimensions .....	20-2

---

Figure 20-3	20-DIP-300A Package Dimensions .....	20-3
Figure 20-4	20-SOP-375 Package Dimensions .....	20-4
Figure 20-5	20-SSOP-225 Package Dimensions.....	20-5
Figure 21-1	S3F8S28/S3F8S24 Pin Assignments (24-DIP/24-SOP) .....	21-2
Figure 21-2	S3F8S28/S3F8S24 Pin Assignments (20-DIP/20-SOP/20-SSOP) .....	21-2
Figure 21-3	PCB Design Guide for on Board Programming .....	21-4
Figure 22-1	Emulator-based Development System Configuration.....	22-6
Figure 22-2	TB8S19/8S28/8S39 Target Board Configuration.....	22-8
Figure 22-3	DIP Switch for Smart Option .....	22-12
Figure 22-4	24-Pin Connector for TB8S19/8S28/8S39.....	22-13
Figure 22-5	S3F8S28/S3F8S24 Probe Adapter for 24 Pin Package .....	22-13
Figure 22-6	Zilog Development Platform.....	22-14
Figure 22-7	PCB Design Guide for In System Programming.....	22-15

## List of Tables

Table Number	Title	Page Number
Table 1-1	S3F8S28/S3F8S24 Pin Descriptions .....	1-8
Table 1-2	Descriptions of Pins Used to Read/Write the Flash ROM .....	1-9
Table 2-1	Register Type Summary .....	2-5
Table 4-1	System and Peripheral Control Registers, Set1 .....	4-2
Table 4-2	System and Peripheral Control Registers, Set1, Bank0 .....	4-3
Table 4-3	System and Peripheral Control Registers, Set1, Bank1 .....	4-4
Table 5-1	Interrupt Control Register Overview.....	5-5
Table 5-2	Interrupt Source Control and Data Registers.....	5-7
Table 6-1	Instruction Group Summary .....	6-2
Table 6-2	Flag Notation Conventions.....	6-7
Table 6-3	Instruction Set Symbols .....	6-7
Table 6-4	Instruction Notation Conventions .....	6-8
Table 6-5	OPCODE Quick Reference.....	6-9
Table 6-6	OPCODE Quick Reference.....	6-10
Table 6-7	Condition Codes.....	6-11
Table 8-1	Register Values After a Reset, Set1 .....	8-7
Table 8-2	Register Values After a Reset, Set1, Bank0 .....	8-8
Table 8-3	System and Peripheral Control Registers, Set1, Bank1 .....	8-9
Table 9-1	S3F8S28/S3F8S24 Port Configuration Overview .....	9-1
Table 9-2	Port Data Register Summary .....	9-2
Table 12-1	Watchdog Timer Prescaler Select .....	12-5
Table 13-1	PWM Control and Data Registers.....	13-3
Table 13-2	PWM Output "Stretch" Values for Extension Data Bits Ext1 (PWM0EX.7-2, PWM1EX.7-2).....	13-5
Table 13-3	PWM Output "Stretch" Values for Extension Data Bits Ext0 (PWM0EX.7-6, PWM1EX.7-6).....	13-7
Table 13-4	PWM Output "Stretch" Values for Extension Data Bits Ext1 (PWM0EX.7-2, PWM1EX.7-2).....	13-9
Table 15-1	Commonly Used Baud Rates Generated by 8-Bit BRDATA.....	15-5
Table 16-1	Sample Timing Calculations for the IIC Bus Transmit Clock (SCL).....	16-3
Table 18-1	Descriptions of Pins Used to Read/Write the Flash in Tool Program Mode .....	18-2
Table 18-2	ISP Sector Size .....	18-6
Table 18-3	Reset Vector Address .....	18-6
Table 19-1	Absolute Maximum Ratings .....	19-1
Table 19-2	DC Electrical Characteristics.....	19-2
Table 19-3	AC Electrical Characteristics.....	19-3
Table 19-4	Oscillator Characteristics .....	19-4
Table 19-5	Oscillation Stabilization Time .....	19-5
Table 19-6	Data Retention Supply Voltage in Stop Mode.....	19-6
Table 19-7	UART Timing Characteristics in Mode 0 (10MHz) .....	19-7
Table 19-8	A/D Converter Electrical Characteristics .....	19-8
Table 19-9	LVD Circuit Characteristics .....	19-9
Table 19-10	LVR Circuit Characteristics .....	19-9
Table 19-11	Flash Memory AC Electrical characteristics.....	19-10
Table 19-12	ESD Characteristics .....	19-11
Table 21-1	Descriptions of Pins Used to Read/Write the EPROM .....	21-3
Table 21-2	Comparison of S3F8S28/S3F8S24 Features .....	21-3

---

Table 21-3	Reference Table for Connection .....	21-5
Table 22-1	Components of TB8S19/8S28/8S39 .....	22-9
Table 22-2	Device Selection Settings for TB8S19/8S28/8S39 .....	22-9
Table 22-3	Power Selection Settings for TB8S19/8S28/8S39 .....	22-10
Table 22-4	The SMDS2+ Tool Selection Setting .....	22-10
Table 22-5	Using Single Header Pins to Select Clock Source/PWM/Operation Mode .....	22-11
Table 22-6	Using Single Header Pins as the Input Path for External Trigger Sources .....	22-12
Table 22-7	ISP II Circuit Recommended Values .....	22-15



# List of Examples

<b>Example Number</b>	<b>Title</b>	<b>Page Number</b>
Example 2-1	Smart Option Setting .....	2-4
Example 2-2	Setting the Register Pointers .....	2-10
Example 2-3	Using the RPs to Calculate the Sum of a Series of Registers .....	2-11
Example 2-4	Addressing the Common Working Register Area .....	2-15
Example 2-5	Standard Stack Operations Using PUSH and POP .....	2-20
Example 8-1	Sample S3F8S28/S3F8S24 Initialization Routine .....	8-10
Example 10-1	Configuring the Basic Timer .....	10-6
Example 11-1	Using the Timer 1 .....	11-8
Example 13-1	Programming the PWM Module to Sample Specifications .....	13-14
Example 14-1	Configuring A/D Converter .....	14-8
Example 17-1	LVD Using Method .....	17-4
Example 18-1	Sector Erase .....	18-8
Example 18-2	1Byte Programming .....	18-12
Example 18-3	Reading .....	18-14
Example 18-4	Hard Lock Protection .....	18-15

# 1 Product Overview

## 1.1 S3C8/S3F8 Series Microcontrollers

Zilog's S3C8/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.

## 1.2 S3F8S28/S3F8S24 Microcontroller

The S3F8S28/S3F8S24 single-chip CMOS micro-controller is fabricated using a highly advanced CMOS process and is based on Zilog'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 S3F8S28/S3F8S24 is a micro-controller with 8K/4Kbyte multi-time-programmable Flash ROM embedded.

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

- Three configurable I/O ports (22 pins)
- 18 interrupt sources with 18 vectors and 8 interrupt levels
- A 16-bit Timer 0 with one 16-bit timer or two 8-bit timer mode.
- A 16-bit Timer 1 with interval & Capture function.
- A free running Watchdog Timer with interrupt and Reset.
- Analog to digital converter with thirteen input channels (MAX.) and 12-bit resolution
- One UART module
- One IIC module
- Two PWM outputs with three optional mode: 12-bit (6 + 6); 8-bit (6 + 2); 14-bit (8 + 6);

The S3F8S28/S3F8S24 microcontroller is ideal for use in a wide range of electronic applications requiring simple timer/counter, PWM, ADC. S3F8S28/S3F8S24 is available in a 24/20-pin SOP Package and a 24-pin TSSOP package and a 20-pin DIP package.

## 1.3 Features

- CPU
  - SAM8RC CPU core
- Memory
  - Internal multi-time program Full-Flash memory:
    - 8K × 8 bits program memory (S3F8S28)
    - 4K × 8 bits program memory (S3F8S24)
      - Sector size: 128 bytes
      - User programmable by "LDC" instruction
      - Sector erase available
      - Fast programming time
      - External serial programming support
      - Endurance: 10,000 erase/program cycles
      - 10 Years data retention
  - 272byte general-purpose register area
- Instruction Set
  - 78 instructions
  - Idle and Stop instructions added for power-down modes
- Instruction Execution Time
  - 333ns at 12MHz  $f_{OSC}$  (minimum)
- Interrupts
  - 8 interrupt levels and 17 interrupt sources (8 external interrupt and 9 internal interrupt)
  - Fast interrupt processing feature
  - Watchdog interrupt can release Stop Mode.
- General I/O
  - Three I/O ports (Max. 22 pins)
  - Bit programmable ports
- 2-ch High-speed PWM with Three Selectable Resolutions
  - 12-bit PWM: 6-bit base + 6-bit extension
  - 8-bit PWM: 6-bit base + 2-bit extension
  - 14-bit PWM: 8-bit base + 6-bit extension

- Timer/Counters
  - One 8-bit basic timer for watchdog function
  - One 16-bit timer(Timer 0) or two 8-bit timers A/B with time interval mode
  - One 16-bit timer/counter (Timer 1) with two operating modes; Interval mode, Capture mode
  - One free running Watchdog Timer with programmable timer-out period. It can be used to generate RESET or release STOP when clocked by Ring Oscillator.
  
- A/D Converter
  - Thirteen analog input pins (Max.)
  - 12-bit conversion resolution
  - Integrated sample and hold circuitry
  
- Asynchronous UART
  - Programmable baud rate generator
  - Support serial data transmit/receive operations with 8-bit, 9-bit UART
  
- Multi-Master IIC-Bus
  - Serial Peripheral Interface
  - Serial, 8-bit Data Transfers
  - Programmable Clock Prescale
  
- Oscillation Frequency
  - 0.1MHz to 1MHz external low gain (LG) crystal oscillator
  - 0.4MHz to 12MHz external high gain (HG) crystal oscillator
  - Internal RC: 0.5MHz (typ.), 1 MHz (typ.), 2MHz (typ.), 4MHz (typ.), 8MHz (typ.), in VDD = 5V with 1% tolerance
  - On-Chip Ring oscillator with 32kHz frequency for free running Watchdog Timer.
  - Maximum 12MHz CPU clock
  
- Built-in RESET Circuit (LVR)
  - Low-Voltage check to make system reset
  - $V_{LVR} = 1.9/2.3/3.0/3.9V$  (by Smart Option)
  
- Low Voltage Detect Circuit (LVD)
  - Programmable detection voltage
  - $V_{LVD} = 2.1/2.5/3.2/4.1V$
  - En/Disable S/W selectable.
  
- Operating Temperature Range
  - $-40^{\circ}C$  to  $+85^{\circ}C$

- Operating Voltage Range
  - 1.8V to 5.5V @ 0.1 – 4MHz
  - 2.7V to 5.5V @ 0.1 – 12MHz
  
- Smart Option
  - LVR enable/disable
  - Oscillator selection
  
- Package Types
  - S3F8S28/F8S24:
    - 24-SOP-375
    - 24-TSSOP-BD44
    - 20-DIP-300A
    - 20-SOP-375
    - 20-SSOP-225

### 1.4 Block Diagram

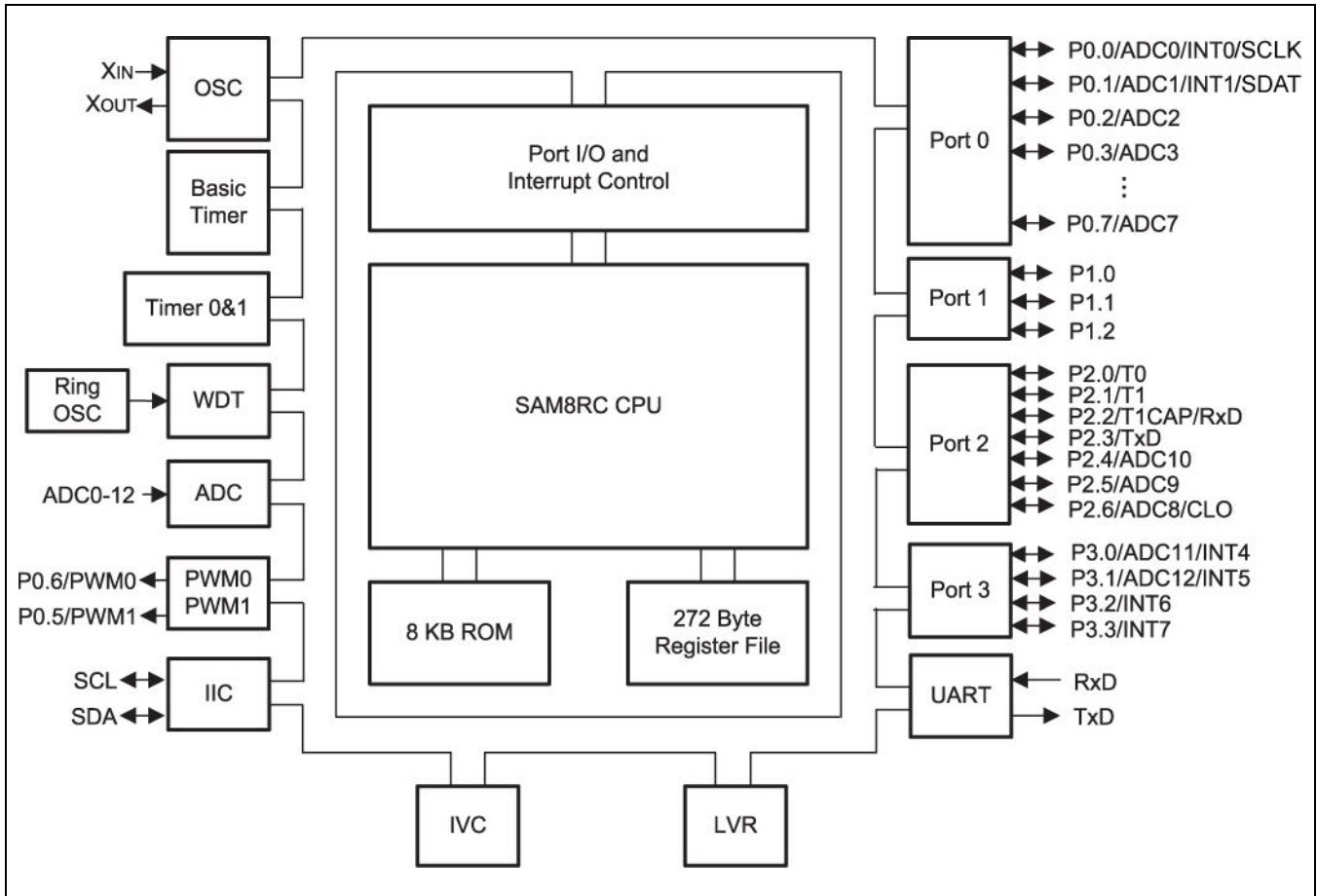


Figure 1-1 Block Diagram

Note: The Internal Voltage Converter (IVC) for the S3F8S28/S3F8S24 MCU's 0.13µm process is not configurable.

## 1.5 Pin Assignments

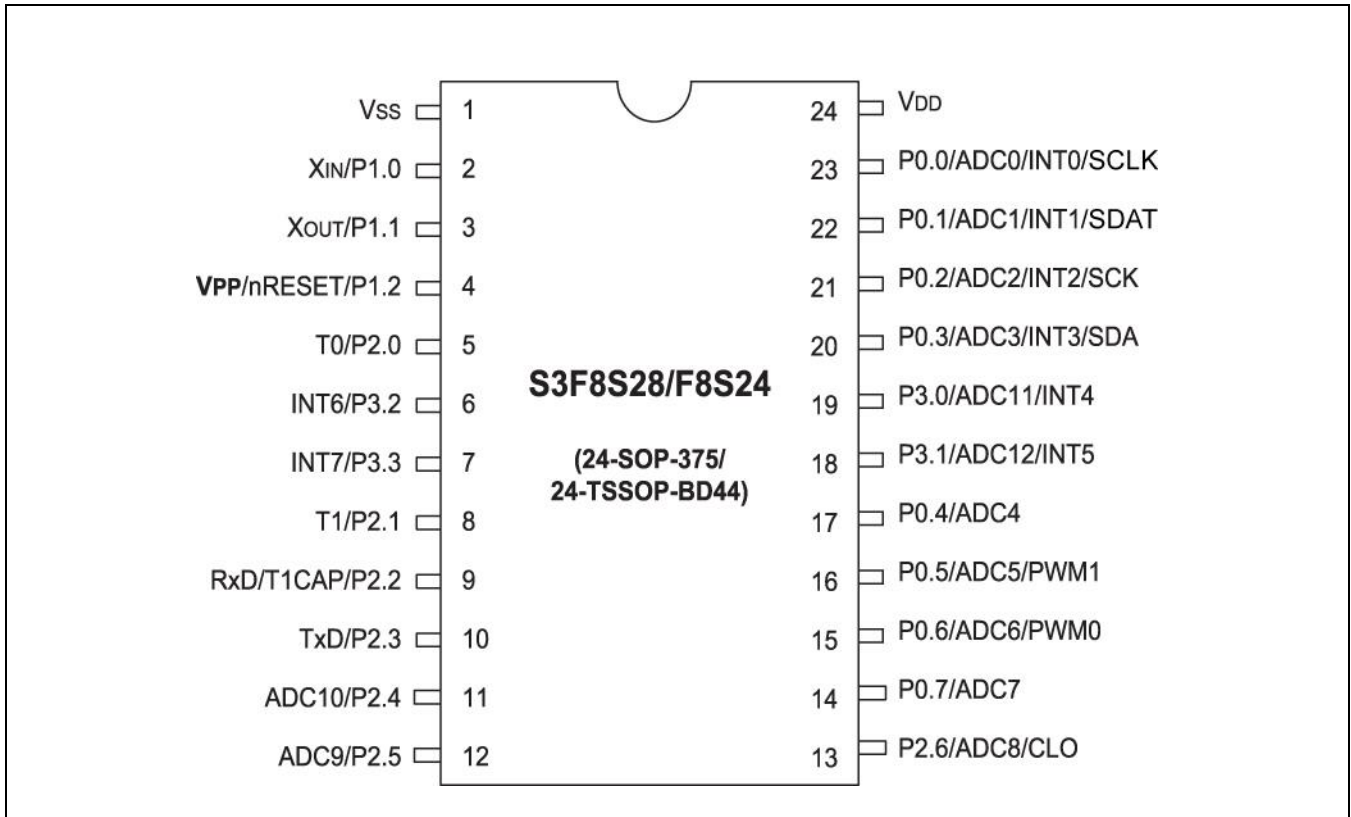
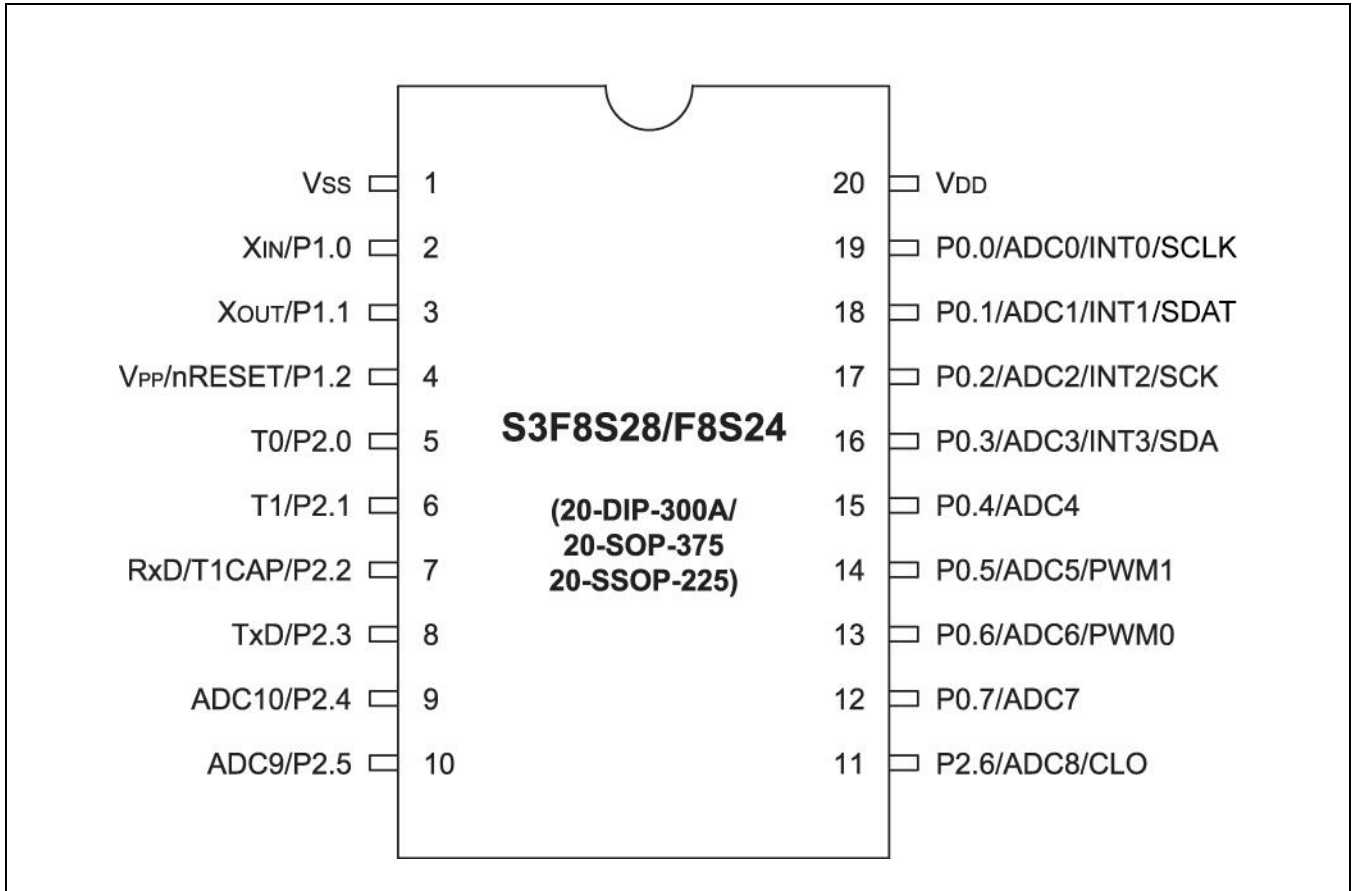


Figure 1-2 Pin Assignment Diagram (24-Pin SOP Package)



**Figure 1-3 Pin Assignment Diagram (24-Pin DIP/SOP/SSOP Package)**



## 1.6 Pin Descriptions

**Table 1-1 S3F8S28/S3F8S24 Pin Descriptions**

Pin Name	Input/Output	Pin Description	Pin Type	Share Pins
P0.0 to 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, external interrupt input P0.2-3 shared with IIC ports SCK and SDA	E-1	ADC0 to ADC7 INT0 to INT3 PWM, SCK SDA
P1.0 to P1.2	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. P1.2 is used as Schmitt trigger input port and Open-drain output	E-2 B	X <sub>IN</sub> , X <sub>OUT</sub> RESET
P2.0 to P2.6	I/O	Bit-programmable I/O port for Schmitt trigger input or push-pull, open-drain output(P2.6, P2.3-.0). Pull-up resistors are assignable by software. P2.2 can be used for T1CAP input. P2.2-3 shared with UART ports RxD and TxD	E	ADC8 to ADC10 CLO T0; T1; T1CAP; RxD,TxD
P3.0 to P3.3	I/O	Bit-programmable I/O port for Schmitt trigger input or push-pull. Pull-up resistors are assignable by software. Port3 pins can also be used as A/D converter input, external interrupt input	E-1	ADC11 to ADC12 INT4 to INT7
X <sub>IN</sub> , X <sub>OUT</sub>	–	Crystal/Ceramic oscillator signal for system clock.	–	P1.0 to P1.1
nRESET	I	Internal LVR or external RESET	B	P1.2
V <sub>DD</sub> , V <sub>SS</sub>	–	Voltage input pin and ground	–	–
CLO	O	System clock output port	E	P2.6
INT0 to INT7	I	External interrupt input port	E-1	P0.0 to P0.3 P3.0 to P3.3
PWM0	O	8-Bit high speed PWM0 output	E-1	P0.6
PWM1	O	8-Bit high speed PWM1 output	E-1	P0.5
T0	O	Timer0/A match output	E	P2.0
T1	O	Timer1 match output	E	P2.1
T1CAP	I	T1 capture input	E	P2.2
ADC0 to DC12	I	A/D converter input	E-1 E	P0.0 to P0.7 P2.4 to P2.6 P3.0 to P3.1
RxD	I/O	Serial data RxD pin for receive input and transmit output (mode 0)	E-1	P2.2
TxD	O	Serial data TxD pin for transmit output and shift clock output (mode 0)	E-1	P2.3
SCK, SDA	I/O	IIC Pins	E-1	P0.2, P0.3

**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	SDAT	22 (24-pin), 18 (20-pin)	I/O	Serial data pin (output when reading, Input when writing) Input and push-pull output port can be assigned
P0.0	SCLK	23(24-pin), 19 (20-pin)	I	Serial clock pin (input only pin)
RESET, P1.2	VPP	4	I	Power supply pin for Flash ROM cell writing (indicates that MTP enters into tool mode). When 11V is applied, MTP is in tool mode.
V <sub>DD</sub> /V <sub>SS</sub>	V <sub>DD</sub> /V <sub>SS</sub>	24 (24-pin), 20 (20-pin) 1 (24-pin), 1 (20-pin)	I	Logic power supply pin.

## 1.7 Pin Circuits

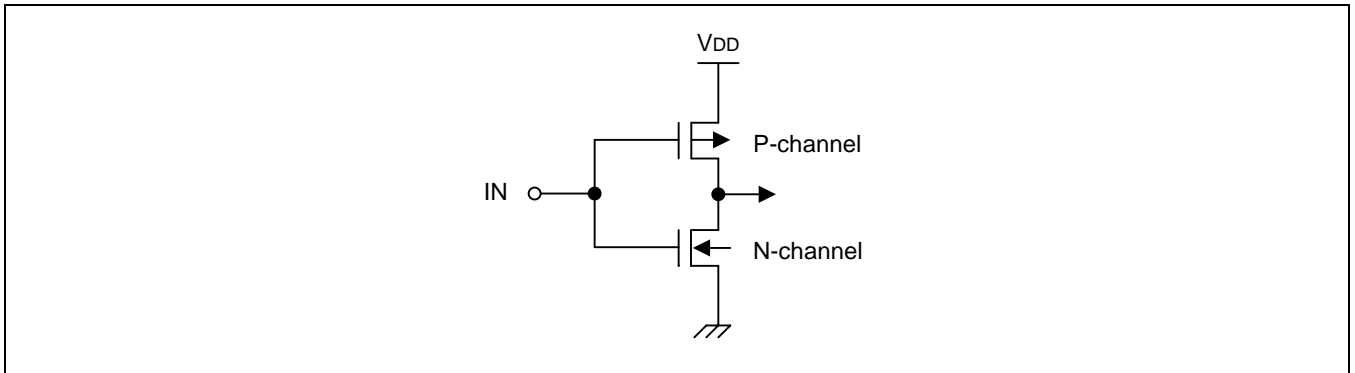


Figure 1-4 Pin Circuit Type A

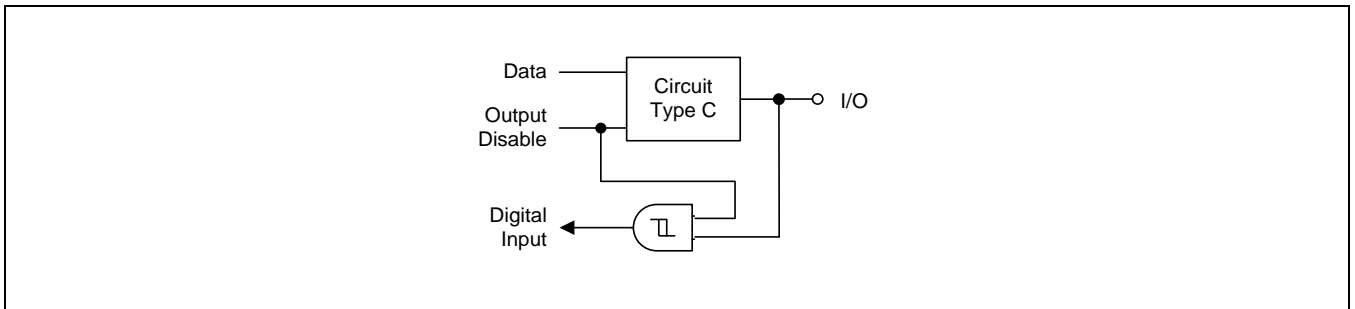


Figure 1-5 Pin Circuit Type B

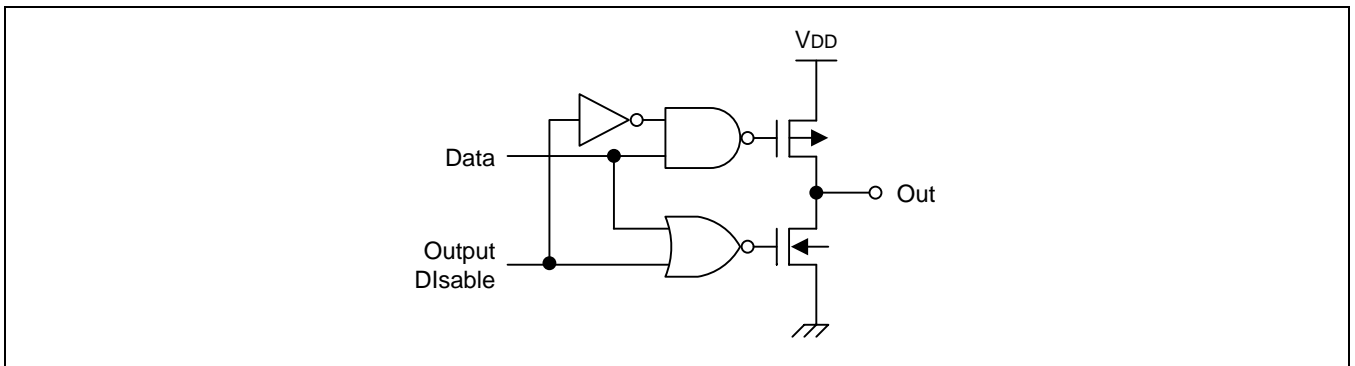


Figure 1-6 Pin Circuit Type C

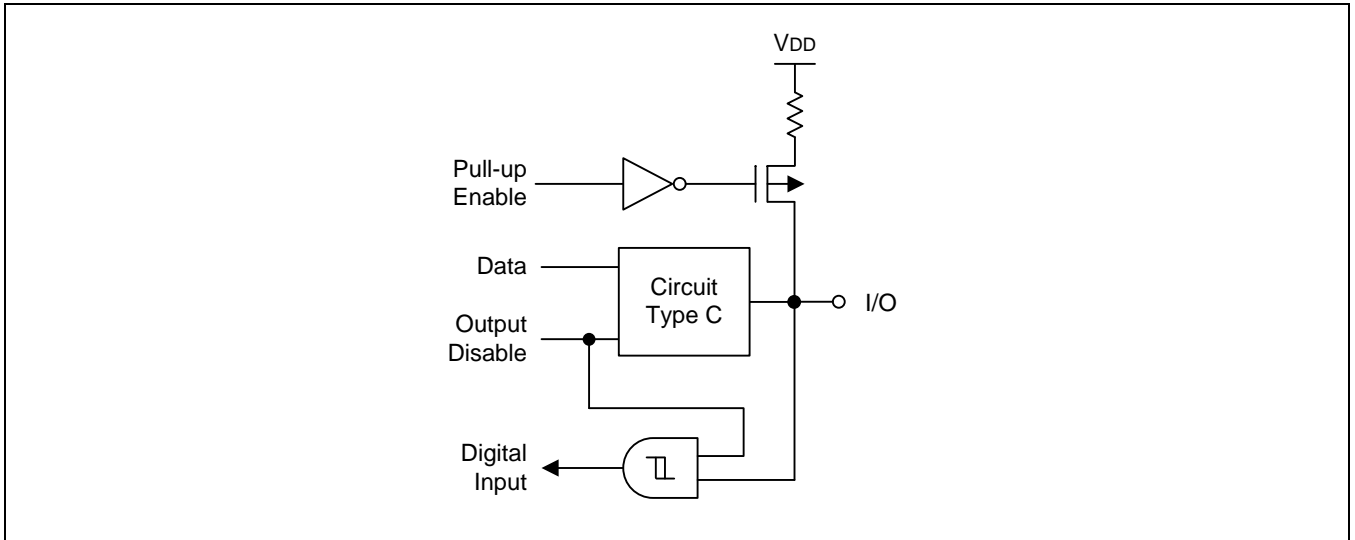


Figure 1-7 Pin Circuit Type D

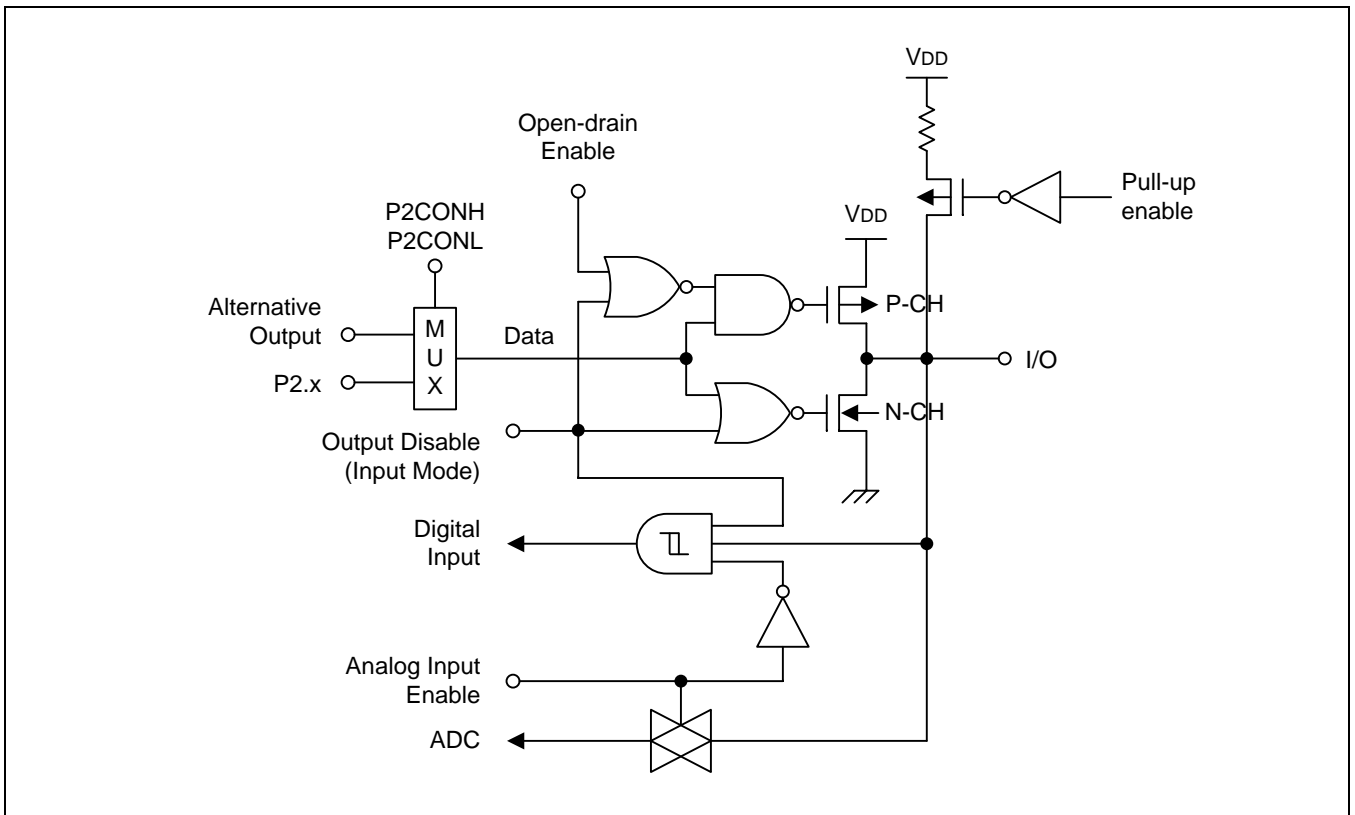


Figure 1-8 Pin Circuit Type E

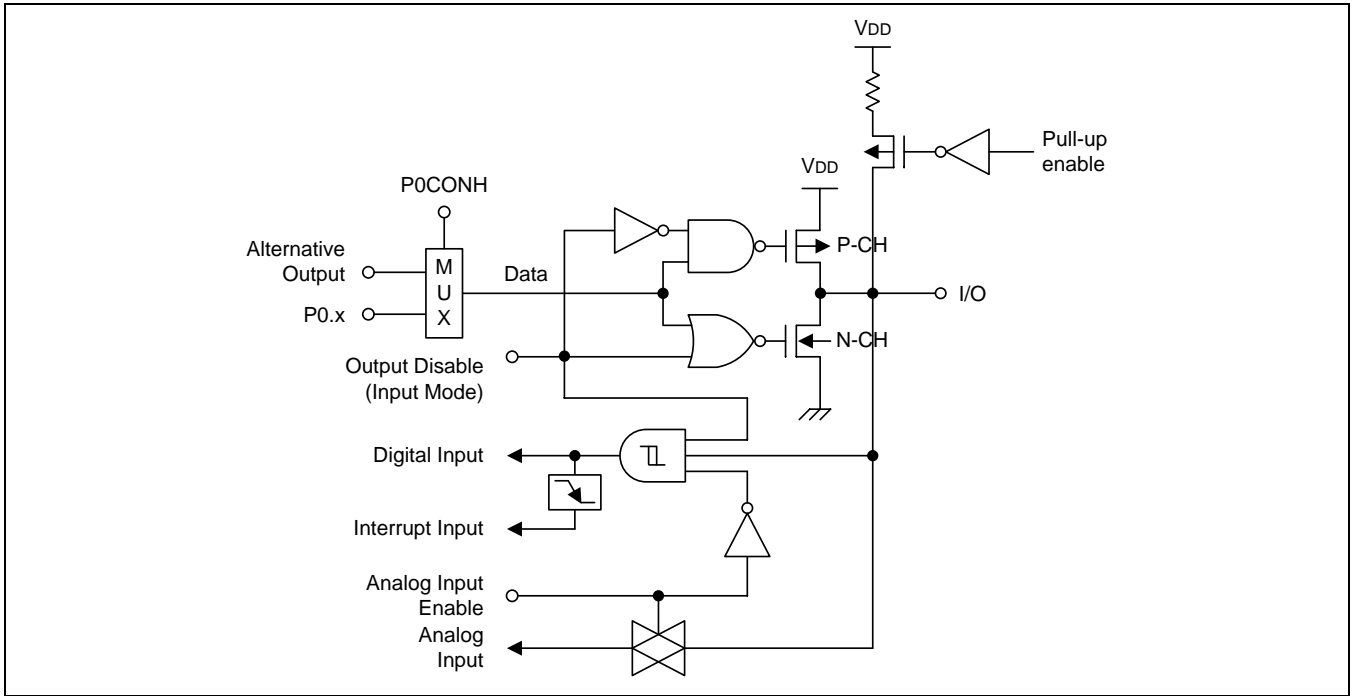


Figure 1-9 Pin Circuit Type E-1

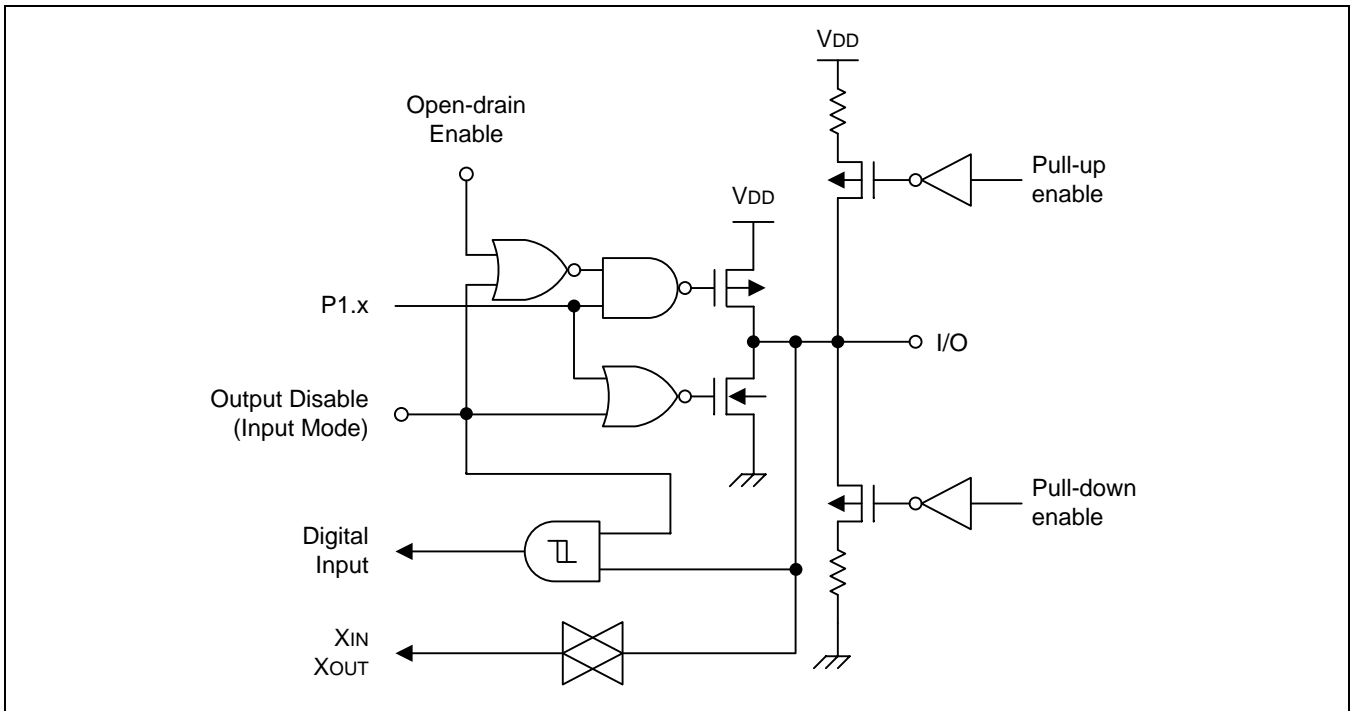


Figure 1-10 Pin Circuit Type E-2

# 2 Address Spaces

## 2.1 Overview

The S3F8S28/S3F8S24 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 S3F8S28/S3F8S24 have 8K/4Kbytes of multi-time-programmable Flash program memory: which is configured as the Internal ROM mode, all of the 8K/4Kbyte internal program memory is used.

The S3F8S28/S3F8S24 microcontroller has 272 general-purpose registers in its internal register file. 69bytes in the register file are mapped for system and peripheral control functions.

## 2.2 Program Memory (ROM)

### 2.2.1 Normal Operating Mode

The S3F8S28/S3F8S24 MCU has 8K/4Kbytes (locations 0H to 0FFFH) of internal multi-time-programmable Flash program memory.

The first 256 bytes of the ROM (0H to 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.

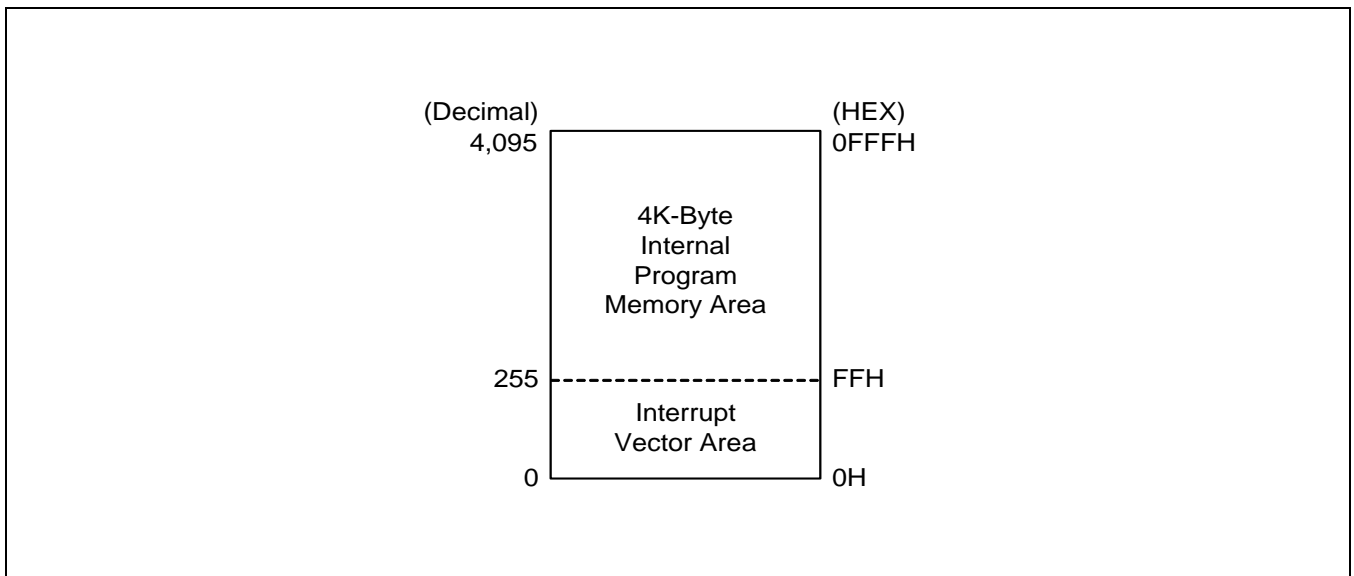
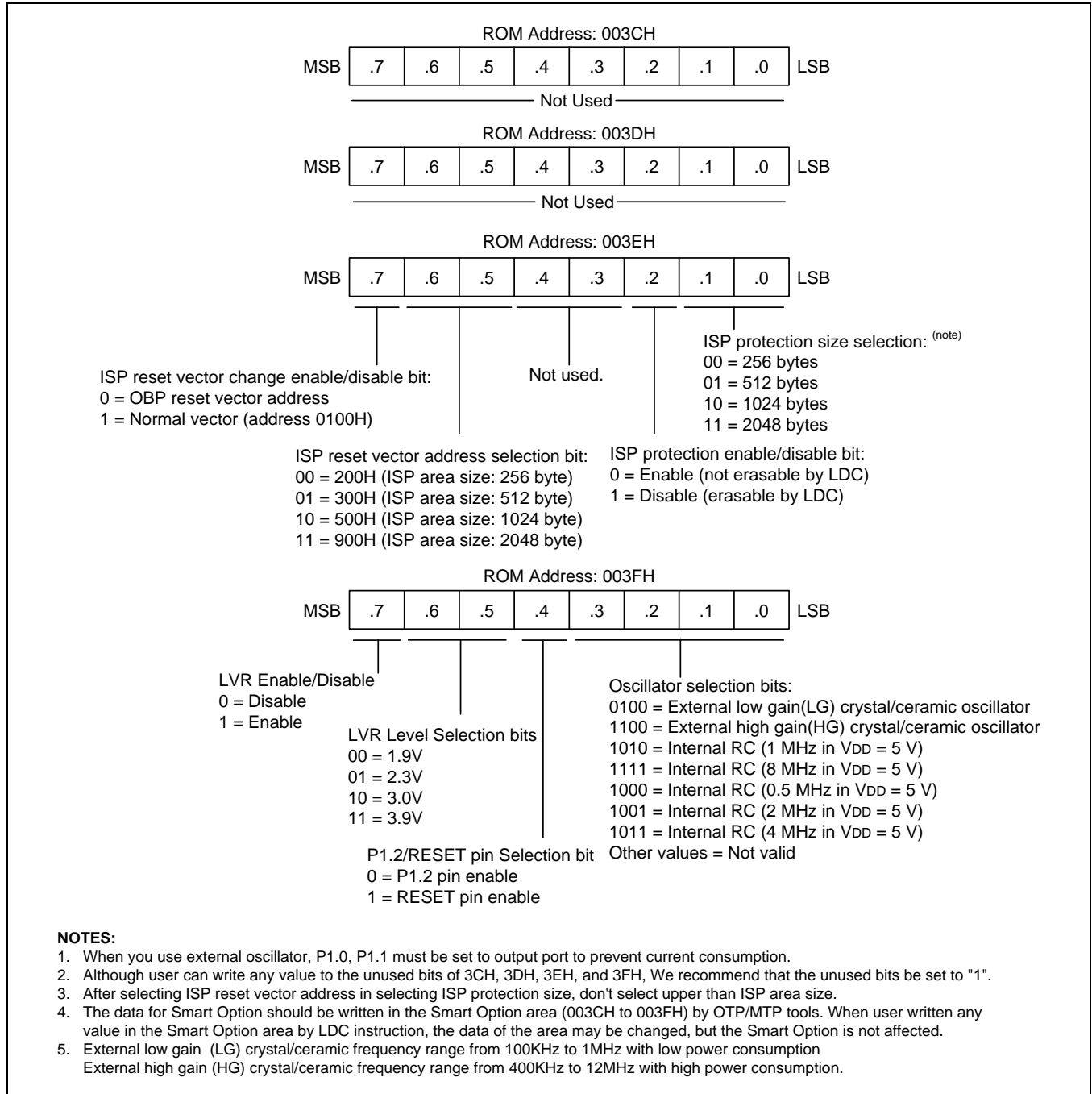


Figure 2-1 Program Memory Address Space

**2.2.2 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 S3F8S28/S3F8S24 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).



**Figure 2-2 Smart Option**



**Example 2-1 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, Internal RC oscillator 1MHz

;-----<< Interrupt Vector Address >>
VECTOR 0F6H, INT_TIMER0      ; Timer 0 interrupt
; << Reset >>
ORG          0100H
RESET:      DI
            .
            .
            .
```

## 2.3 Register Architecture

In the S3F8S28/S3F8S24 implementation, the upper 64byte area of register files is expanded two 64byte areas, called set 1 and set 2. The upper 32byte area of set 1 is further expanded two 32byte register banks (bank 0 and bank 1), and the lower 32byte area is a single 32byte common area.

The 64bytes of set 1 are addressed as working registers, system control registers and peripheral control registers. The 64bytes of set 2 are for general-purpose use, and commonly used for stack operations. You must use Register Indirect addressing mode or Indexed addressing mode to access registers in set 2.

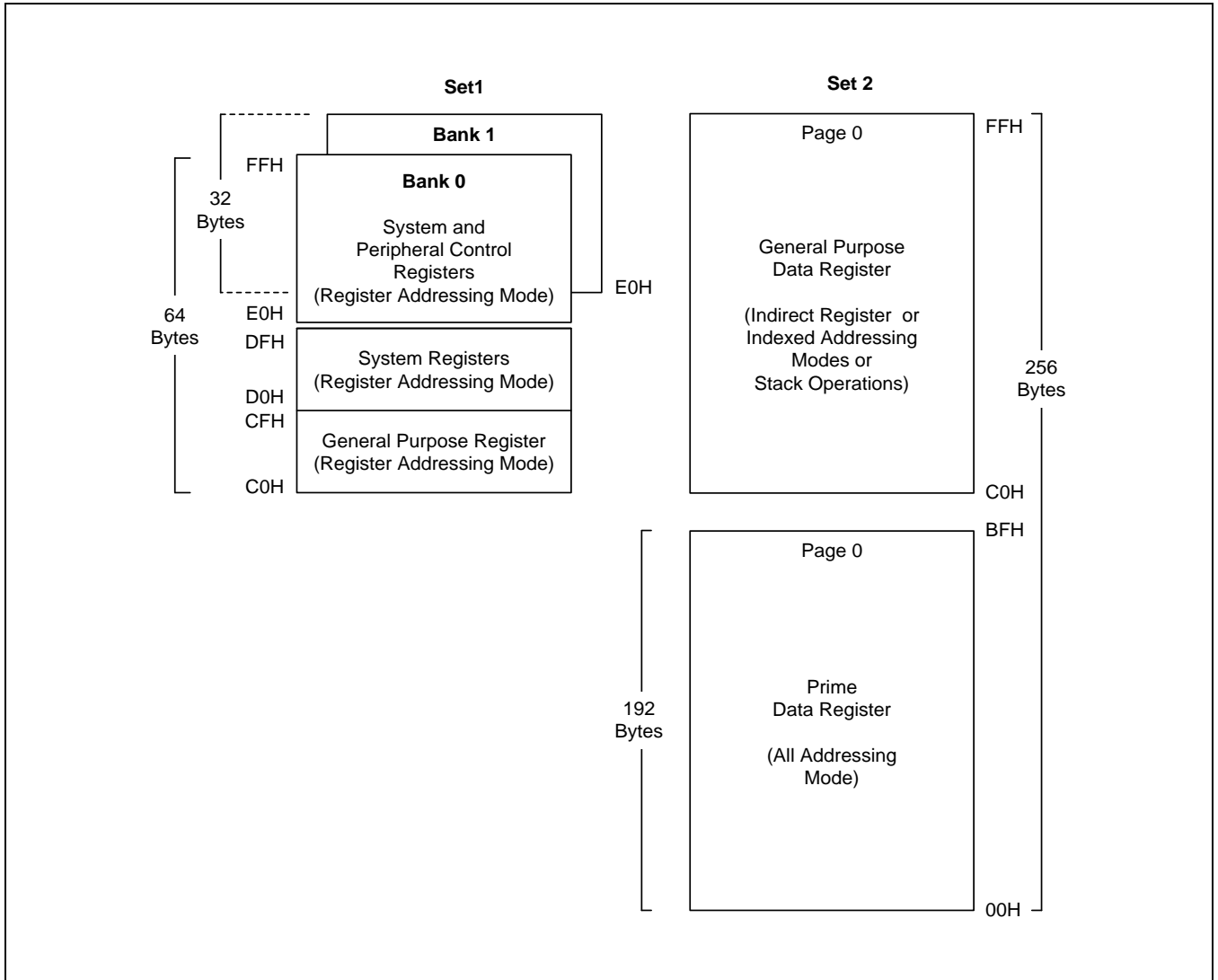
In case of S3F8S28/S3F8S24 the total number of addressable 8-bit registers is 341. Of these 341 registers, 69bytes are for CPU and system control registers and peripheral control and data registers, 16 bytes are used as shared working registers, and 256 registers are for general-purpose use.

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 to BFH: page0). This register file expansion is not implemented in the S3F8S28/S3F8S24 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	69
General-purpose registers (including the 16-bit common working register area)	272
Total Addressable Bytes	341

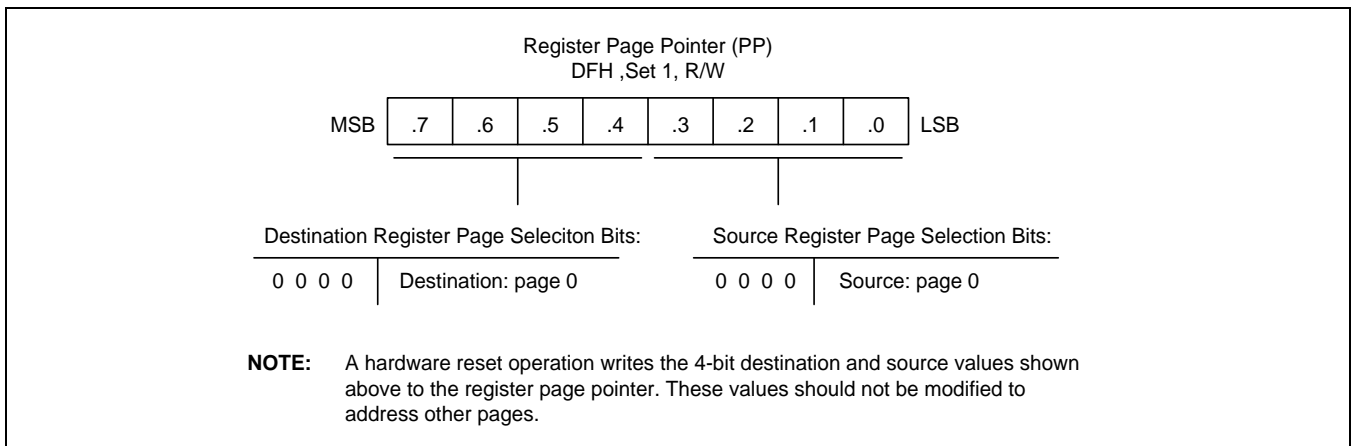


**Figure 2-3 Internal Register File Organization**

### 2.3.1 Register Page Pointer (PP)

The S3C8/S3F8 Series architecture supports the logical expansion of the physical 314byte internal register files (using an 8-bit data bus) into as many as 16 separately addressable register pages. Page addressing is controlled by the register page pointer PP (DFH, Set 1, Bank0). In the S3F8S28/S3F8S24 microcontroller, a paged register file expansion is not implemented and the register page pointer settings therefore always point to "page 0".

Following a reset, the page pointer's source value (lower nibble) and destination value (upper nibble) are always "0000" automatically. Therefore, S3F8S28/S3F8S24 is always selected page 0 as the source and destination page for register addressing. These page pointer (PP) register settings, as shown in [Figure 2-4](#), should not be modified during normal operation.



**Figure 2-4 Register Page Pointer (PP)**

### 2.3.2 Register Set 1

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

The upper 32byte area of set 1, (E0H to FFH) contains 27 mapped system and peripheral control registers. The lower 32byte area contains 15 system registers (D0H to DFH) and a 16byte common working register area (C0H to CFH). You can use the common working register area as a "scratch" area for data operations being performed in other areas of the register file.

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

### 2.3.3 Register Set 2

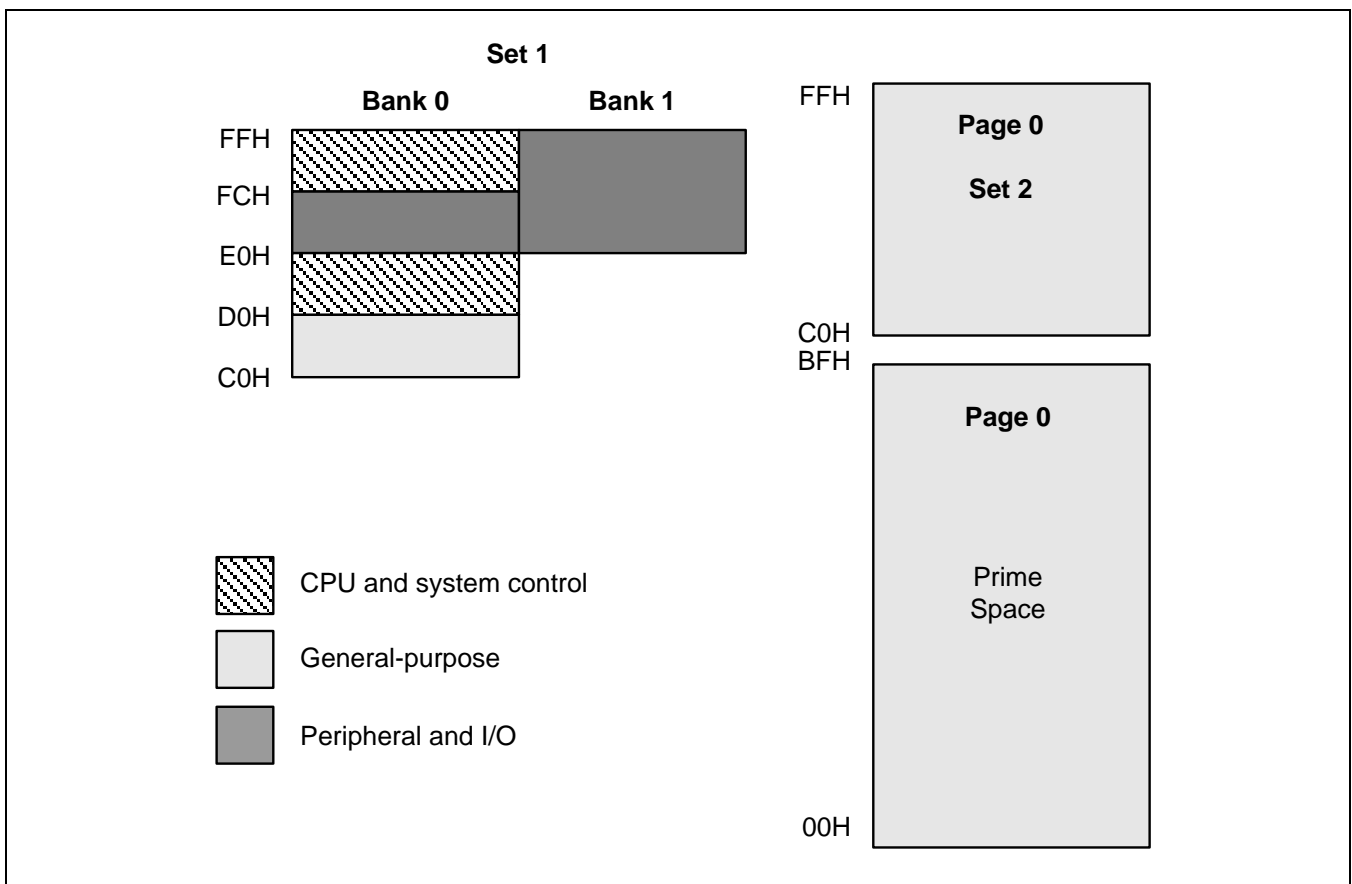
The same 64byte physical space that is used for set 1 locations C0H to FFH is logically duplicated to add another 64 bytes of register space. This expanded area of the register file is called set 2. The set 2 locations (C0H to FFH) is accessible on page 0 in the S3F8S28/S3F8S24 register space.

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

The set 2 register area is commonly used for stack operations.

### 2.3.4 Prime Register Space

The lower 192 bytes of the 256byte physical internal register file (00H to BFH) are called the prime register space or, more simply, the prime area. You can access registers in this address using any addressing mode. (In other words, there is no addressing mode restriction for these registers, as is the case for set 1 and set 2 registers.). The prime register area on page 0 is immediately addressable following a reset.



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

### 2.3.5 Working Registers

Instructions can access specific 8-bit registers or 16-bit register pairs using either 4-bit or 8-bit address fields. When 4-bit working register addressing is used, the 256byte register file can be seen by the programmer as consisting of 328byte register groups or "slices." Each slice consists of eight 8-bit registers.

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

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

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

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

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

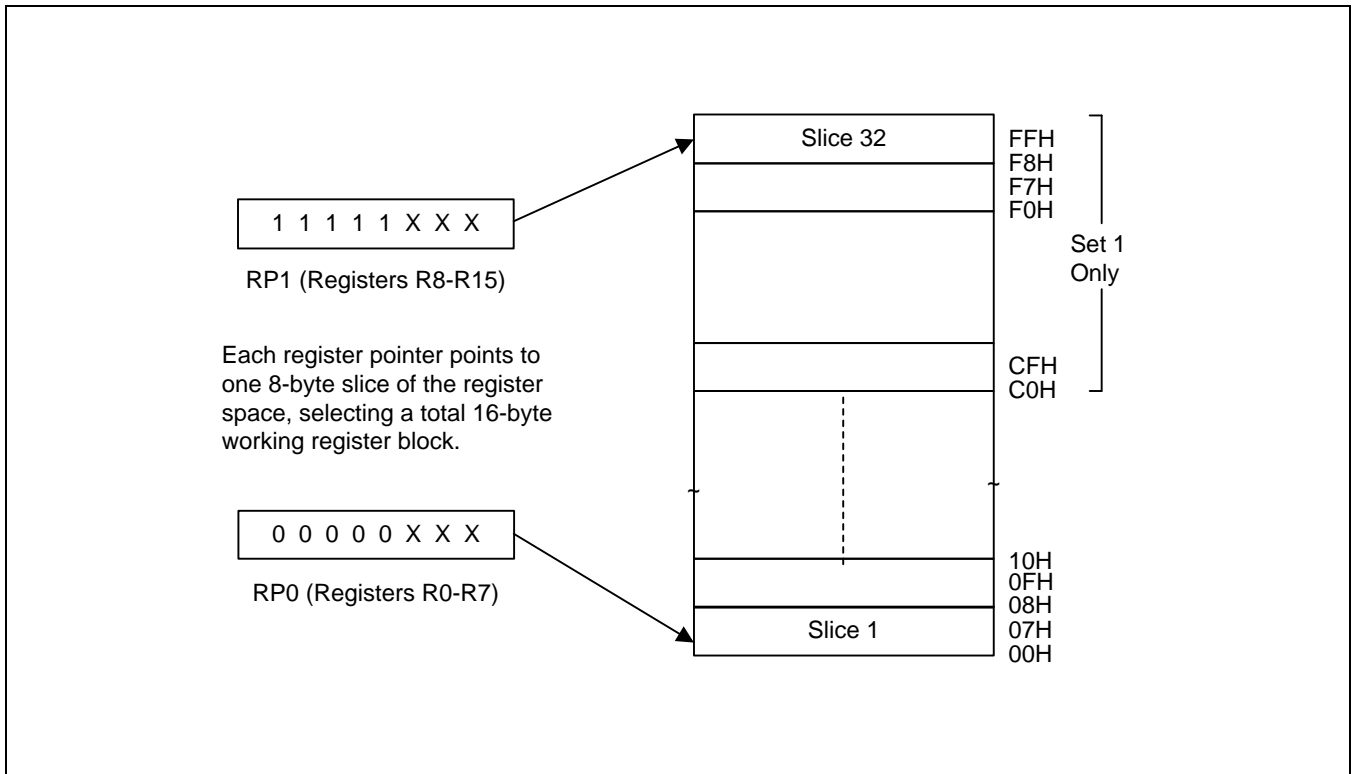


Figure 2-6 8-Byte Working Register Areas (Slices)

### 2.3.6 Using the Register Pointers

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

To change a register pointer value, you load a new value to RP0 and/or RP1 using an SRP or LD instruction (see [Figure 2-7](#) and [Figure 2-8](#)).

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

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

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

#### Example 2-2 Setting the Register Pointers

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

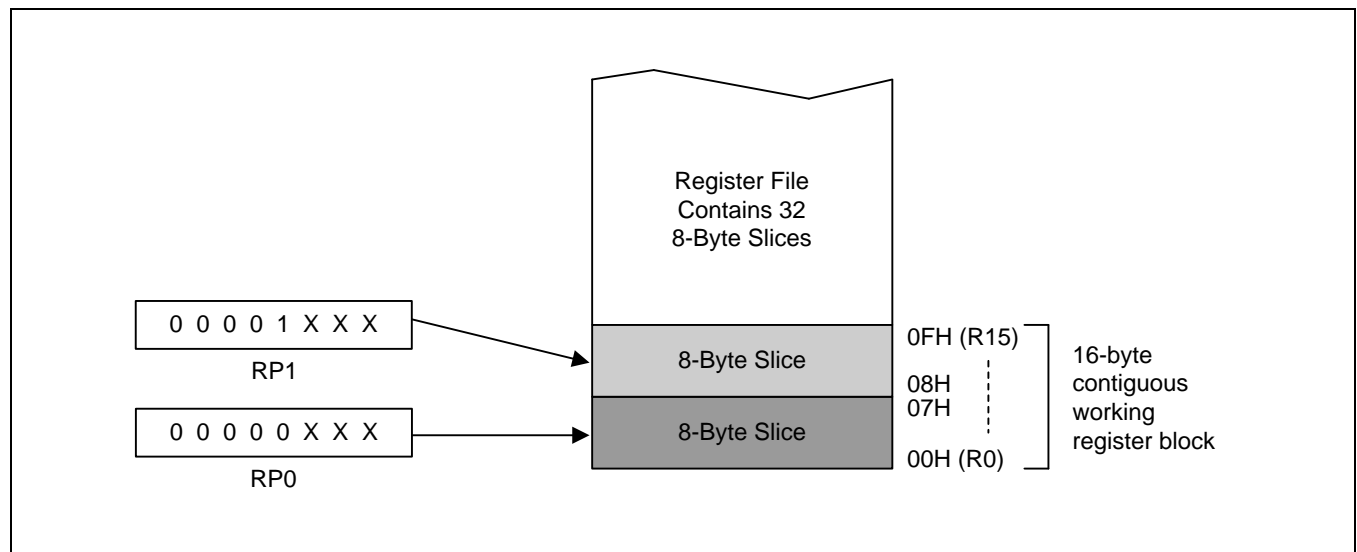


Figure 2-7 Contiguous 16-Byte Working Register Block

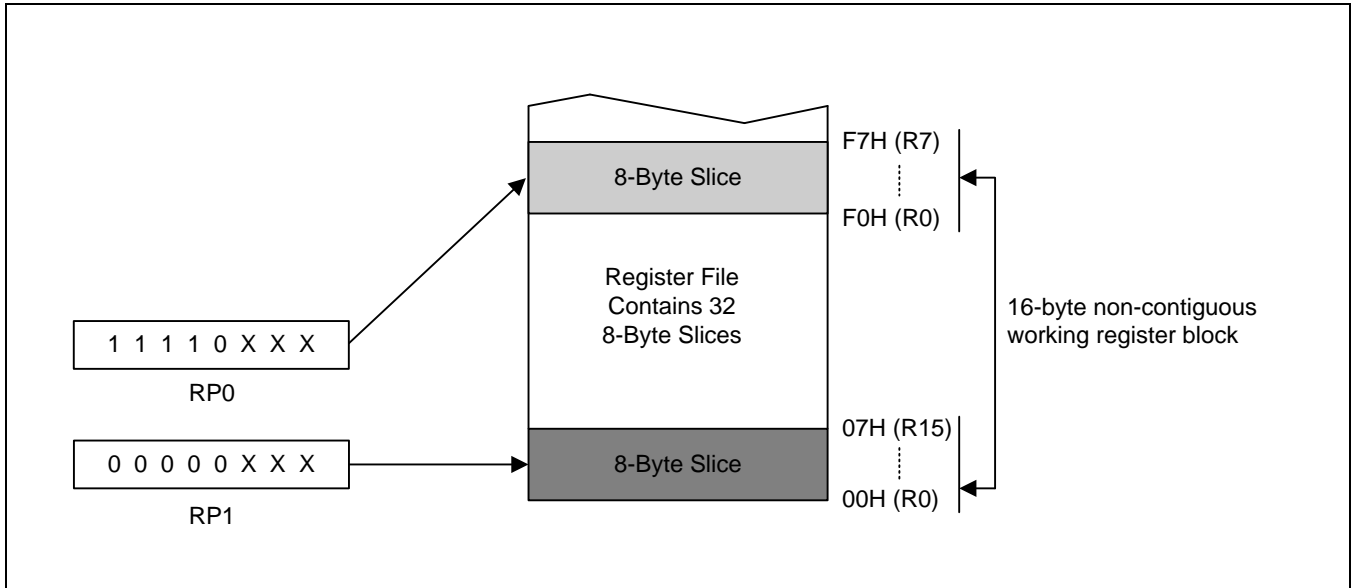


Figure 2-8 Non-Contiguous 16-Byte Working Register Block

**Example 2-3 Using the RPs to Calculate the Sum of a Series of Registers**

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

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

The sum of these six registers, 6FH, is located in the register R0 (80H). The instruction string used in this example takes 12 bytes of instruction code and its execution time is 36 cycles. If the register pointer is not used to calculate the sum of these registers, the following instruction sequence would have to be used:

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

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



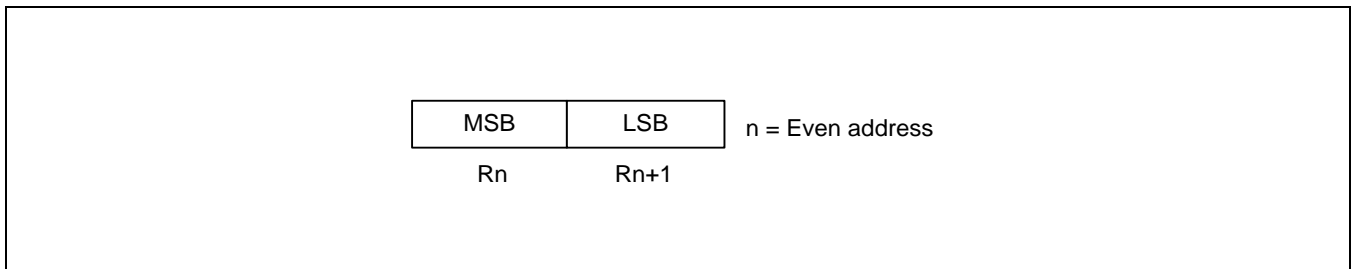
## 2.4 Register Addressing

The S3C8 Series register architecture provides an efficient method of working register addressing that takes full advantage of shorter instruction formats to reduce execution time.

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

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

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



**Figure 2-9 16-Bit Register Pair**

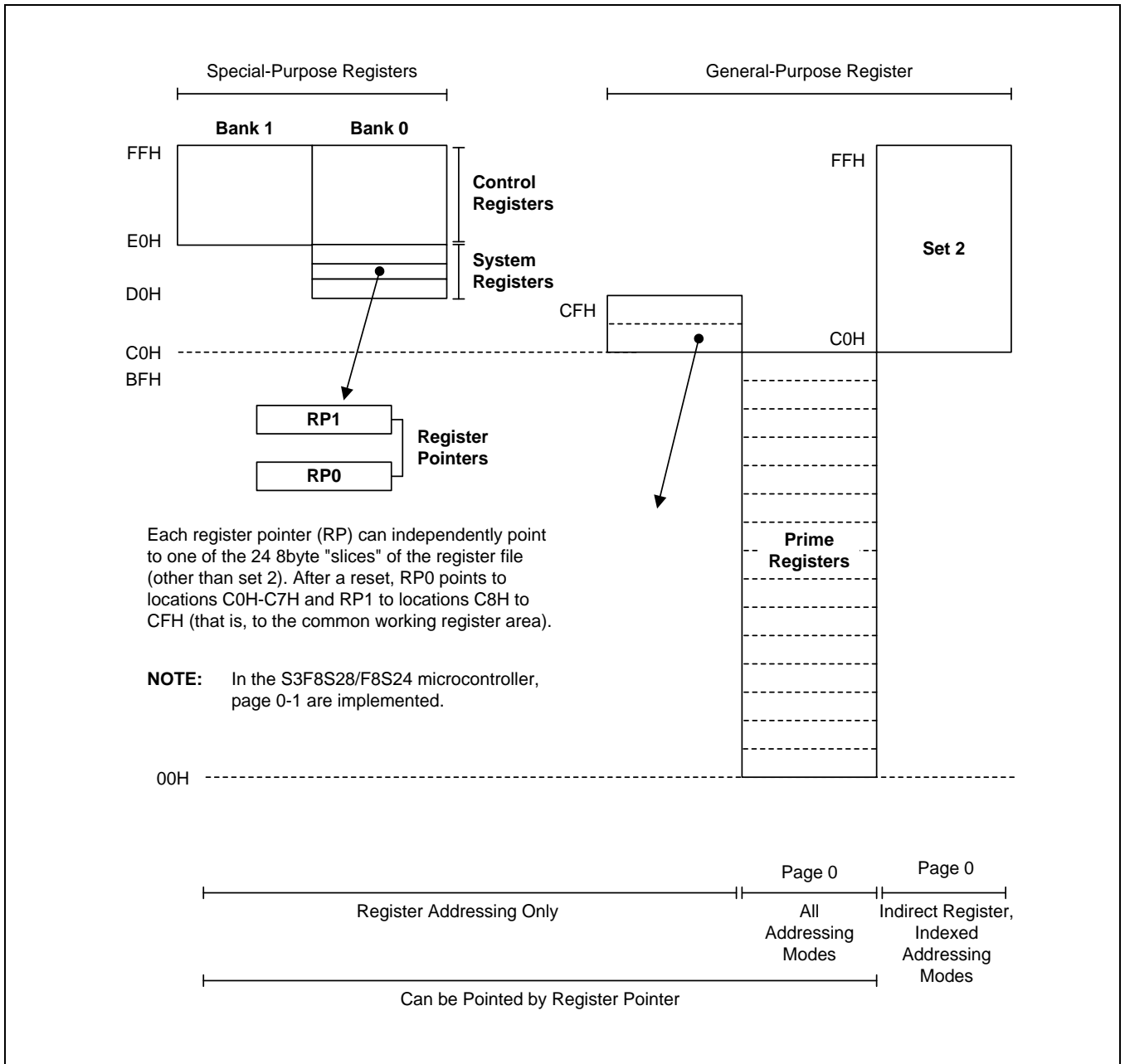


Figure 2-10 Register File Addressing

### 2.4.1 Common Working Register Area (C0H to CFH)

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

- RP0 → C0H to C7H
- RP1 → C8H to CFH

This 16byte 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.

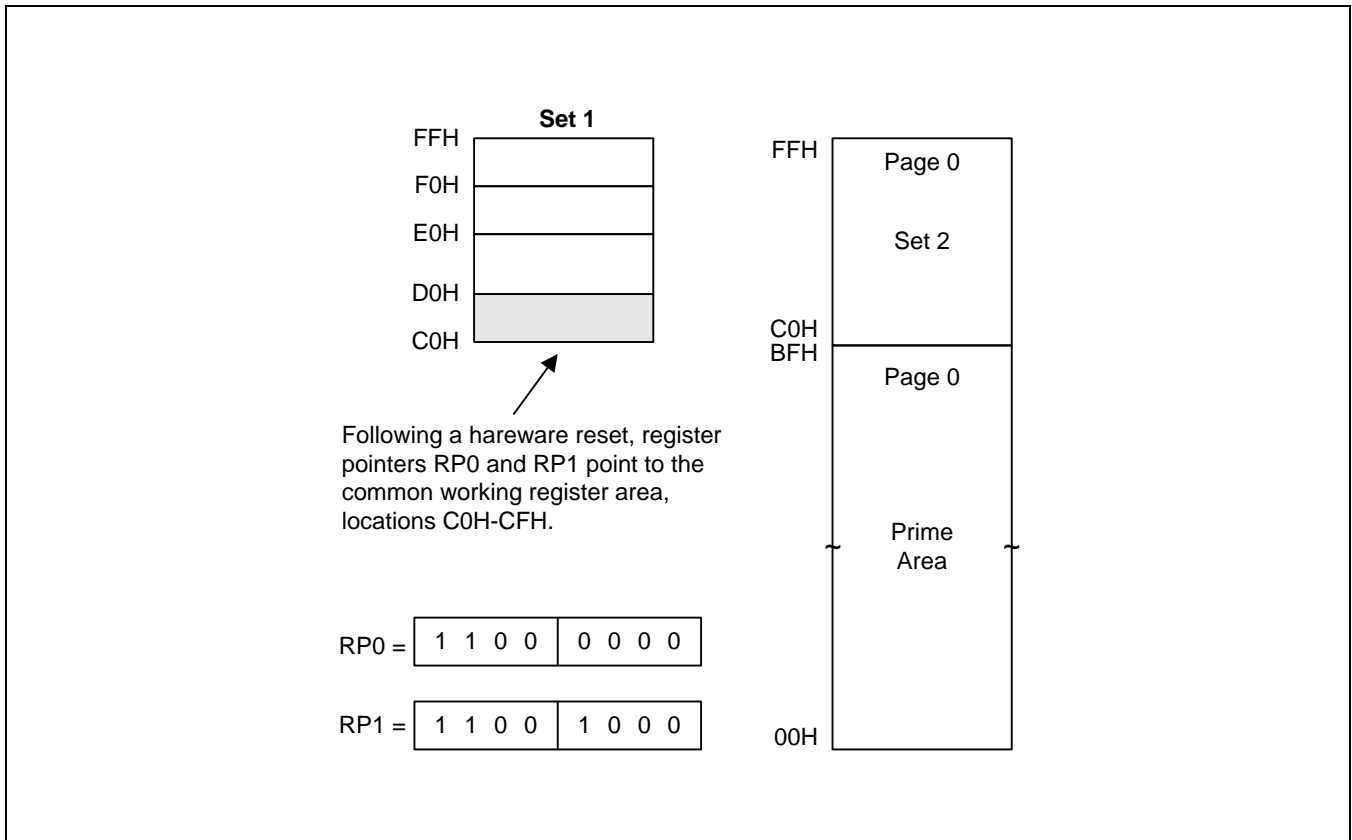


Figure 2-11 Common Working Register Area

**Example 2-4 Addressing the Common Working Register Area**

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

**Example 1:**

```
LD    0C2H,40H      ; Invalid addressing mode!
```

Use working register addressing instead:

```
SRP   #0C0H
```

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

**Example 2:**

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

Use working register addressing instead:

```
SRP   #0C0H
```

```
ADD   R3,#45H      ; R3 (C3H) ← R3 + 45H
```

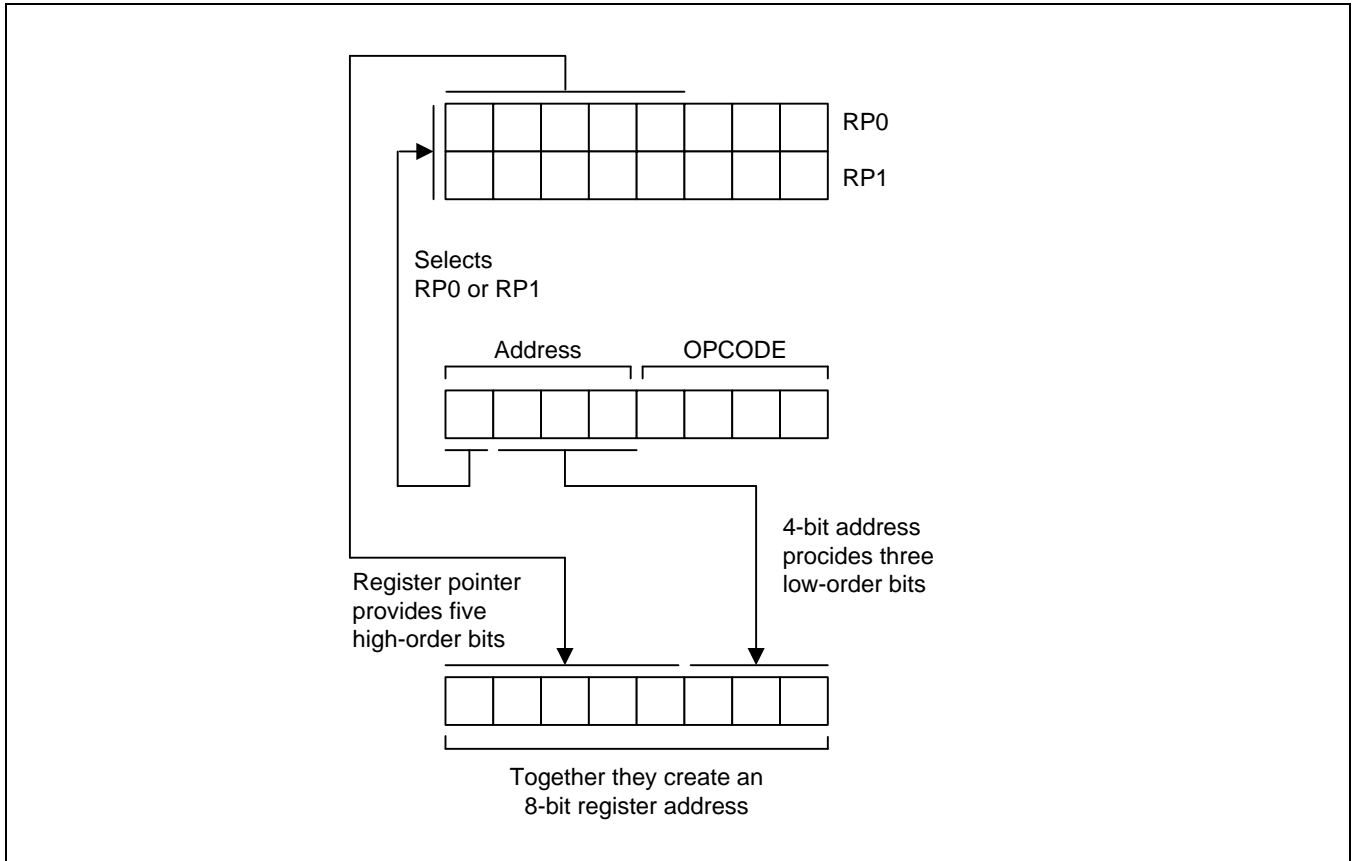
**2.4.2 4-Bit Working Register Addressing**

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

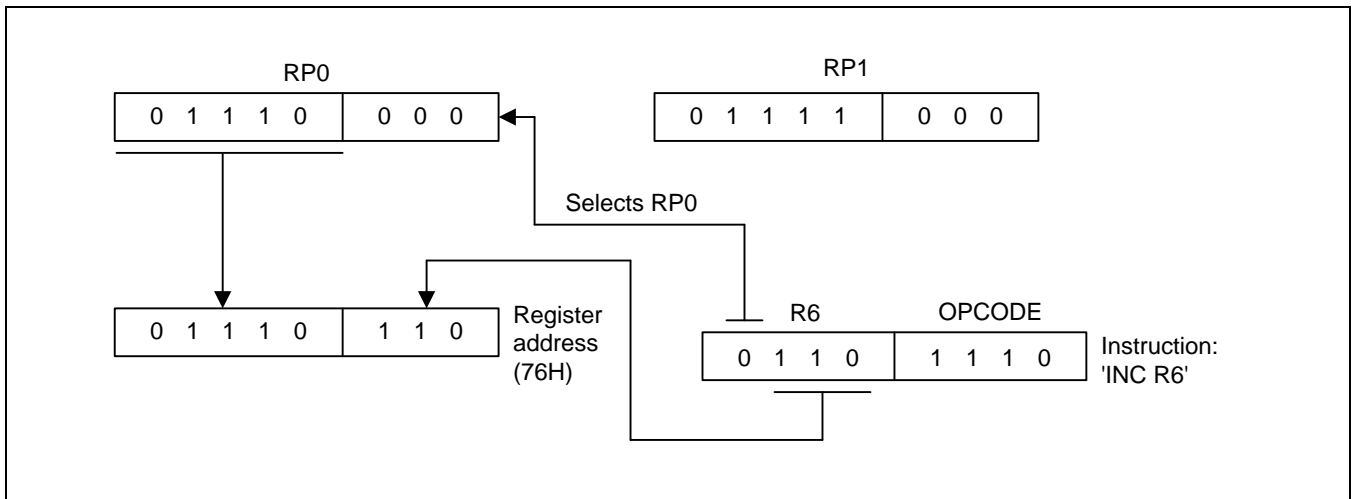
- The high-order bit of the 4-bit address selects one of the register pointers ("0" selects RP0; "1" selects RP1);
- The five high-order bits in the register pointer select an 8byte slice of the register space;
- The three low-order bits of the 4-bit address select one of the eight registers in the slice.

As shown in [Figure 2-12](#), the result of this operation is that the five high-order bits from the register pointer are concatenated with the three low-order bits from the instruction address to form the complete address. As long as the address stored in the register pointer remains unchanged, the three bits from the address will always point to an address in the same 8-byte register slice.

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



**Figure 2-12 4-Bit Working Register Addressing**



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

### 2.4.3 8-Bit Working Register Addressing

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

As shown in [Figure 2-14](#), the lower nibble of the 8-bit address is concatenated in much the same way as for 4-bit addressing: Bit 3 selects either RP0 or RP1, which then supplies the five high-order bits of the final address. The three low-order bits of the complete address are provided by the original instruction.

[Figure 2-15](#) shows an example of 8-bit working register addressing. The four high-order bits of the instruction address (1100B) specify 8-bit working register addressing. Bit 4 ("1") selects RP1 and the five high-order bits in RP1 (10101B) become the five high-order bits of the register address. The three low-order bits of the register address (011) are provided by the three low-order bits of the 8-bit instruction address. The five-address bits from RP1 and the three address bits from the instruction are concatenated to form the complete register address, 0ABH (10101011B).

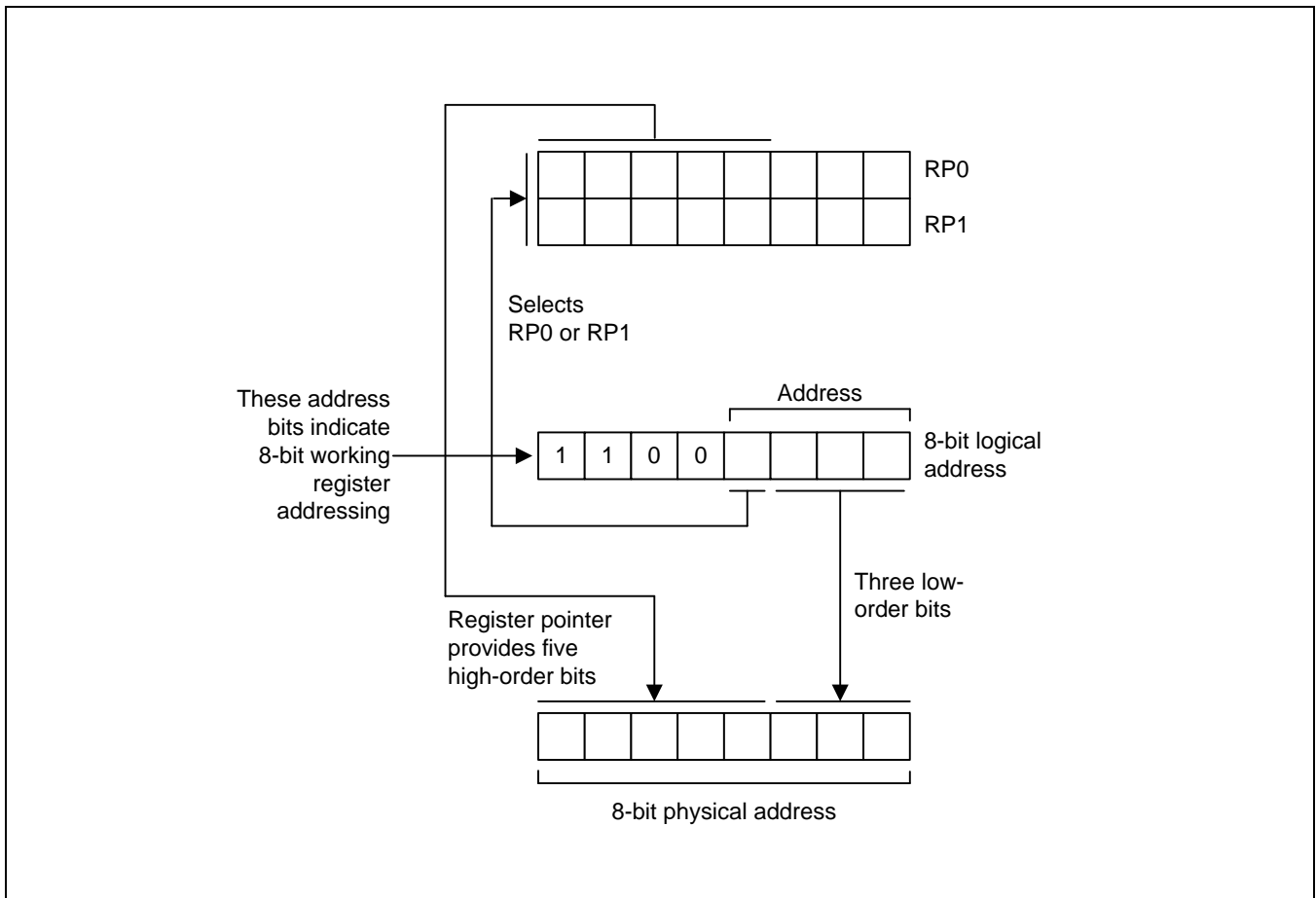


Figure 2-14 8-Bit Working Register Addressing

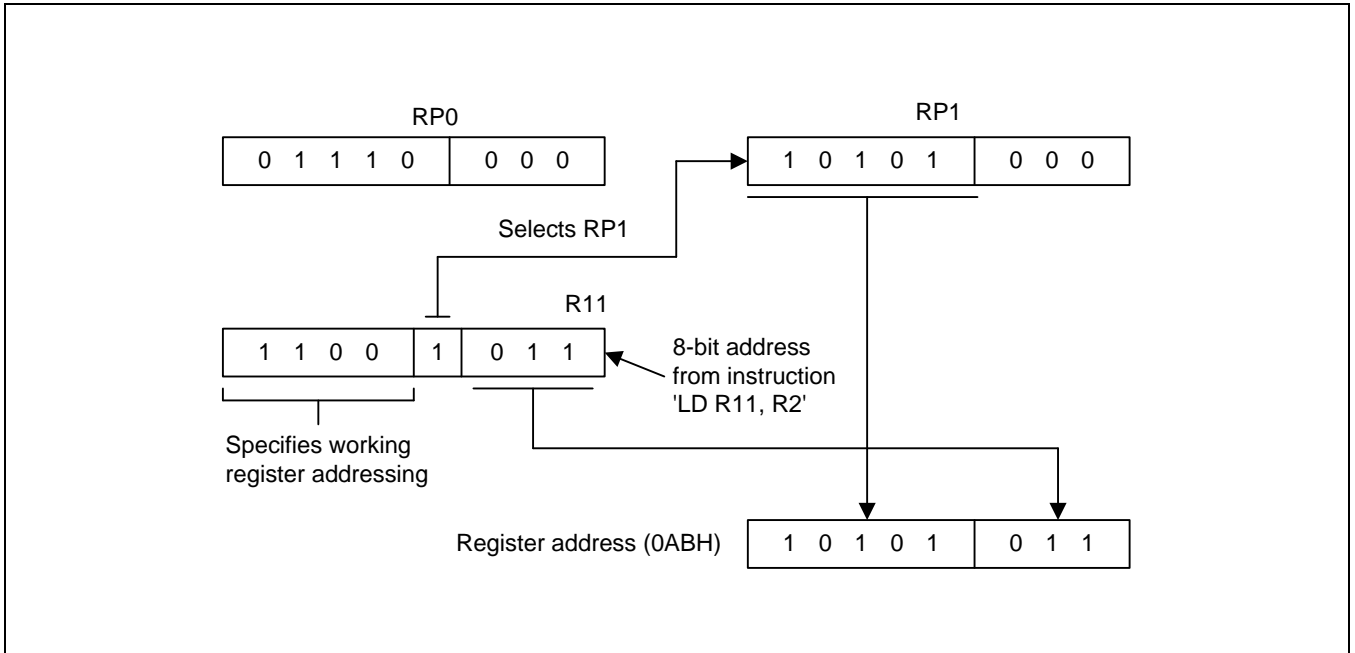


Figure 2-15 8-Bit Working Register Addressing Example

## 2.5 System and User Stacks

S3C8 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 S3F8S28/S3F8S24 architecture supports stack operations in the internal register file.

### 2.5.1 Stack Operations

Return addresses for procedure calls, interrupts and data are stored on the stack. The contents of the PC are saved to stack by a CALL instruction and restored by the RET instruction. When an interrupt occurs, the contents of the PC and the FLAGS registers are pushed to the stack. The IRET instruction then pops these values back to their original locations. The stack address value is always decreased by one before a push operation and increased by one after a pop operation. The stack pointer (SP) always points to the stack frame stored on the top of the stack, as shown in [Figure 2-16](#).

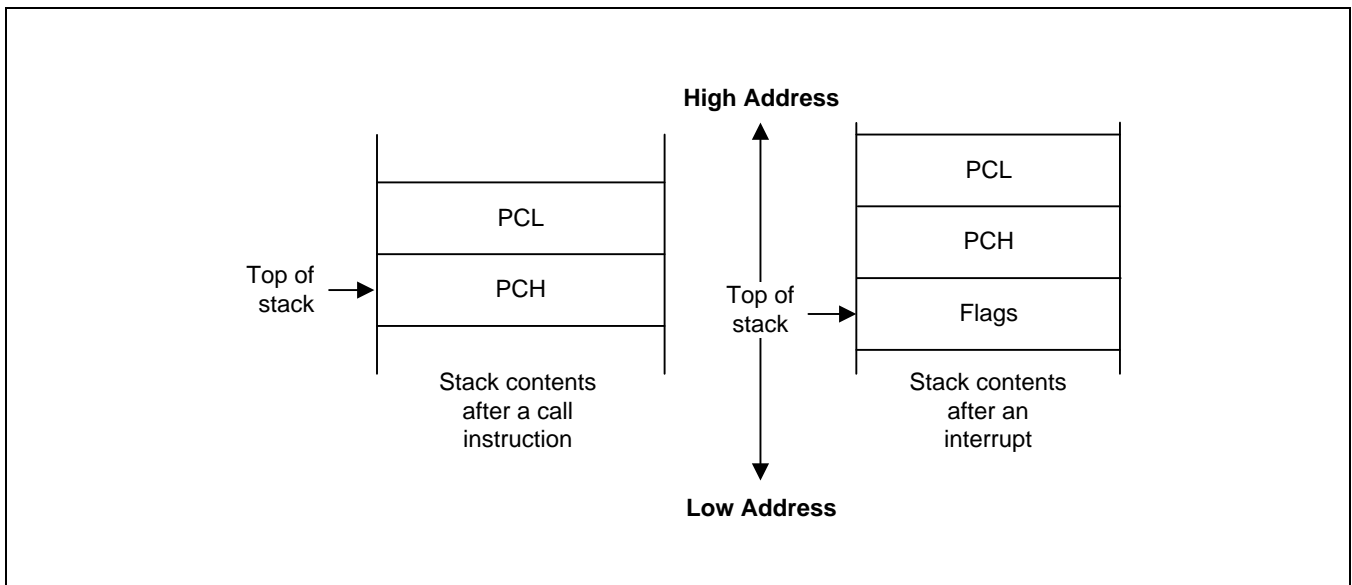


Figure 2-16 Stack Operations

### 2.5.2 User-Defined Stacks

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

### 2.5.3 Stack Pointers (SPL)

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 256byte is implemented in The S3F8S28/S3F8S24, the SPL must be initialized to an 8-bit value in the range 00 to FFH.



**Example 2-5 Standard Stack Operations Using PUSH and POP**

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

```
LD      SPL,#0FFH      ; SP:FFH (Normally, the SP is set to FFH 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

## 3.1 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.

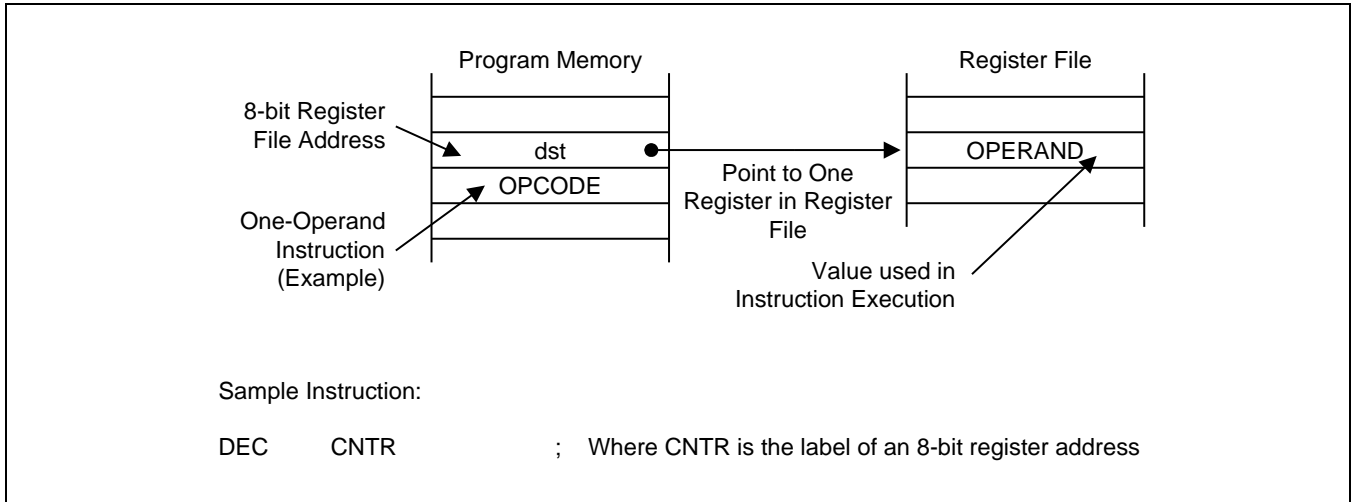
The S3F Series instruction set supports seven explicit addressing modes. Not all of these addressing modes are available for each instruction. The seven addressing modes and their symbols are:

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

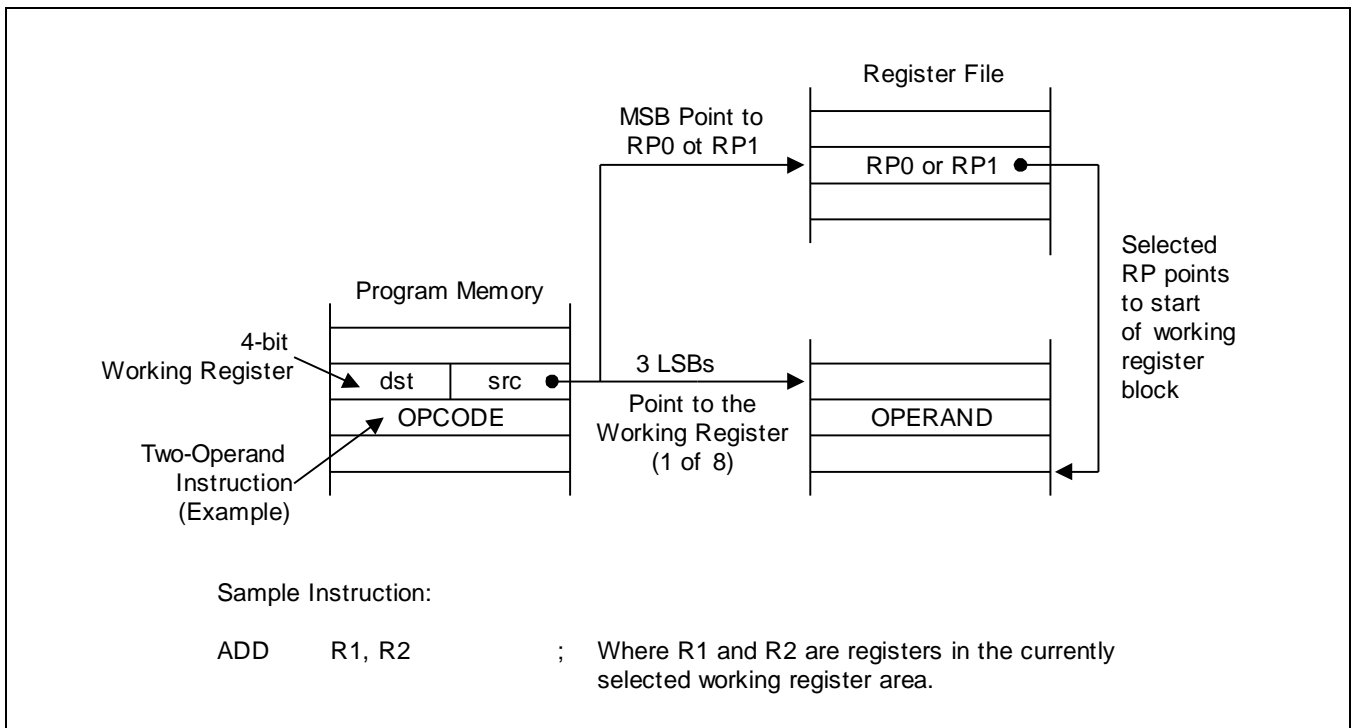
### 3.2 Register Addressing Mode (R)

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 8byte working register space in the register file and an 8-bit register within that space (see [Figure 3-2](#)).



**Figure 3-1 Register Addressing**



**Figure 3-2 Working Register Addressing**

### 3.3 Indirect Register 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 [Figure 3-3](#) through [Figure 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 to FFH in set 1 using the Indirect Register addressing mode.

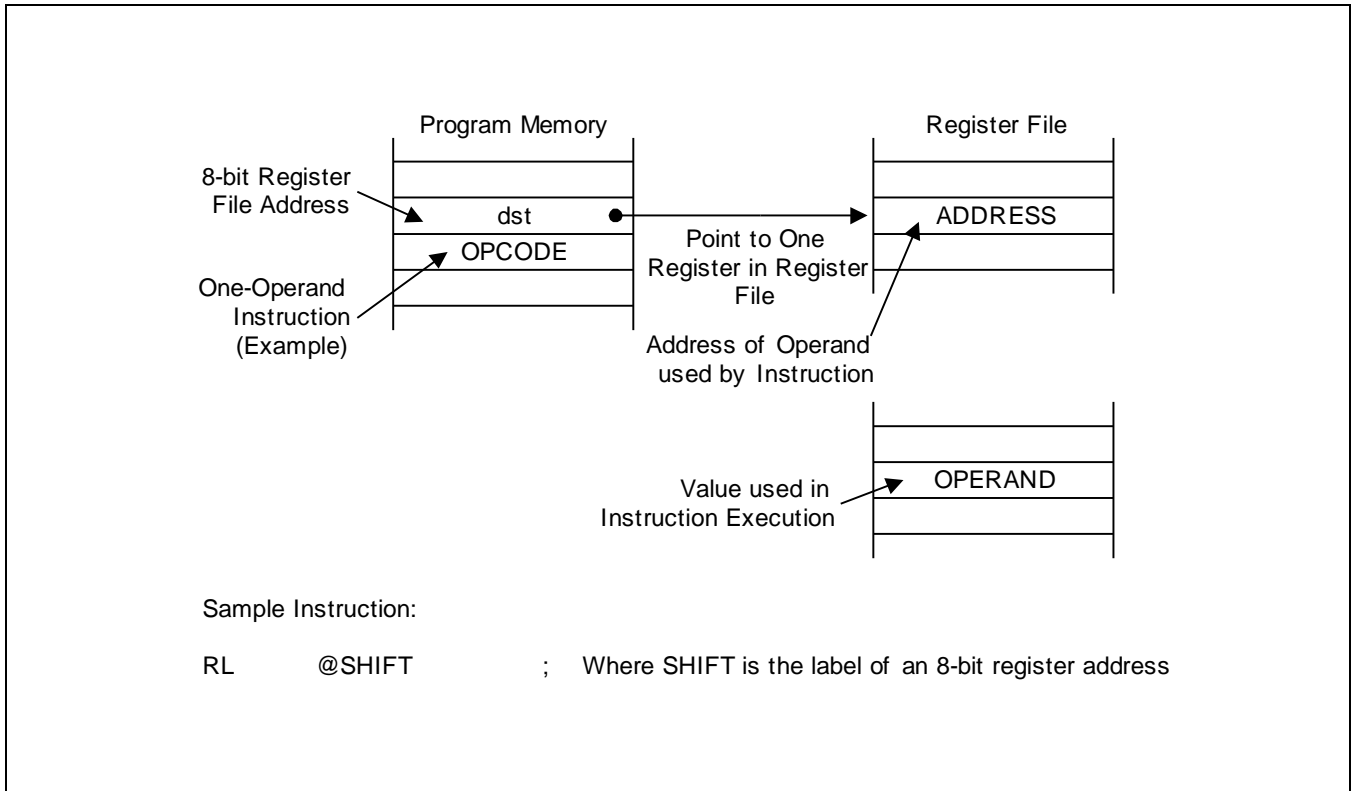


Figure 3-3 Indirect Register Addressing to Register File

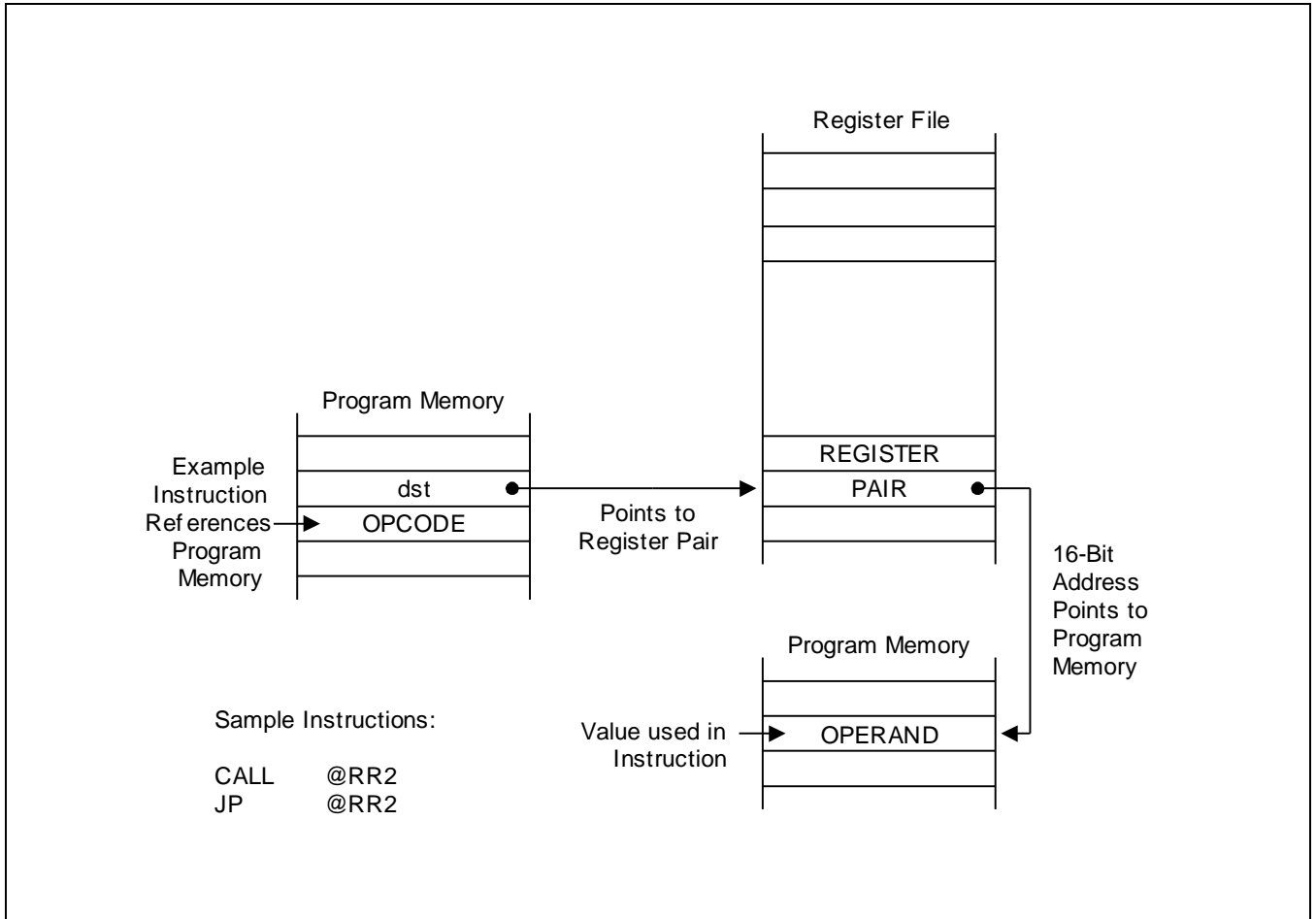
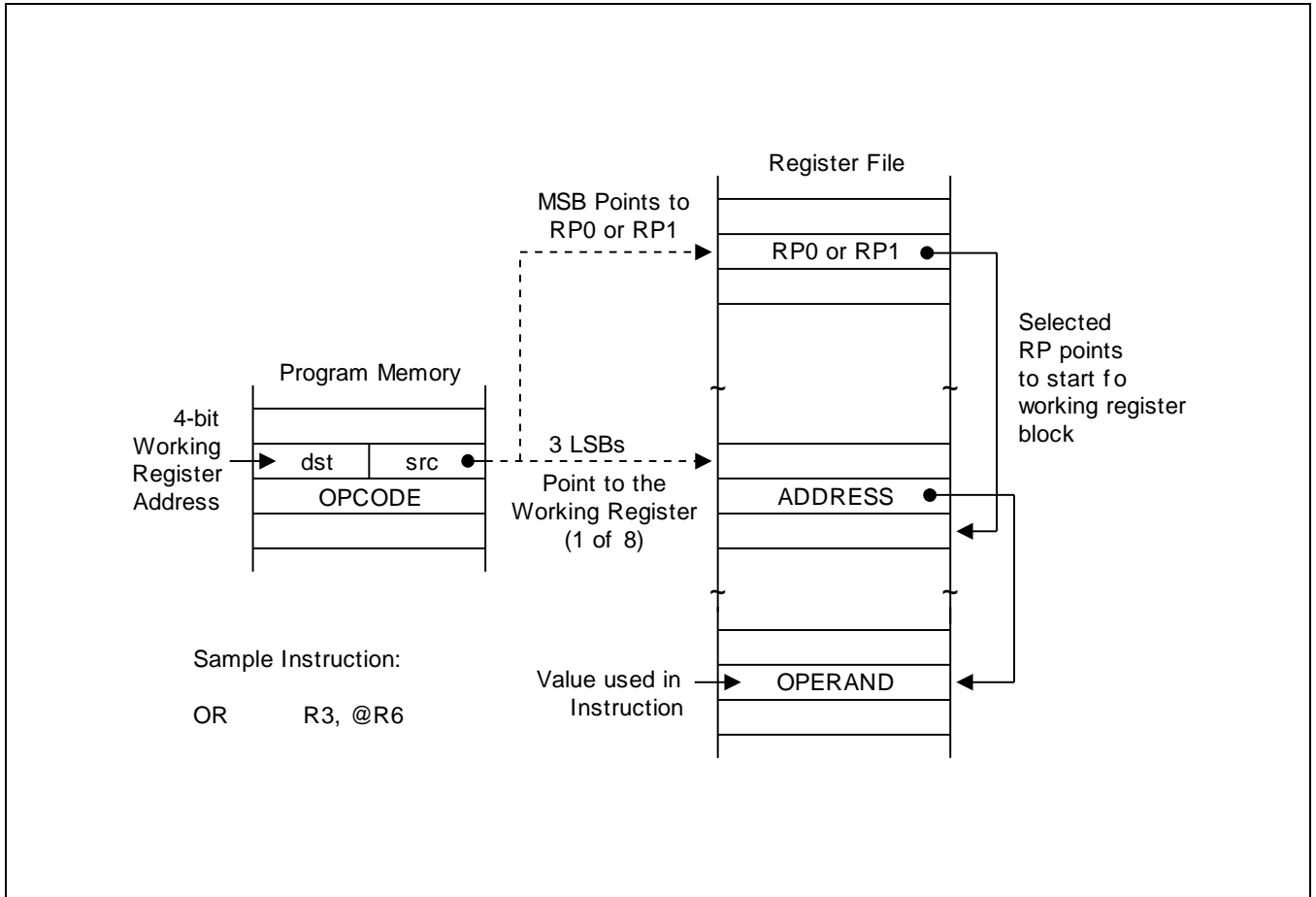


Figure 3-4 Indirect Register Addressing to Program Memory



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

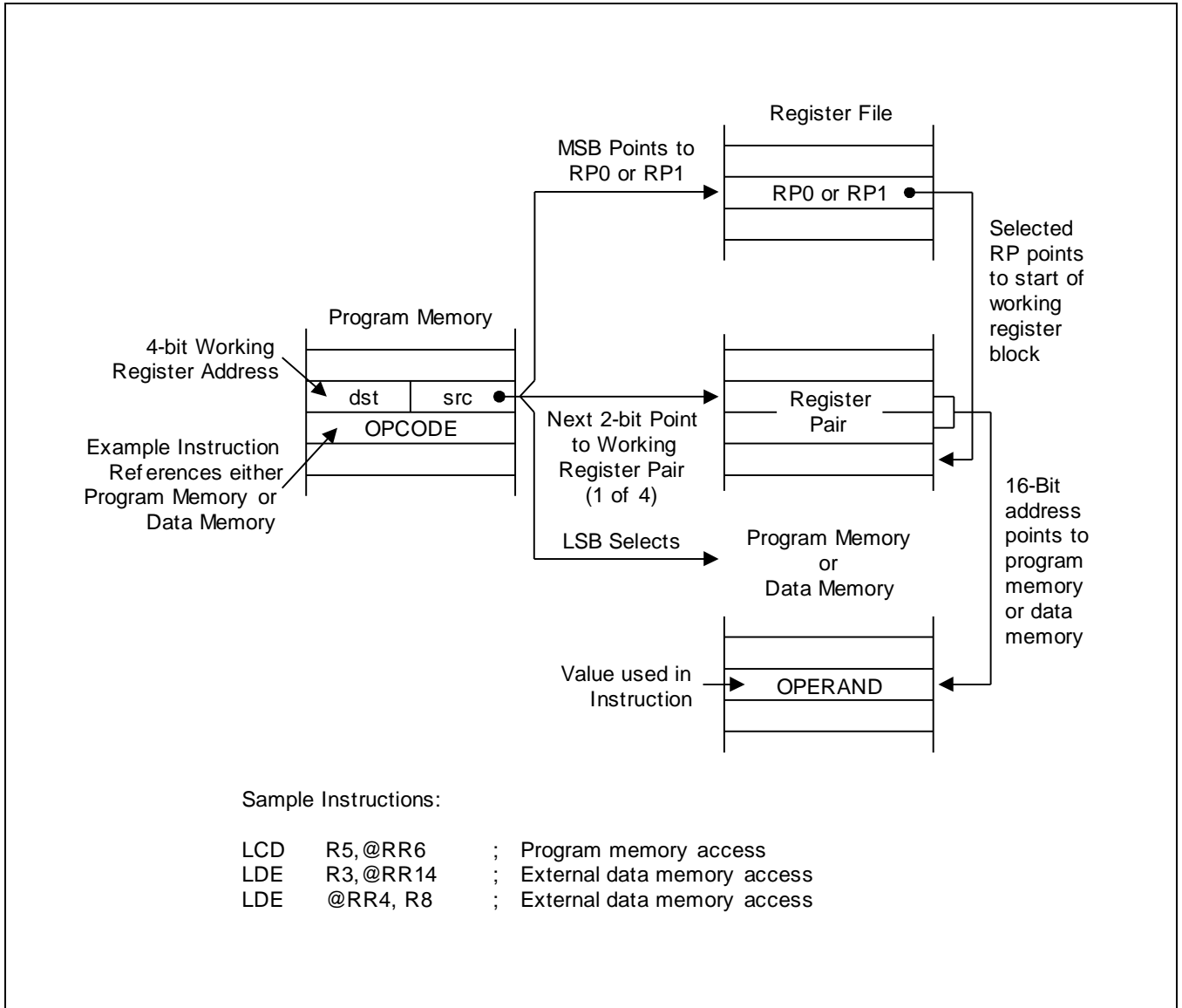


Figure 3-6 Indirect Working Register Addressing to Program or Data Memory

### 3.4 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 to 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.

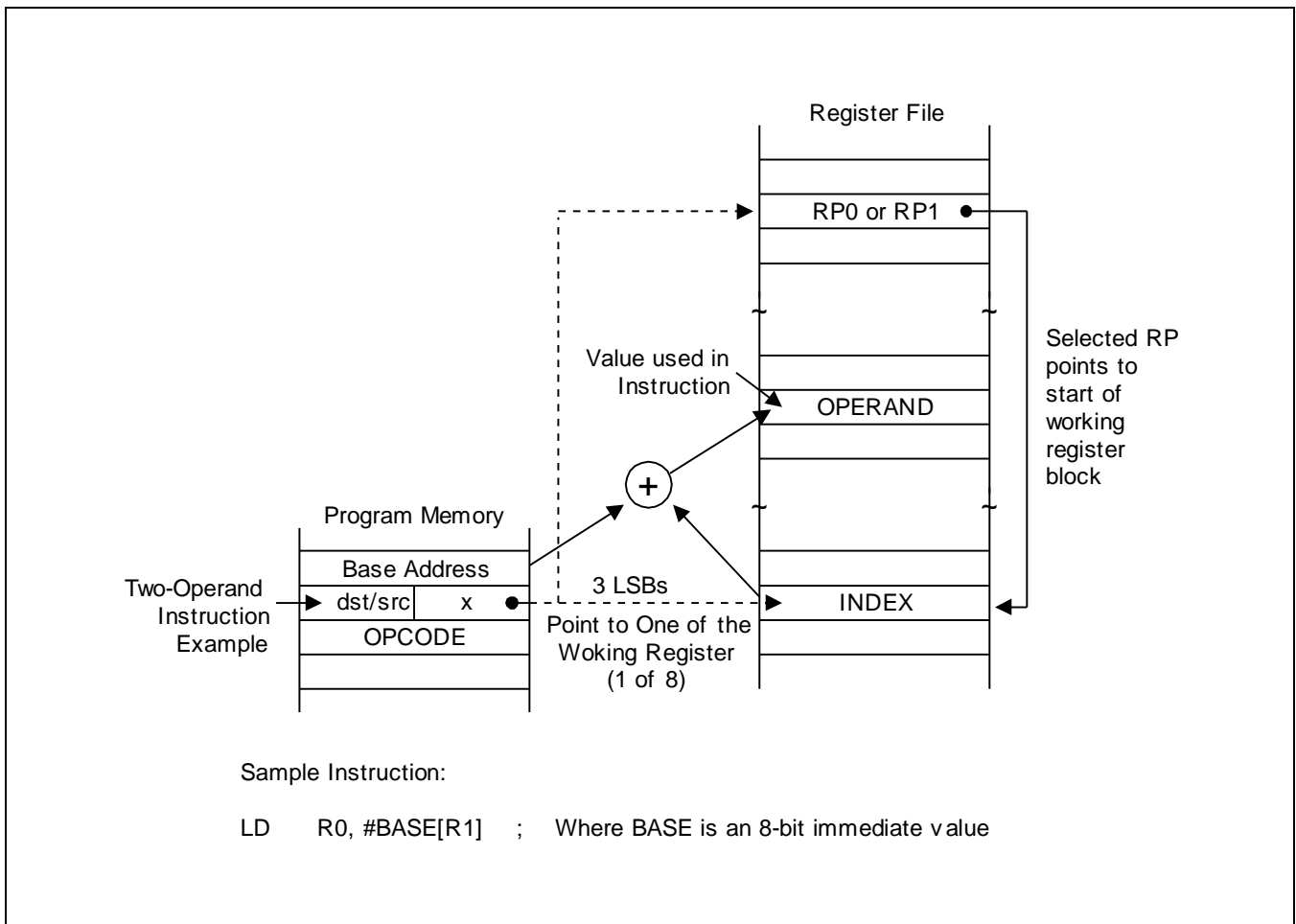


Figure 3-7 Indexed Addressing to Register File



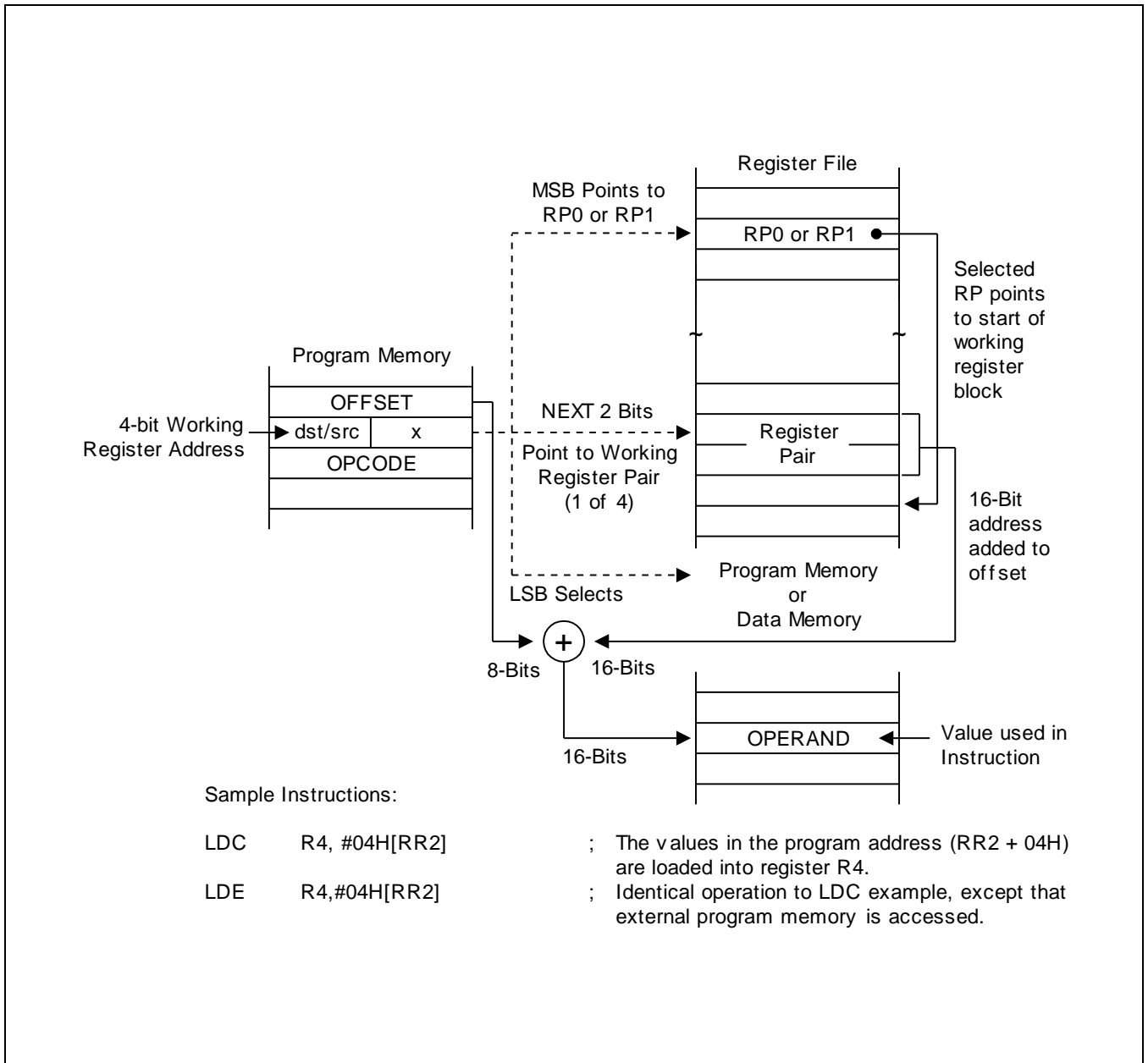


Figure 3-8 Indexed Addressing to Program or Data Memory with Short Offset

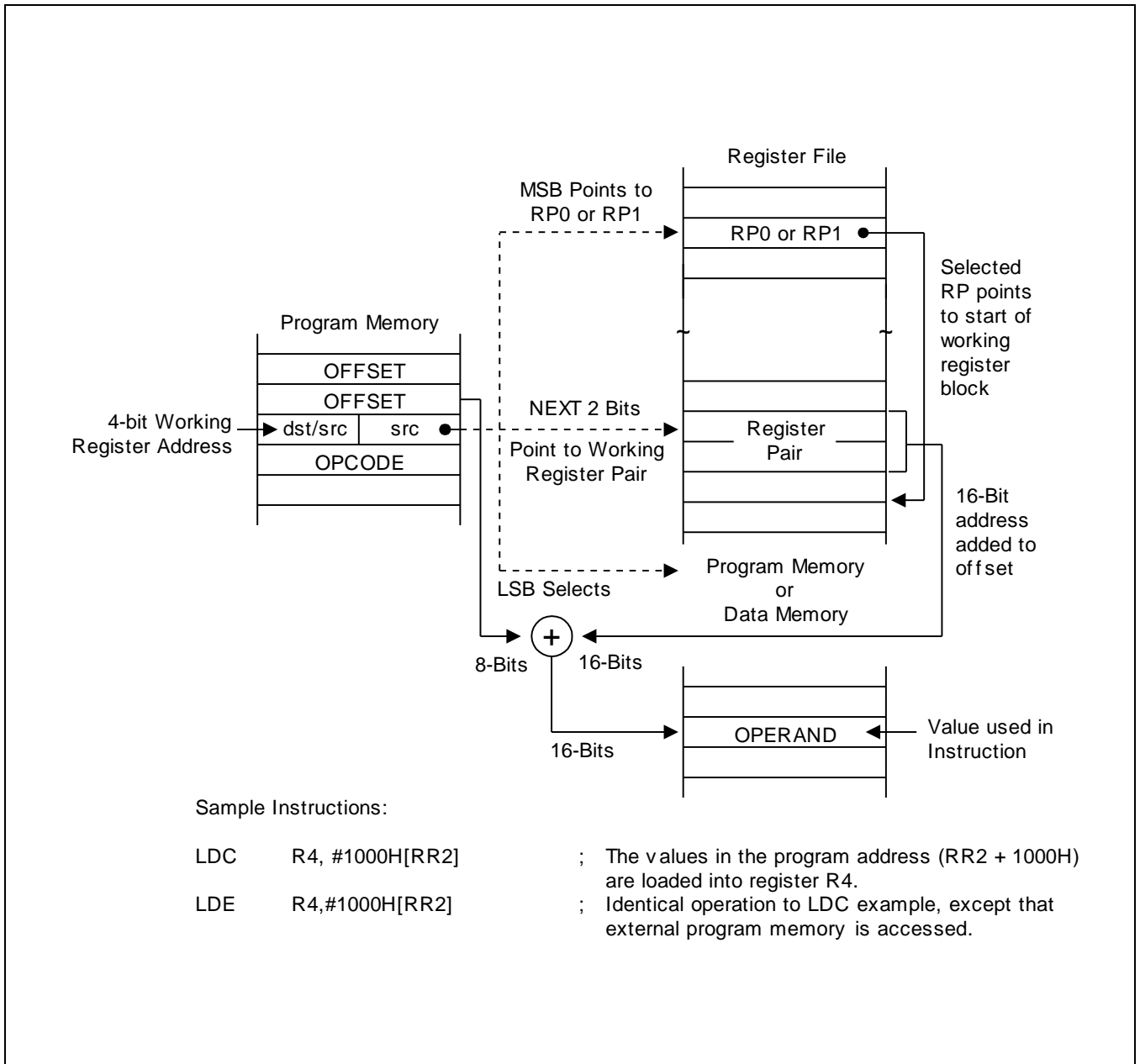
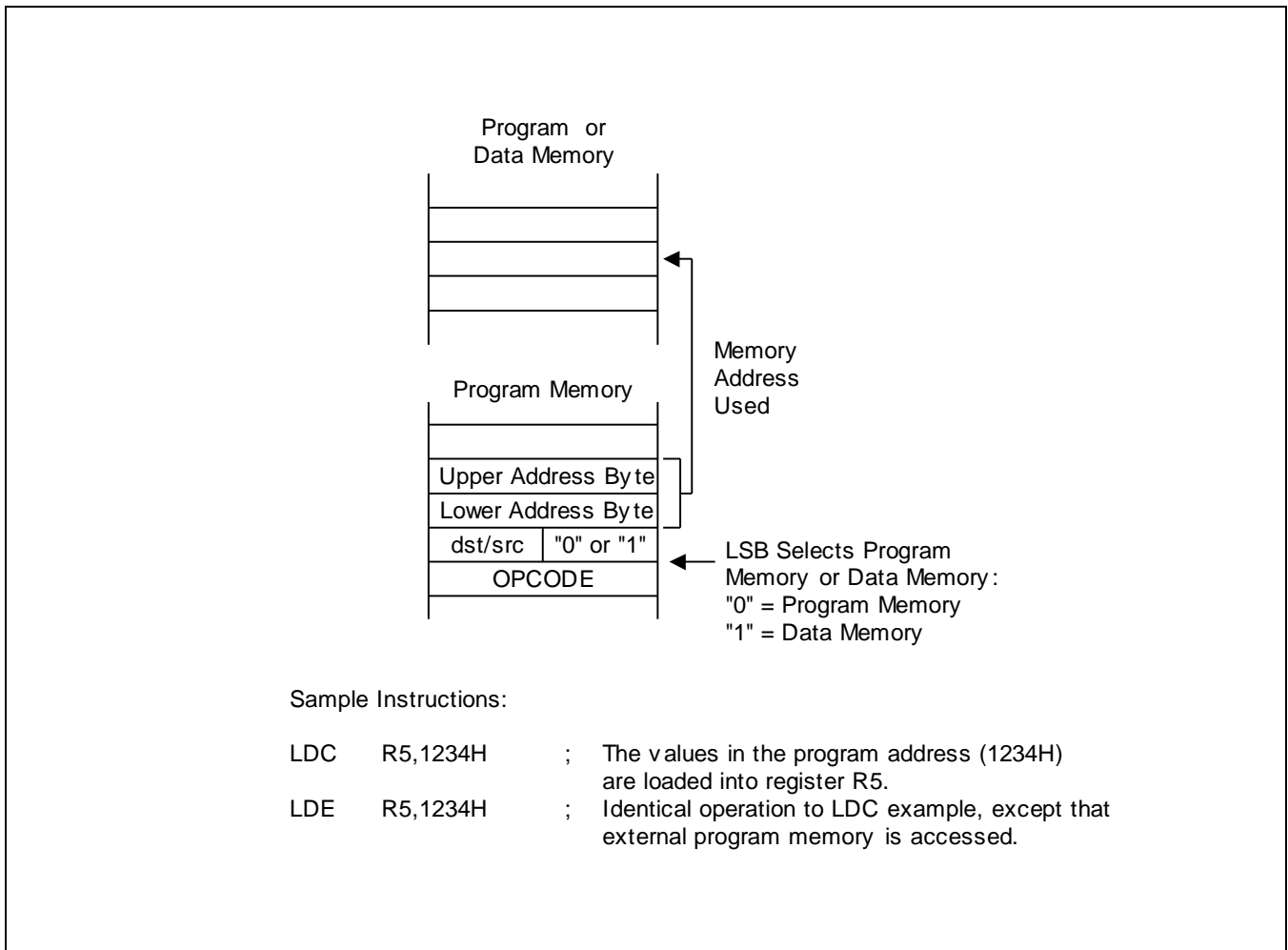


Figure 3-9 Indexed Addressing to Program or Data Memory

### 3.5 Direct Address Mode (DA)

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

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



**Figure 3-10 Direct Addressing for Load Instructions**

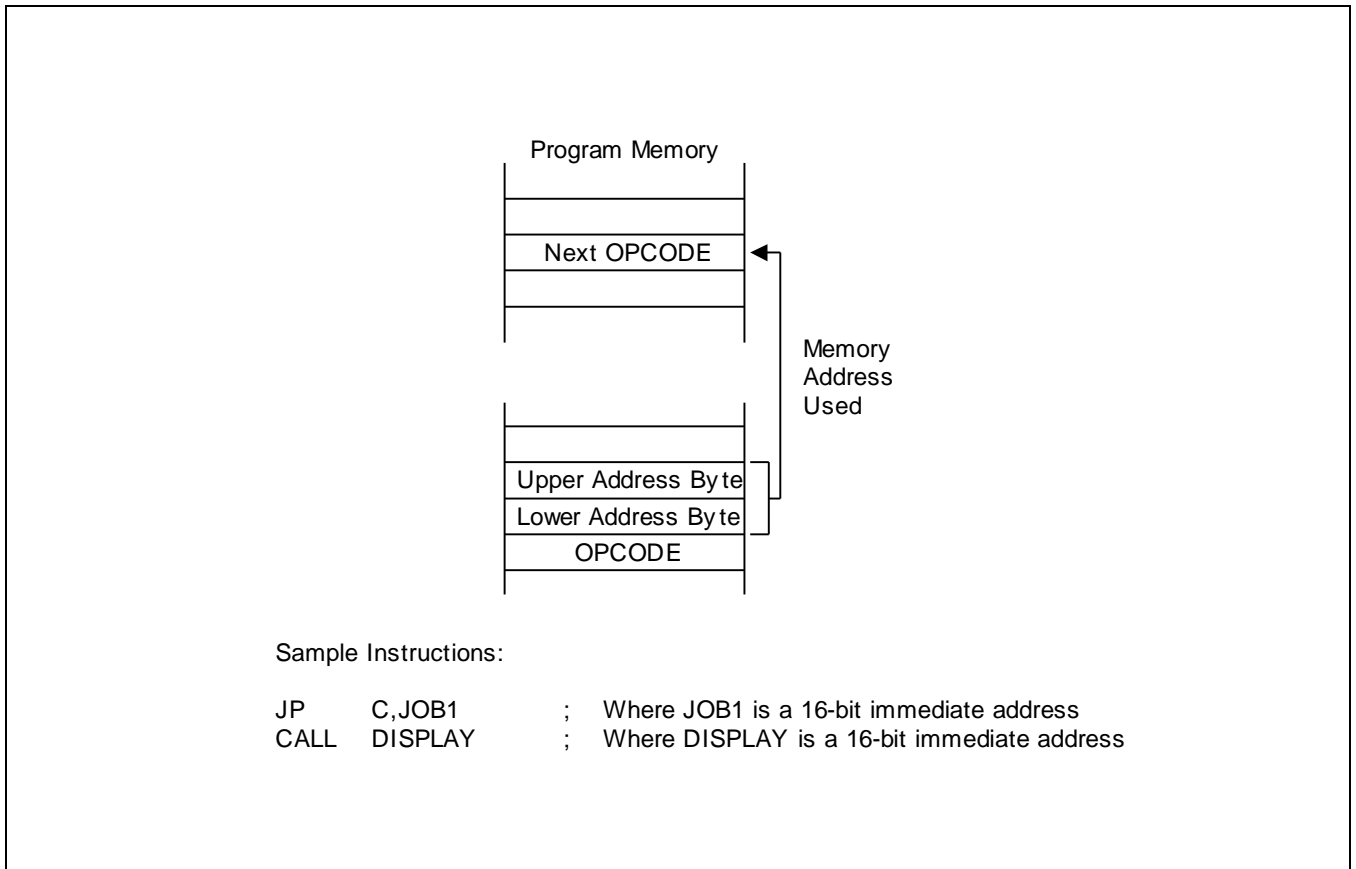


Figure 3-11 Direct Addressing for Call and Jump Instructions

### 3.6 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.

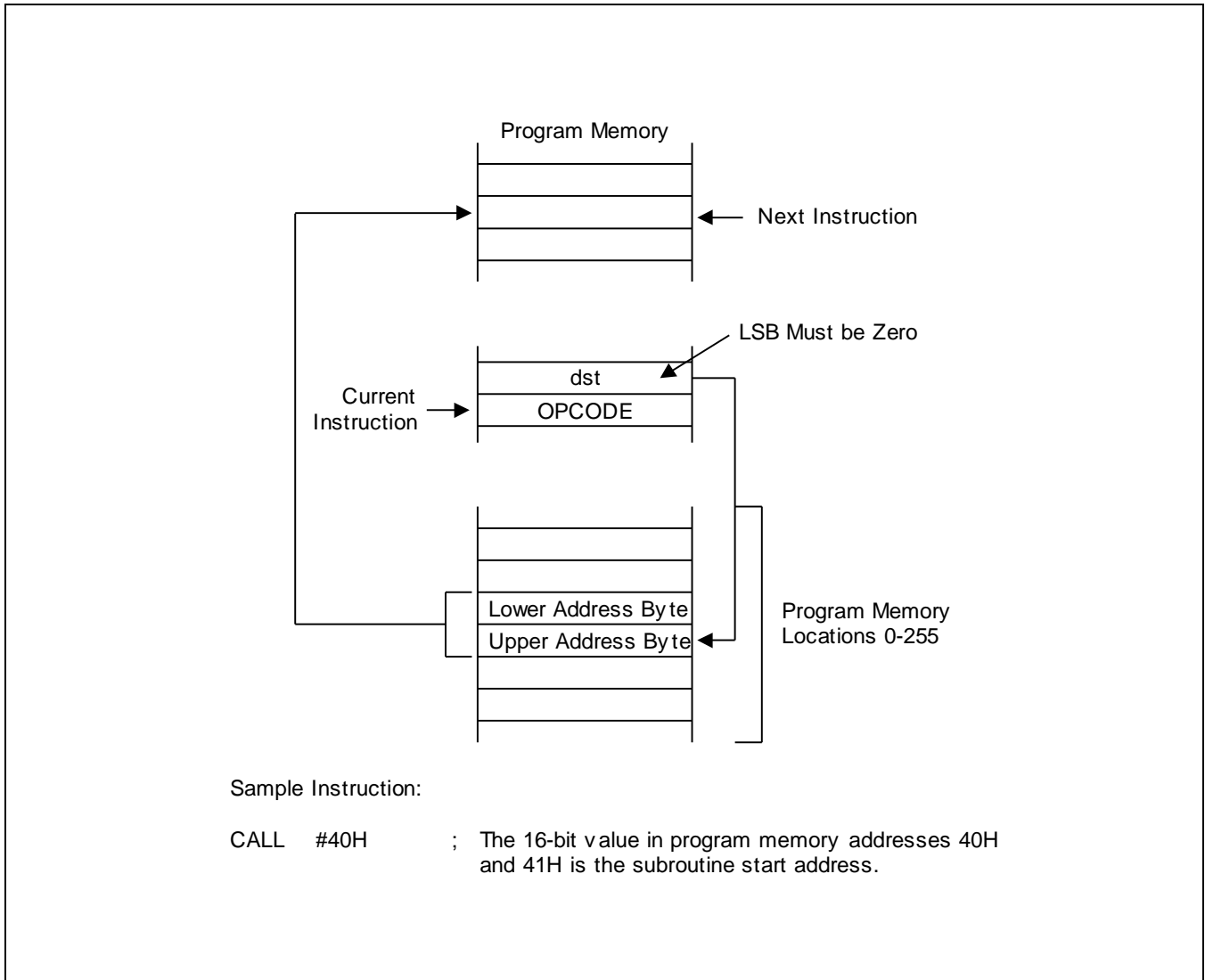
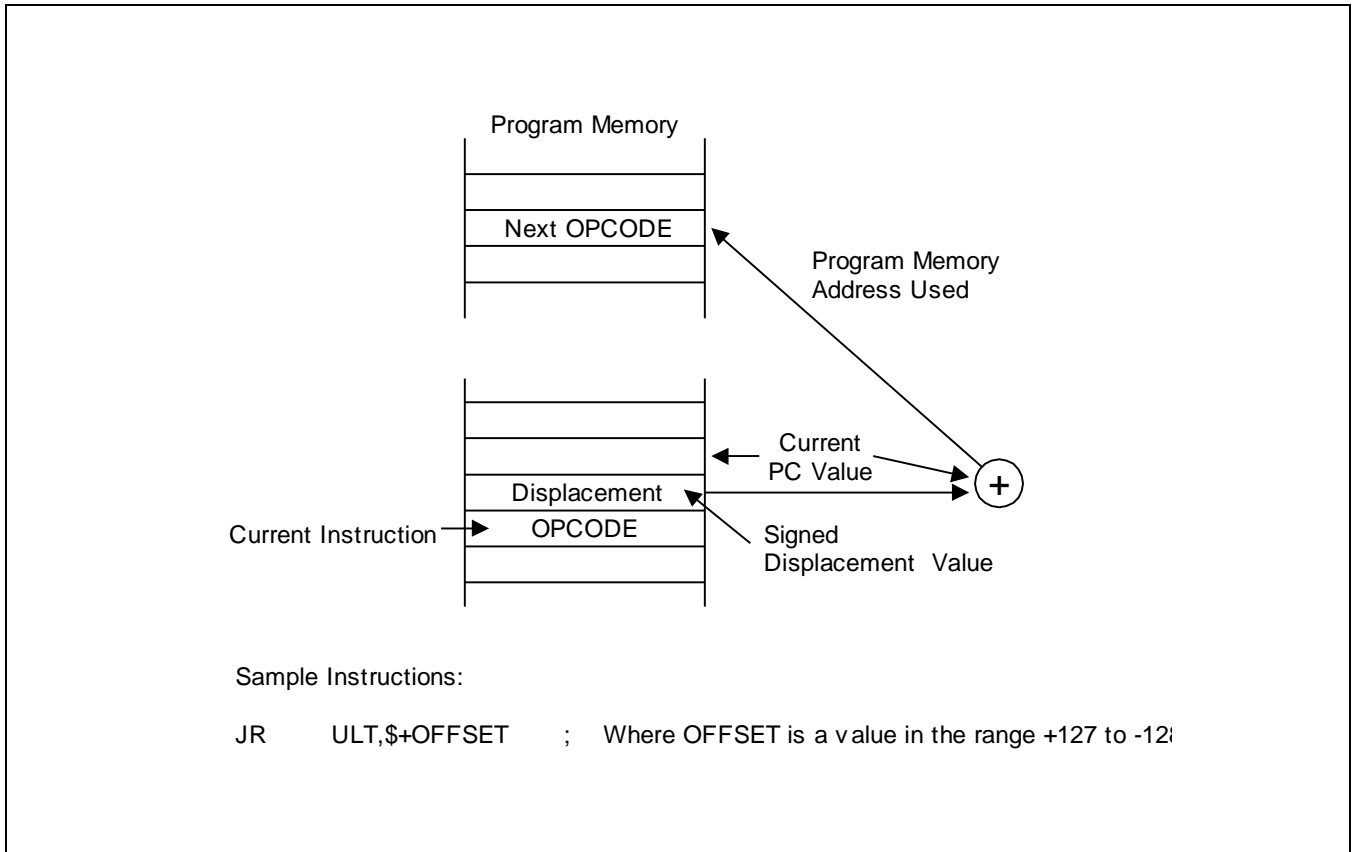


Figure 3-12 Indirect Addressing

### 3.7 Relative Address Mode (RA)

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



**Figure 3-13 Relative Addressing**

### 3.8 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.

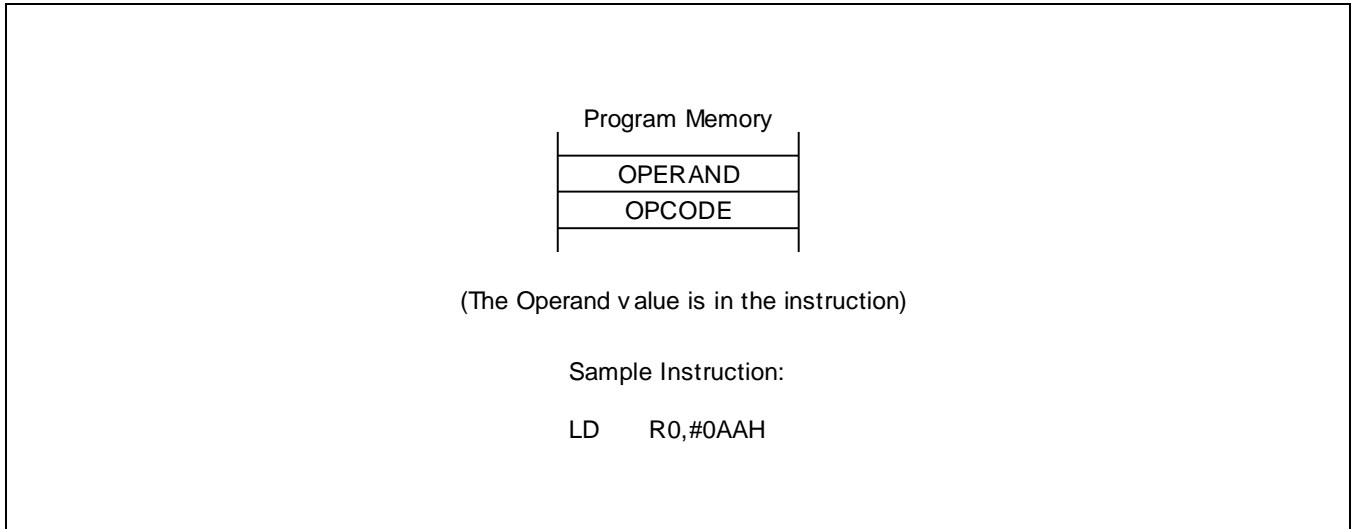


Figure 3-14 Immediate Addressing

# 4 Control Registers

## 4.1 Overview

In this section, detailed descriptions of the S3F8S28/S3F8S24 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](#), [Table 4-2](#), and [Table 4-3](#). [Figure 4-1](#) illustrates the important features of the standard register description format.

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



**Table 4-1 System and Peripheral Control Registers, Set1**

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

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

Table 4-2 System and Peripheral Control Registers, Set1, Bank0

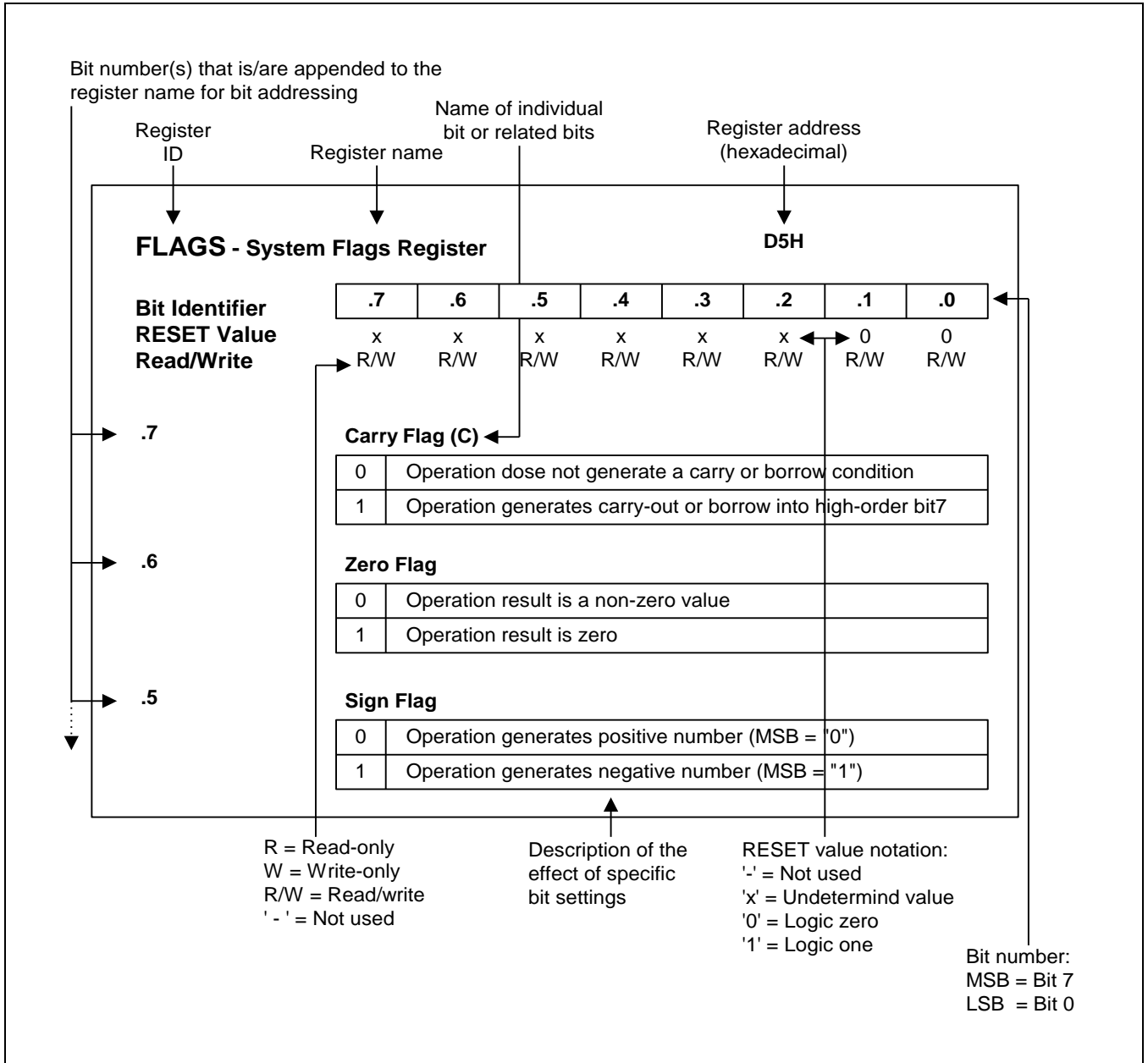
Register Name	Mnemonic	Address & Location		RESET Value (Bit)								
		Address	RW	7	6	5	4	3	2	1	0	
Port 0 data register	P0	E0H	RW	0	0	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	RW	-	-	-	-	-	0	0	0	0
Port 2 data register	P2	E2H	RW	-	0	0	0	0	0	0	0	0
Port 3 data register	P3	E3H	RW	-	-	-	-	0	0	0	0	0
Port 2 pull-up resistor enable register	P2PUR	E4H	RW	-	0	0	0	0	0	0	0	0
Port 0 pull-up resistor enable register	P0PUR	E5H	RW	0	0	0	0	0	0	0	0	0
Port 0 control register (High Byte)	P0CONH	E6H	RW	0	0	0	0	0	0	0	0	0
Port 0 control register (Low Byte)	P0CONL	E7H	RW	0	0	0	0	0	0	0	0	0
Port 0 interrupt pending register	P0PND	E8H	RW	-	-	-	-	0	0	0	0	0
Port 1 control register	P1CON	E9H	RW	0	0	-	-	0	0	0	0	0
Port 2 control register (High Byte)	P2CONH	EAH	RW	-	0	0	0	0	0	0	0	0
Port 2 control register (Low Byte)	P2CONL	EBH	RW	0	0	0	0	0	0	0	0	0
Timer B counter register	TBCNT	ECH	R	0	0	0	0	0	0	0	0	0
Timer B data register	TBDATA	EDH	RW	1	1	1	1	1	1	1	1	1
Timer B control register	TBCON	EEH	RW	-	-	0	0	0	0	0	0	0
Port 3 interrupt pending register	P3PND	EFH	RW	0	0	0	0	0	0	0	0	0
Port 3 control register	P3CON	F0H	RW	0	0	0	0	0	0	0	0	0
PWM0 extension data register	PWM0EX	F1H	RW	0	0	0	0	0	0	0	0	0
PWM0 data register	PWM0DATA	F2H	RW	0	0	0	0	0	0	0	0	0
PWM0 control register	PWM0CON	F3H	RW	0	0	-	0	0	0	0	0	0
STOP control register	STOPCON	F4H	RW	0	0	0	0	0	0	0	0	0
Ring Oscillator control register	ROSCCON	F5H	RW	0	0	0	0	0	0	0	0	0
Watchdog Timer control register	WDTCON	F6H	RW	0	0	0	0	0	0	0	0	0
A/D control register	ADCON	F7H	RW	1	1	1	1	0	0	0	0	0
A/D converter data register (High)	ADDATAH	F8H	R	x	x	x	x	x	x	x	x	x
A/D converter data register (Low)	ADDATAH	F9H	R	0	0	0	0	0	0	0	x	x
Locations FAH to FCH are not mapped												
Basic Timer counter	BTCNT	FDH	R	0	0	0	0	0	0	0	0	0
External memory timing register	EMT	FEH	RW	0	0	0	0	0	0	0	0	0
Interrupt priority register	IPR	FFH	RW	x	x	x	x	x	x	x	x	x

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

**Table 4-3 System and Peripheral Control Registers, Set1, Bank1**

Register Name	Mnemonic	Address & Location		RESET Value (Bit)								
		Address	RW	7	6	5	4	3	2	1	0	
Timer 1 Data Register (High Byte)	T1DATAH	E0H	RW	1	1	1	1	1	1	1	1	1
Timer 1 Data Register (Low Byte)	T1DATAL	E1H	RW	1	1	1	1	1	1	1	1	1
Timer 1 Counter Register (High Byte)	T1CNTH	E2H	R	0	0	0	0	0	0	0	0	0
Timer 1 Counter Register (Low Byte)	T1CNTL	E3H	R	0	0	0	0	0	0	0	0	0
Timer 1 Control Register	T1CON	E4H	RW	0	0	0	0	0	0	0	0	0
Timer 1 prescaler register	T1PS	E5H	RW	0	0	0	0	0	0	0	0	0
PWM1 extension data register	PWM1EX	E6H	RW	0	0	0	0	0	0	0	0	0
PWM1 data register	PWM1DATA	E7H	RW	0	0	0	0	0	0	0	0	0
PWM1 control register	PWM1CON	E8H	RW	0	0	–	0	0	0	0	0	0
Locations E9H are not mapped												
Reset source indicating register	RESETID	EAH	RW	Refer to the detail description								
Flash memory control register	FMCON	ECH	RW	0	0	0	0	0	–	–	0	0
Flash memory user programming enable register	FMUSR	EDH	RW	0	0	0	0	0	0	0	0	0
Flash memory sector address register (High Byte)	FMSECH	EEH	RW	0	0	0	0	0	0	0	0	0
Flash memory sector address register (Low Byte)	FMSECL	EFH	RW	0	0	0	0	0	0	0	0	0
IIC Control Register	ICCR	F0H	RW	0	0	0	0	1	1	1	1	1
IIC Status Register	ICSR	F1H	RW	0	0	0	0	0	0	0	0	0
IIC Data Shift Register	IDSR	F2H	RW	x	x	x	x	x	x	x	x	x
IIC Address Register	IAR	F3H	RW	x	x	x	x	x	x	x	x	x
Low Voltage Detector Control Register	LVDCON	F4H	RW	0	–	0	–	–	–	–	0	0
UART control register	UARTCON	F5H	RW	0	0	0	0	0	0	0	0	0
UART pending register	UARTPND	F6H	RW	–	–	–	–	–	–	–	0	0
UART Baud rate data register	BRDATA	F7H	RW	1	1	1	1	1	1	1	1	1
UART data register	UDATA	F8H	RW	x	x	x	x	x	x	x	x	x
Location F9H to FFH is not mapped												

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



**Figure 4-1 Register Description Format**

**4.1.1 ADCON**

- A/D Converter Control Register: F7H, SET 1, BANK 0

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

.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	ADC9 (P2.5)
1	0	1	0	ADC10 (P2.)
1	0	1	1	ADC11 (P3.0)
1	1	0	0	ADC12 (P3.1)
1	1	0	1	Disable ADC (Power Down)
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	$f_{osc}/16$ ( $f_{osc} \leq 12\text{MHz}$ )
0	1	$f_{osc}/12$ ( $f_{osc} \leq 10\text{MHz}$ )
1	0	$f_{osc}/8$ ( $f_{osc} \leq 4\text{MHz}$ )
1	1	$f_{osc}/4$ ( $f_{osc} \leq 3.2\text{MHz}$ )

.0

Conversion Start Bit

0	No meaning
1	A/D conversion start

**NOTE:**

- Maximum ADC clock input = 850kHz.

2. When you select one ADC channel, the ADC module was enabled; when disable ADC, the ADC enter Power Down mode.

#### 4.1.2 BTCON

- Basic Timer Control Register: D3H, SET 1

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

.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".

### 4.1.3 CLKCON

- Clock Control Register: D4H, SET 1

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

.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 S3F8S28/S3F8S24

.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 S3F8S28/S3F8S24

#### 4.1.4 EMT

- External Memory Timing Register: FEH, SET 1, BANK 0

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

.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 S3F8S28/S3F8S24

**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



**4.1.5 FLAGS**

- System Flags Register: D5H, SET 1

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

#### 4.1.6 FMCON

- Flash Memory Control Register: ECH, SET 1, BANK 1

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

.7–.4

Flash Memory Mode Selection Bits

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

.3–.1

Not used for the S3F8S28/S3F8S24

.0

Flash Operation Start Bit

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

#### 4.1.7 FMSECH

- Flash Memory Sector Address Register (High Byte): EEH, SET 1, BANK 1

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

Flash Memory Sector Address Bits (High Byte)

The 15th to 8th bits to select a sector of Flash ROM

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

### 4.1.8 FMSECL

- Flash Memory Sector Address Register (Low Byte): EFH, SET 1, BANK 1

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

.7 Flash Memory Sector Address Bit (Low Byte)

The 7th bit to select a sector of Flash ROM
---

.6–.0 Bits 6–0

Don't care
------------

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

### 4.1.9 FMUSR

- Flash Memory User Programming Enable Register: EDH, SET 1, BANK 1

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

.7–.0 Flash Memory User Programming Enable Bits

10100101	Enable user programming mode
Other values	Disable user programming mode

**4.1.10 ICCR**

- Multi-master IIC Bus Clock Control Register: F0H, SET1, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
nRESET Value	0	0	0	0	1	1	1	1
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW
Addressing Mode	Register addressing mode only							

.7 Acknowledgement Enable Bit

0	Acknowledgement disable mode
1	Acknowledgement enable mode

.6 Tx Clock Selection Bit

0	$f_{osc}/16$
1	$f_{osc}/512$

.5 Multi-master IIC Bus Tx/Rx Interrupt Enable Bit

0	Disable
1	Enable

.4 Multi-master

0	Interrupt request is not pending; (when read) pending bit clear when write 0
1	Interrupt request is pending (when read)

.3–.0 ICCR.3–0: Transmit Clock 4-Bit prescaler Bits

$SCL\ clock = IICLK/CCR[3:0] + 1$ where, $IICLK = f_{osc}/16$ when IICR.6 is "0", $IICLK = f_{osc}/512$ when IICR.6 is "1"	
---	--

**4.1.11 ICSR**

- IIC Status Register: F1H, SET1, BANK1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
nRESET Value	0	0	0	0	0	0	0	0
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW
Addressing Mode	Register addressing mode only							

.7–.6

IIC Bus Master/Slave Tx/Rx Mode Selection Bits

0	0	Slave receiver mode (default mode)
0	1	Slave transmitter mode
1	0	Master receiver mode
1	1	Master transmitter mode

.5

IIC Bus Busy Bit

0	IIC-bus is not busy
0	Stop condition generation
1	IIC-bus is busy (when read)
1	Stop condition generation (when write)

.4

IIC-bus Interface Module Enable Bit

0	Disable IIC-bus data transmit/receive
1	Enable IIC-bus data transmit/receive

.3

Arbitration Lost Bit

This bit is set by H/W when the serial I/O interface, in master transmit mode, loses a bus arbitration procedure. In slave mode this flag is set to "1" when ICCR.5 is "1" and ICSR.2 is "0"

.2

Address Match Bit

0	When Start or Stop or Reset
1	When received slave address matches to IAR register or general call

.1

General Call Bit

0	When Start/Stop condition is generated
1	When received slave address is "00000000" (general call)

.0

Received Acknowledge Bit

0	ACK is received
1	ACK is not received

**4.1.12 IMR**

- Interrupt Mask Register: DDH, SET 1

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

.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.

#### 4.1.13 IPH

- Instruction Pointer (High Byte): DAH, SET 1

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

.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 to IP8). The lower byte of the IP address is located in the IPL register (DBH).

#### 4.1.14 IPL

- Instruction Pointer (Low Byte): DBH, SET 1

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

.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 to IP0). The upper byte of the IP address is located in the IPH register (DAH).

#### 4.1.15 IPR

- Interrupt Priority Register: FFH, SET 1, BANK 0

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

.7, .4 and .1

Priority Control Bits for Interrupt Groups A, B, and C (NOTE)

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

.6

Interrupt Subgroup C Priority Control Bit

0	IRQ6 > IRQ7
1	IRQ7 > IRQ6

.5

Interrupt Group C Priority Control Bit

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

.3

Interrupt Subgroup B Priority Control Bit

0	IRQ3 > IRQ4
1	IRQ4 > IRQ3

.2

Interrupt Group B Priority Control Bit

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

.0

Interrupt Group A Priority Control Bit

0	IRQ0 > IRQ1
1	IRQ1 > IRQ0

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



**4.1.16 IRQ**

- Interrupt Request Register: DCH, SET 1

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

.7 Level 7 (IRQ7) Request Pending Bit;

0	Not pending
1	Pending

.6 Level 6 (IRQ6) Request Pending Bit;

0	Not pending
1	Pending

.5 Level 5 (IRQ5) Request Pending Bit;

0	Not pending
1	Pending

.4 Level 4 (IRQ4) Request Pending Bit;

0	Not pending
1	Pending

.3 Level 3 (IRQ3) Request Pending Bit;

0	Not pending
1	Pending

.2 Level 2 (IRQ2) Request Pending Bit;

0	Not pending
1	Pending

.1 Level 1 (IRQ1) Request Pending Bit;

0	Not pending
1	Pending

.0 Level 0 (IRQ0) Request Pending Bit;

0	Not pending
1	Pending

**4.1.17 LVDCON**

- Interrupt Request Register: F4H, SET 1, BANK 1

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

.7	LVD Enable/Disable Bit	
	0	LVD Disable
	1	LVD Enable

.6	Not Used in S3F8S28/S3F8S24
----	-----------------------------

.5	LVD Output Bit (Read Only)	
	0	$V_{DD} > V_{LVD}$
	1	$V_{DD} < V_{LVD}$

.4	Not Used in S3F8S28/S3F8S24 (must be kept as "0")
----	---

.3-.2	Not Used in S3F8S28/S3F8S24
-------	-----------------------------

.1–.0	Detection Voltage Level Selection Bits		
	0	0	$V_{LVD0} = 4.1V$
	0	1	$V_{LVD1} = 3.2V$
	1	0	$V_{LVD2} = 2.5V$
	1	1	$V_{LVD3} = 2.1V$

**4.1.18 P0CONH**

- Port 0 Control Register (High Byte): E6H, SET 1, BANK 0

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

.7–.6

Port 0, P0.7/INT7 Configuration Bits

0	x	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/PWM0 Configuration Bits

0	0	Schmitt trigger input
0	1	Alternative function (PWM0 output)
1	0	Push-pull output
1	1	A/D converter input (ADC6); Schmitt trigger input off

.3–.2

Port 0, P0.5/ADC5/PWM1 Configuration Bits

0	0	Schmitt trigger input
0	1	Alternative function (PWM1 output)
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	x	Schmitt trigger input
1	0	Push-pull output
1	1	A/D converter input (ADC4); Schmitt trigger input off

**4.1.19 P0CONL**

- Port 0 Control Register (Low Byte): E7H, SET 1, BANK 0

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

.7–.6

Port 0, P0.3/INT3 Configuration Bits

0	0	Schmitt trigger input/falling edge interrupt input
0	1	Alternative function: SDA input
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/falling edge interrupt input
0	1	Alternative function: SCK input
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	x	Schmitt trigger input/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	x	Schmitt trigger input/falling edge interrupt input
1	0	Push-pull output
1	1	A/D converter input (ADC0); Schmitt trigger input off

**4.1.20 P0PND**

- Port 0 Interrupt Pending Register: E8H, SET 1, BANK 0

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

.7 Port 0.3/ADC3/INT3 Interrupt Enable Bit

0	INT3 falling edge interrupt disable
1	INT3 falling edge interrupt enable

.6 Port 0.3/ADC3/INT3 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)

.5 Port 0.2/ADC2/INT2 Interrupt Enable Bit

0	INT2 falling edge interrupt disable
1	INT2 falling edge interrupt enable

.4 Port 0.2/ADC2/INT2 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)

.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)

**4.1.21 P0PUR**

- Port 0 Pull-up Resistor Enable Register: E5H, SET 1, BANK 0

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

.7 Port 0.7 Pull-up Resistor Enable Bit

0	Enable Pull-up Resistor
1	Disable Pull-up Resistor

.6 Port 0.6 Pull-up Resistor Enable Bit

0	Enable Pull-up Resistor
1	Disable Pull-up Resistor

.5 Port 0.5 Pull-up Resistor Enable Bit

0	Enable Pull-up Resistor
1	Disable Pull-up Resistor

.4 Port 0.4 Pull-up Resistor Enable Bit

0	Enable Pull-up Resistor
1	Disable Pull-up Resistor

.3 Port 0.3 Pull-up Resistor Enable Bit

0	Disable Pull-up Resistor
1	Enable Pull-up Resistor

.2 Port 0.2 Pull-up Resistor Enable Bit

0	Disable Pull-up Resistor
1	Enable Pull-up Resistor

.1 Port 0.1 Pull-up Resistor Enable Bit

0	Disable Pull-up Resistor
1	Enable Pull-up Resistor

.0 Port 0.0 Pull-up Resistor Enable Bit

0	Disable Pull-up Resistor
1	Enable Pull-up Resistor

**4.1.22 P1CON**

- Port 1 Control Register: E9H, SET 1, BANK 0

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

.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 Port 1.0 N-channel open-drain Enable Bit

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

.5

Not used for S3F8S28/S3F8S24	
------------------------------	--

.4 Port 1.2 Configuration Bit

0	Configure P1.2 as a Schmitt trigger input;
1	Configure P1.2 as a open-drain output

.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 Port 1, P1.0 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

**NOTE:** When you use external oscillator, P1.0, P1.1 must be set to output port to prevent current consumption.



**4.1.23 P2CONH**

- Port 2 Control Register (High Byte): EAH, SET 1, BANK 0

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

.7	Not used for the S3F8S28/S3F8S24
----	----------------------------------

.6–.4	Port 2, P2.6/ADC8/CLO Configuration Bits			
	0	0	x	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/ADC9 Configuration Bits		
	0	0	Schmitt trigger input
	0	1	Alternative function: ADC Input
	1	0	Push-pull output
1	1	Invalid	

.1–.0	Port 2, 2.4/ADC10 Configuration Bits		
	0	0	Schmitt trigger input
	0	1	Alternative function: ADC Input
	1	0	Push-pull output
1	1	Invalid	

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

**4.1.24 P2CONL**

- Port 2 Control Register (Low Byte): EBH, SET 1, BANK 0

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

.7–.6

Part 2, P2.3 Configuration Bits

0	0	Schmitt trigger input
0	1	Alternative function: TxD output
1	0	Push-pull output
1	1	Open-drain output

.5–.4

Port 2, P2.2 Configuration Bits

0	0	Schmitt trigger input T1 capture input; RxD input
0	1	Alternative function: RxD output
1	0	Push-pull output
1	1	Open-drain output

.3–.2

Port 2, P2.1 Configuration Bits

0	0	Schmitt trigger input
0	1	Alternative function:T1 match output
1	0	Push-pull output
1	1	Open-drain output

.1–.0

Port 2, P2.0 Configuration Bits

0	0	Schmitt trigger input
0	1	Alternative function:T0 match output
1	0	Push-pull output
1	1	Open-drain output

4.1.25 P2PUR

- Port 2 Pull-up Resistor Enable Register: E4H, SET 1, BANK 0

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

.7	Not used for the S3F8S28/S3F8S24
----	----------------------------------

.6	Port 2.6 Pull-up Resistor Enable Bit	
	0	Enable Pull-up Resistor
	1	Disable Pull-up Resistor

.5	Port 2.5 Pull-up Resistor Enable Bit	
	0	Enable Pull-up Resistor
	1	Disable Pull-up Resistor

.4	Part 2.4 Pull-up Resistor Enable Bit	
	0	Enable Pull-up Resistor
	1	Disable Pull-up Resistor

.3	Part 2.3 Pull-up Resistor Enable Bit	
	0	Enable Pull-up Resistor
	1	Disable Pull-up Resistor

.2	Part 2.2 Pull-up Resistor Enable Bit	
	0	Enable Pull-up Resistor
	1	Disable Pull-up Resistor

.1	Part 2.1 Pull-up Resistor Enable Bit	
	0	Enable Pull-up Resistor
	1	Disable Pull-up Resistor

.0	Part 2.0 Pull-up Resistor Enable Bit	
	0	Enable Pull-up Resistor
	1	Disable Pull-up Resistor

**4.1.26 P3CON**

- Port 3 Control Register: F0H, SET 1, BANK 0

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

.7–.6

Part 3, P3.3 Configuration Bits

0	0	Schmitt trigger input/falling edge interrupt input
0	1	Schmitt trigger input with pull-up/falling edge interrupt input
1	x	Push-pull output

.5–.4

Port 3, P3.2 Configuration Bits

0	0	Schmitt trigger input/falling edge interrupt input
0	1	Schmitt trigger input with pull-up /falling edge interrupt input
1	x	Push-pull output

.3–.2

Port 3, P3.1/ADC12 Configuration Bits

0	0	Schmitt trigger input/falling edge interrupt input
0	1	Schmitt trigger input with pull-up /falling edge interrupt input
1	0	Push-pull output
1	1	Alternative function: ADC input

.1–.0

Port 3, P3.0/ADC11 Configuration Bits

0	0	Schmitt trigger input/falling edge interrupt input
0	1	Schmitt trigger input with pull-up /falling edge interrupt input
1	0	Push-pull output
1	1	Alternative function: ADC input

**4.1.27 P3PND**

- Port 3 Interrupt Pending Register: EFH, SET 1, BANK 0

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

.7 Port 3.3/ADC12/INT7 Interrupt Enable Bit

0	INT7 falling edge interrupt disable
1	INT7 falling edge interrupt enable

.6 Port 3.3/ADC12/INT7 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)

.5 Port 3.2/ADC11/INT6 Interrupt Enable Bit

0	INT6 falling edge interrupt disable
1	INT6 falling edge interrupt enable

.4 Port 3.2/ADC11/INT6 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)

.3 Port 3.1/ADC10/INT5 Interrupt Enable Bit

0	INT5 falling edge interrupt disable
1	INT5 falling edge interrupt enable

.2 Port 3.1/ADC10/INT5 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 3.0/ADC9/INT4 Interrupt Enable Bit

0	INT4 falling edge interrupt disable
1	INT4 falling edge interrupt enable

.0

Port 3.0/ADC9/INT4 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)

#### 4.1.28 PP

- Register Page Pointer: DFH, SET 1

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

.7-.0

Not used for the S3F8S28/S3F8S24.

**NOTE:** In S3F8S28/S3F8S24, only page 0 settings are valid. Register page pointer values for the source and destination register page are automatically set to "00F" following a hardware reset. These values should not be changed during normal operation.

#### 4.1.29 PWM0CON

- PWM0 Control Register: F3H, SET 1, BANK 0

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

.7–.6

PWM Input Clock Selection Bits

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

.5

Not used for S3F8S28/S3F8S24

.4

PWM0DATA Reload Interval Selection Bit

0	Reload from extension up counter overflow
1	Reload from base up counter overflow

.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:** PWM0CON.3 is not autocleared. You must pay attention when clear pending bit. (Refer to page [13-14](#)).

### 4.1.30 PWM1CON

- PWM1 Control Register: E8H, SET 1, BANK 1

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

.7–.6

PWM Input Clock Selection Bits

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

.5

Not used for S3F8S28/S3F8S24

.4

PWM1DATA Reload Interval Selection Bit

0	Reload from extension up counter overflow
1	Reload from base up counter overflow

.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:** PWM1CON.3 is not autocleared. You must pay attention when clear pending bit. (Refer to page [13-14](#)).



### 4.1.31 PWM0EX

- PWM0 Extension Register: F1H, SET 1, BANK 0

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

.7–.2

PWM Extension Bits

PWM extension bits: Bit .7–.2 for 6+6 resolution and 8+6 resolution; Bit .7–.6 for 6+2 resolution
---

.1–.0

PWM Base/extension Control bits:

0	0	Base 6-bit (PWM0DATA.7–.2) + Extension 6-bit (PWM0EX.7–.2)
1	0	
0	1	Base 6-bit (PWM0DATA.5–.0) + Extension 2-bit (PWM0EX.7–.6)
1	1	Base 8-bit (PWM0DATA.7–.0) + Extension 6-bit (PWM0EX.7–.2)

### 4.1.32 PWM1EX

- PWM1 Extension Register: E6H, SET 1, BANK 1

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

.7–.2

PWM Extension Bits

PWM extension bits: Bit .7–.2 for 6+6 resolution and 8+6 resolution Bit .7–.6 for 6+2 resolution
--

.1–.0

PWM Base/extension Control bits:

0	0	Base 6-bit (PWM1DATA.7–.2) + Extension 6-bit (PWM1EX.7–.2)
1	0	
0	1	Base 6-bit (PWM1DATA.5–.0) + Extension 2-bit (PWM1EX.7–.6)
1	1	Base 8-bit (PWM1DATA.7–.0) + Extension 6-bit (PWM1EX.7–.2)

**4.1.33 RESETID**

- Reset Source Indicating Register: EAH, SET 1, BANK1

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

.7–.4	Not used for the S3F8S28/S3F8S24
-------	----------------------------------

.3	Watchdog Timer Reset Indicating Bit	
	0	Reset is not generated by Watch dog (when read)
	1	Reset is generated by Watch dog (when read)

.2	nReset pin Indicating Bit	
	0	Reset is not generated by nReset pin (when read)
	1	Reset is generated by nReset pin (when read)

.1	Basic Timer Reset Indicating Bit	
	0	Reset is not generated by Basic Timer (when read)
	1	Reset is generated by Basic Timer (when read)

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

State of RESETID depends on reset source

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

**NOTE:**

- When LVR is disabled (Smart Option 3FH.7 = 0), RESETID.0 is invalid; when P1.2 is set to be IO (Smart Option 3FH.4 = 0), RESETID.3 is invalid.
- To clear an indicating register, write a "0" to indicating flag bit; writing a "1" has no effect.
- Once a LVR reset happens, RESETID.1 will be set and all the other bits will be cleared to "0" at the same time.
- Once a WDT reset, Basic Timer reset or nRESET pin reset happens, corresponding bit will be set, but leave all other indicating bits unchanged.

**4.1.34 ROSCCON**

- Ring Oscillator Control Register: F5H, SET 1, BANK 0

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

.7 Ring OSC Enable Bit

0	Disable Ring OSC
1	Enable Ring OSC

.6 Free Running Watchdog Timer Clock Source Selection Bit

0	System Clock: Fosc
1	Ring OSC clock

.5–.0 Ring OSC Frequency Trimming Bits

000000	Maximum frequency
111111	Minimum frequency.

**4.1.35 RP0**

- Register Pointer 0: D6H, SET 1

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

.7–.3 Register Pointer 0 Address Value

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

.2–.0

Not used for the S3F8S28/S3F8S24
----------------------------------

**4.1.36 RP1**

- Register Pointer 1: D7H, SET 1

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

.7–.3

Register Pointer 1 Address Value

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

.2–.0

Not used for the S3F8S28/S3F8S24

**4.1.37 SPL**

- Stack Pointer: D9H, SET 1

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

.7–.0

Stack Pointer Address (Low Byte)

The SP value is undefined following a reset.

**4.1.38 STOPCON**

- Stop Mode Control Register: F4H, SET 1, BANK 0

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

.7–.0

Watchdog Timer Function Enable Bit

10100101	Enable STOP instruction
Other value	Disable STOP instruction

**NOTE:**

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

**4.1.39 SYM**

- System Mode Register: DEH, SET 1

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

.7 Tri-state External Interface Control Bit <sup>(1)</sup>

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 S3F8S28/S3F8S24

.4–.2 Fast Interrupt Level Selection Bits <sup>(2)</sup>

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 <sup>(3)</sup>

0	Disable fast interrupt processing
1	Enable fast interrupt processing

.0 Global Interrupt Enable Bit <sup>(4)</sup>

0	Disable all interrupt processing
1	Enable all interrupt processing

**NOTE:**

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

**4.1.40 T1CON**

- Timer 1 Control Register: E4H, SET 1, Bank1

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

.7–.6

Timer 1 Operating Mode Selection Bits

0	0	Interval timer mode (counter cleared by match signal)
0	1	Capture mode (rising edges, counter running, OVF interrupt can occur)
1	0	Capture mode (falling edges, counter running, OVF interrupt can occur)
1	1	Capture mode (both falling edges and rising edges, counter running, OVF interrupt can occur)

.5

Timer 1 Counter Run Enable Bit

0	Stop Timer 1 (Disable Counter Run)
1	Start Timer 1 (Enable Counter Run)

.4

Timer 1 Counter Clear Bit

0	No effect
1	Clear T1 counter, T1CNT (when write, After clearing, return to "0")

.3

Timer 1 Overflow Interrupt Enable Bit

0	Disable T1 overflow interrupt
1	Enable T1 overflow interrupt

.2

Timer 1 Overflow Interrupt Pending Bit

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

.1

Timer 1 Match/Capture Interrupt Enable Bit

0	Disable T1 match/capture interrupt
1	Enable T1 match/capture interrupt

.0

Timer 1 Match/Capture Interrupt Pending Bit

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

**NOTE:** A Timer 1 overflow interrupt pending condition is automatically cleared by hardware. However, the Timer 1 match/capture interrupt, IRQ3, vector ECH, must be cleared by the interrupt service routine (S/W).

#### 4.1.41 T1PS

- Timer 1 Prescaler Register (Low Byte): E5H, SET 1, Bank1

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

.7-.4

Not used for the S3F8S28/S3F8S24

.3-.0

Timer 1 prescaler bits

T1 clock =  $F_{OSC}/(2T1PS[3-0])$  prescaler values above 12 are invalid

#### 4.1.42 TACON

- Timer 0/A Control Register: D2H, SET 1

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

.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)

**NOTE:**

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.



#### 4.1.43 TBCON

- Timer B Control Register: EEH, SET 1, BANK 0

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

.7 and .6	Not used for the S3F8S28/S3F8S24
-----------	----------------------------------

.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".

**4.1.44 UARTCON**

- UART Control Register: F5H, Set 1, Bank 1

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

.7–.6 Operating mode and baud rate selection bits

0	0	Mode 0: SIO mode [ $f_{xx}/(16 \times (BRDATA + 1))$ ]
0	1	Mode 1: 8-bit UART [ $f_{xx}/(16 \times (BRDATA + 1))$ ]
1	0	Mode 2: 9-bit UART [ $f_{xx}/16$ ]
1	1	Mode 3: 9-bit UART [ $f_{xx}/(16 \times (BRDATA + 1))$ ]

.5 Multiprocessor communication <sup>(1)</sup> enable bit (for modes 2 and 3 only)

0	Disable
1	Enable

.4 Serial data receive enable bit

0	Disable
1	Enable

.3 Location of the 9th data bit to be transmitted in UART Mode 2 or 3 ("0" or "1")

--	--

.2 Location of the 9th data bit that was received in UART Mode 2 or 3 ("0" or "1")

--	--

.1 Receive interrupt enable bit

0	Disable Receive interrupt
1	Enable Receive interrupt

.0 Transmit interrupt enable bit

0	Disable Transmit interrupt
1	Enable Transmit Interrupt

**NOTE:**

1. In mode 2 or 3, if the MCE (UARTCON.5) bit is set to "1", then the receive interrupt will not be activated if the received 9th data bit is "0". In mode 1, if MCE = "1", then the receive interrupt will not be activated if a valid stop bit was not received. In mode 0, the MCE (UARTCON.5) bit should be "0".
2. The descriptions for 8-bit and 9-bit UART Mode do not include start and stop bits for serial data receive and transmit.

**4.1.45 UARTPND**

- UART Pending and parity control: F6H, Set 1, Bank 1

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

.7–.2	Not used for the S3F8S28/S3F8S24
-------	----------------------------------

.1	UART receive interrupt pending flag	
	0	Not pending
	0	Clear pending bit (when write)
	1	Interrupt pending

.0	UART transmit interrupt pending flag	
	0	Not pending
	0	Clear pending bit (when write)
	1	Interrupt pending

**NOTE:**

1. In order to clear a data transmit or receive interrupt pending flag, you must write a "0" to the appropriate pending bit.
2. To avoid programming errors, we recommend using load instruction (except for LDB), when manipulating UARTPND values.

#### 4.1.46 WDTCN

- Watchdog Timer Control Register: F6H, Set 1, Bank 0

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

.7 Watchdog Timer Enable bit

0	Disable Watchdog Timer
1	Enable Watchdog Timer

.6 Watchdog Time Overflow Reset Enable bit

0	Disable Overflow Reset
1	Enable Overflow Reset

.5 Watchdog Timer Interrupt Enable bit

0	Disable Interrupt
1	Enable Interrupt

.4 Watchdog Timer Counter bit 10 Clear bit

0	No effect
1	Clear counter bit 10 (when write)

.3–.0 Watchdog clock prescaler bits

Watchdog clock = FLCLK/(2WDPS[3–0])	
-------------------------------------	--

**NOTE:** FLCLK means the clock source for free running Watchdog Timer, that was selected by ROSCCON.6

# 5 Interrupt Structure

## 5.1 Overview

The S3C8/S3F8 Series interrupt structure has three basic components: levels, vectors, and sources. The SAM8RC CPU recognizes up to eight interrupt levels and supports up to 128 interrupt vectors. When a specific interrupt level has more than one vector address, the vector priorities are established in hardware. A vector address can be assigned to one or more sources.

### 5.1.1 Levels

Interrupt levels are the main unit for interrupt priority assignment and recognition. All peripherals and I/O blocks can issue interrupt requests. In other words, peripheral and I/O operations are interrupt-driven. There are eight possible interrupt levels: IRQ0 to IRQ7, also called level 0 to 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 S3F8S28/S3F8S24 interrupt structure recognizes eight interrupt levels.

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

### 5.1.2 Vectors

Each interrupt level can have one or more interrupt vectors, or it may have no vector address assigned at all. The maximum number of vectors that can be supported for a given level is 128 (The actual number of vectors used for S3C8/S3F8 Series devices is always much smaller). If an interrupt level has more than one vector address, the vector priorities are set in hardware. S3F8S28/S3F8S24 uses 17 vectors.

### 5.1.3 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 S3F8S28/S3F8S24 interrupt structure, there are 17 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.

## 5.2 Interrupt Types

The three components of the S3C8/S3F8 interrupt structure described before - levels, vectors, and sources - are combined to determine the interrupt structure of an individual device and to make full use of its available interrupt logic. There are three possible combinations of interrupt structure components, called interrupt types 1, 2, and 3. The types differ in the number of vectors and interrupt sources assigned to each level (see [Figure 5-1](#)):

- Type 1: One level (IRQn) + one vector (V1) + one source (S1)
- Type 2: One level (IRQn) + one vector (V1) + multiple sources (S1 – Sn)
- Type 3: One level (IRQn) + multiple vectors (V1 – Vn) + multiple sources (S1 – Sn, Sn + 1 – Sn + m)

In the S3F8S28/S3F8S24 microcontroller, two interrupt types are implemented.

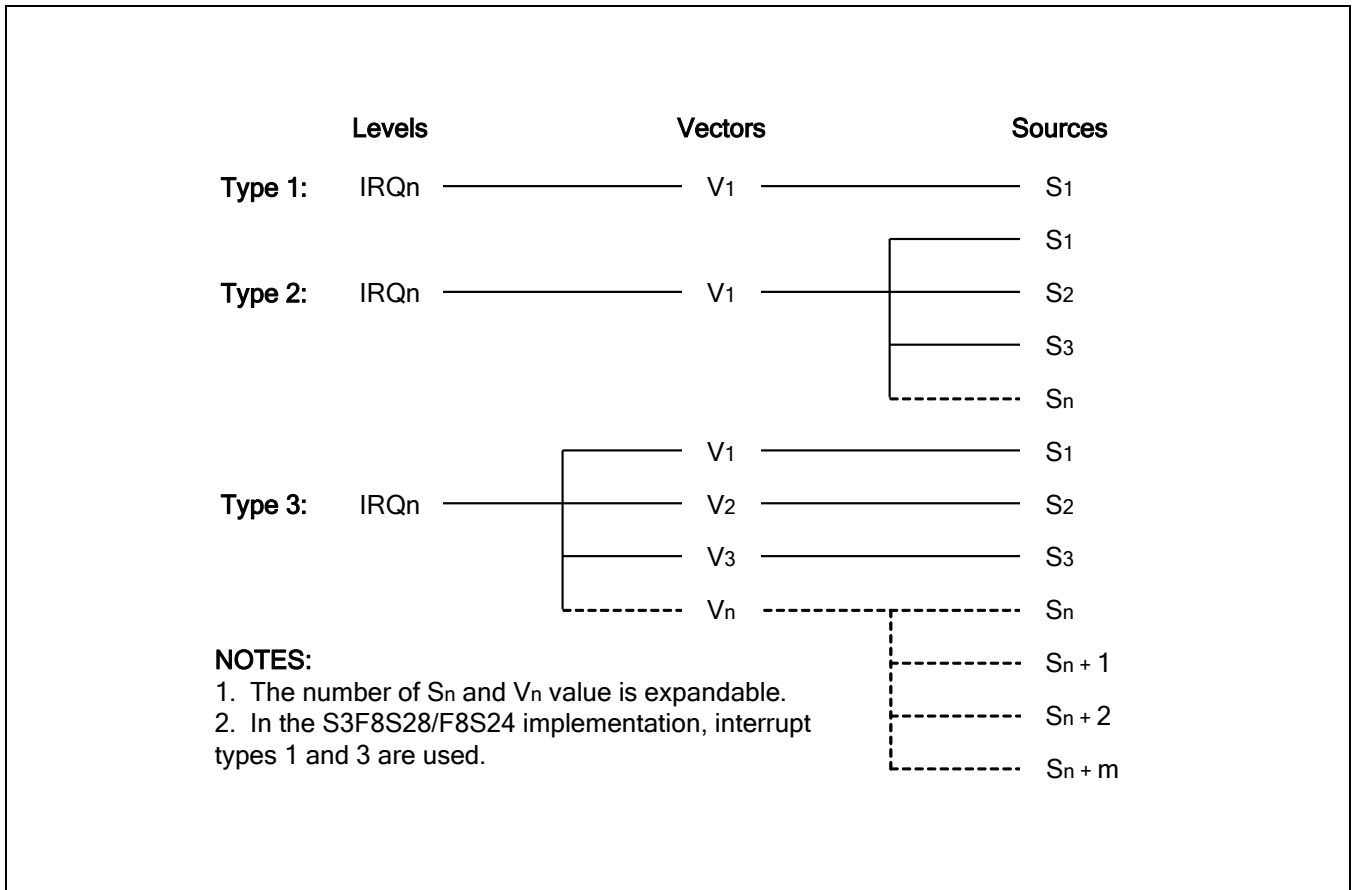


Figure 5-1 S3C8/S3F8 Series Interrupt Types

### 5.3 S3F8S28/S3F8S24 Interrupt Structure

The S3F8S28/S3F8S24 microcontroller supports 17 interrupt sources. Every interrupt source has a corresponding interrupt address. Eight interrupt levels are recognized by the CPU in this device-specific interrupt structure, as shown in [Figure 5-2](#).

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

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

Levels	Vectors	Sources	Reset/Clear
RESET	100H	Basic timer overflow	H/W
IRQ0	FEH	External interrupt 0	S/W
	FCH	External interrupt 1	S/W
	FAH	External interrupt 2	S/W
	F8H	External interrupt 3	S/W
IRQ1	F6H	Timer 0/A match interrupt	S/W
	F4H	Timer B match interrupt	S/W
IRQ2	F2H	PWM0 overflow interrupt	S/W
	F0H	PWM1 overflow interrupt	S/W
IRQ3	ECH	Timer 1 match/capture interrupt	S/W
	EAH	Timer 1 overflow interrupt	H/W
IRQ4	E6H	Watchdog interrupt	H/W
IRQ5	E4H	UART transmit interrupt	S/W
	E2H	UART Receive interrupt	S/W
IRQ6	E0H	IIC transmit / receive interrupt	S/W
IRQ7	DEH	External interrupt 4	S/W
	DCH	External interrupt 5	S/W
	DAH	External interrupt 6	S/W
	D8H	External interrupt 7	S/W

**NOTE:** External interrupts are triggered by a falling edge.

**Figure 5-2 S3F8S28/S3F8S24 Interrupt Structure**

### 5.3.1 Interrupt Vector Addresses

All interrupt vector addresses for the S3F8S28/S3F8S24 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.

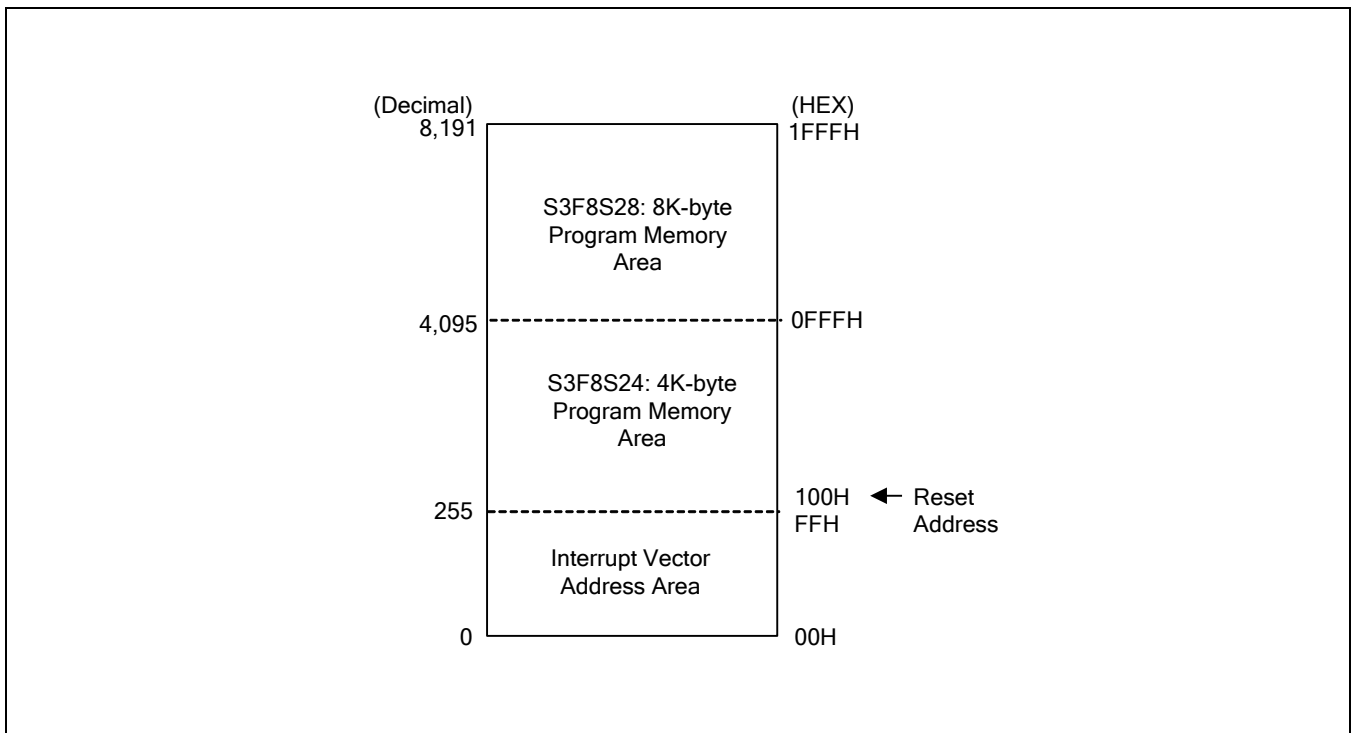


Figure 5-3 ROM Vector Address Area

### 5.3.2 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.



## 5.4 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	RW	Function Description
Interrupt mask register	IMR	RW	Bit settings in the IMR register enable or disable interrupt processing for each of the eight interrupt levels: IRQ0 to IRQ7.
Interrupt priority register	IPR	RW	Controls the relative processing priorities of the interrupt levels. The eight levels of S3F8S28/S3F8S24 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	RW	This register enables/disables fast interrupt processing, and dynamic global interrupt processing.

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

## 5.5 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.

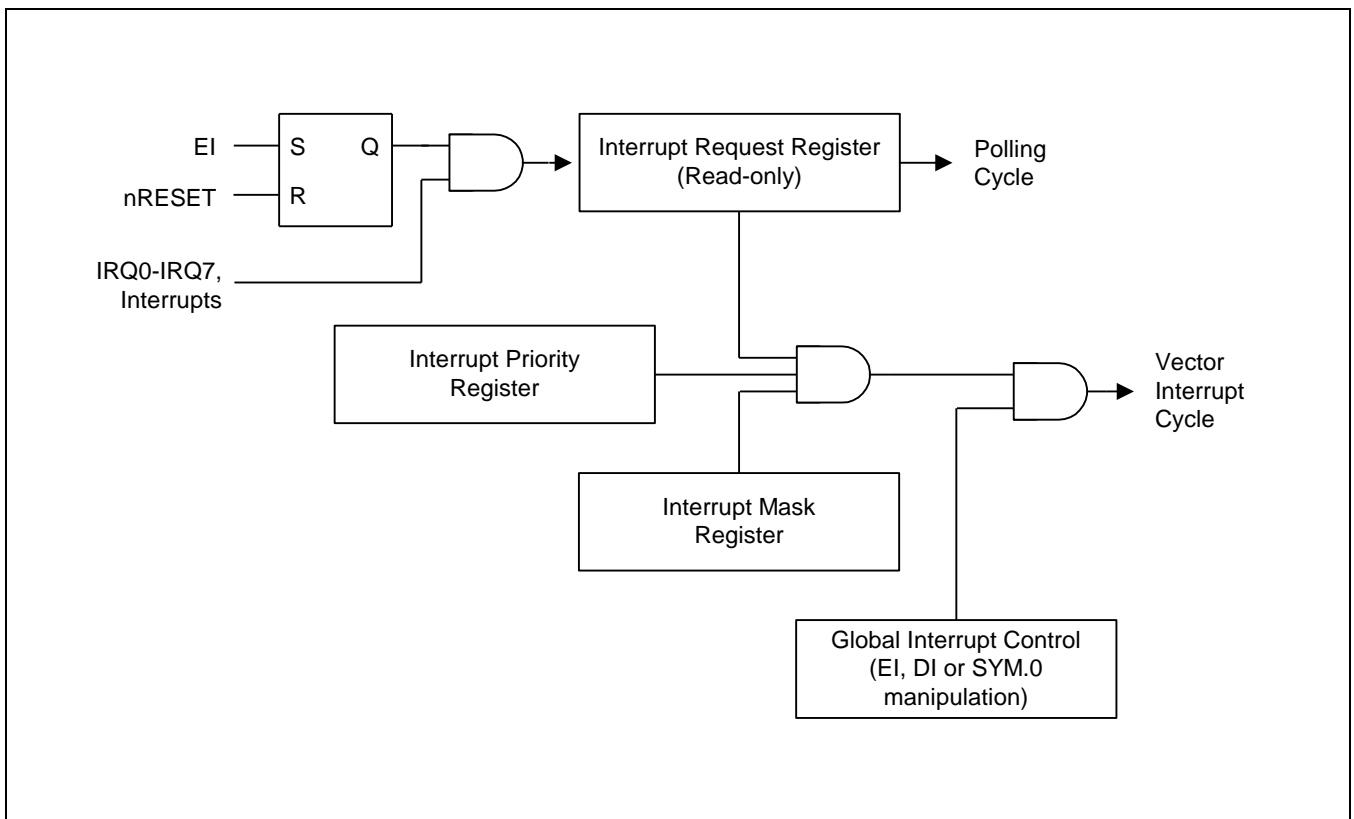


Figure 5-4 Interrupt Function Diagram

## 5.6 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**

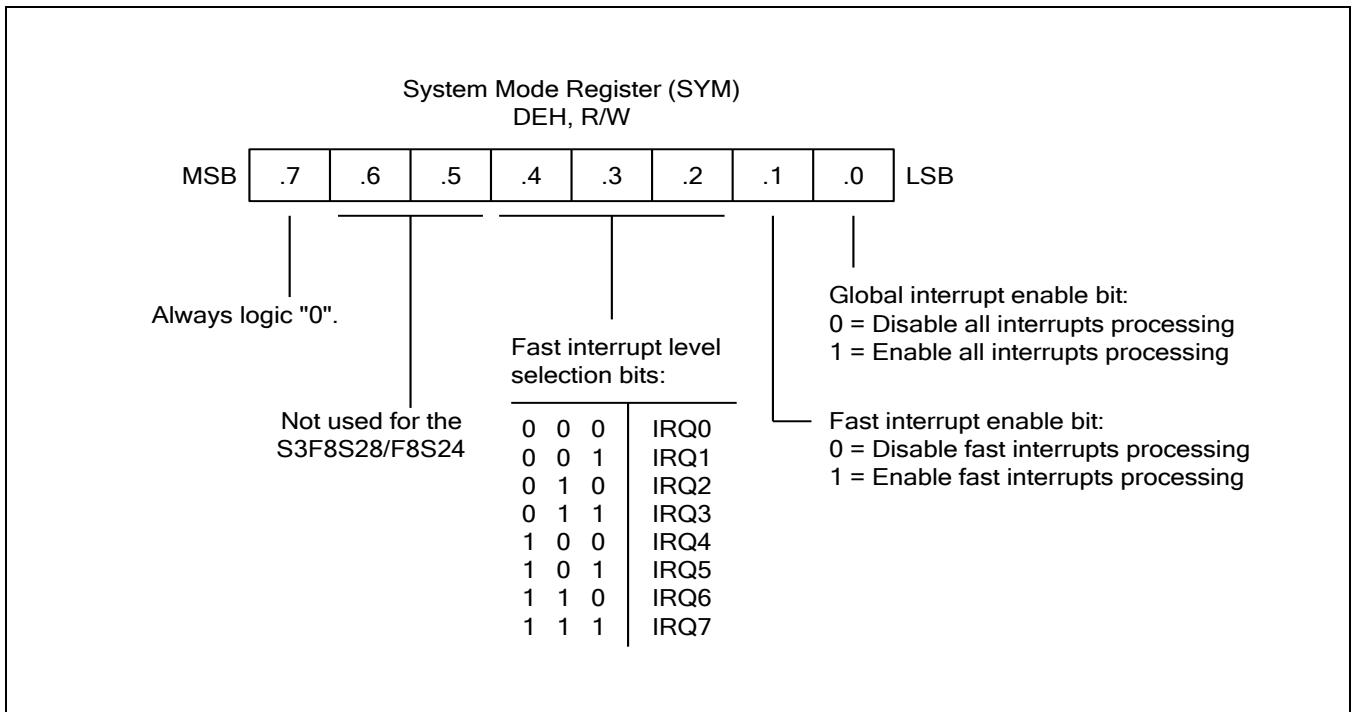
Interrupt Source	Interrupt Level	Register (s)	Location (s)
P0.0 to P0.3 external interrupt	IRQ0	P0CONL P0PND	E7H, Bank0 E8H, Bank0
Timer 0/A match interrupt Timer B match interrupt	IRQ1	TACON P2CONL TBCON	D0H, Bank0 EBH, Bank0 D1H, Bank0
PWM0 overflow interrupt PWM1 overflow interrupt	IRQ2	PWM0CON PWM1CON P0CONH	F3H, Bank0 E8H, Bank1 E6H, Bank0
Timer 1 match/capture interrupt Timer 1 overflow interrupt	IRQ3	T1CON	E4H, Bank1
Watchdog interrupt	IRQ4	WDTCON	F6H, Bank0
UART Transmit interrupt UART Receive interrupt	IRQ5	UARTCON UARTPND BRDATA UDATA	F5H, Bank1 F6H, Bank1 F7H, Bank1 F8H, Bank1
IIC Transmit/Receive interrupt	IRQ6	ICCR ICSR IDSR IAR	F0H, Bank1 F1H, Bank1 F2H, Bank1 F3H, Bank1
P3.0 to P3.3 external interrupt	IRQ7	P3CON P3PND	F0H, Bank0 EFH, Bank0

## 5.7 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 to 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.



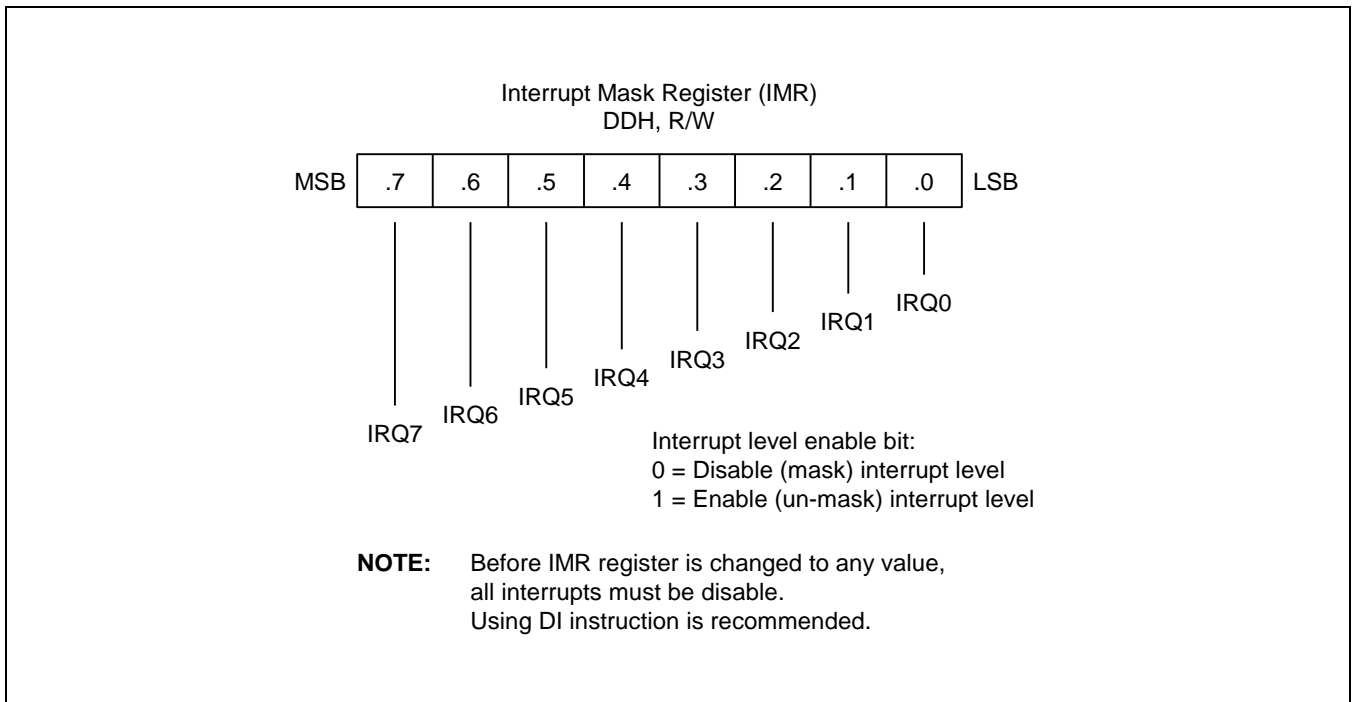
**Figure 5-5 System Mode Register (SYM)**

## 5.8 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)**

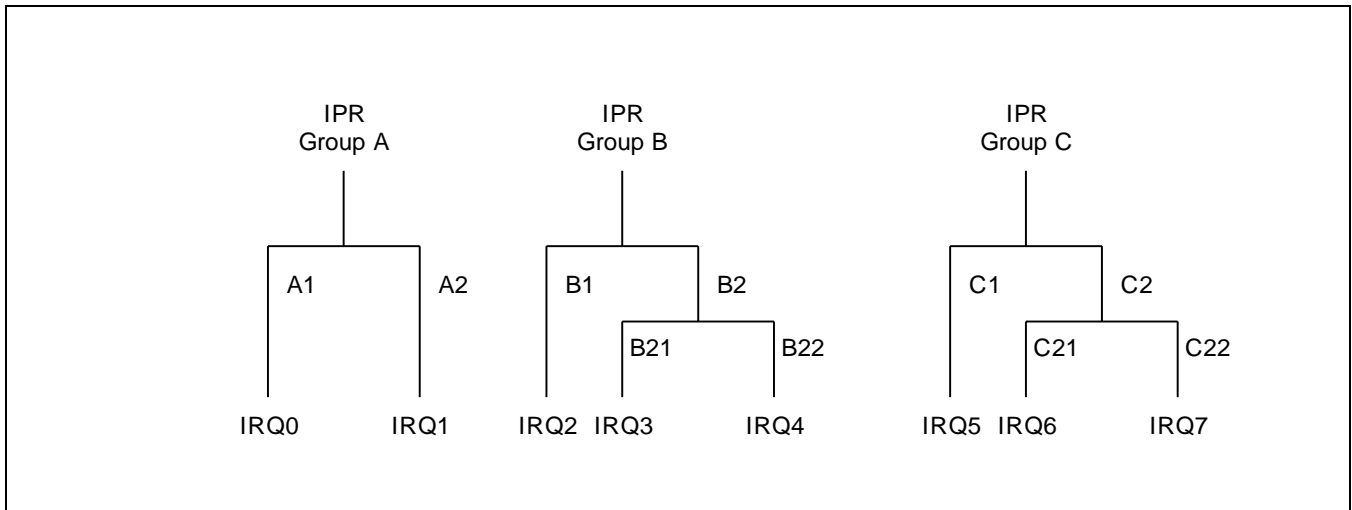
## 5.9 Interrupt Priority Register (IPR)

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

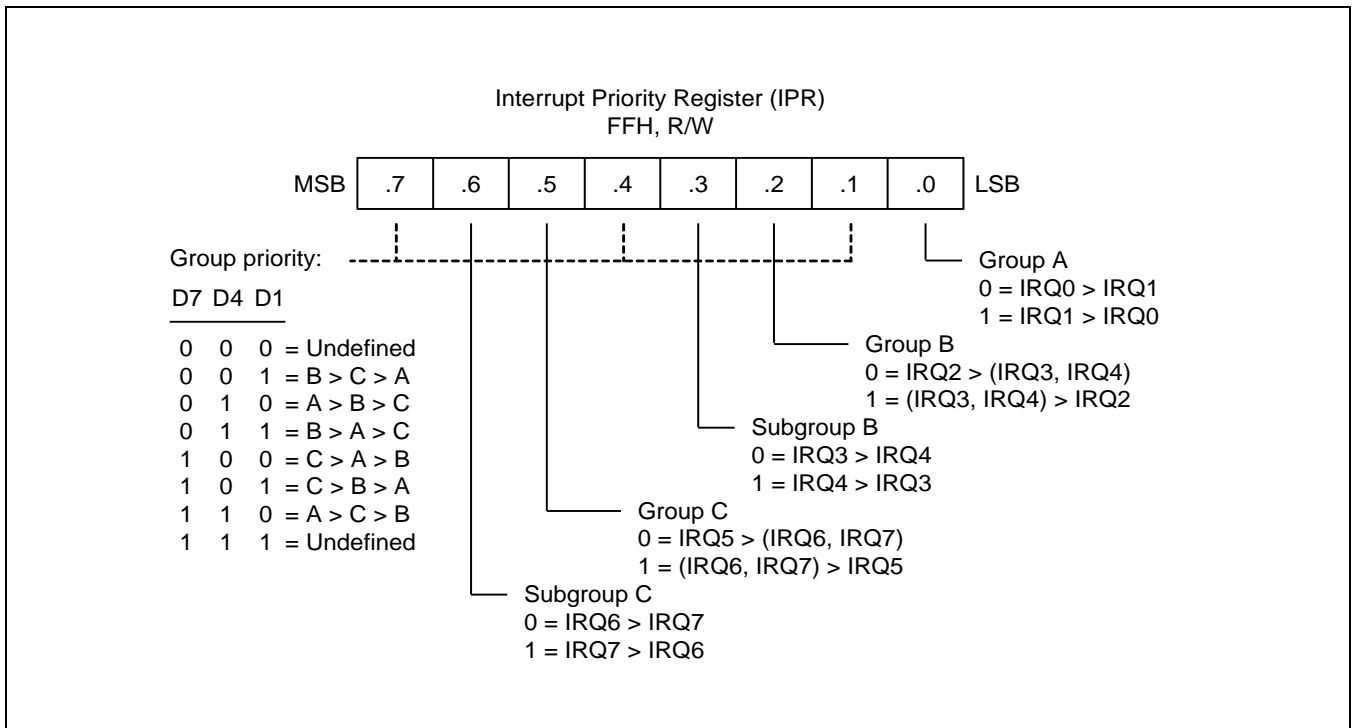


**Figure 5-7** Interrupt Request Priority Groups

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.



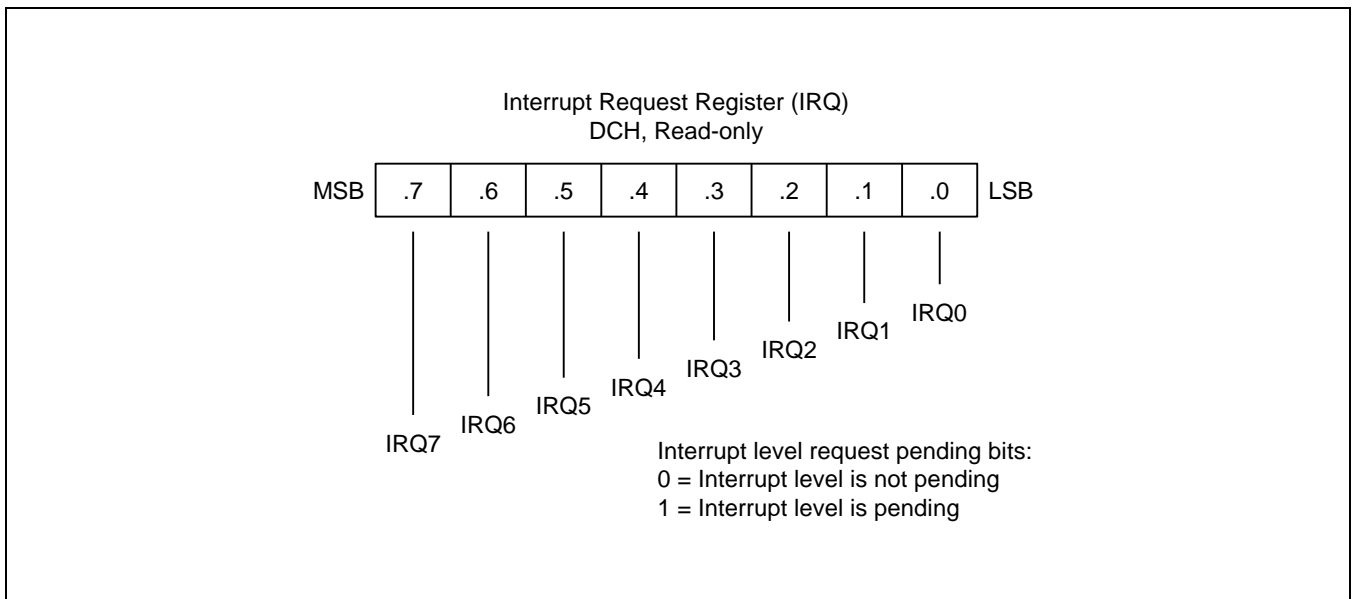
**Figure 5-8 Interrupt Priority Register (IPR)**

## 5.10 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)



## 5.11 Interrupt Pending Function Types

### 5.11.1 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.

### 5.11.2 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 S3F8S28/S3F8S24 interrupt structure, Timer 1 overflow interrupt and Watchdog Timer interrupt belong to this category of interrupts in which pending condition is cleared automatically by hardware.

### 5.11.3 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.

## 5.12 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.

## 5.13 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.

## 5.14 Generating Interrupt Vector Addresses

The interrupt vector area in the ROM (00H to 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 to FFH.

## 5.15 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.

## 5.16 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 to IP8) and IPL (low byte, IP7 to IP0).

## 5.17 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 to SYM.2. Then, to enable fast interrupt processing for the selected level, you set SYM.1 to "1".

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 S3F8S28/S3F8S24 microcontroller, the service routine for any one of the eight interrupt levels: IRQ0 to IRQ7, can be selected for fast interrupt processing.

## 5.18 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 to SYM.2)
3. Write a "1" to the fast interrupt enable bit in the SYM register.

## 5.19 Fast Interrupt Service Routine

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

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

## 5.20 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.

## 5.21 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

## 6.1 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

### 6.1.1 Data Types

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

### 6.1.2 Register Addressing

To access an individual register, an 8-bit address in the range 0 to 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](#).

### 6.1.3 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
<b>Load Instructions</b>		
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 predecrement
LDCPD	dst,src	Load program memory with predecrement
LDEPI	dst,src	Load external data memory with preincrement
LDCPI	dst,src	Load program memory with preincrement
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)
<b>Arithmetic Instructions</b>		
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
<b>Logic Instructions</b>		
AND	dst,src	Logical AND
COM	dst	Complement

Mnemonic	Operands	Instruction
OR	dst,src	Logical OR
XOR	dst,src	Logical exclusive OR
<b>Program Control Instructions</b>		
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
<b>Bit Manipulation Instructions</b>		
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
<b>Rotate and Shift Instructions</b>		
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
<b>CPU Control Instructions</b>		
CCF	–	Complement carry flag
DI	–	Disable interrupts



Mnemonic	Operands	Instruction
EI	–	Enable interrupts
IDLE	–	Enter Idle mode
NOP	–	No operation
RCF	–	Reset carry flag
SB0	–	Set bank 0
SB1	–	Set bank 1
SCF	–	Set carry flag
SRP	src	Set register pointers
SRP0	src	Set register pointer 0
SRP1	src	Set register pointer 1
STOP	–	Enter Stop mode

## 6.2 Flags Register (FLAGS)

The flags register FLAGS contains eight bits that describe the current status of CPU operations. Four of these bits, FLAGS.7 to 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.

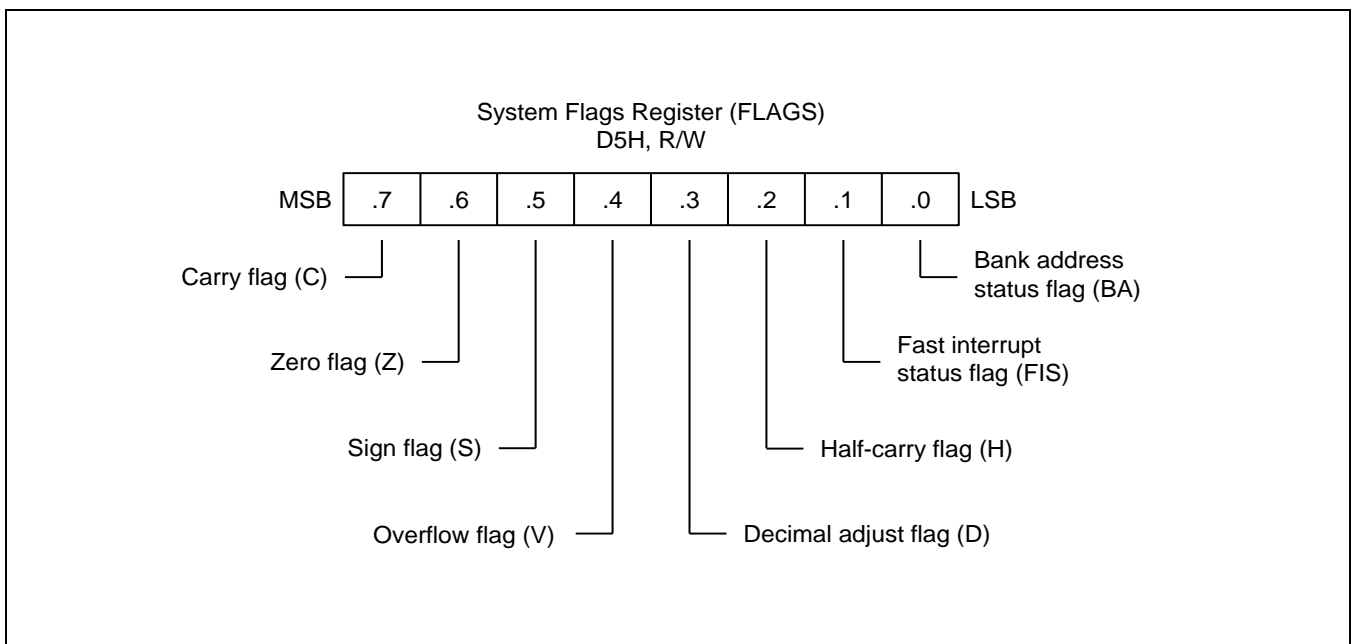


Figure 6-1 System Flags Register (FLAGS)

### 6.2.1 Flag Descriptions

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

### 6.3 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 <a href="#">Table 6-7</a> .
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 to 255, n = 0 to 15)
Rb	Bit (b) of register or working register	reg.b (reg = 0 to 255, b = 0 to 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 to 254, even number only)
lr	Indirect working register only	@Rn (n = 0 to 15)
IR	Indirect register or indirect working register	@Rn or @reg (reg = 0 to 255, n = 0 to 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 to 254, even only, where p = 0, 2, ..., 14)
X	Indexed addressing mode	#reg [Rn] (reg = 0 to 255, n = 0 to 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 to 65535, where p = 0, 2, ..., 14)
da	Direct addressing mode	addr (addr = range 0 to 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 to 255)
iml	Immediate (long) addressing mode	#data (data = range 0 to 65535)

**Table 6-5 OPCODE Quick Reference**

OPCODE Map									
Lower Nibble (HEX)									
	-	0	1	2	3	4	5	6	7
U	0	DEC R1	DEC IR1	ADD r1,r2	ADD r1,lr2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	BOR r0-Rb
P	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,lr2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	BCP r1.b, R2
P	2	INC R1	INC IR1	SUB r1,r2	SUB r1,lr2	SUB R2,R1	SUB IR2,R1	SUB R1,IM	BXOR r0-Rb
E	3	JP IRR1	SRP/0/1 IM	SBC r1,r2	SBC r1,lr2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	BTJR r2.b, RA
R	4	DA R1	DA IR1	OR r1,r2	OR r1,lr2	OR R2,R1	OR IR2,R1	OR R1,IM	LDB r0-Rb
	5	POP R1	POP IR1	AND r1,r2	AND r1,lr2	AND R2,R1	AND IR2,R1	AND R1,IM	BITC r1.b
N	6	COM R1	COM IR1	TCM r1,r2	TCM r1,lr2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	BAND r0-Rb
I	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,lr2	TM R2,R1	TM IR2,R1	TM R1,IM	BIT r1.b
B	8	DECW RR1	DECW IR1	PUSHUD IR1,R2	PUSHUI IR1,R2	MULT R2,RR1	MULT IR2,RR1	MULT IM,RR1	LD r1, x, r2
B	9	RL R1	RL IR1	POPUD IR2,R1	POPUI IR2,R1	DIV R2,RR1	DIV IR2,RR1	DIV IM,RR1	LD r2, x, r1
L	A	INCW RR1	INCW IR1	CP r1,r2	CP r1,lr2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, lrr2, xL
E	B	CLR R1	CLR IR1	XOR r1,r2	XOR r1,lr2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, lrr2, xL
	C	RRC R1	RRC IR1	CPIJE lr,r2,RA	LDC r1,lrr2	LDW RR2,RR1	LDW IR2,RR1	LDW RR1,IML	LD r1, lr2
H	D	SRA R1	SRA IR1	CPIJNE lrr,r2,RA	LDC r2,lrr1	CALL IA1		LD IR1,IM	LD lrr1, r2
E	E	RR R1	RR IR1	LDCD r1,lrr2	LDCI r1,lrr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, lrr2, xs
X	F	SWAP R1	SWAP IR1	LDCPD r2,lrr1	LDCPI r2,lrr1	CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, lrr1, xs

Table 6-6 OPCODE Quick Reference

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

## 6.4 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 6-7](#).

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

**Table 6-7 Condition Codes**

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

**NOTE:**

1. It indicates 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.



## 6.5 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

### 6.5.1 ADC (Add with Carry)

**ADC** dst, src

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

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

**Format:**

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

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

```

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

In the first example, destination register R1 contains the value 10H, 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.

### 6.5.2 ADD (Add)

**ADD** dst, src

**Operation:**  $dst \leftarrow dst + src$   
 The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed.

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

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode				
					dst	src			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst   src</td> </tr> </table>	opc	dst   src		2	4	02	r	r	
	opc	dst   src							
			6	03	r	lr			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst		3	6	04	R	R
	opc	src	dst						
			6	05	R	IR			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src		3	6	06	R	IM
opc	dst	src							

**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.

### 6.5.3 AND (Logical AND)

**AND** dst, src

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

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

**Format:**

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

### 6.5.4 BAND (Bit AND)

**BAND** dst, src.b

**BAND** dst.b, src

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

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

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

**Format:**

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

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

**Examples:** Given: R1 = 07H and Register 01H = 05H:

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

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

### 6.5.5 BCP (Bit Compare)

**BCP** dst, src.b

**Operation:** dst (0) – src (b)  
 The specified bit of the source is compared to (subtracted from) bit zero (LSB) of the destination. The zero flag is set if the bits are the same; otherwise it is cleared. The contents of both operands are unaffected by the comparison.

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

**Format:**

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

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

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

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

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

### 6.5.6 BITC (Bit Complement)

**BITC**            dst.b

**Operation:**    dst (b) ← NOT dst (b)  
 This instruction complements the specified bit within the destination without affecting any other bits 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		
<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst   b   0</td> </tr> </table>	opc	dst   b   0	2	4	57	rb
opc	dst   b   0					

**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 = 7H

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.

### 6.5.7 BITR (Bit Reset)

**BITR**            dst.b

**Operation:**    dst (b) ← 0  
 The BITR instruction clears the specified bit within the destination without affecting any other bits in the destination.

**Flags:**            No flags are affected.

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode
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).



### 6.5.8 BITS (Bit Set)

**BITS**            dst.b

**Operation:**    dst (b) ← 1  
 The BITS instruction sets the specified bit within the destination without affecting any other bits in the destination.

**Flags:**            No flags are affected.

**Format:**

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst		
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst   b   1</td> </tr> </table>	opc	dst   b   1	2	4	77	rb
opc	dst   b   1					

**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).

### 6.5.9 BOR (Bit OR)

**BOR** dst, src.b

**BOR** dst.b, src

**Operation:** dst (0) ← dst (0) OR src (b)  
                   or  
                   dst (b) ← dst (b) OR src(0)

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

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

**Format:**

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	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 3byte 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.

### 6.5.10 BTJRF (Bit Test, Jump Relative on False)

**BTJRF**          dst, src.b

**Operation:**    If src (b) is a "0", then  $PC \leftarrow PC + dst$   
 The specified bit within 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)			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	src   b   0	dst	3	10	37	RA	rb

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

**Example:**      Given: R1 = 07H:

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

If working register R1 contains the value 07H (00000111B), the statement "BTJRF SKIP, R1.3" tests 3-bit. 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.)

### 6.5.11 BTJRT (Bit Test, Jump Relative on True)

**BTJRT**          dst, src.b

**Operation:**    If src (b) is a "1", then  $PC \leftarrow PC + dst$   
 The specified bit within the source operand is tested. If 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)			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	src   b   1	dst	3	10	37	RA	rb

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

**Example:**      Given: R1 = 07H:

BTJRT SKIP, R1.1

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

### 6.5.12 BXOR (Bit XOR)

**BXOR** dst, src.b

**BXOR** dst.b, src

**Operation:** dst (0) ← dst (0) XOR src (b)  
 or

dst (b) ← dst (b) XOR src (0)

The specified bit of 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:**

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

**NOTE:** In the second byte of the 3byte 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.

### 6.5.13 CALL (Call Procedure)

**CALL**            dst

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

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

**Flags:**            No flags are affected.

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode 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 = 000H (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.

### 6.5.14 CCF (Complement Carry Flag)

#### CCF

**Operation:**  $C \leftarrow \text{NOT } C$   
 The carry flag (C) is complemented. If C = "1", the value of the carry flag is changed to logic zero; if C = "0", the value of the carry flag is changed to logic one.

**Flags:** **C:** Complementated.  
 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.

### 6.5.15 CLR (Clear)

**CLR**            dst

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

**Flags:**        No flags are affected.

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode 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.



### 6.5.16 COM (Complement)

**COM**           dst

**Operation:**   dst ← NOT dst  
 The contents of the destination location are complemented (one's complement); all "1s" are changed to "0s", and vice-versa.

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

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode		
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 5px;">opc</td> <td style="padding: 5px;">dst</td> </tr> </table>	opc	dst		2	4	60	R
	opc	dst					
			4	61	IR		

**Examples:**   Given: R1 = 07H and Register 07H = 0F1H:

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

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

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

### 6.5.17 CP (Compare)

**CP** dst, src

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

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

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode	dst	src
opc	dst   src	2	4	A2	r		r
			6	A3	r	lr	
opc	src	3	6	A4	R		R
			6	A5	R	IR	
opc	dst	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.

### 6.5.18 CPIJE (Compare, Increment, and Jump on Equal)

**CPIJE**          dst, src, RA

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

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

**Flags:**          No flags are affected.

**Format:**

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

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

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

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

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

### 6.5.19 CPIJNE (Compare, Increment, and Jump on Non-Equal)

**CPIJNE**      dst, src, RA

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

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

**Flags:**        No flags are affected.

**Format:**

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

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

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

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

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

6.5.20 DA (Decimal Adjust)

DA dst

Operation: dst ← DA dst

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

Instruction	Carry Before DA	Bits 4–7 Value (Hex)	H Flag Before DA	Bits 0–3 Value (Hex)	Number Added to Byte	Carry After DA
–	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
opc	dst	2	4	40	R
			4	41	IR

**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:

```
  0001  0101  15
+ 0010  0111  27
-----
  0011  1100  =  3CH
```

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

```
  0011  1100
+ 0000  0110
-----
  0100  0010  =   42
```

Assuming the same values given above, the statements

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

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

**6.5.21 DEC (Decrement)**

**DEC**            dst

**Operation:**     $dst \leftarrow dst - 1$   
 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		
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 5px;">opc</td> <td style="padding: 5px;">dst</td> </tr> </table>	opc	dst		2	4	00	R
	opc	dst					
			4	01	IR		

**Examples:**    Given: R1 = 03H and Register 03H = 10H:

DEC R1    →    R1 = 02H  
 DEC @R1 →    Register 03H = 0FH

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

### 6.5.22 DECW (Decrement Word)

**DECW**          dst

**Operation:**     $dst \leftarrow dst - 1$   
 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.

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
```



### 6.5.23 DI (Disable Interrupts)

#### DI

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

**Flags:** No flags are affected.

#### Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	8F

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

DI

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

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

### 6.5.24 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  $\geq 2^8$ , 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  $\geq 2^8$  or if divisor = "0"; cleared otherwise.  
**D:** Unaffected.  
**H:** Unaffected.

**Format:**

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

**NOTE:** Execution takes 10 cycles if 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).

### 6.5.25 DJNZ (Decrement and Jump if Non-Zero)

**DJNZ**            r, dst

**Operation:**

$r \leftarrow r - 1$

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

The working register 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.

### 6.5.26 EI (Enable Interrupts)

#### EI

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

**Flags:** No flags are affected.

#### Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	9F

**Example:** Given: SYM = 00H:

EI

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

6.5.27 ENTER (Enter)

ENTER

**Operation:** SP ← SP – 2  
 @SP ← IP  
 IP ← PC  
 PC ← @IP  
 IP ← IP + 2

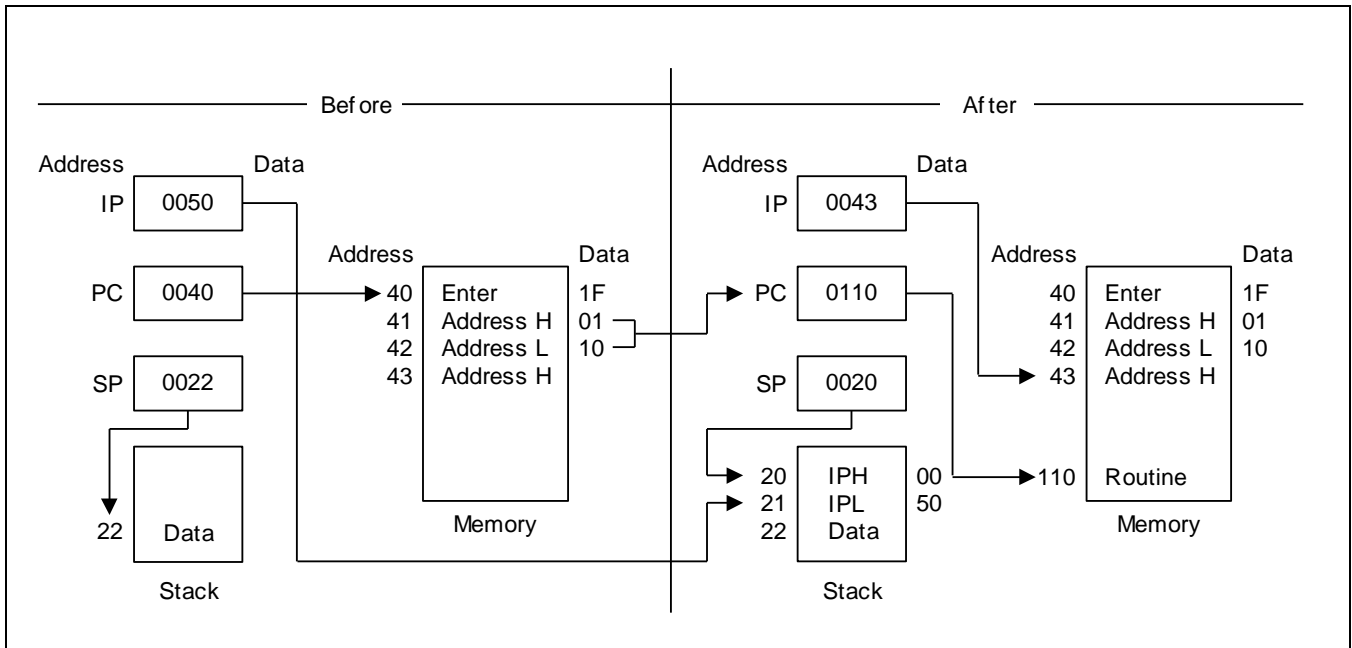
This instruction is useful when implementing threaded-code languages. The contents of the instruction pointer are pushed to the stack. The program counter (PC) value is then written to the instruction pointer. The program memory word that is pointed to by the instruction pointer is loaded into the PC,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



6.5.28 EXIT (Exit)

EXIT

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

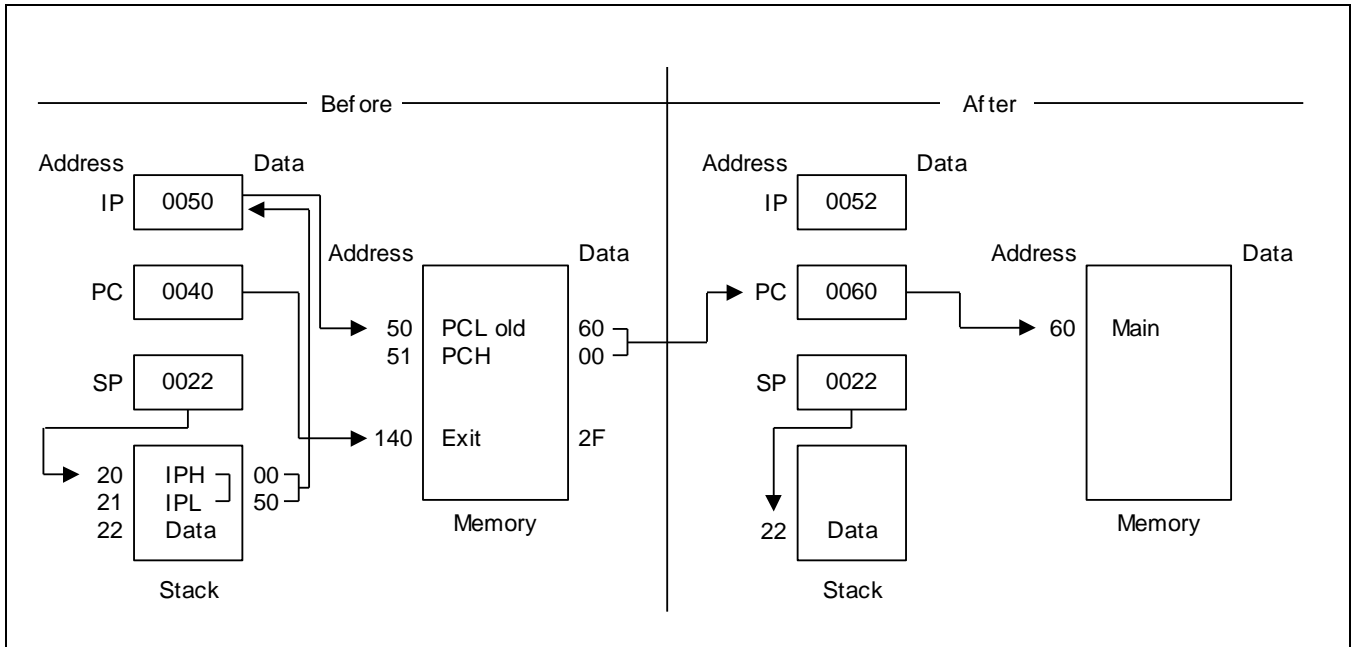
This instruction is useful when implementing threaded-code languages. The stack value is popped and loaded into the instruction pointer. The program memory word that is pointed to by the instruction pointer is then loaded into the program counter, and the instruction pointer is incremented by two.

**Flags:** No flags are affected.

**Format:**

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

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



### 6.5.29 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 adequate time interval for the clock to stabilize before the next instruction is executed. If three or more NOP instructions are not used after IDLE instruction, leakage current could be flown because of the floating state in the internal bus.

**Flags:** No flags are affected.

**Format:**

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

**Example:** The instruction  
 IDLE ; stops the CPU clock but not the system clock  
 NOP  
 NOP  
 NOP

### 6.5.30 INC (Increment)

**INC**            dst

**Operation:**    dst ← dst + 1  
 The contents of the destination operand are incremented by one.

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

**Format:**

	Bytes	Cycles	Opcode (Hex)	Addr Mode
<div style="border: 1px solid black; padding: 5px; display: inline-block;">dst   opc</div>	1	4	rE r = 0 to F	r
<div style="display: inline-block; border: 1px solid black; padding: 5px;">opc</div> <div style="display: inline-block; border: 1px solid black; padding: 5px; margin-left: 20px;">dst</div>	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.



### 6.5.31 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
```

**6.5.32 IRET (Interrupt Return)**

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

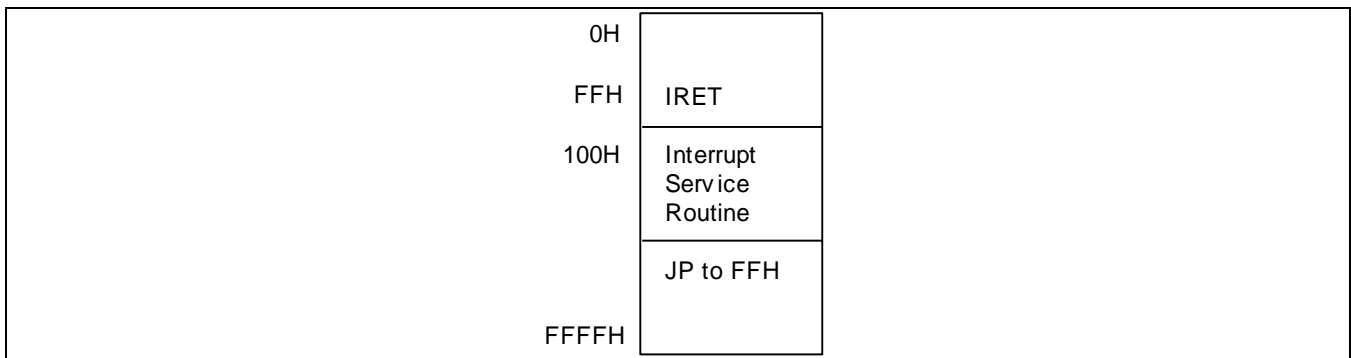
This instruction is used at the end of an interrupt service routine. It restores the flag register and the program counter. It also reenables global interrupts. A "normal IRET" is executed only if the fast interrupt status bit (FIS, bit one of the FLAGS register, 0D5H) is cleared (= "0"). If a fast interrupt 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) 12 (internal stack)	BF
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 preceded by a clearing of the interrupt status (as with a reset of the IPR register).

### 6.5.33 JP (Jump)

**JP** cc, dst (Conditional)

**JP** dst (Unconditional)

**Operation:** If cc is true, PC ← dst  
 The conditional JUMP instruction transfers program control to the destination address if the condition specified by 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)

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

**NOTE:**

1. The 3byte format is used for a conditional jump and the 2byte 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.

### 6.5.34 JR (Jump Relative)

**JR**            cc, dst

**Operation:**    If cc is true,  $PC \leftarrow PC + dst$   
 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:**

(NOTE)	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
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.

**6.5.35 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:**

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

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

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

### 6.5.36 LDB (Load Bit)

**LDB** dst, src.b

**LDB** dst.b, src

**Operation:** dst (0) ← src (b)  
 or

dst (b) ← src (0)

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

**Flags:** No flags are affected.

**Format:**

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	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.

**6.5.37 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 "lrr" or "rr" values an even number for program memory and odd an odd number for data memory.

**Flags:** No flags are affected.

**Format:**

				Bytes	Cycles	Opcode (Hex)	Addr Mode	dst	src
1.	opc	dst   src		2	10	C3	r		lrr
2.	opc	src   dst		2	10	D3	lrr		r
3.	opc	dst   src	XS	3	12	E7	r		XS [rr]
4.	opc	src   dst	XS	3	12	F7	XS [rr]		r
5.	opc	dst   src	XL <sub>L</sub>	XL <sub>H</sub>	4	14	A7	r	XL [rr]
6.	opc	src   dst	XL <sub>L</sub>	XL <sub>H</sub>	4	14	B7	XL [rr]	r
7.	opc	dst   0000	DA <sub>L</sub>	DA <sub>H</sub>	4	14	A7	r	DA
8.	opc	src   0000	DA <sub>L</sub>	DA <sub>H</sub>	4	14	B7	DA	r
9.	opc	dst   0001	DA <sub>L</sub>	DA <sub>H</sub>	4	14	A7	r	DA
10.	opc	src   0001	DA <sub>L</sub>	DA <sub>H</sub>	4	14	B7	DA	r

**NOTE:**

1. The source (src) or working register pair [rr] for formats 5 and 6 cannot use register pair 0–1.
2. For formats 3 and 4, the destination 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.

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

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

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



**6.5.38 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 "lrr" an even number for program memory and an odd number for data memory.

**Flags:** No flags are affected.

**Format:**

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

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

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

**6.5.39 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 "lrr" even for program memory and odd for data memory.

**Flags:**            No flags are affected.

**Format:**

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

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

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

### 6.5.40 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 "lrr" an even number for program memory and an odd number for external data memory.

**Flags:**            No flags are affected.

**Format:**

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

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

LDCPD @RR6, R0    ; (RR6 ← RR6 – 1)  
                          ; 77H (contents of R0) is loaded into program memory location

                         ; 2FFFH (3000H to 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 to 1H)  
                          ; R0 = 77H, R6 = 2FH, R7 = 0FFH

### 6.5.41 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 "lrr" an even number for program memory and an odd number for data memory.

**Flags:** No flags are affected.

**Format:**

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

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

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

### 6.5.42 LDW (Load Word)

**LDW** dst, src

**Operation:** dst ← src  
 The contents of the source (a word) are loaded into the destination. The contents of the source are unaffected.

**Flags:** No flags are affected.

**Format:**

			Bytes	Cycles	Opcode (Hex)	Addr Mode	dst	src			
<table border="1" style="display: inline-table;"> <tr> <td style="width: 30px; text-align: center;">opc</td> <td style="width: 30px; text-align: center;">src</td> <td style="width: 30px; text-align: center;">dst</td> </tr> </table>			opc	src	dst	3	8	C4	RR	RR	
opc	src	dst									
				8	C5	RR	IR				
<table border="1" style="display: inline-table;"> <tr> <td style="width: 30px; text-align: center;">opc</td> <td style="width: 30px; text-align: center;">dst</td> <td style="width: 30px; text-align: center;">src</td> </tr> </table>			opc	dst	src	4	8	C6	RR	IML	
opc	dst	src									

**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.

### 6.5.43 MULT (Multiply-Unsigned)

**MULT**        dst, src

**Operation:**     $dst \leftarrow dst \times src$   
 The 8-bit destination operand (even register of the register pair) is multiplied by the source operand (8 bits) and the product (16 bits) is stored in the register pair specified by the destination address. Both operands are treated as unsigned integers.

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

**Format:**

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

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

**6.5.44 NEXT (Next)**

**NEXT**

**Operation:** PC ← @ IP  
 IP ← IP + 2

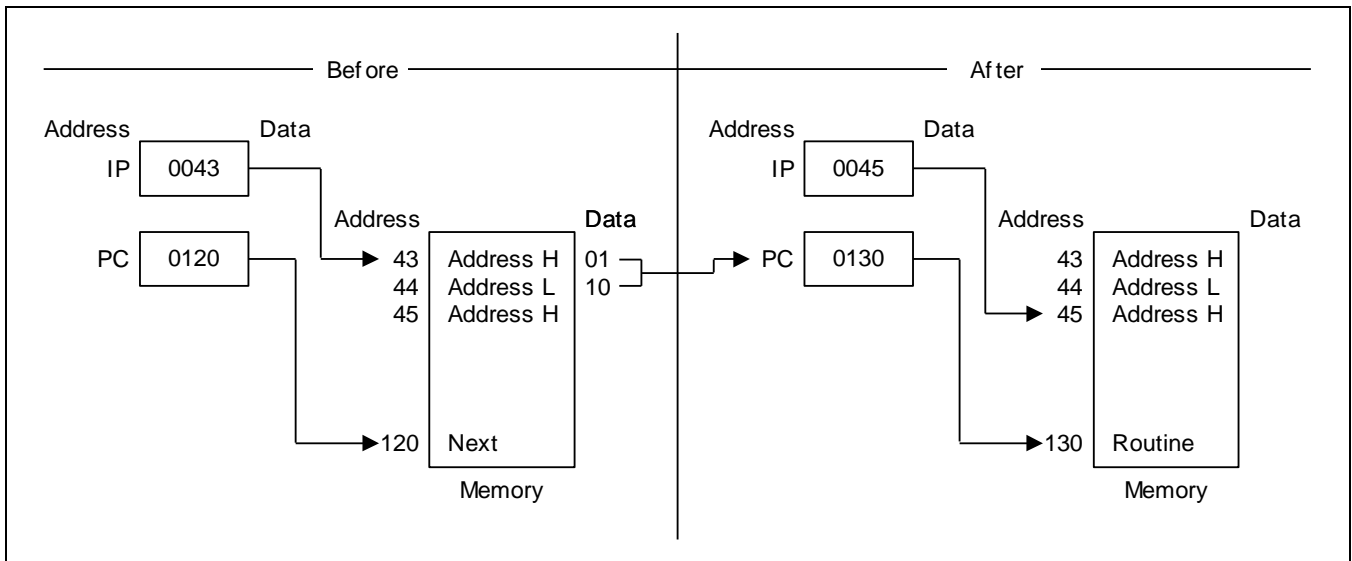
The NEXT instruction is useful when implementing threaded-code languages. The program memory word that is pointed to by the instruction pointer is loaded into the program counter. The instruction pointer is then incremented by two.

**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.



### 6.5.45 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)	
<table border="1"><tr><td>opc</td></tr></table>	opc	1	4	FF
opc				

**Example:** When the instruction

NOP

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



### 6.5.46 OR (Logical OR)

**OR**            dst,src

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

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

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	dst   src	2	4	42	r	r
			6	43	r	lr
opc	src    dst	3	6	44	R	R
			6	45	R	IR
opc	dst    src	3	6	46	R	IM

**Examples:**    Given: R0 = 15H, R1 = 2AH, R2 = 01H, Register 00H = 08H, Register 01H = 37H, and Register 08H = 8AH:

```
OR   R0, R1      →   R0 = 3FH, R1 = 2AH
OR   R0, @R2     →   R0 = 37H, R2 = 01H, Register 01H = 37H
OR   00H, 01H   →   Register 00H = 3FH, Register 01H = 37H
OR   01H, @00H  →   Register 00H = 08H, Register 01H = 0BFH
OR   00H, #02H  →   Register 00H = 0AH
```

In the first example, if working register R0 contains the value 15H and register R1 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.

### 6.5.47 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		
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	dst		2	8	50	R
	opc	dst					
			8	51	IR		

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

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

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

**6.5.48 POPUD (Pop User Stack-Decrementing)**

**POPUD** dst, src

**Operation:** dst ← src  
 IR ← IR – 1

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

**Flags:** No flags are affected.

**Format:**

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

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

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

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

### 6.5.49 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	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.



### 6.5.51 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	src
opc	dst	src	3	8	82	IR	R

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

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

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

### 6.5.52 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	src
opc	dst	src	3	8	83	IR	R

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

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

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

### 6.5.53 RCF (Reset Carry Flag)

**RCF**            RCF

**Operation:**     $C \leftarrow 0$   
 The carry flag is cleared to logic zero, regardless of its previous value.

**Flags:**        **C:** Cleared to "0".  
 No other flags are affected.

**Format:**

	Bytes	Cycles	Opcode (Hex)
opc	1	4	CF

**Example:**     Given:  $C = "1"$  or  $"0"$ :  
 The instruction RCF clears the carry flag (C) to logic zero.



### 6.5.54 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) 10 (internal stack)	AF

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

RET → PC = 101AH, SP = 00FEH

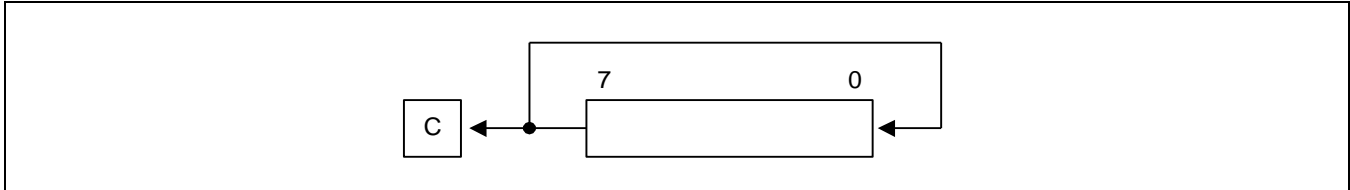
The statement "RET" pops the contents of stack pointer location 00FCH (10H) into the high byte of the program counter. The stack pointer then pops the value in location 00FEH (1AH) into the PC's low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location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

**6.5.55 RL (Rotate Left)**

**RL**            dst

**Operation:**

$C \leftarrow \text{dst}(7)$   
 $\text{dst}(0) \leftarrow \text{dst}(7)$   
 $\text{dst}(n + 1) \leftarrow \text{dst}(n), n = 0-6$   
 The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit zero (LSB) position and also replaces the carry flag.



**Flags:**

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

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst	2	4	90	R
			4	91	IR

**Examples:**

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

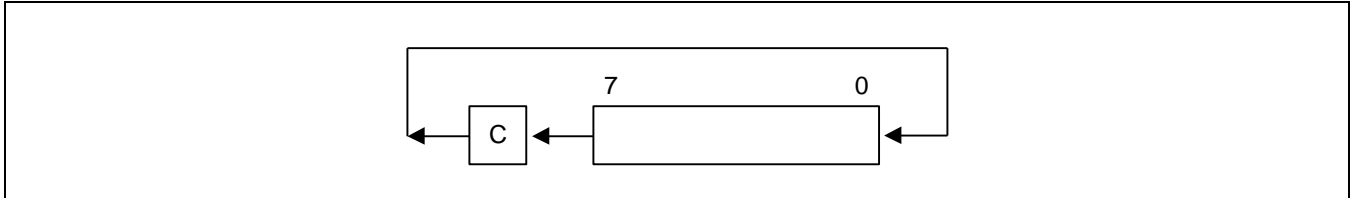
RL 00H → Register 00H = 55H, C = "1"  
 RL @01H → Register 01H = 02H, Register 02H = 2EH, C = "0"

In the first example, if general register 00H contains the value 0AAH (10101010B), the statement "RL 00H" rotates the 0AAH value left one bit position, leaving the new value 55H (01010101B) and setting the carry and overflow flags.

**6.5.56 RLC (Rotate Left Through Carry)**

**RLC**            dst

**Operation:**    dst (0) ← C  
                   C ← dst (7)  
                   dst (n + 1) ← dst (n), n = 0 to 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 (7-bit) was "1".
- Z:** Set if the result is "0"; cleared otherwise.
- S:** Set if the result 7-bit is set; cleared otherwise.
- V:** Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
- D:** Unaffected.
- H:** Unaffected.

**Format:**

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

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

RLC 00H → Register 00H = 54H, C = "1"  
 RLC @01H → Register 01H = 02H, Register 02H = 2EH, C = "0"

In the first example, if general register 00H has the value 0AAH (10101010B), the statement "RLC 00H" rotates 0AAH one bit position to the left. The initial value of 7-bit 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.

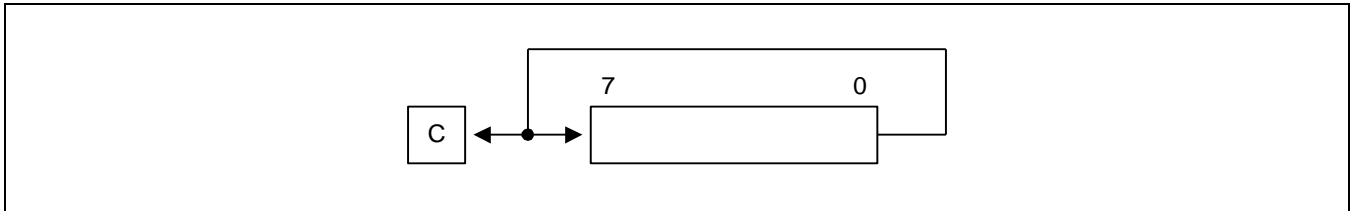
**6.5.57 RR (Rotate Right)**

**RR**            dst

**Operation:**

$C \leftarrow \text{dst}(0)$   
 $\text{dst}(7) \leftarrow \text{dst}(0)$   
 $\text{dst}(n) \leftarrow \text{dst}(n + 1), n = 0 \text{ to } 6$

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
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"

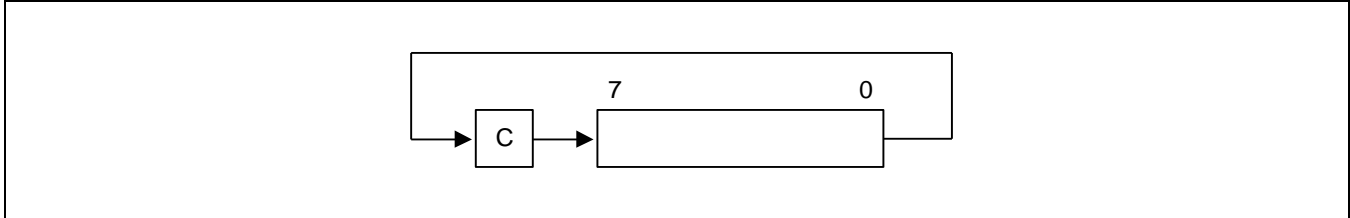
In the first example, if general register 00H contains the value 31H (00110001B), the statement "RR 00H" rotates this value one bit position to the right. The initial value of bit zero is moved to bit 7, leaving the new value 98H (10011000B) in the destination register. The initial bit zero also resets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

**6.5.58 RRC (Rotate Right Through Carry)**

**RRC**            dst

**Operation:**

dst (7) ← C  
 C ← dst (0)  
 dst (n) ← dst (n + 1), n = 0 to 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
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".

### 6.5.59 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.

### 6.5.60 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.

### 6.5.61 SBC (Subtract with Carry)

**SBC** dst, src

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

**Flags:**

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

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode	dst	src			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst   src</td> </tr> </table>	opc	dst   src		2	4	32		r	r	
	opc	dst   src								
			6	33		r	lr			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst		3	6	34		R	R
	opc	src	dst							
			6	35		R	IR			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src		3	6	36		R	IM
opc	dst	src								

**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.



### 6.5.62 SCF (Set Carry Flag)

#### SCF

**Operation:**  $C \leftarrow 1$   
The carry flag (C) is set to logic one, regardless of its previous value.

**Flags:** **C:** Set to "1".  
No other flags are affected.

**Format:**

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

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

6.5.63 SRA (Shift Right Arithmetic)

**SRA** dst

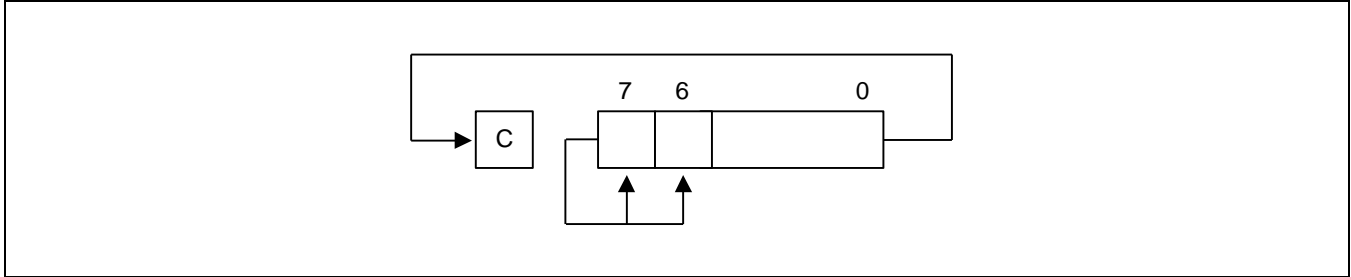
**Operation:**

dst (7) ← dst (7)

C ← dst (0)

dst (n) ← dst (n + 1), n = 0 to 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
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.

### 6.5.64 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 to 7 of the selected register pointer are written unless both register pointers are selected. RP0.3 is then cleared to logic zero and RP1.3 is set to logic one.

**Flags:** No flags are affected.

**Format:**

	Bytes	Cycles	Opcode (Hex)	Addr Mode 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.

### 6.5.65 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 adequate time interval for the clock to stabilize before the next instruction is executed. If three or more NOP instructions are not used after STOP instruction, leakage current could be flown because of the floating state in the internal bus.

**Flags:** No flags are affected.

**Format:**

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

**Example:** The statement

STOP ; Halts all microcontroller operations

NOP  
 NOP  
 NOP

### 6.5.66 SUB (Subtract)

**SUB** dst, src

**Operation:** dst ← dst – src

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

**Flags:**

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

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode	dst	src			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst   src</td> </tr> </table>	opc	dst   src		2	4	22		r	r	
	opc	dst   src								
			6	23		r	lr			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst		3	6	24		R	R
	opc	src	dst							
			6	25		R	IR			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src		3	6	26		R	IM
opc	dst	src								

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

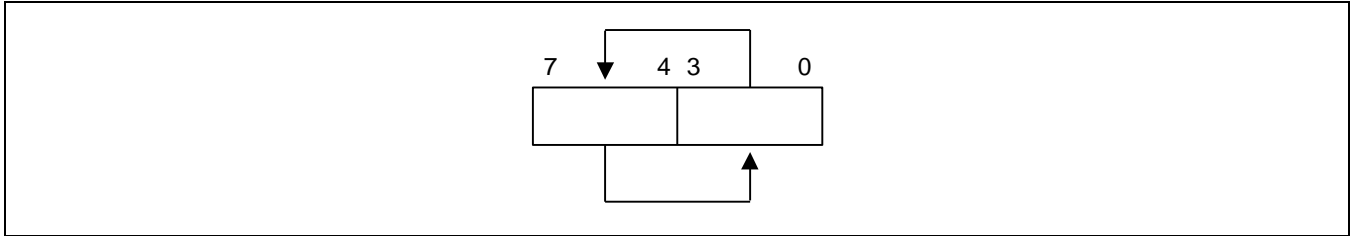
- SUB R1, R2 →□ R1 = 0FH, R2 = 03H
- SUB R1, @R2 →□ R1 = 08H, R2 = 03H
- SUB 01H, 02H →□ Register 01H = 1EH, Register 02H = 03H
- SUB 01H, @02H →□ Register 01H = 17H, Register 02H = 03H
- SUB 01H, #90H →□ Register 01H = 91H; C, S, and V = "1"
- SUB 01H, #65H →□ Register 01H = 0BCH; C and S = "1", V = "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.

**6.5.67 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
opc	dst	2	4	F0	R
			4	F1	IR

**Examples:** Given: Register 00H = 3EH, Register 02H = 03H, and Register 03H = 0A4H:

SWAP 00H □→ Register 00H = 0E3H  
 SWAP @02H □→ Register 02H = 03H, Register 03H = 4AH

In the first example, if general register 00H contains the value 3EH (00111110B), the statement "SWAP 00H" swaps the lower and upper four bits (nibbles) in the 00H register, leaving the value 0E3H (11100011B).

### 6.5.68 TCM (Test Complement under Mask)

**TCM** dst, src

**Operation:** (NOT dst) AND src

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

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

**Format:**

			Bytes	Cycles	Opcode (Hex)	Addr Mode	dst	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.

**6.5.69 TM (Test under Mask)**

**TM** dst, src

**Operation:** dst AND src

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

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

**Format:**

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



**6.5.70 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.

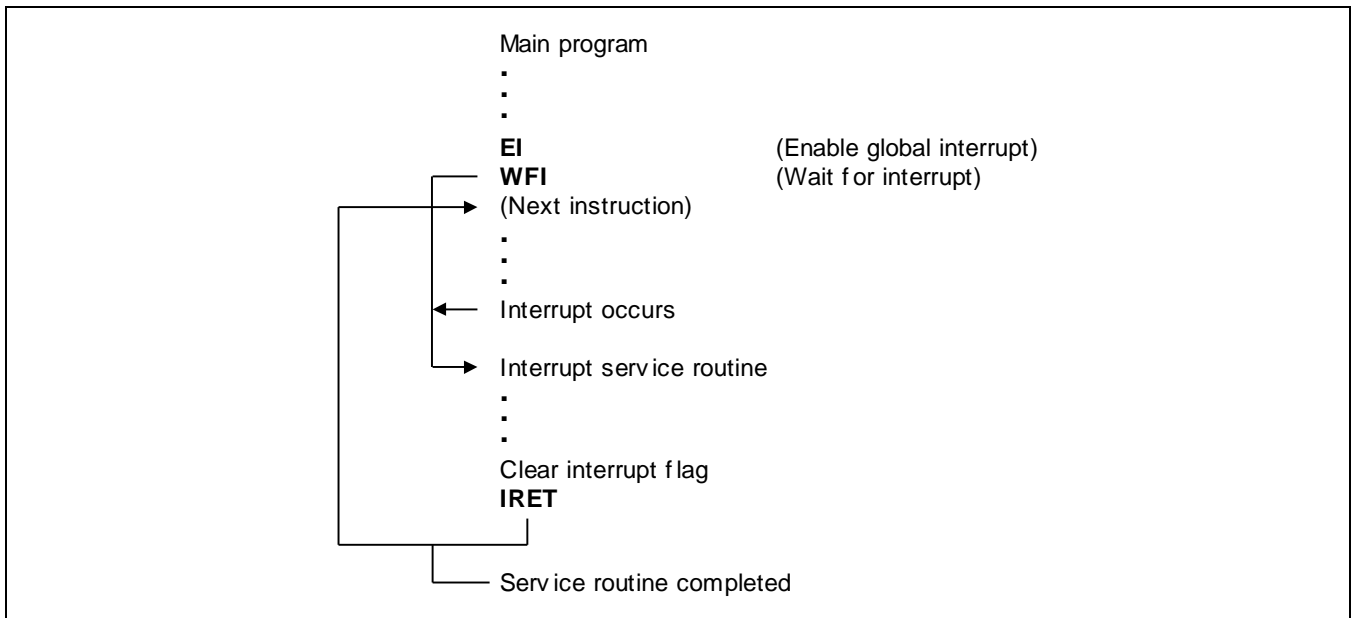
**Flags:** No flags are affected.

**Format:**

	<b>Bytes</b>	<b>Cycles</b>	<b>Opcode (Hex)</b>
opc	1	4n	3F

**NOTE:** n = 1, 2, 3, ...

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



**6.5.71 XOR (Logical Exclusive OR)**

**XOR** dst, src

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

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

**Format:**

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src			
<table border="1"> <tr> <td>opc</td> <td>dst   src</td> </tr> </table>	opc	dst   src		2	4	B2	r	r	
	opc	dst   src							
			6	B3	r	lr			
<table border="1"> <tr> <td>opc</td> <td>src</td> <td>dst</td> </tr> </table>	opc	src	dst		3	6	B4	R	R
	opc	src	dst						
			6	B5	R	IR			
<table border="1"> <tr> <td>opc</td> <td>dst</td> <td>src</td> </tr> </table>	opc	dst	src		3	6	B6	R	IM
opc	dst	src							

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

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

In the first example, if working register R0 contains the value 0C7H and 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

## 7.1 Overview

By Smart Option (3FH.3–.0 in ROM), user can select internal RC oscillator or external oscillator. In using internal oscillator, XIN (P1.0), XOUT (P1.1) can be used by normal I/O pins. An internal RC oscillator source provides a typical 8MHz, 4MHz, 3.2MHz, 2MHz, 1MHz or 0.5MHz (in  $V_{DD} = 5V$ ) depending on Smart Option.

An external RC oscillation source provides a typical 4MHz clock for S3F8S28/S3F8S24. An internal capacitor supports the RC oscillator circuit. A low gain external crystal or ceramic oscillation source provides a maximum 1 MHz clock with low system power consumption. A high gain external crystal or ceramic oscillation source provides a maximum 12MHz clock, these two different crystal/ceramic oscillation is selected by Smart Option (3F.3–3F.0). The XIN and XOUT pins connect the oscillation source to the on-chip clock circuit. Simplified external RC oscillator and crystal/ceramic oscillator circuits are shown in [Figure 7-1](#) and [Figure 7-2](#). When you use external oscillator, P1.0, P1.1 must be set to output port to prevent current consumption

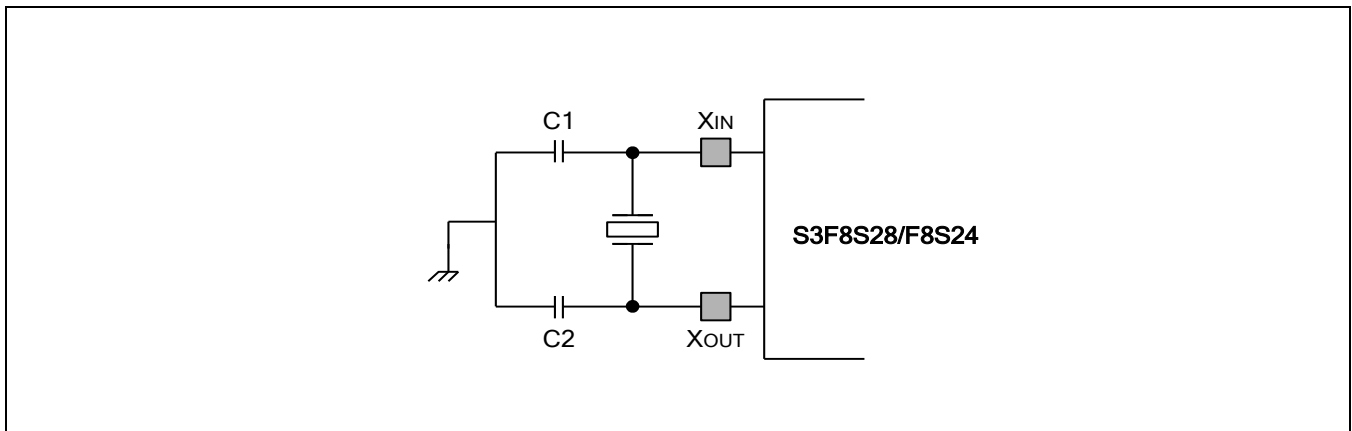


Figure 7-1 Main Oscillator Circuit (Crystal/Ceramic Oscillator)

## 7.2 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.

## 7.3 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, by Watchdog Timer interrupt by an external interrupt with RC-delay noise filter (for S3F8S28/S3F8S24, INT0 to INT7).
- 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).

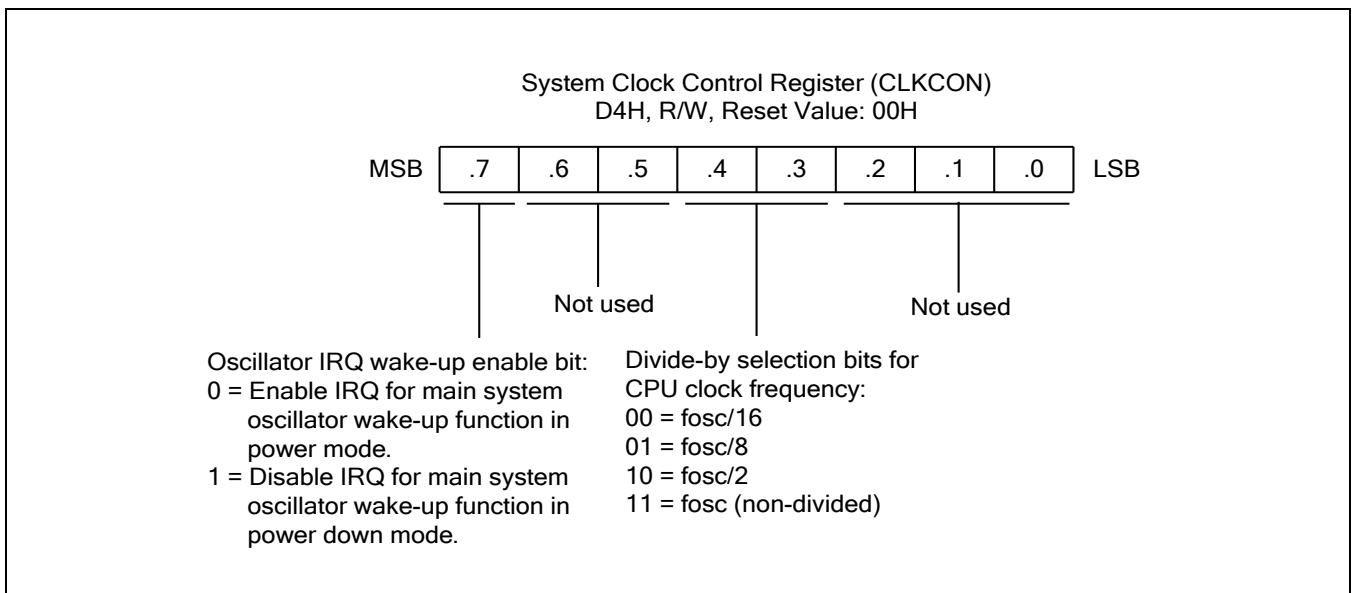
## 7.4 System Clock Control Register (CLKCON)

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 7-2 System Clock Control Register (CLKCON)**

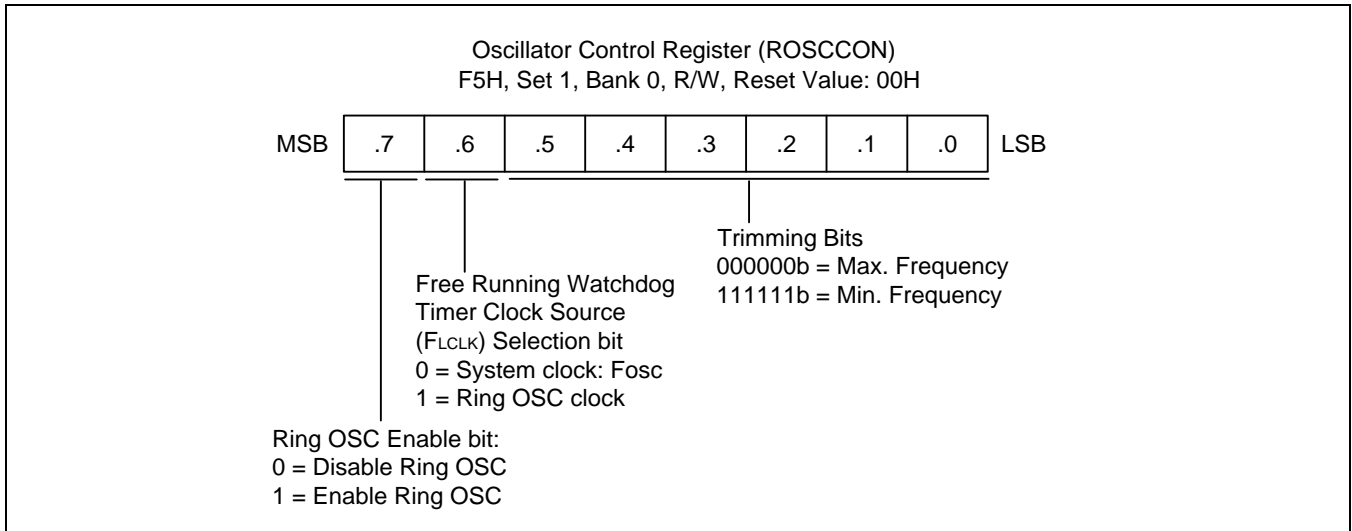
## 7.5 Ring Oscillator Control Register (ROSCCON)

S3F8S28/S3F8S24 has an internal 32K (typ.) Ring oscillator for Watchdog Timer, that can be enabled and run in Stop Mode, it is useful for system wakeup in Stop Mode within setting period.

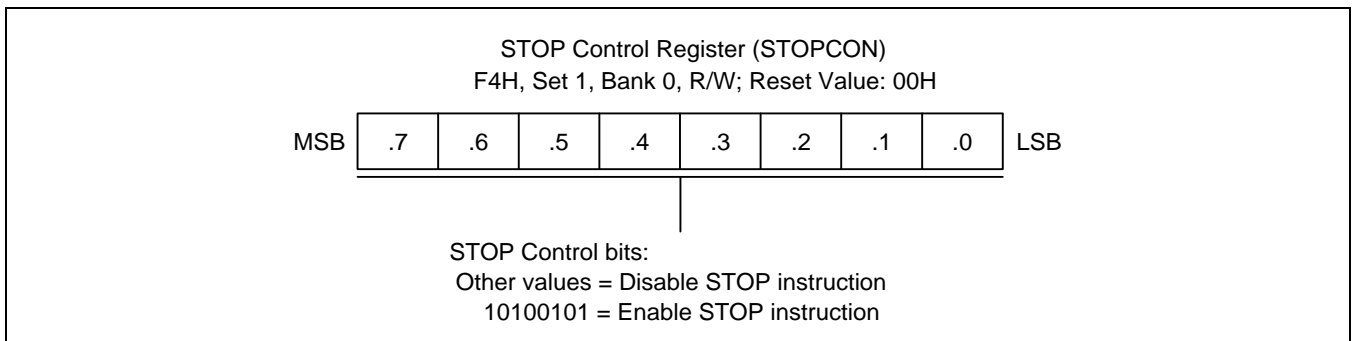
The frequency distribution of the Ring oscillator is very large, so the trimming bits (ROSCCON.5–.0) are provided to adjust the frequency; the reset value of ROSCCON.5-.0 is "000000b", which set the max frequency, so user need to adjust the frequency by setting the trimming bits.

The Ring oscillator control register, ROSCCON, is located in location F5H, Set 1, Bank 0. It is read/write addressable and has the following functions:

- Ring OSC Enable/Disable (ROSCCON.7)
- Free running Watchdog Timer clock source select (ROSCCON.6)
- Ring OSC frequency trimming bits (ROSCCON.5–.0)



**Figure 7-3 Ring Oscillator Control Register (ROSCCON)**



**Figure 7-4 Stop Control Register (STOPCON)**

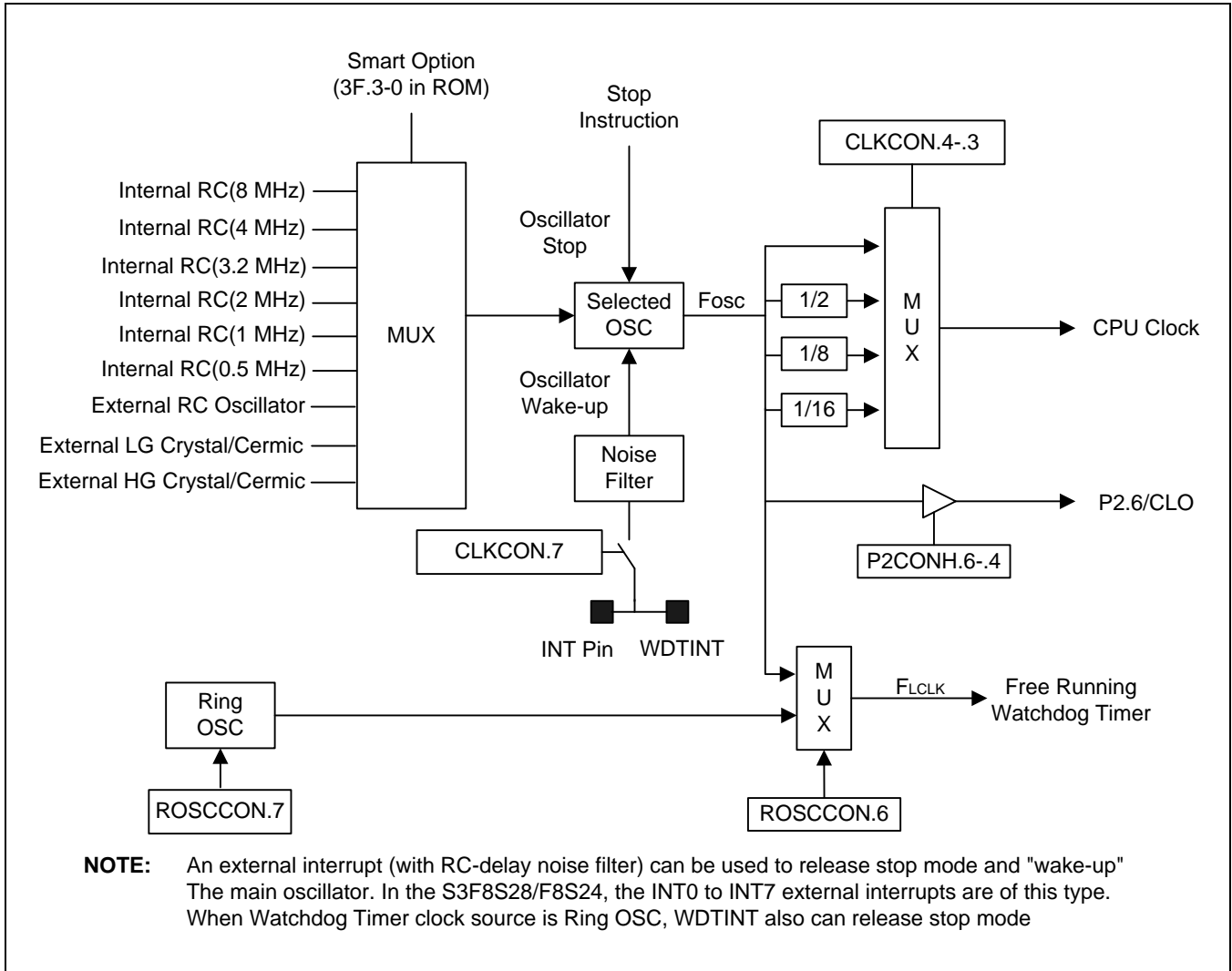


Figure 7-5 System Clock Circuit Diagram

# 8

## RESET and Power-Down

### 8.1 System Reset

#### 8.1.1 Overview

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

The S3F8S28/S3F8S24 can be RESET in five ways:

- By external power-on-reset
- By the external nRESET input pin pulled low
- By the digital Basic Timer overflow
- By the digital free-running watchdog peripheral timing out
- By Low Voltage Reset (LVR)

During a 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 input through a Schmitt trigger circuit where it is then synchronized with the CPU clock. This brings the S3F8S28/S3F8S24 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.

After the nRESET pin is released, the S3F8S28/S3F8S24 MCU will enter an idle state for a minimum time interval to allow time for internal CPU clock oscillation to stabilize. The minimum required oscillation stabilization time for a reset is approximately 52.4ms (@  $2^{19}/f_{OSC}$ ,  $f_{OSC} = 10\text{MHz}$ ).

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](#) to [Table 8-3](#)).

The Basic Timer provides a watchdog function in order to ensure graceful recovery from software malfunction in RUN & IDLE modes. If Basic Timer counter is not refreshed before an end-of-counter condition (overflow) is reached, the internal reset will be activated. The free running Watchdog Timer also can be used generate Reset to ensure system recovery, it's clock source can be set to free running Ring Oscillator, so it can reset chip in Stop Mode.

The on-chip Low Voltage Reset, features static Reset when supply voltage is below a reference value (Typ. 1.9, 2.3, 3.0, 3.9V). 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 a 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 when LVR enable in Smart Option.



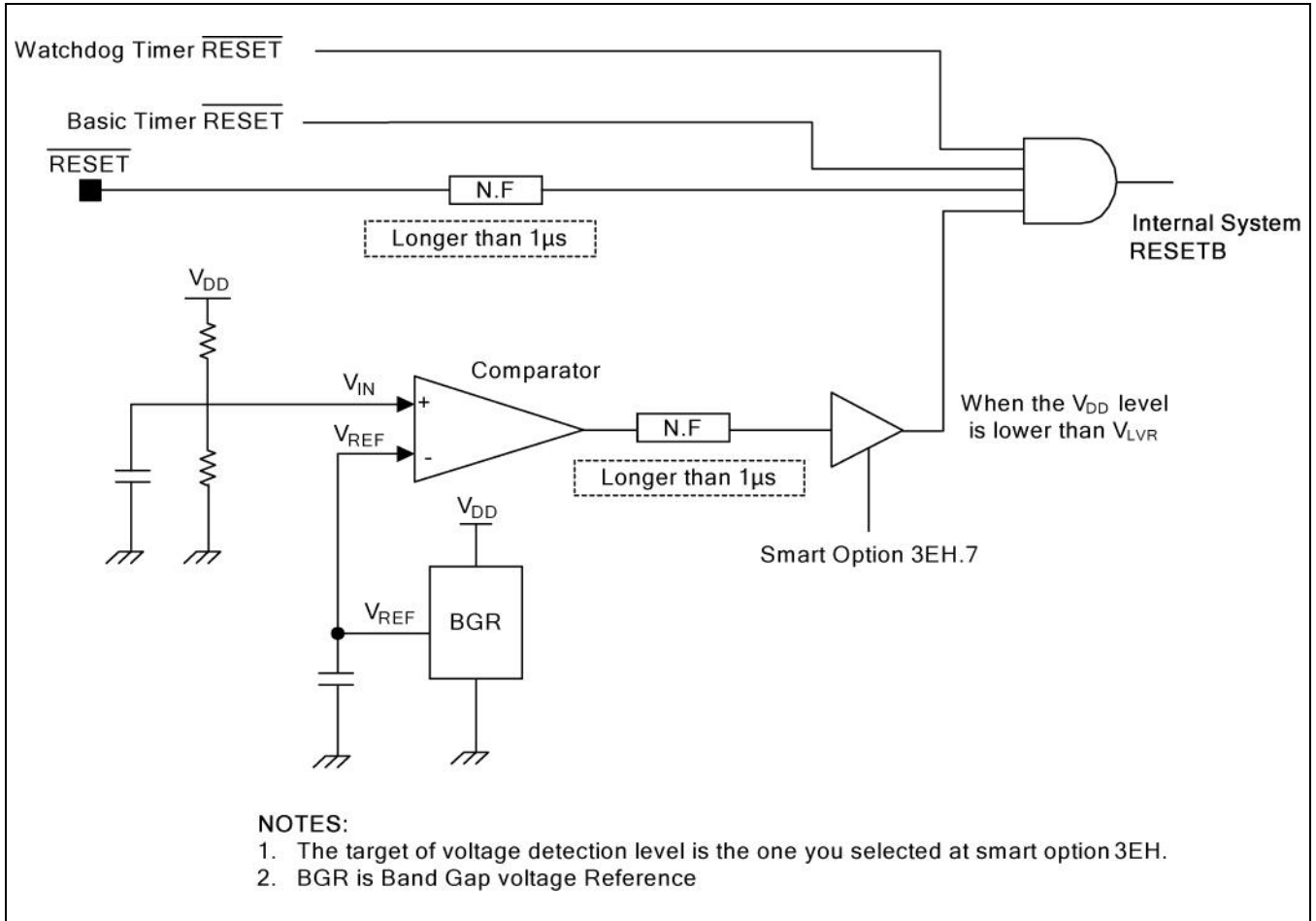


Figure 8-1 Low Voltage Reset Circuit

**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.

### 8.1.2 External RESET Pin

When the nRESET pin transiting from  $V_{IL}$  (low input level of reset pin) to  $V_{IH}$  (high input level of reset pin), the reset pulse is generated.

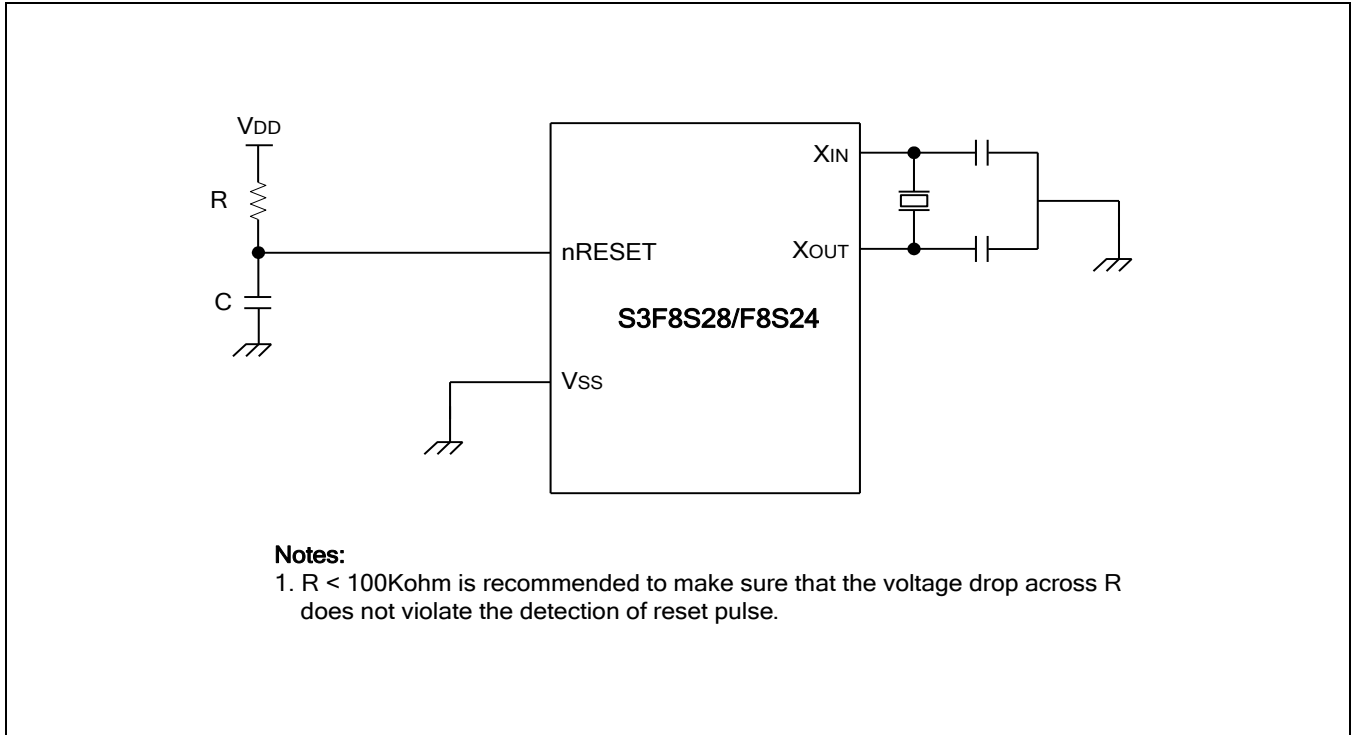


Figure 8-2 Recommended External Reset Circuit

### 8.1.3 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 to 2 are set to input mode
- Peripheral control and data registers are disabled and reset to their initial values (see [Table 8-1](#) to [Table 8-3](#)).
- 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.

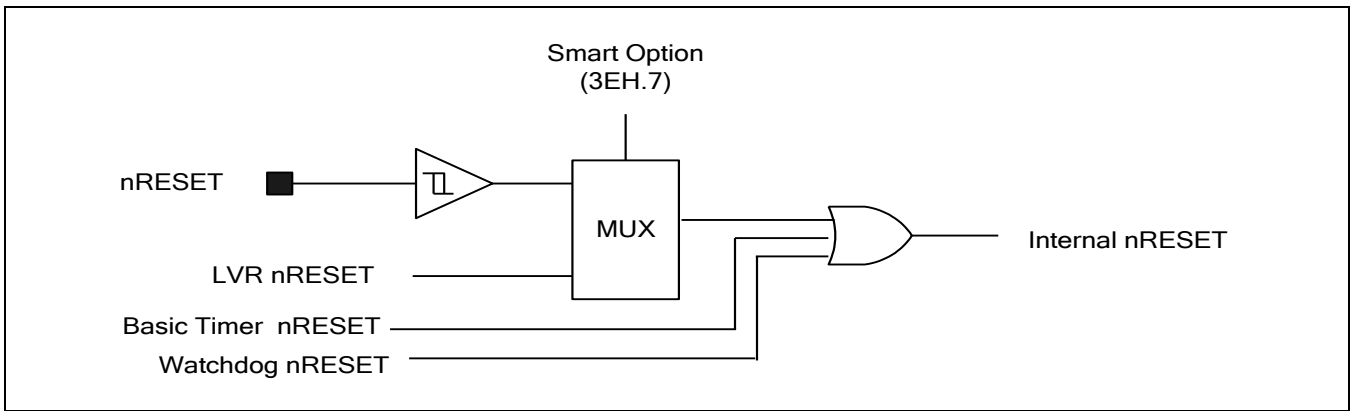


Figure 8-3 Reset Block Diagram

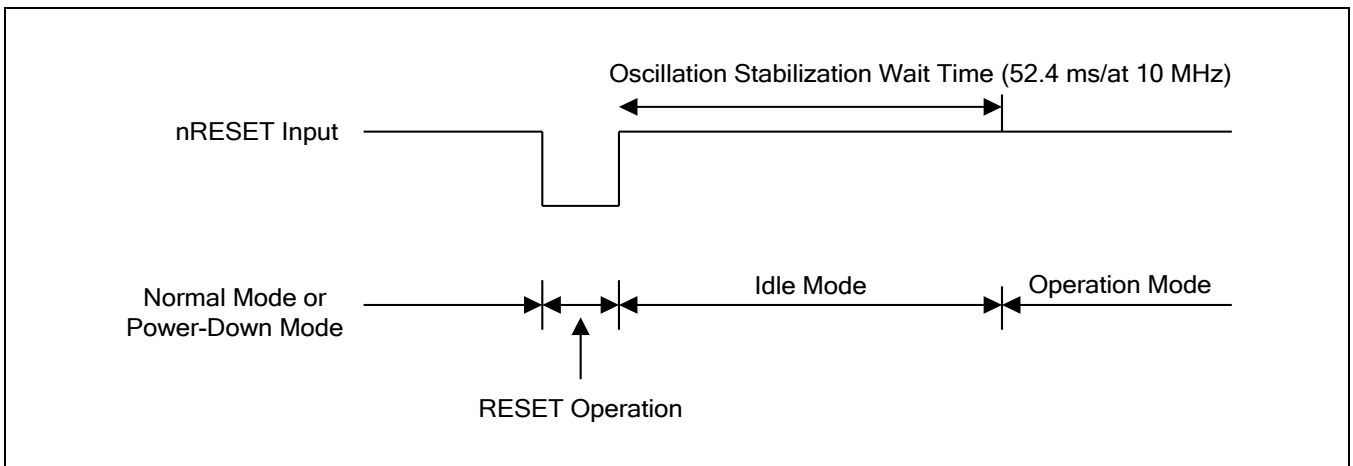


Figure 8-4 Timing for S3F8S28/S3F8S24 After Reset

## 8.2 Power-Down Modes

### 8.2.1 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 4 $\mu$ A 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 three ways: by a nRESET signal, by an external interrupt or by Watchdog Timer interrupt.

**NOTE:** Before execute the STOP instruction, must set the STPCON register as "10100101b".

### 8.2.2 Sources to Release Stop Mode

Stop mode is released when following sources go active:

- System Reset by external reset pin (nRESET)
- External Interrupt (INT0 to INT7)
- Watchdog Timer Interrupt (WDTINT)

#### 8.2.2.1 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.

#### 8.2.2.2 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 to INT7 in the S3F8S28/S3F8S24 interrupt structure meet this criterion.

### 8.2.2.3 Using Watchdog Timer Interrupt to Release Stop Mode

Watchdog timer overflow interrupt can be used to release stop mode: **WDTINT**

Please note the following conditions for Stop mode release:

- If you release Stop mode using an external interrupt or Watchdog Timer interrupt, the current values in system and peripheral control registers are unchanged.
- If you use an external interrupt or Watchdog Timer interrupt for Stop mode release, you can also program the duration of the oscillation stabilization interval. To do this, you must make the appropriate control and clock settings before entering Stop mode.
- When the Stop mode is released by external interrupt or Watchdog Timer interrupt, the CLKCON.4 and CLKCON.3 bit-pair setting remains unchanged and the currently selected clock value is used. The CLKCON.1 and CLKCON.0 bits are also remain previous values for Ring OSC setting and Watchdog Timer clock source selection.
- The external interrupt or Watchdog Timer 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.

### 8.2.2.4 How to Enter into Stop Mode

There are two steps to enter into Stop mode:

- Handling STOPCON register to appropriate value (10100101B).
- Writing Stop instruction (keep the order).
- Waiting several clocks (insert several "NOP" instructions)

### 8.2.3 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.

#### NOTE:

1. External interrupts that are not clock-related and interrupts of free running Watchdog Timer 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.

### 8.3 Hardware Reset Values

[Table 8-1](#) to [Table 8-3](#) 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, Set1**

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

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

**Table 8-2 Register Values After a Reset, Set1, Bank0**

Register Name	Mnemonic	Address & Location		Bit Values After RESET								
		Address	RW	7	6	5	4	3	2	1	0	
Port 0 data register	P0	E0H	RW	0	0	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	RW	-	-	-	-	-	0	0	0	0
Port 2 data register	P2	E2H	RW	-	0	0	0	0	0	0	0	0
Port 3 data register	P3	E3H	RW	-	-	-	-	0	0	0	0	0
Port 2 pull-up resistor enable register	P2PUR	E4H	RW	-	0	0	0	0	0	0	0	0
Port 0 pull-up resistor enable register	P0PUR	E5H	RW	0	0	0	0	0	0	0	0	0
Port 0 control register (High Byte)	P0CONH	E6H	RW	0	0	0	0	0	0	0	0	0
Port 0 control register (Low Byte)	P0CONL	E7H	RW	0	0	0	0	0	0	0	0	0
Port 0 interrupt pending register	P0PND	E8H	RW	-	-	-	-	0	0	0	0	0
Port 1 control register	P1CON	E9H	RW	0	0	-	-	0	0	0	0	0
Port 2 control register (High Byte)	P2CONH	EAH	RW	-	0	0	0	0	0	0	0	0
Port 2 control register (Low Byte)	P2CONL	EBH	RW	0	0	0	0	0	0	0	0	0
Timer B counter register	TBCNT	ECH	R	0	0	0	0	0	0	0	0	0
Timer B data register	TBDATA	EDH	RW	1	1	1	1	1	1	1	1	1
Timer B control register	TBCON	EEH	RW	-	-	0	0	0	0	0	0	0
Port 3 interrupt pending register	P3PND	EFH	RW	0	0	0	0	0	0	0	0	0
Port 3 control register	P3CON	F0H	RW	0	0	0	0	0	0	0	0	0
PWM0 extension data register	PWM0EX	F1H	RW	0	0	0	0	0	0	0	0	0
PWM0 data register	PWM0DATA	F2H	RW	0	0	0	0	0	0	0	0	0
PWM0 control register	PWM0CON	F3H	RW	0	0	-	0	0	0	0	0	0
PWM extension data register	PWMEX	F1H	RW	0	0	0	0	0	0	0	0	0
PWM data register	PWMDATA	F2H	RW	0	0	0	0	0	0	0	0	0
PWM control register	PWMCON	F3H	RW	0	0	-	0	0	0	0	0	0
STOP control register	STOPCON	F4H	RW	0	0	0	0	0	0	0	0	0
Ring Oscillator control register	ROSCCON	F5H	RW	0	0	0	0	0	0	0	0	0
Watchdog Timer control register	WDTCON	F6H	RW	0	0	0	0	0	0	0	0	0
A/D control register	ADCON	F7H	RW	1	1	1	1	0	0	0	0	0
A/D converter data register (High)	ADDATAH	F8H	R	x	x	x	x	x	x	x	x	x
A/D converter data register (Low)	ADDATAAL	F9H	R	0	0	0	0	0	0	0	x	x
Locations FAH to FCH are not mapped												
Basic Timer counter	BTCNT	FDH	R	0	0	0	0	0	0	0	0	0
External memory timing register	EMT	FEH	RW	0	0	0	0	0	0	0	0	0
Interrupt priority register	IPR	FFH	RW	x	x	x	x	x	x	x	x	x

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

**Table 8-3 System and Peripheral Control Registers, Set1, Bank1**

Register Name	Mnemonic	Address & Location		RESET Value (Bit)								
		Address	RW	7	6	5	4	3	2	1	0	
Timer 1 Data Register (High Byte)	T1DATAH	E0H	RW	1	1	1	1	1	1	1	1	1
Timer 1 Data Register (Low Byte)	T1DATAL	E1H	RW	1	1	1	1	1	1	1	1	1
Timer 1 Counter Register (High Byte)	T1CNTH	E2H	R	0	0	0	0	0	0	0	0	0
Timer 1 Counter Register (Low Byte)	T1CNTL	E3H	R	0	0	0	0	0	0	0	0	0
Timer 1 Control Register	T1CON	E4H	RW	0	0	0	0	0	0	0	0	0
Timer 1 Prescaler Register	T1PS	E5H	RW	–	–	–	–	0	0	0	0	0
PWM1 extension data register	PWM1EX	E6H	RW	0	0	0	0	0	0	0	0	0
PWM1 data register	PWM1DATA	E7H	RW	0	0	0	0	0	0	0	0	0
PWM1 control register	PWM1CON	E8H	RW	0	0	–	0	0	0	0	0	0
Locations E9H are not mapped												
Reset source indicating register	RESETID	EAH	RW	Refer to the detail description								
Flash memory control register	FMCON	ECH	RW	0	0	0	0	0	–	–	0	0
Flash memory user programming enable register	FMUSR	EDH	RW	0	0	0	0	0	0	0	0	0
Flash memory sector address register (High Byte)	FMSECH	EEH	RW	0	0	0	0	0	0	0	0	0
Flash memory sector address register (Low Byte)	FMSECL	EFH	RW	0	0	0	0	0	0	0	0	0
IIC Control Register	ICCR	F0H	RW	0	0	0	0	1	1	1	1	1
IIC Status Register	ICSR	F1H	RW	0	0	0	0	0	0	0	0	0
IIC Data Shift Register	IDSR	F2H	RW	x	x	x	x	x	x	x	x	x
IIC Address Register	IAR	F3H	RW	x	x	x	x	x	x	x	x	x
Low Voltage Detector Control Register	LVDCON	F4H	RW	0	–	0	–	–	–	0	0	0
UART control register	UARTCON	F5H	RW	0	0	0	0	0	0	0	0	0
UART pending register	UARTPND	F6H	RW	–	–	–	–	–	–	0	0	0
UART Baud rate data register	BRDATA	F7H	RW	1	1	1	1	1	1	1	1	1
UART data register	UDATA	F8H	RW	x	x	x	x	x	x	x	x	x
Location F9H to FFH is not mapped												

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



**Example 8-1 Sample S3F8S28/S3F8S24 Initialization Routine**

```

;-----<< Interrupt Vector Address >>
    ORG    0000H
    VECTOR 0F2H, PWM0OVF_INT    ;
    VECTOR 0F4H, INT_TIMERB    ;
    VECTOR 0F6H, INT_TIMER_A   ;
    VECTOR 0FCH, INT_EXT1      ;
    VECTOR 0FEH, INT_EXT0      ;

;-----<< 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
    DB     0FEH                ; 003FH, External RC oscillator

;-----<< 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    SP, #0C0H         ; Stack pointer must be set

        LD    P0CONH, #10101010B ;
        LD    P0CONL, #10101010B ; P0.0-P0.7 push-pull output
        LD    POPND, #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      ; CPU = 4MHz, interrupt interval = 6.4msec
        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

;-----<< 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

;-----< Timer A interrupt service routine >
INT_TIMER_A:
    •
    •
    AND    TACON,#1111110B  ; Pending bit clear
    IRET   ; Interrupt return

;-----< Timer B interrupt service routine >
INT_TIMER_B:
    •
    •
    AND    TBCON,#1111110B  ; Pending bit clear
    IRET

;-----< PWM overflow interrupt service routine >
PWM0OVF_INT:
    •
    •
    AND    PWM0CON,#1111110B ; Pending bit clear
    IRET   ; Interrupt return

;-----< External interrupt0 service routine >
INT_EXT0:  •
          •

```

```
        AND    POPND,#11111110B    ; EXT0 Pending bit clear
        IRET                               ; Interrupt return

;-----< External interrupt1 service routine >
INT_EXT1:  •
          •
          AND    POPND,#11111011B    ; EXT1 Pending bit clear
          IRET                               ; Interrupt return
          •
          •
          END                                ;
```

# 9 I/O Ports

## 9.1 Overview

The S3F8S28/S3F8S24 has three I/O ports: with 22 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 10mA).

**Table 9-1 S3F8S28/S3F8S24 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 oscillator input/output or reset input by Smart Option.	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,T1 output,T1 capture input)	Bit
3	Bit-programmable I/O port for schmitt trigger input or push-pull output. Pull-up resistors are assignable by software. Port 3 pins can also be used as alternative function. (ADC input, external interrupt input).	Bit

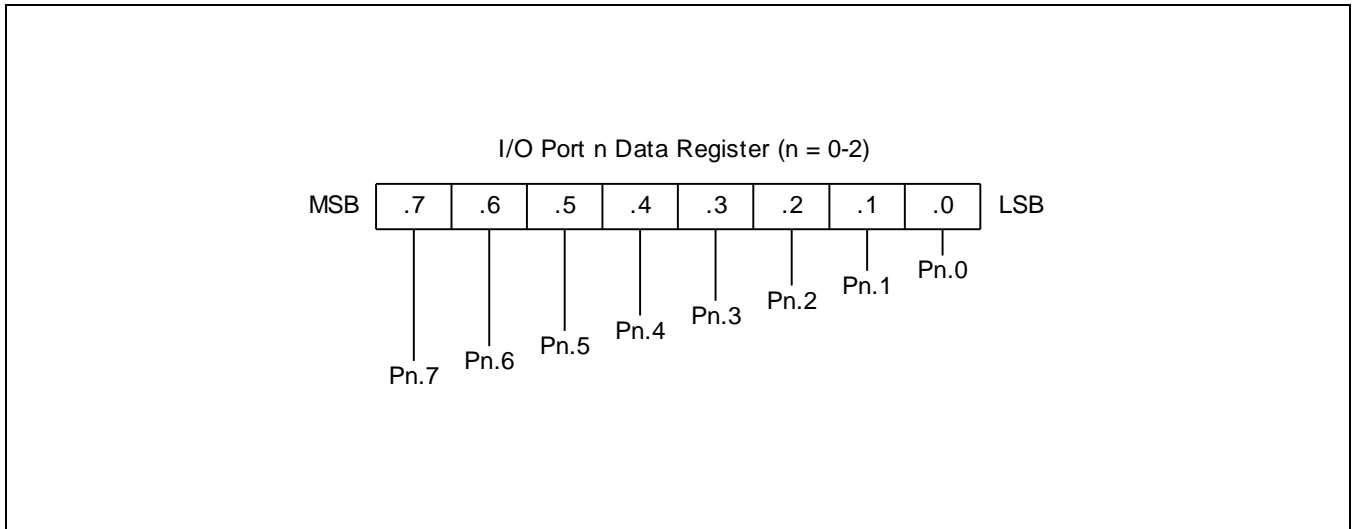
## 9.2 Port Data Registers

[Table 9-2](#) gives you an overview of the port data register names, locations, and addressing characteristics. Data registers for ports 0 to 2 have the structure shown in [Figure 9-1](#).

**Table 9-2 Port Data Register Summary**

Register Name	Mnemonic	Hex	RW
Port 0 data register	P0	E0H	RW
Port 1 data register	P1	E1H	RW
Port 2 data register	P2	E2H	RW
Port 3 data register	P3	E3H	RW

**NOTE:** A reset operation clears the P0 to P2 data register to "00H".



**Figure 9-1 Port Data Register Format**

9.2.1 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 pull-up 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), P0PND (E8H) and P0PUR (E5H).

You access port 0 directly by writing or reading the corresponding port data register, P0 (E0H).

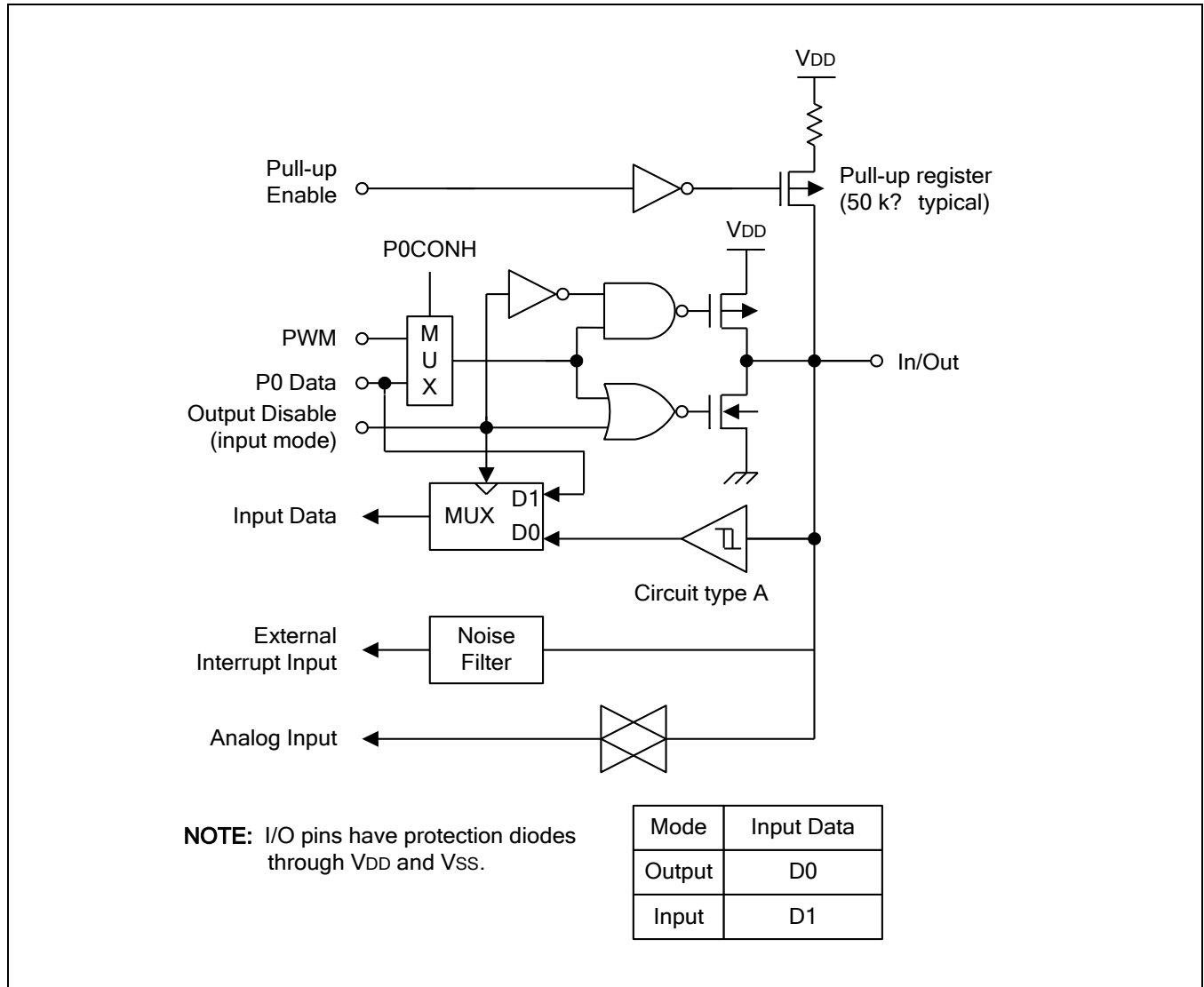


Figure 9-2 Port 0 Circuit Diagram

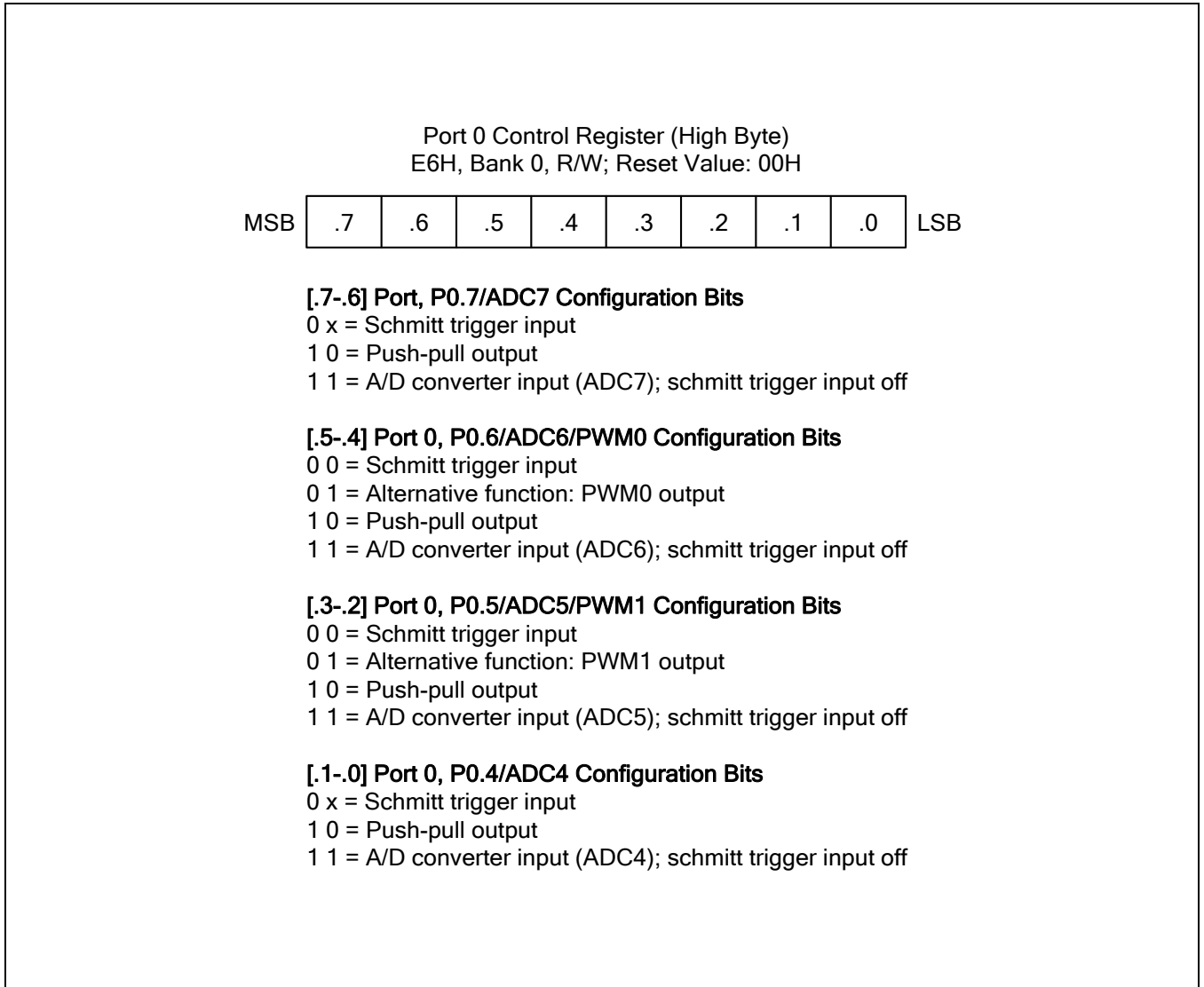


Figure 9-3 Port 0 Control Register (P0CONH, High Byte)

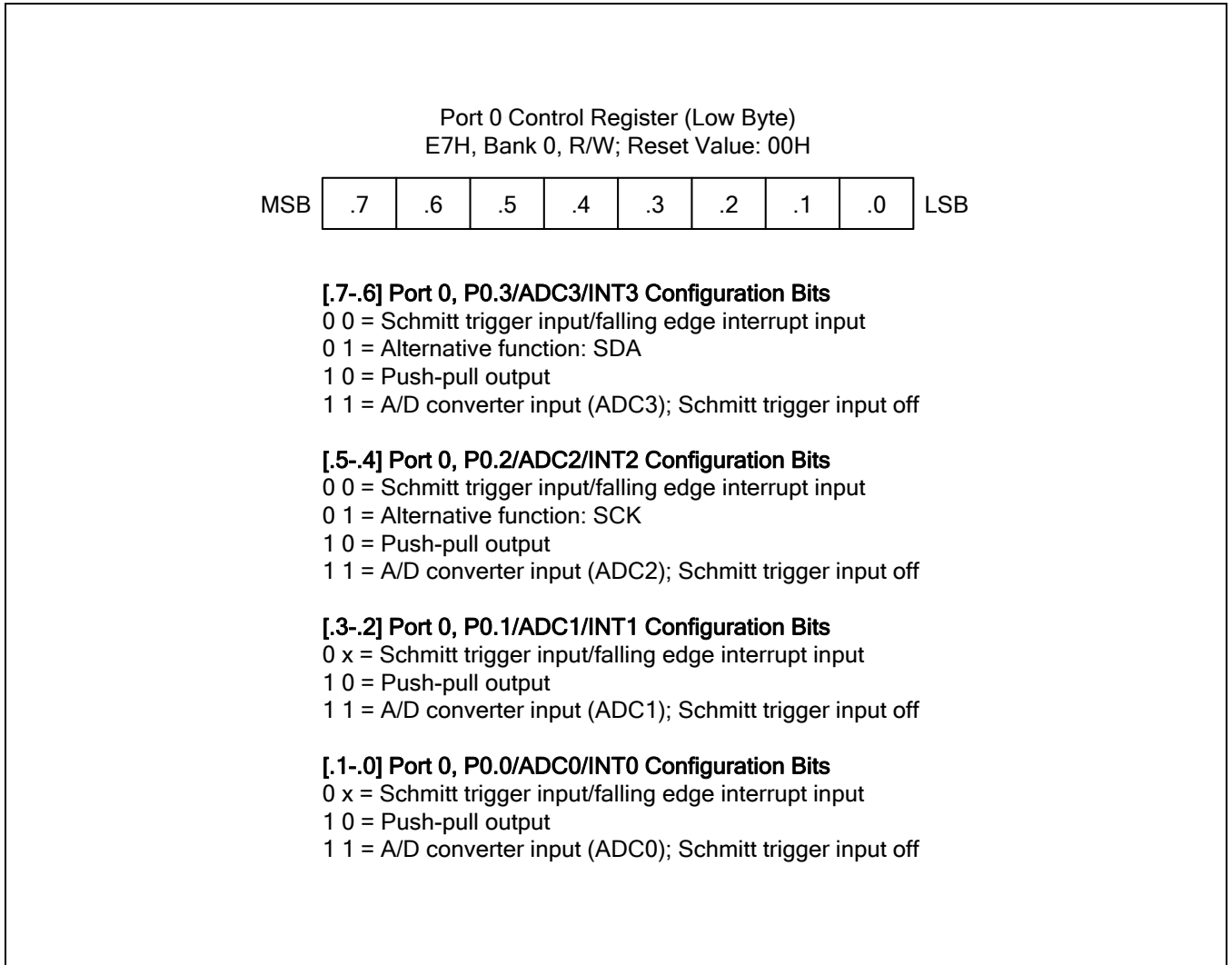
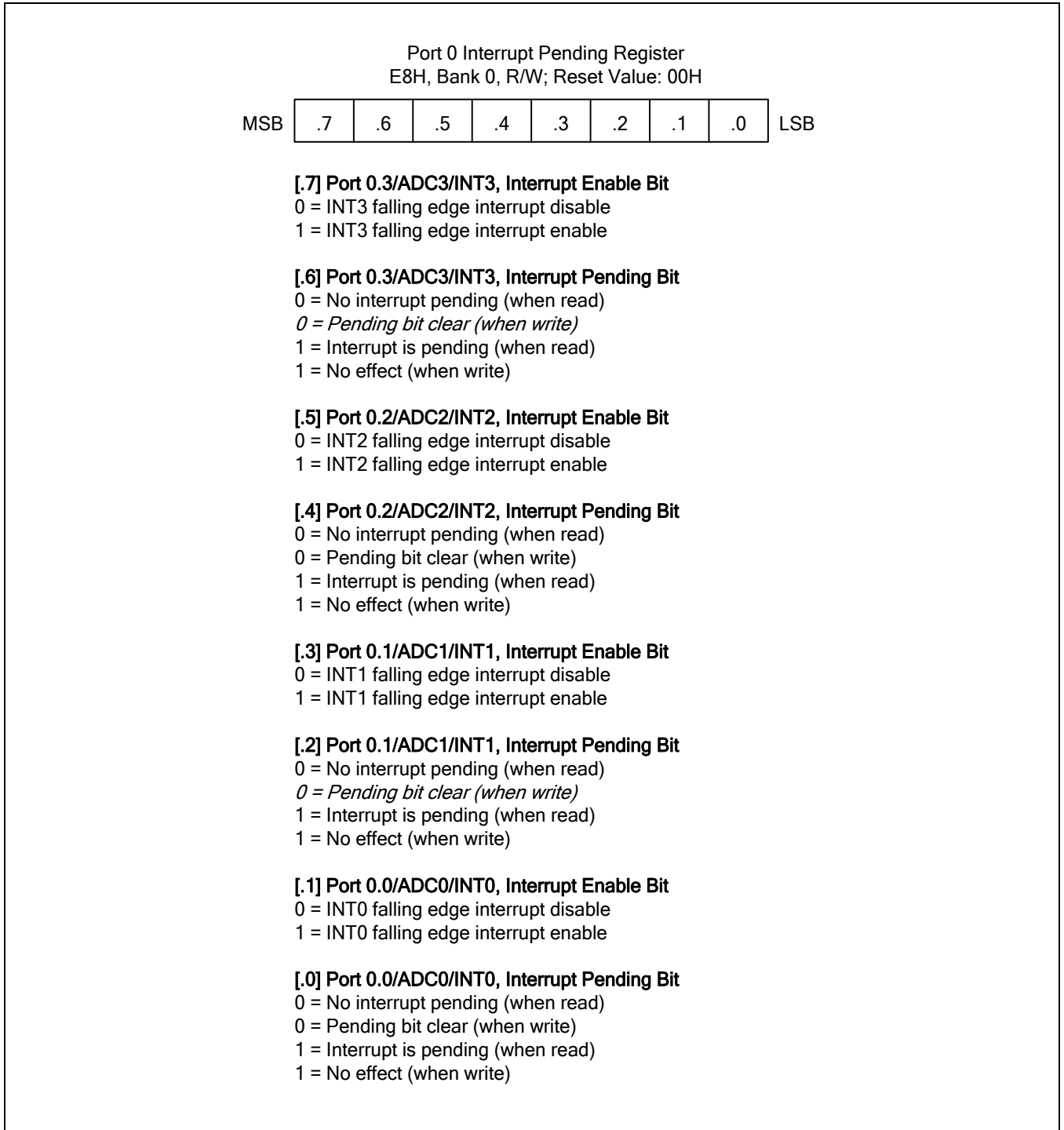


Figure 9-4 Port 0 Control Register (P0CONL, Low Byte)





**Figure 9-5 Port 0 Interrupt Pending Registers (P0PND)**

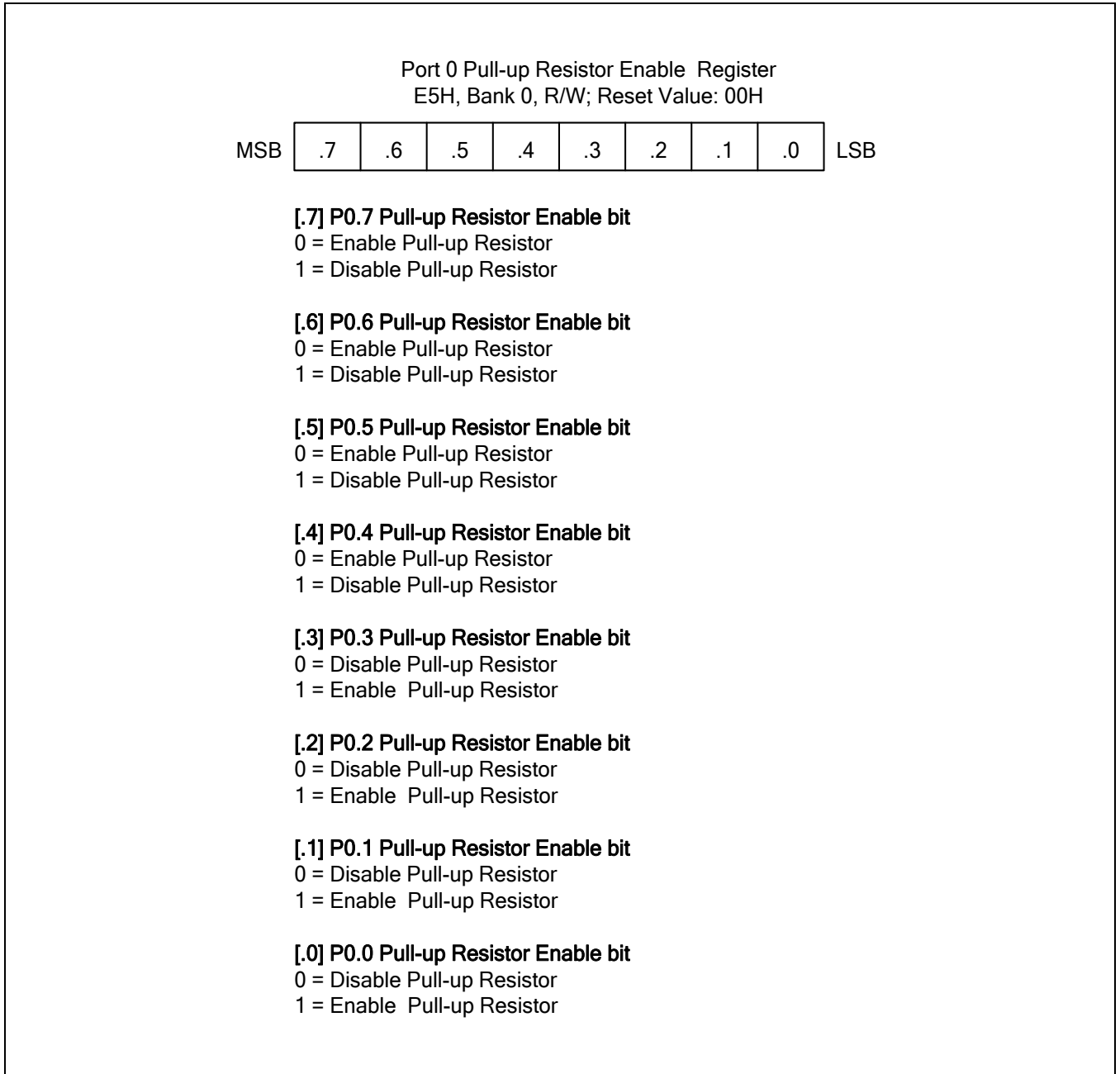


Figure 9-6 Port 0 Pull-Up Resistor Enable Registers (P0PUR)

9.2.2 Port 1

Port 1, is a 3-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. P1.0, P1.1 are used for oscillator input/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 Schmitt trigger input without pull-up or open-drain output.

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, P1.0 and P1.1 must be set to output port to prevent current consumption.

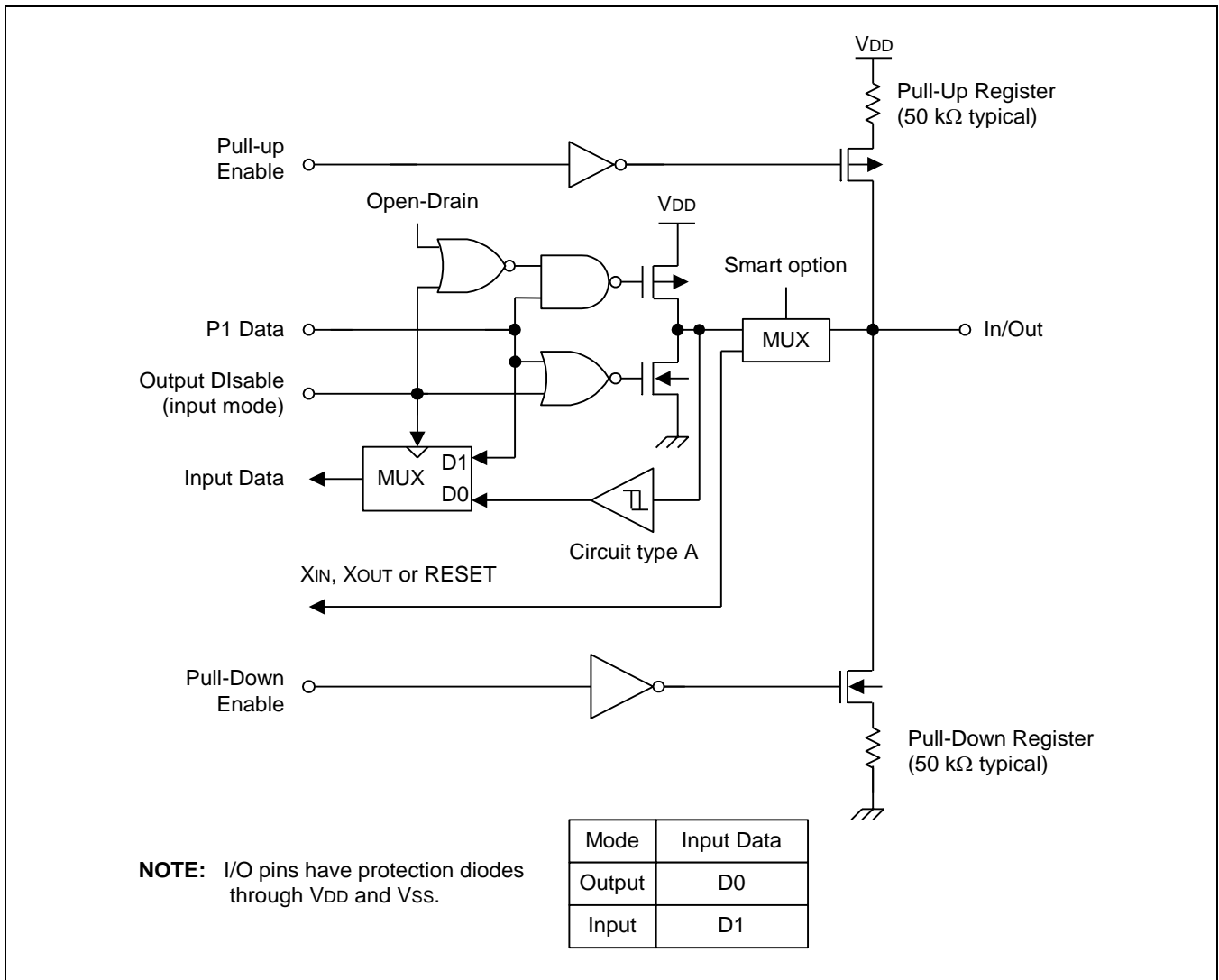
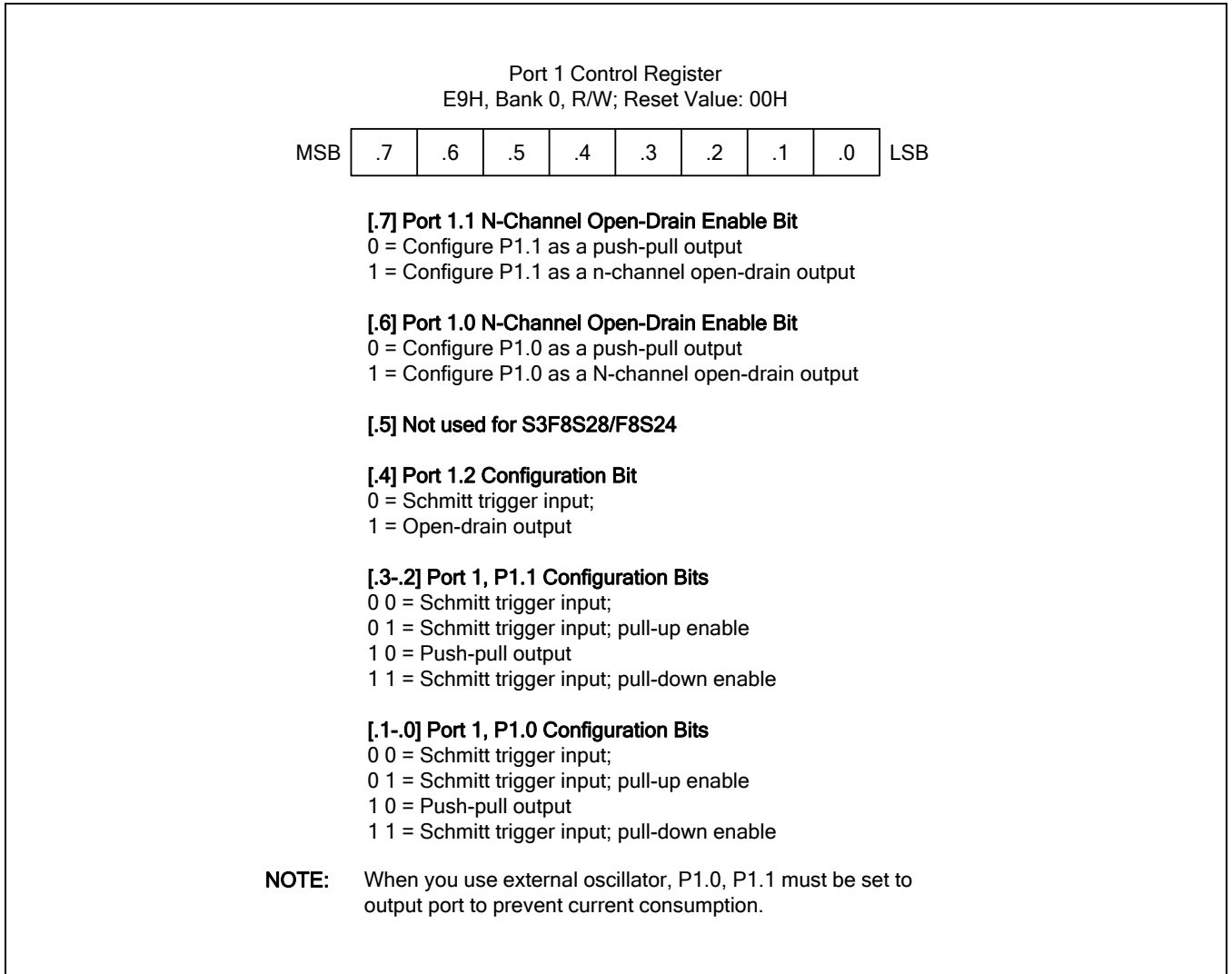


Figure 9-7 Port 1 Circuit Diagram

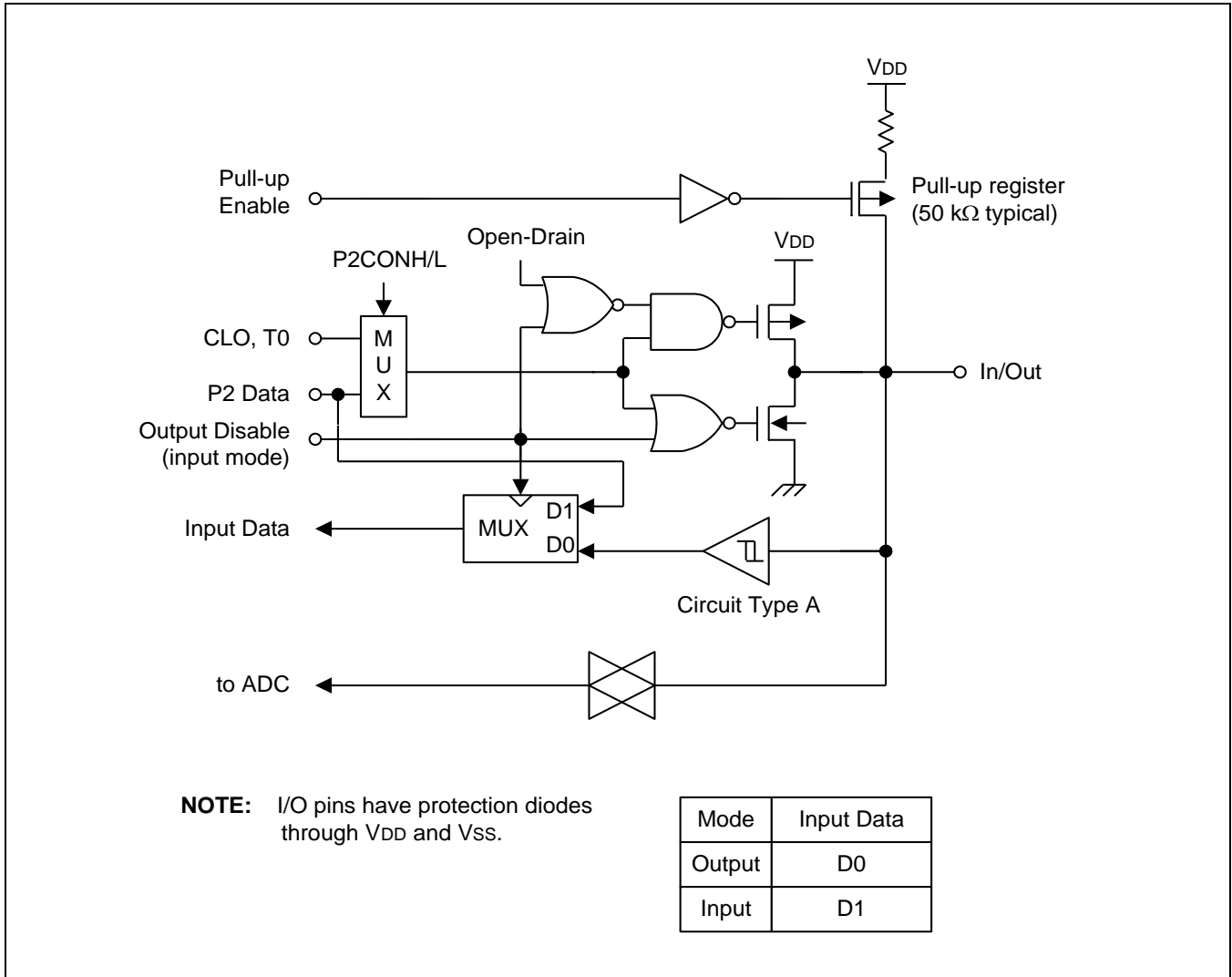


**Figure 9-8 Port 1 Control Register (P1CON)**

**9.2.3 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, T0 match output, T1 capture input, UART ports (RxD, TxD). 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, Bank 0). The port 2 control registers, P2CONH, P2CONL and P2PUR are located at addresses EAH, EBH and E4H of Bank 0 respectively.



**Figure 9-9 Port 2 Circuit Diagram**

**NOTE:**

1. P2.2/T1CAP has a T1CAP input module, and without ADC module.
2. When use P2.2/T1CAP as T1CAP, you must set P2.2/T1CAP in input mode.
3. P2.5 and P2.4 have not Open-drain function.

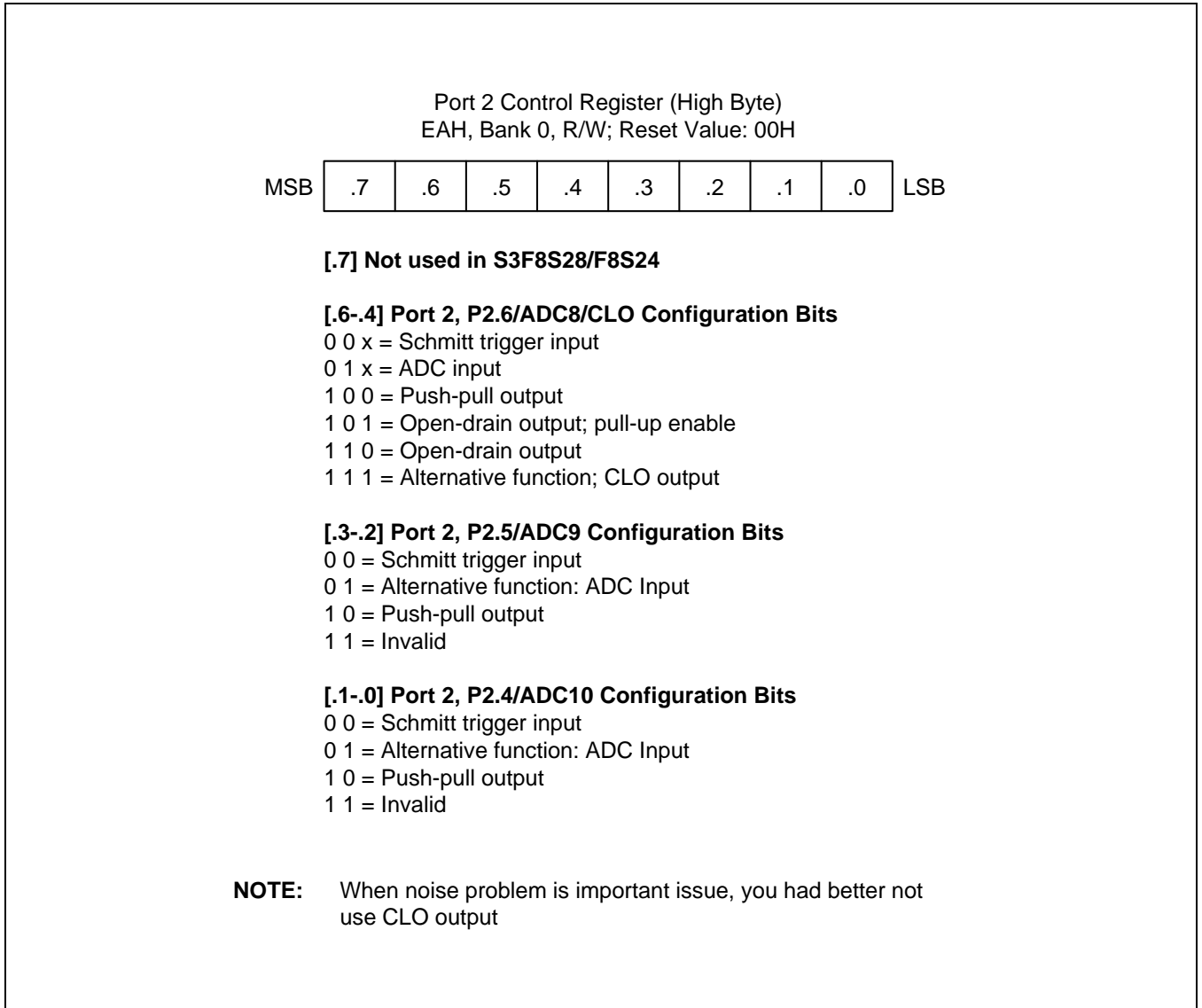


Figure 9-10 Port 2 Control Register (P2CONH, High Byte)

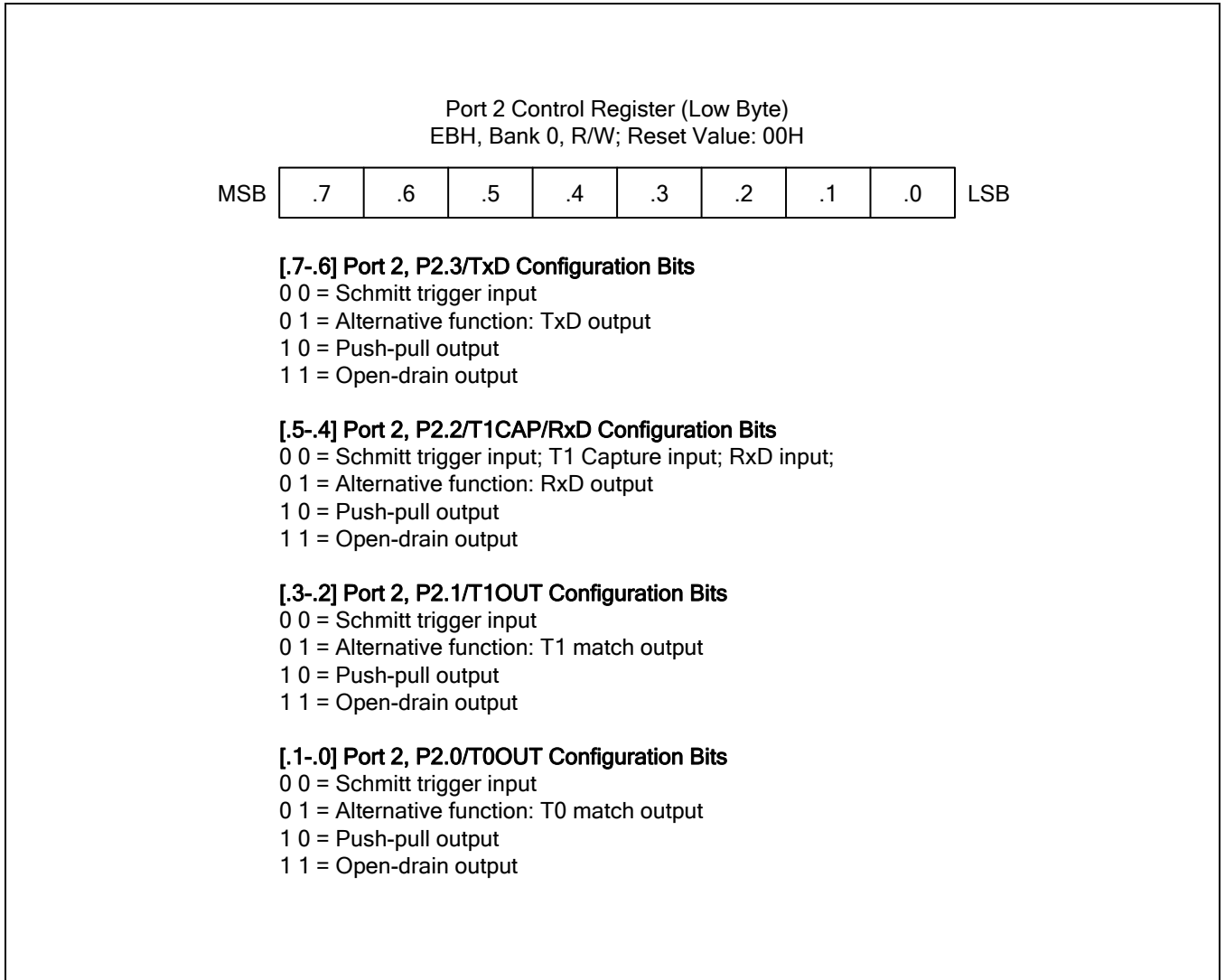
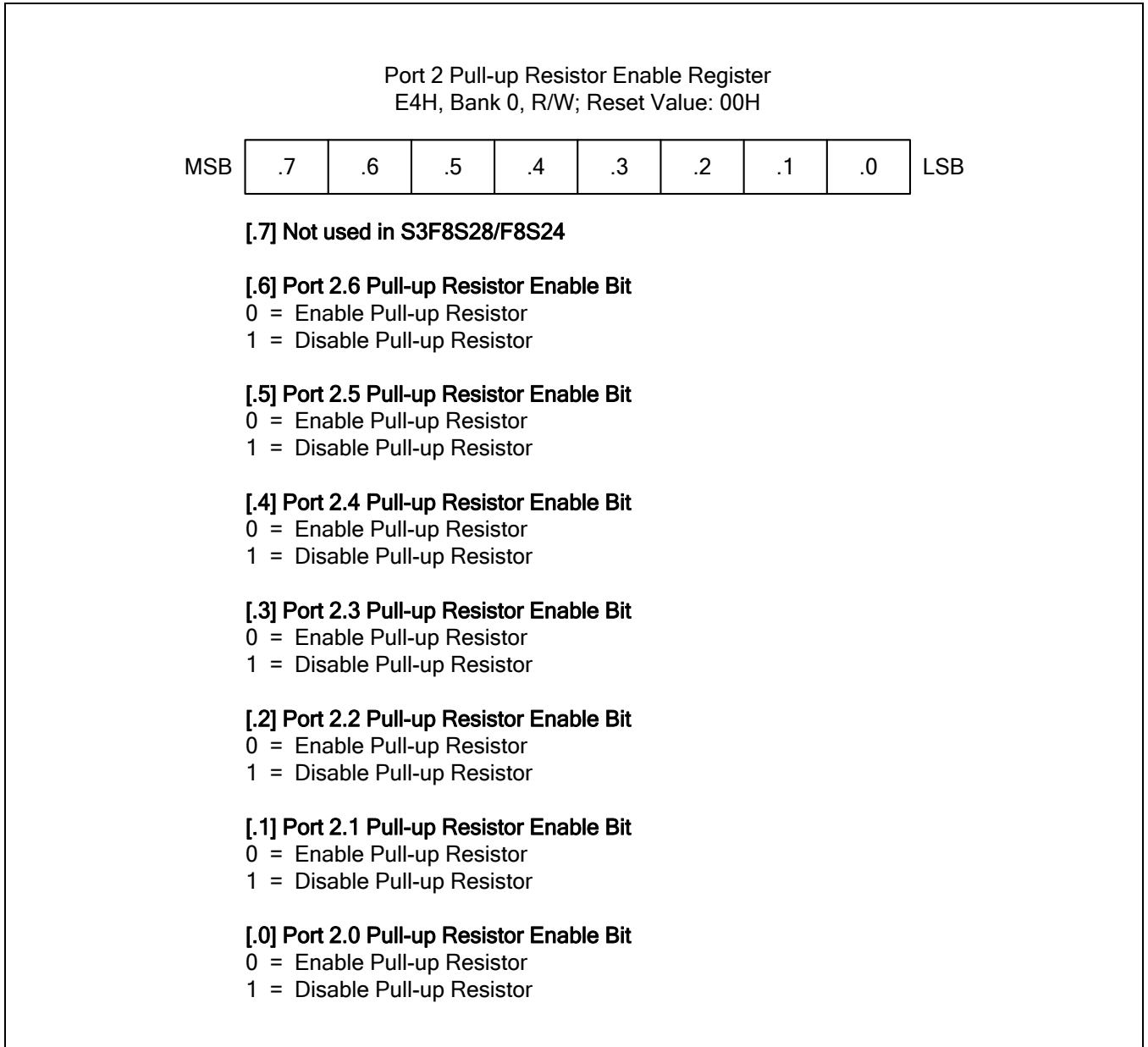


Figure 9-11 Port 2 Control Register (P2CONL, Low Byte)



**Figure 9-12 Port 2 Open-Drain Output Mode Register (P2PUR)**



9.2.4 Port 3

Port 3 is a 4-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 3 as ADC input, external interrupt input. 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 3 bits directly by writing or reading the port 3 data register, P3 (E3H, Bank 0). The port 3 control registers, P3CON and P3PND are located at addresses F0H and EFH of Bank 0 respectively.

You access port 3 directly by writing or reading the corresponding port data register, P0 (E0H).

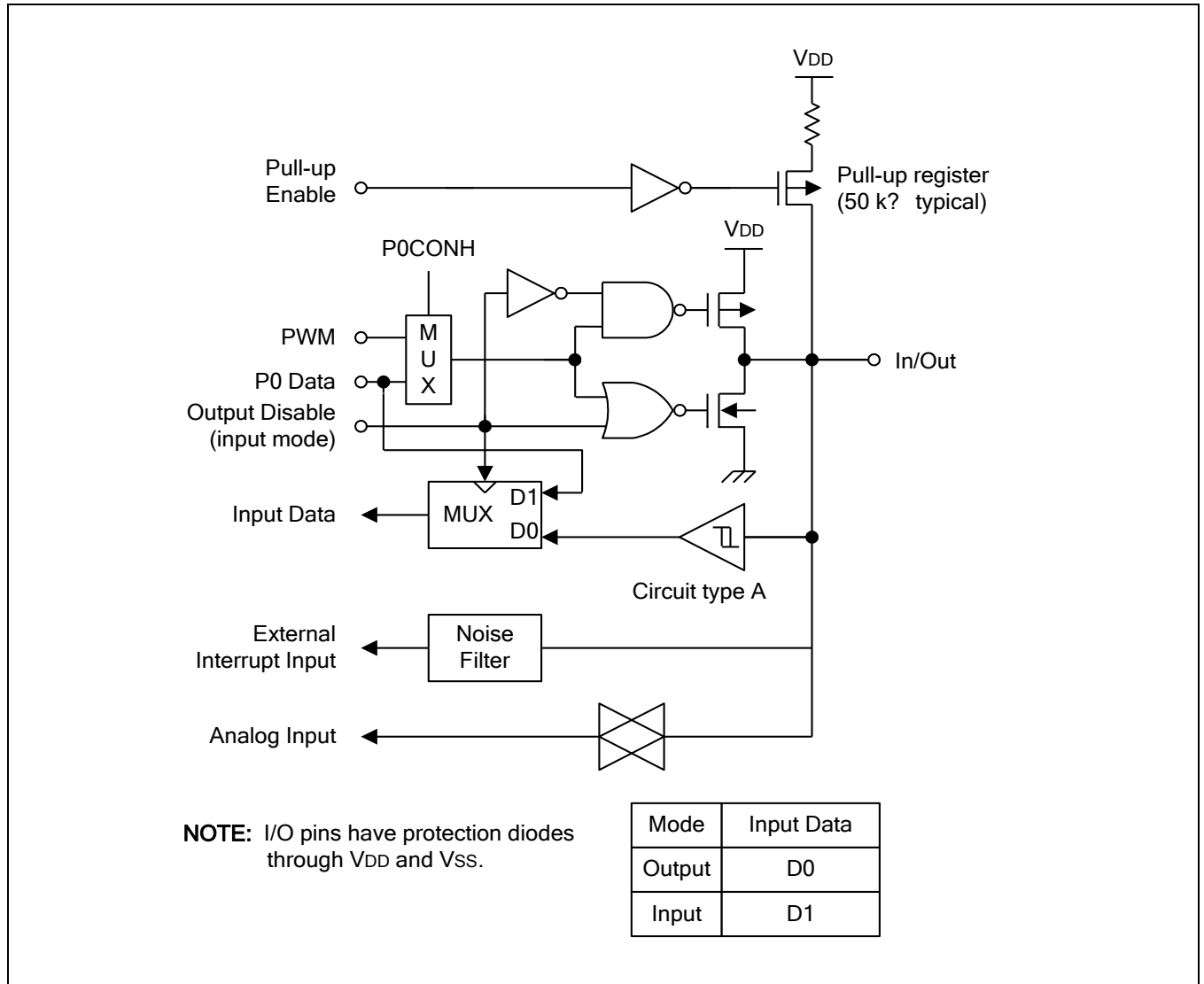
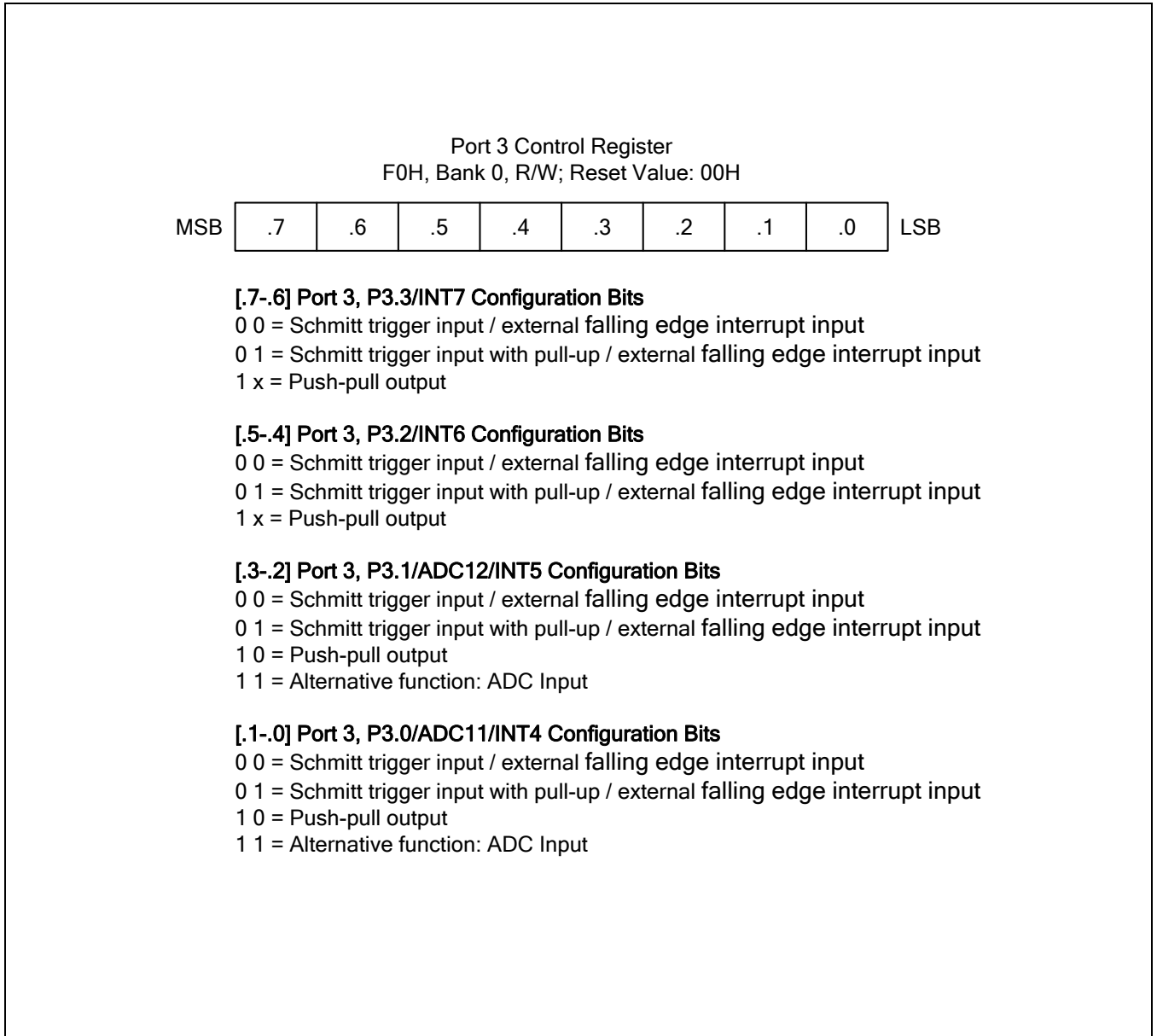
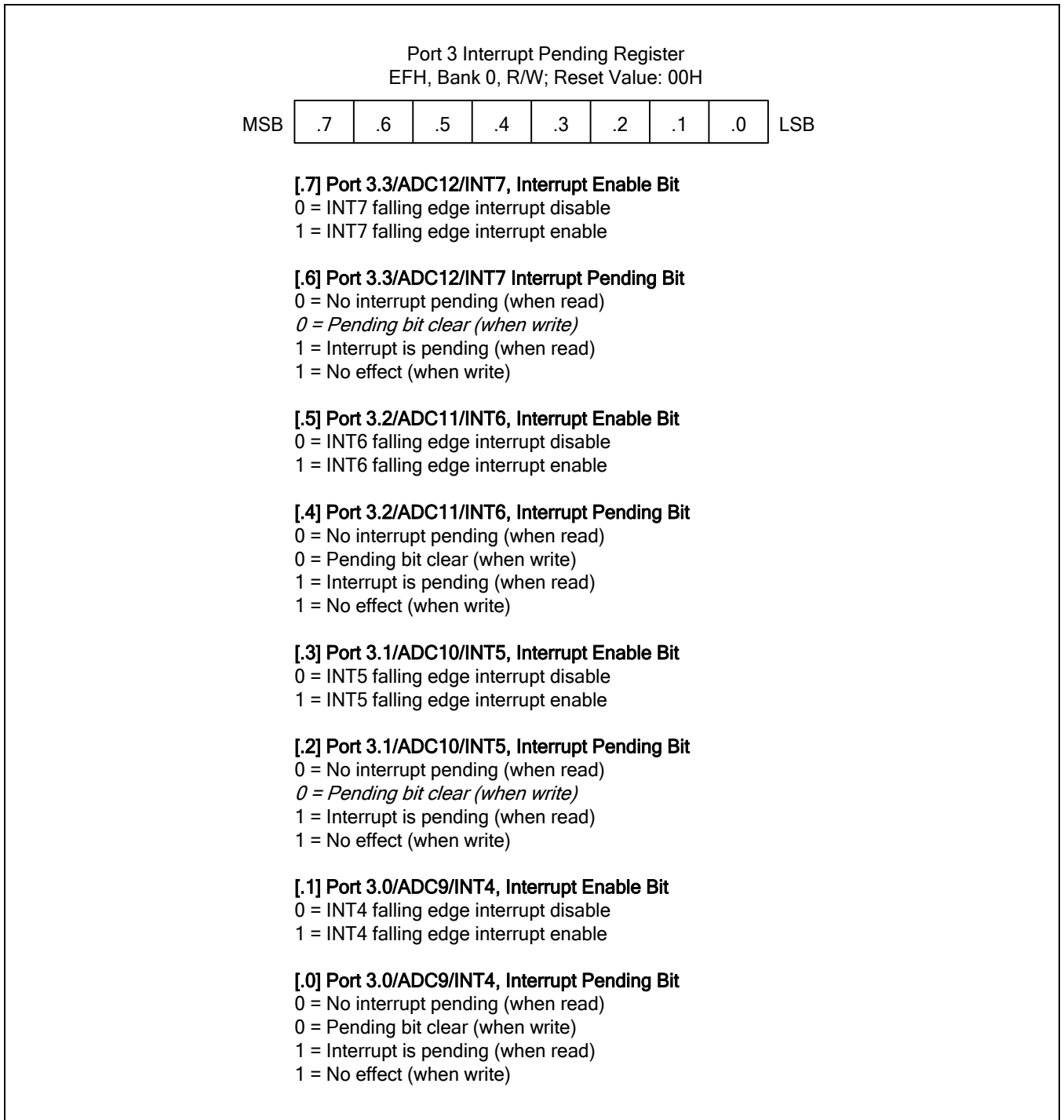


Figure 9-13 Port 3 Circuit Diagram



**Figure 9-14 Port 3 Control Register (P3CON)**



**Figure 9-15 Port 3 Interrupt Pending Register (P3PND)**

# 10 Basic Timer and Timer 0

## 10.1 Module Overview

The S3F8S28/S3F8S24 has two default timers: an 8-bit basic timer, and a 16-bit general-purpose timer, called Timer 0.

### 10.1.1 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)

### 10.1.2 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)

## 10.2 Basic Timer (BT)

### 10.2.1 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 to 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.

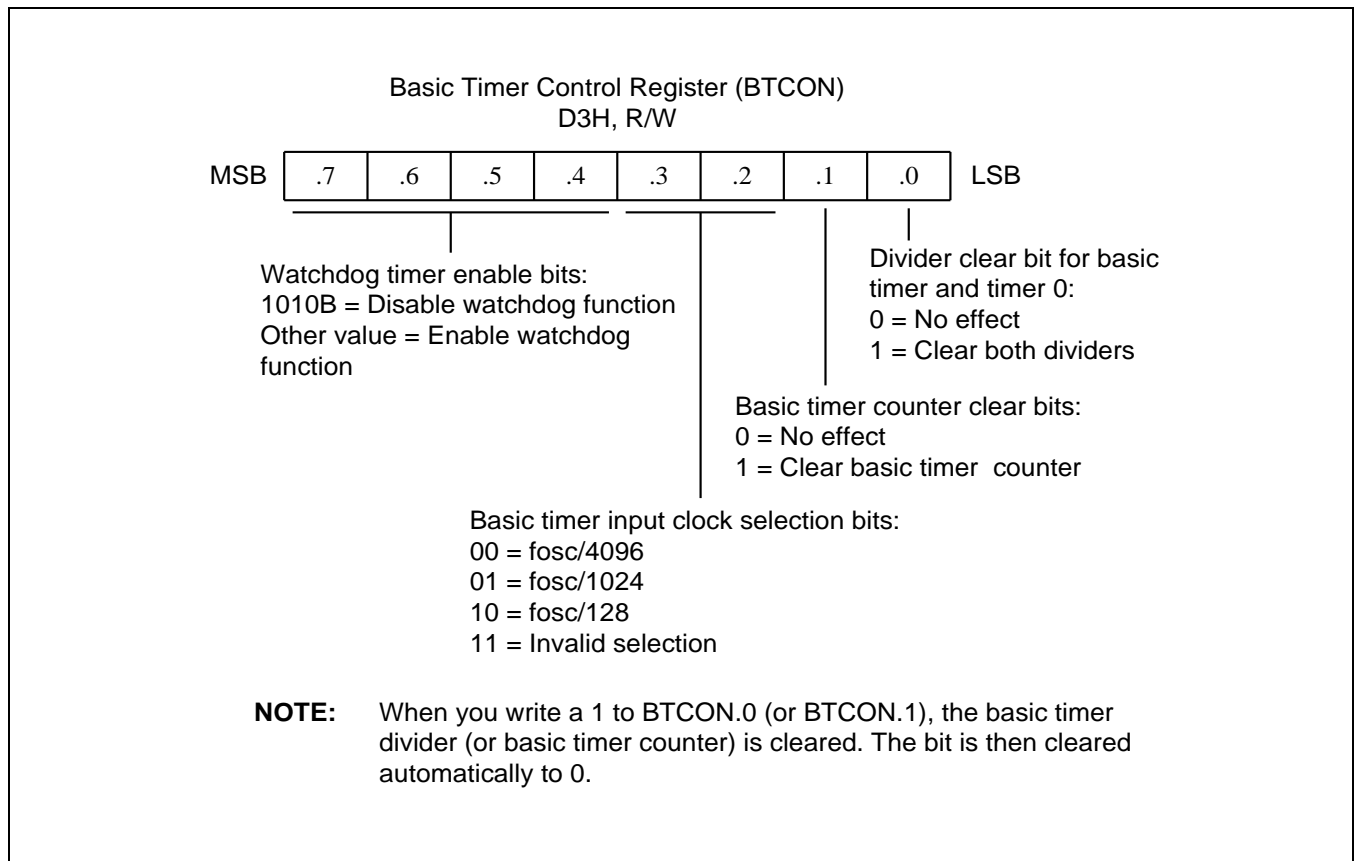


Figure 10-1 Basic Timer Control Register (BTCON)

## 10.2.2 Basic Timer Function Description

### 10.2.2.1 Watchdog Timer Function

You can program the basic timer overflow signal (BTOVF) to generate a Reset by setting BTCON.7–BTCON.4 to any value other than "1010B" (The "1010B" value disables the watchdog function). A Reset clears BTCON to "00H", automatically enabling the watchdog timer function. 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.

### 10.2.2.2 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 [Figure 10-3](#) shows the oscillation stabilization time on RESET and Stop Mode release

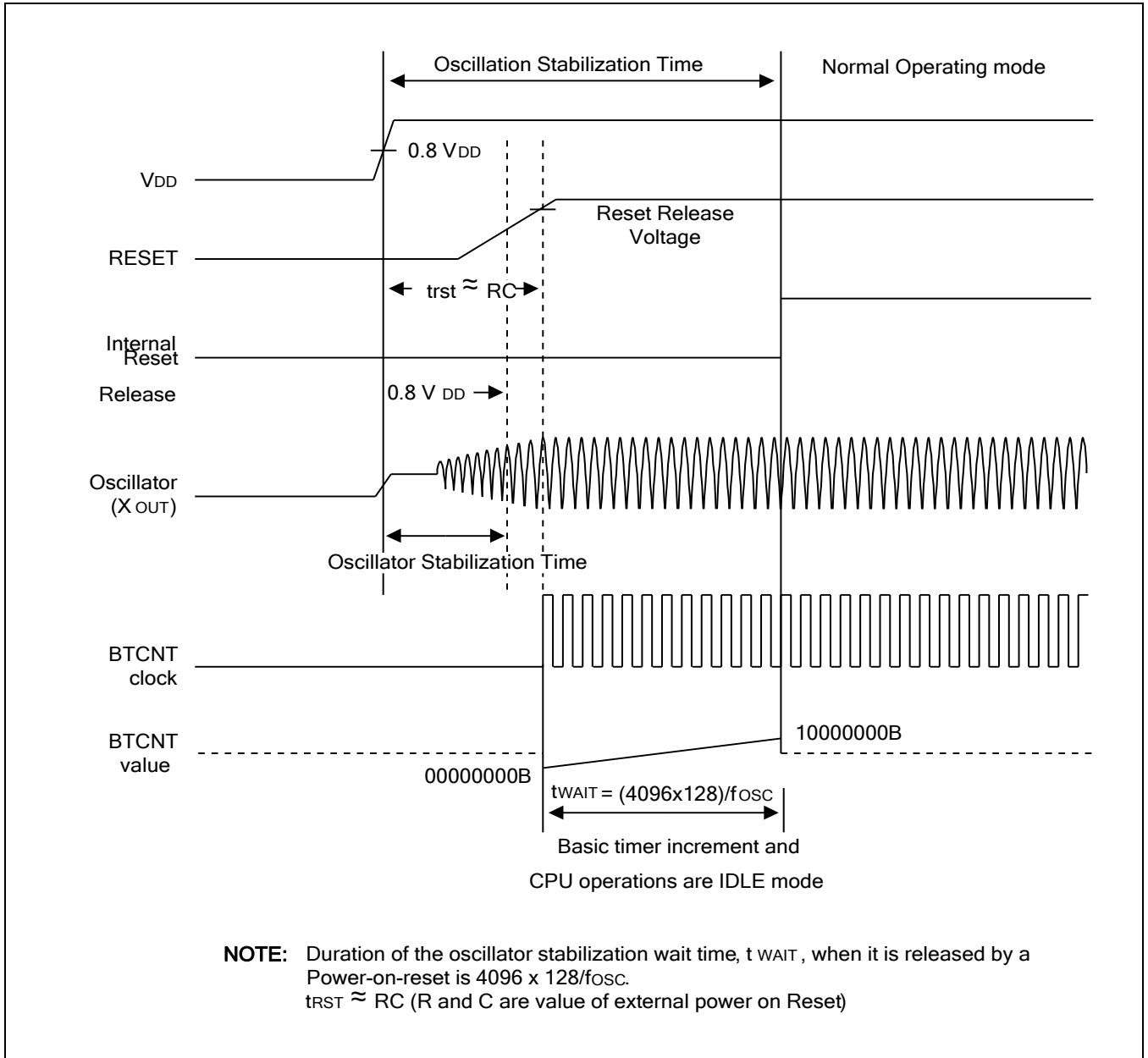


Figure 10-2 Oscillation Stabilization Time on RESET

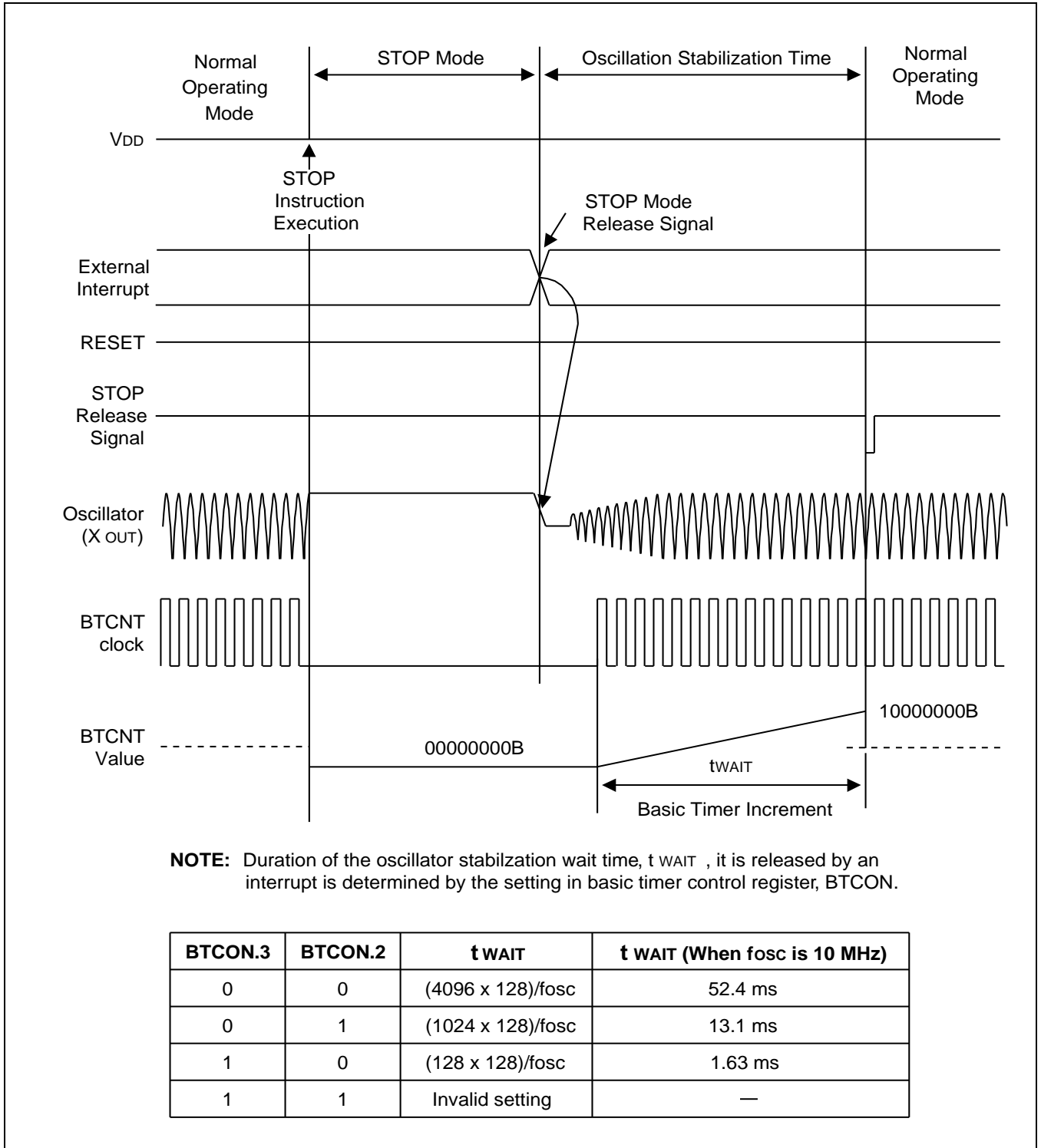


Figure 10-3 Oscillation Stabilization Time on Stop Mode Release



**Example 10-1 Configuring the Basic Timer**

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

```

    ORG    0000H

;-----<< Smart Option >>
    ORG    003CH
    DB     0FFH           ; 003CH, must be initialized to 1
    DB     0FFH           ; 003DH, must be initialized to 1
    DB     0FFH           ; 003EH, enable LVR
    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        ;

```

## 10.3 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)

### 10.3.1 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 (f<sub>xx</sub> 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)

### 10.3.2 Function Description

#### 10.3.2.1 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.

### 10.3.2.2 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  $f_{xx}/256$ , 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.

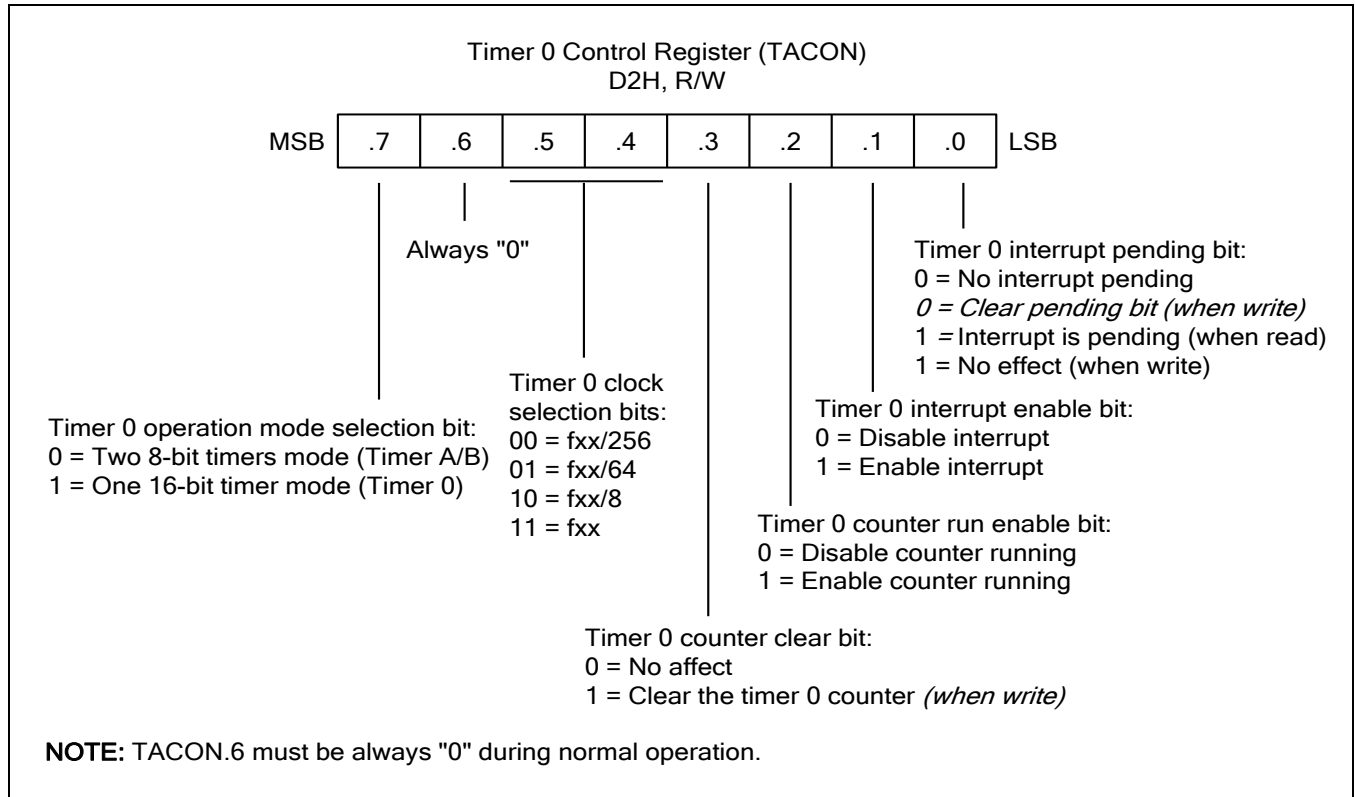


Figure 10-4 Timer 0 Control Register (TACON)

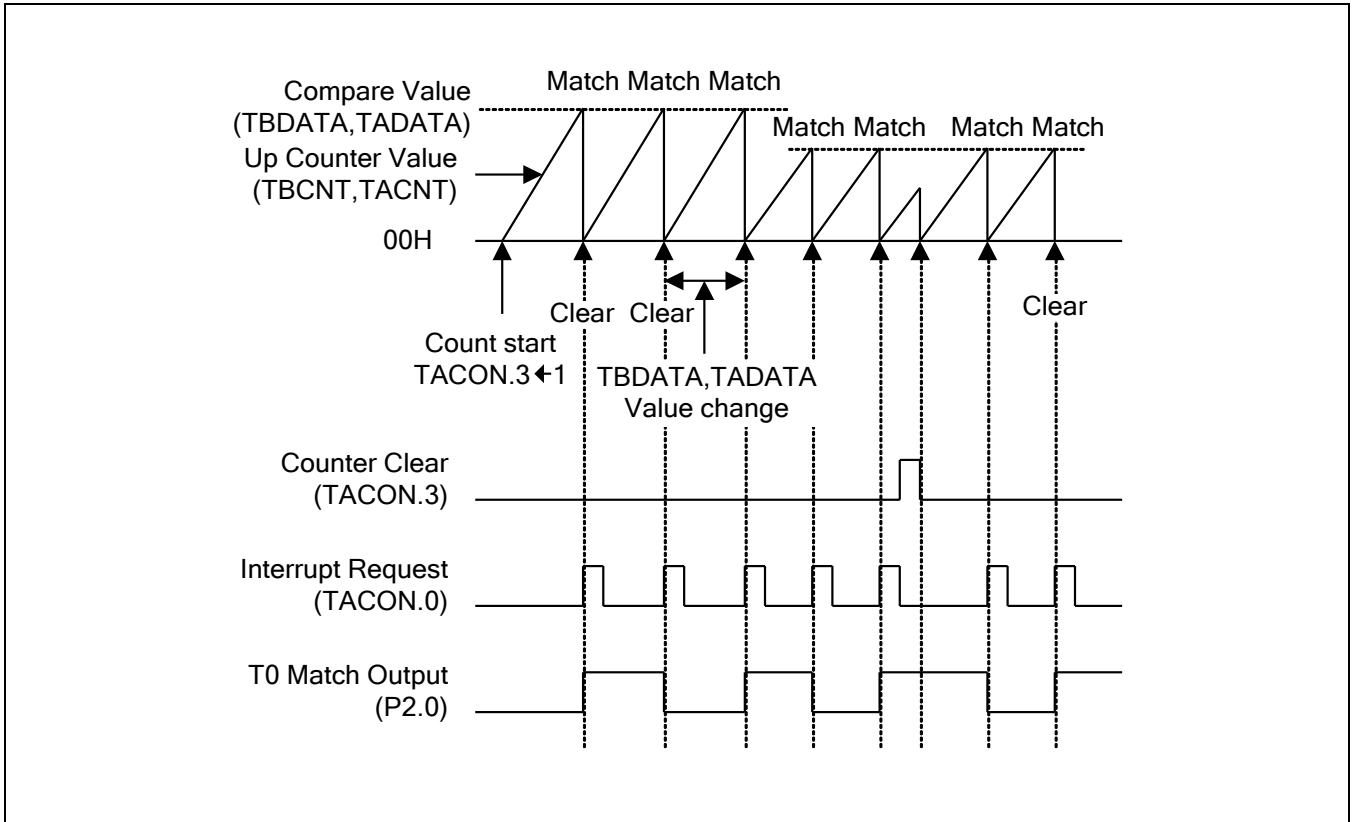


Figure 10-5 Timer 0 Timing Diagram

10.3.3 Block Diagram

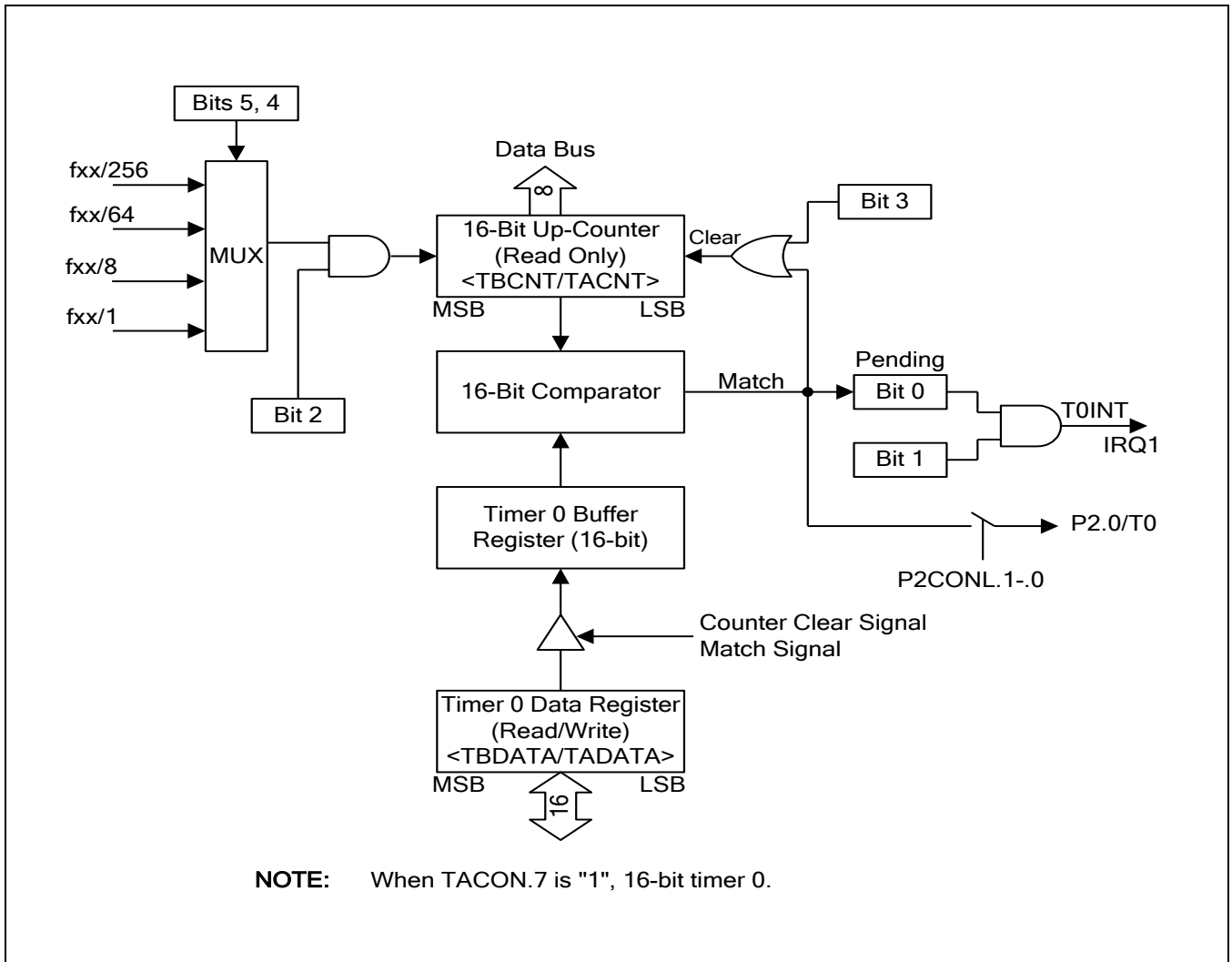


Figure 10-6 Timer 0 Functional Block Diagram

## 10.4 Two 8-Bit Timers Mode (Timer A and B)

### 10.4.1 Overview

The 8-bit timer A and B are the 8-bit general-purpose timers. Timer A and B support interval timer mode using the appropriate TACON and TBCON setting, respectively.

Timer A and Timer 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)

### 10.4.2 Function Description

#### 10.4.2.1 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.

### 10.4.2.2 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  $f_{xx}/256$ , 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 TAIN 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.

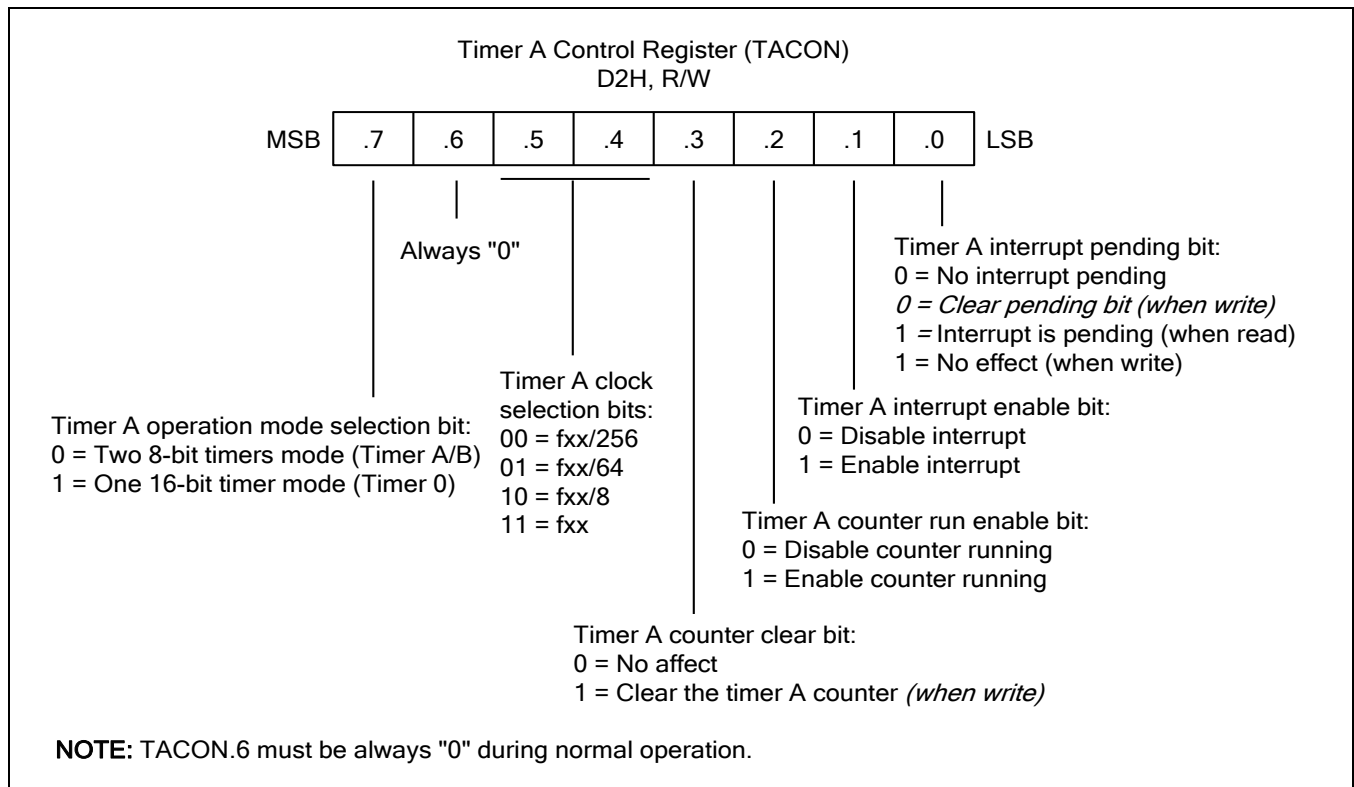


Figure 10-7 Timer A Control Register (TACON)

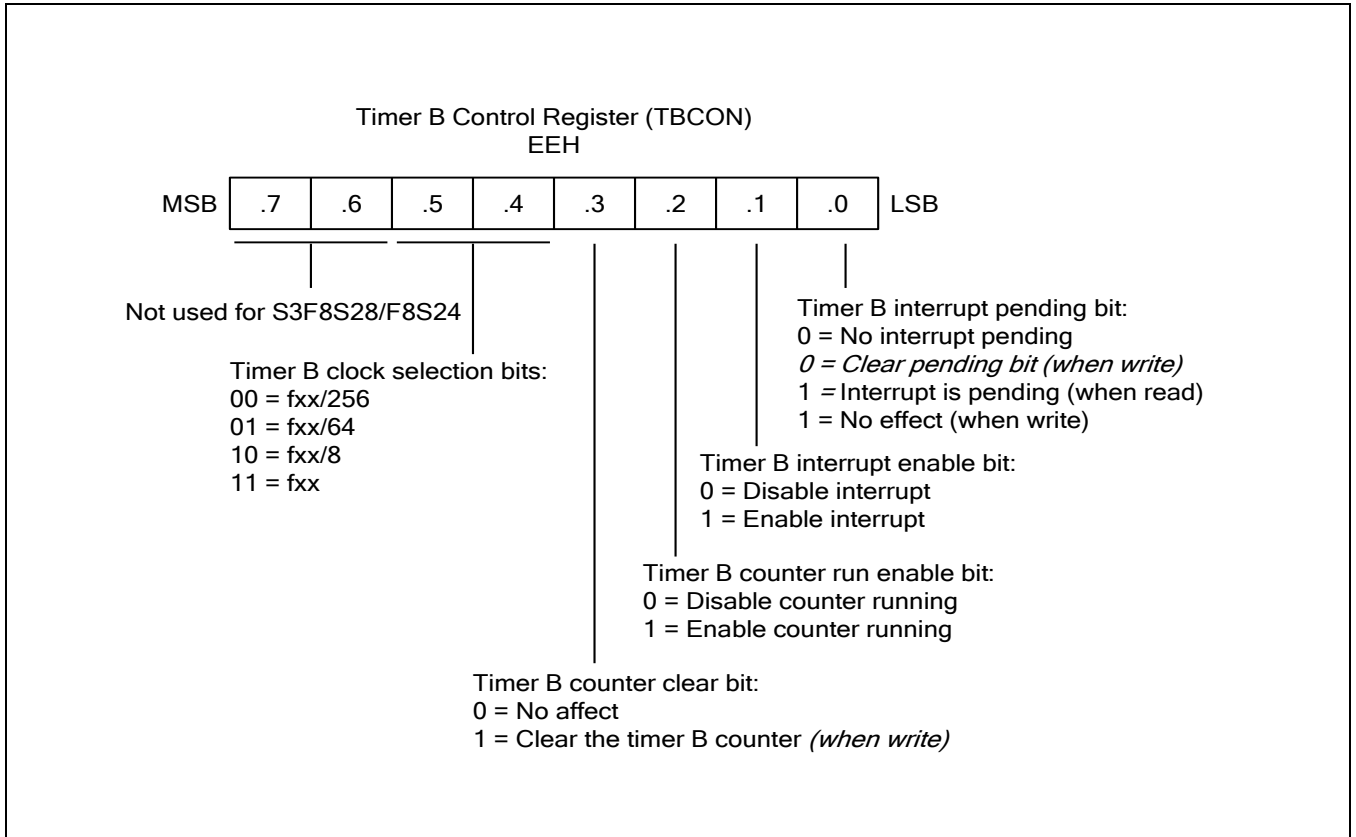


Figure 10-8 Timer B Control Register (TBCON)



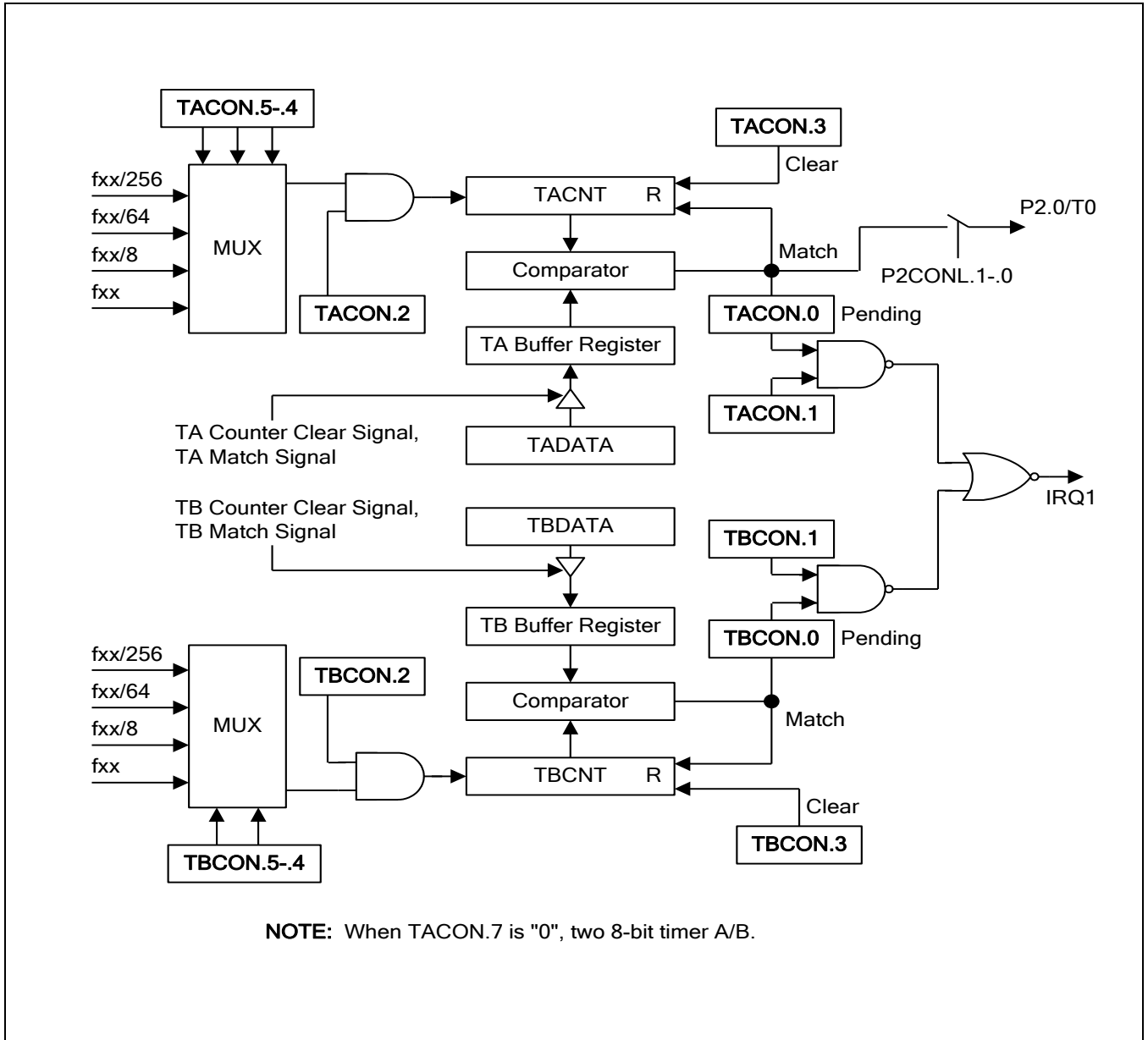


Figure 10-9 Timer A and B Function Block Diagram

# 11

## 16-Bit Timer 1

### 11.1 Overview

The S3F8S28/S3F8S24 has a 16-bit timer/counters-Timer 1. The 16-bit Timer 1 is a 16-bit general-purpose timer/counter. Timer 1 has two operating modes, one of which you select using the appropriate T1CON setting is:

- Interval timer mode
- Capture input mode with a rising or falling edge trigger at the T1CAP pin

Timer 1 has the following functional components:

- Precalar for clock frequency programmable from  $f_{OSC}$  to  $f_{OSC}/4096$
- A 16-bit counter, 16-bit comparator, and two 16-bit reference data register (T1DATAH/L)
- I/O pins for capture input (T1CAP)
- Timer 1 overflow interrupt and match/capture interrupt generation
- Timer 1 control register, T1CON
- Timer 1 Prescaler register, T1PS

You can use Timer 1 in three ways:

- As a normal counter, generating a Timer 1 overflow interrupt (IRQ3, vector EAH) at programmed time intervals.
- To generate a Timer 1 match interrupt (IRQ3, vector ECH) when the 16-bit Timer 1 count value matches the 16-bit value written to the reference data registers.
- To generate a Timer 1 capture interrupt (IRQ3, vector ECH) when a triggering condition exists at the T1CAP (P2.2)

## 11.2 Function Description

### 11.2.1 Timer 1 Interrupts

The Timer 1 module can generate two interrupts, the Timer 1 overflow interrupt (T1OVF), and the Timer 1 match/capture interrupt (T1INT). A Timer 1 overflow interrupt pending condition is cleared by software when it has been serviced. A Timer 1 match/capture interrupt, T1INT pending condition is also cleared by software when it has been serviced.

### 11.2.2 Timer 1 Overflow Interrupt

Timer 1 can be programmed to generate an overflow interrupt (IRQ3, vector EAH) whenever an overflow occurs in the 16-bit up counter. When you set the Timer 1 overflow interrupt enable bit, T1CON.3, to "1", the overflow interrupt is generated each time the 16-bit up counter reaches "FFFFH". After the interrupt request is generated, the counter value is automatically cleared to "00H" and up counting resumes. By writing a "1" to T1CON.4, you can clear/reset the 16-bit counter value at any time during program operation.

### 11.2.3 Interval Mode (Match)

Timer 1 can also be used to generate a match interrupt T1INT (IRQ3, vector ECH) whenever the 16-bit counter value matches the value that is written to the Timer 1 reference data registers, T1DATAH and T1DATA L. When a match condition is detected by the 16-bit comparator, the match interrupt is generated, the counter value is cleared, and up counting resumes from "00H".

In match mode, program software can poll the Timer 1 match/capture interrupt pending bit, T1CON.0, to detect when a Timer 1 match interrupt pending condition exists (T1CON.0 = "1"). When the interrupt request is acknowledged by the CPU and the service routine starts, the interrupt service routine for vector ECH must clear the interrupt pending condition by writing a "0" to T1CON.0.

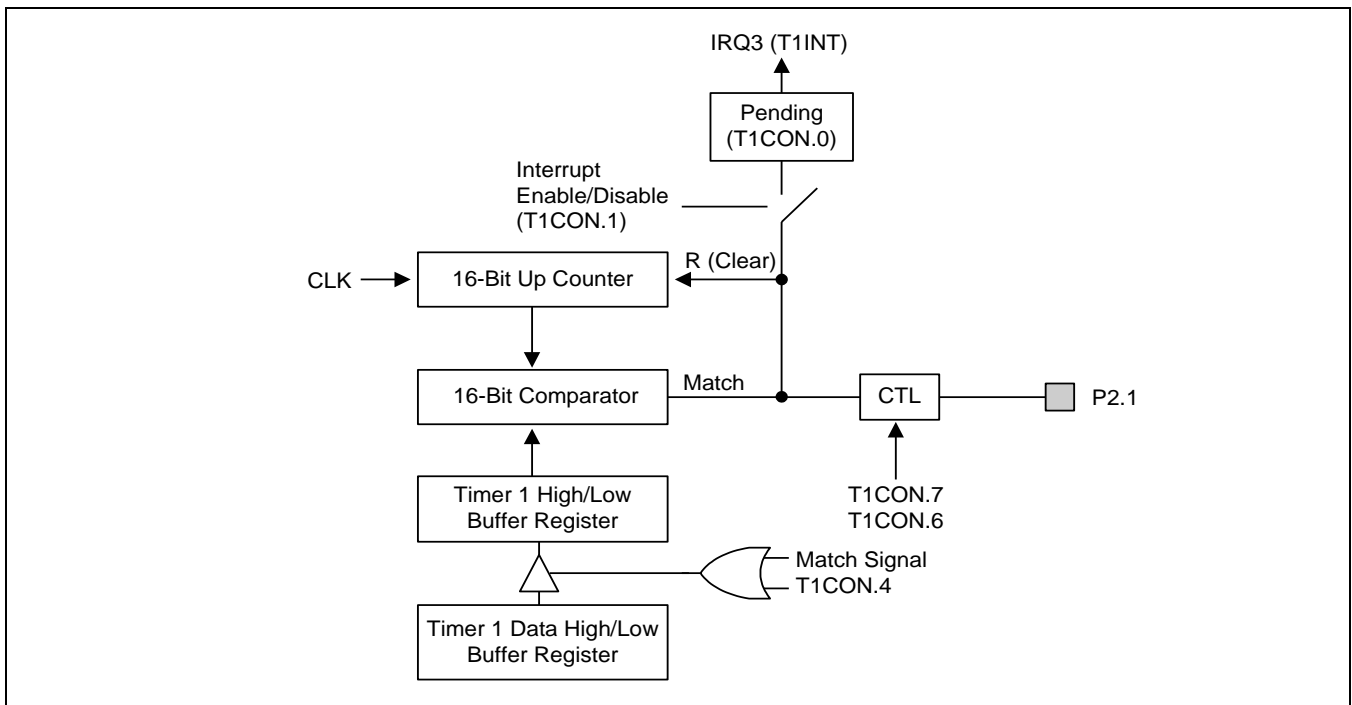


Figure 11-1 Simplified Timer 1 Function Diagram: Interval Mode

### 11.2.4 Capture Mode

Timer 1 also gives you capture input source, the signal edge at the T1CAP (P2.2) pin. You select the capture input by setting the input mode in the port 2 control register, P2CONL

Timer 1 can be used to generate a capture interrupt (IRQ3, vector ECH) whenever a triggering condition is detected at the T1CAP (P2.2) pin. The T1CON.7 and T1CON.6 bit-pair setting is used to select the trigger condition for capture mode operation: rising edges, falling edges or on both falling and rising edge.

In capture mode for Timer 1, a signal edge that is detected at the T1CAP pin opens a gate and loads the current counter value into the T1 data register (T1DATAH/L for rising edge, or falling edge or on both falling and rising edge).

Both kinds of Timer 1 interrupts (T1OVF, T1INT) can be used in capture mode, the Timer 1 overflow interrupt is generated whenever a counter overflow occurs, the Timer 1 capture interrupt is generated whenever the counter value is loaded into the T1 data register (T1DATAH/L).

By reading the captured data value in T1DATAH/L, and assuming a specific value for the Timer 1 clock frequency, you can calculate the pulse width (duration) of the signal that is being input at the T1CAP pin

In capture mode, program software can poll the Timer 1 match/capture interrupt pending bit, T1CON.0, to detect when a Timer 1 capture interrupt pending condition exists (T1CON.0 = "1"). When the interrupt request is acknowledged by the CPU and the service routine starts, the interrupt service routine for vector ECH must clear the interrupt pending condition by writing a "0" to T1CON.0

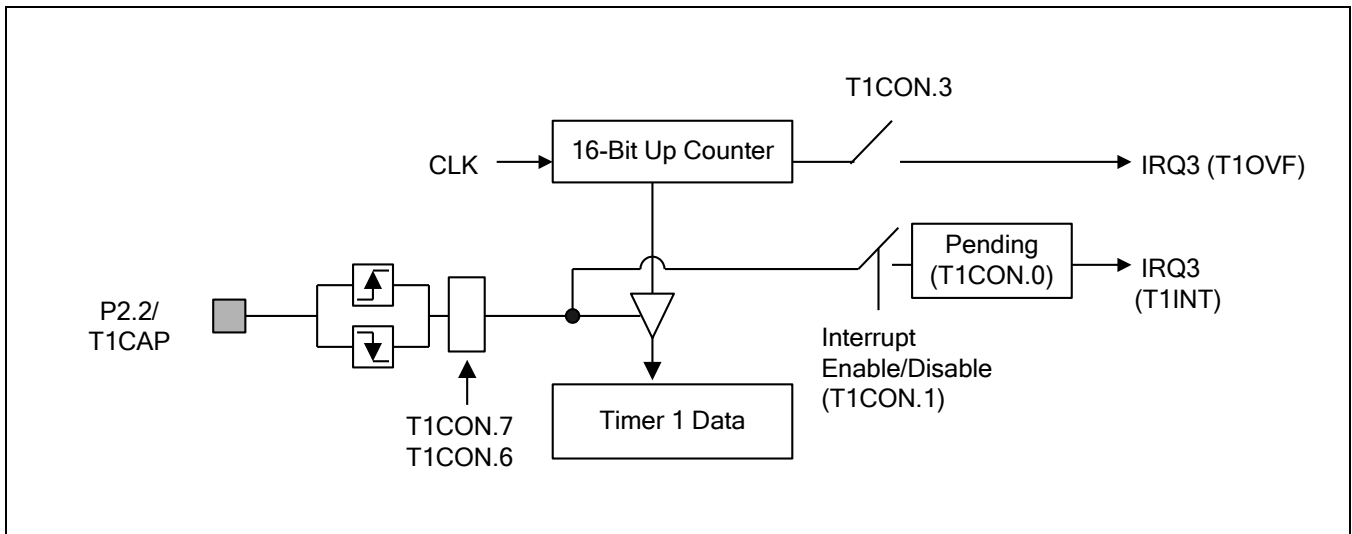


Figure 11-2 Simplified Timer 1 Function Diagram: Capture Mode

### 11.3 Timer 1 Control Register (T1CON)

You use the TIMER 1 control register, T1CON, to

- Select the TIMER 1 operating mode (interval timer, capture mode)
- Timer1 Start/Stop
- Clear the TIMER 1 counter.
- Enable/Disable the Timer 1 overflow interrupt or Timer 1 match/capture interrupt
- Clear Timer1 overflow interrupt or match/capture pending bit

You can use Timer 1 prescaler register, T1PS to

- Program clock prescaler

T1CON is located at address E4H, Bank 1, Set 1, and is read/write addressable using register addressing mode. A reset clears T1CON to "00H". This sets TIMER 1 to normal interval timer mode, disable Timer 1 run; disable Timer 1 overflow and match/capture interrupt.

You can start Timer 1 counter by writing a "1" to T1CON.5. You can clear the Timer 1 counter at any time during normal operation by writing a "1" to T1CON.4. To generate the exact time interval, you should write "1" to T1CON.4 and clear appropriate pending bits of the T1CON register.

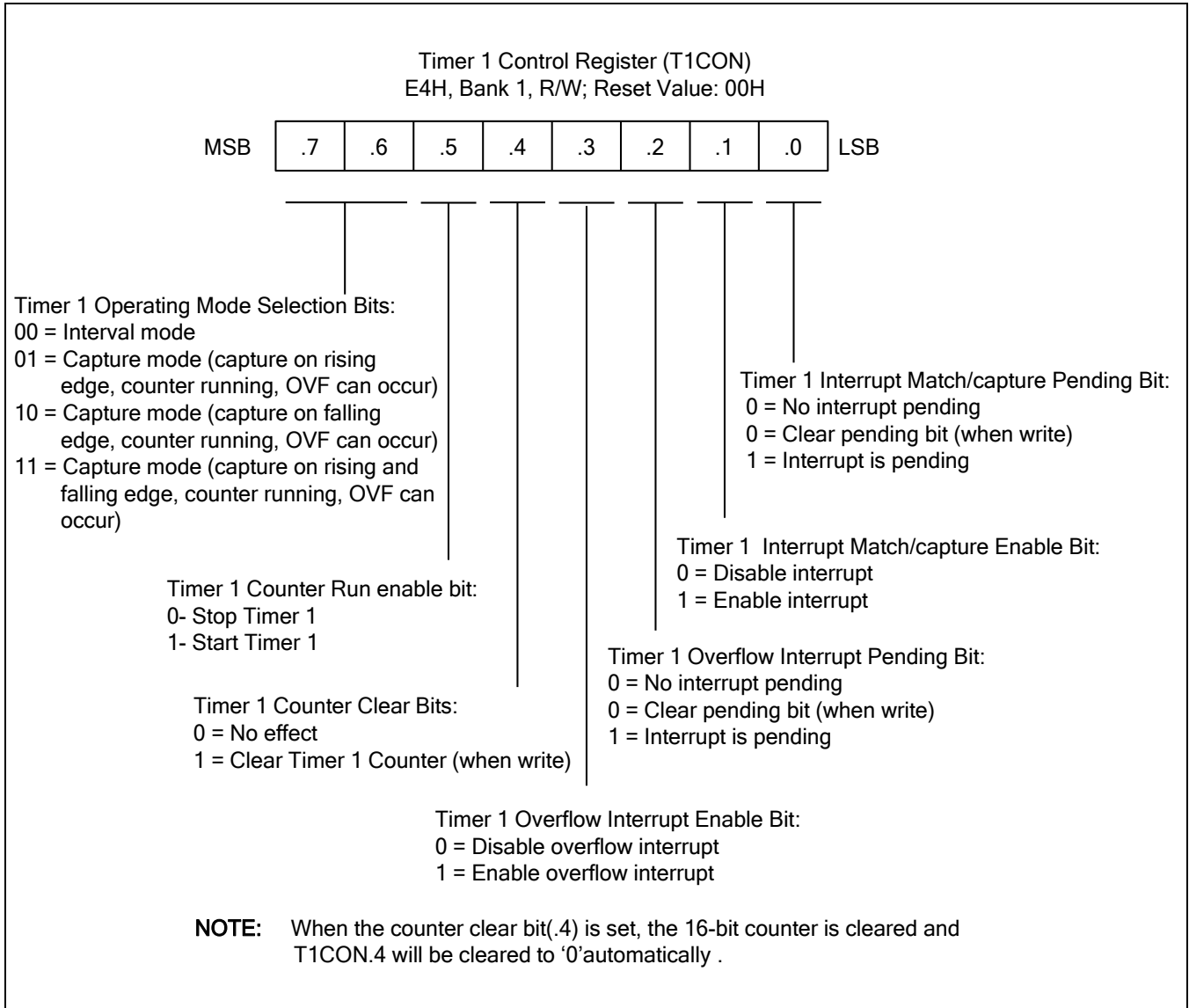
Timer 1 match/capture interrupt is controlled by T1CON.1; you can enable Timer 1 match/capture interrupt by writing a "1" to T1CON.1 or disable it by writing "0" to T1CON.1. Timer 1 overflow (T1OVF) is set by T1CON.3, you can enable Timer 1 overflow interrupt by writing a "1" to T1CON.3 or disable it by writing "0" to T1CON.3.

To detect a match/capture or overflow interrupt pending condition when T1INT or T1OVF is disabled, the application program should poll the pending bit T1CON.0 and T1CON.2. When a "1" is detected, a Timer 1 match/capture or overflow interrupt is pending.

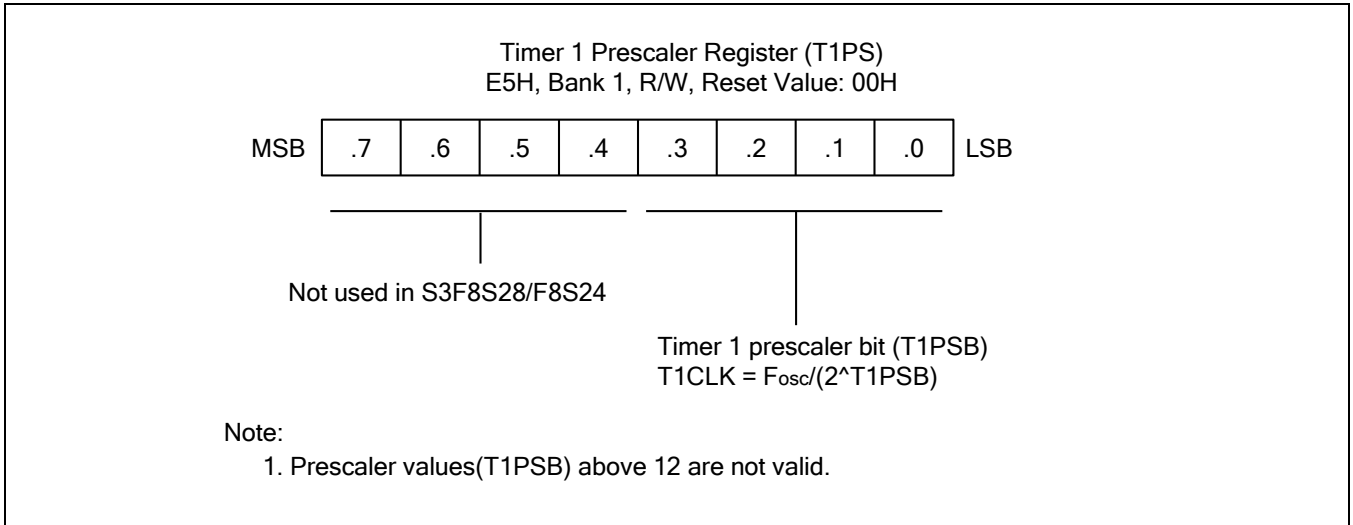
When the sub-routine has been serviced, the pending condition must be cleared by software by writing a "0" to the interrupt pending bit.

T1PS is located at address E5H, Bank 1, Set 1, and is read/write addressable using Register addressing mode.

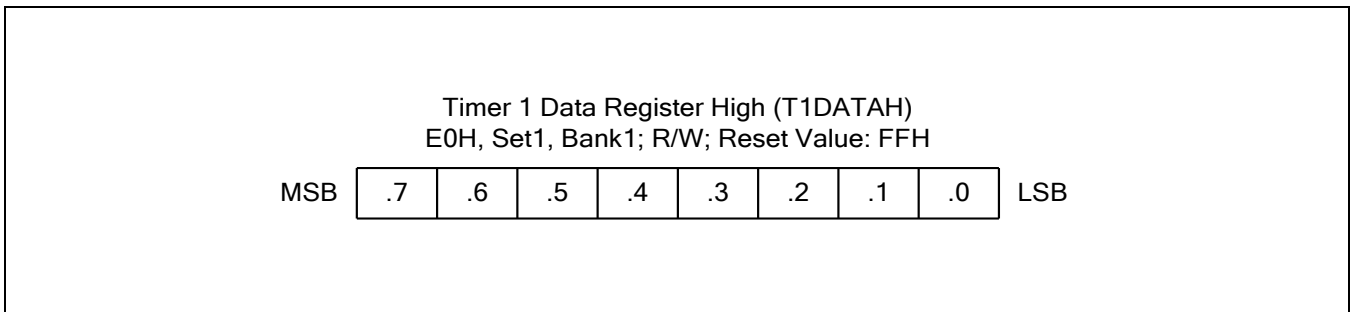
A reset clears T1PS to "00H". This selects the clock frequency of Timer 1 as FLCLK. The clock prescaler value of T1PS should be kept to not larger than 12, the values larger than 12 is not valid.



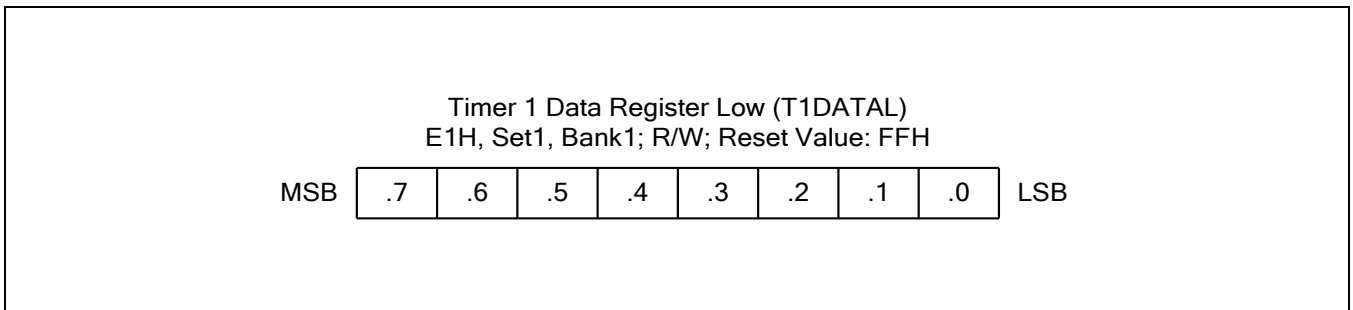
**Figure 11-3 Timer 1 Control Register (T1CON)**



**Figure 11-4 Timer 1 Prescaler Register (T1PS)**



**Figure 11-5 Timer 1 Data Register High (T1DATAH)**



**Figure 11-6 Timer 1 Data Register Low (T1DATAL)**

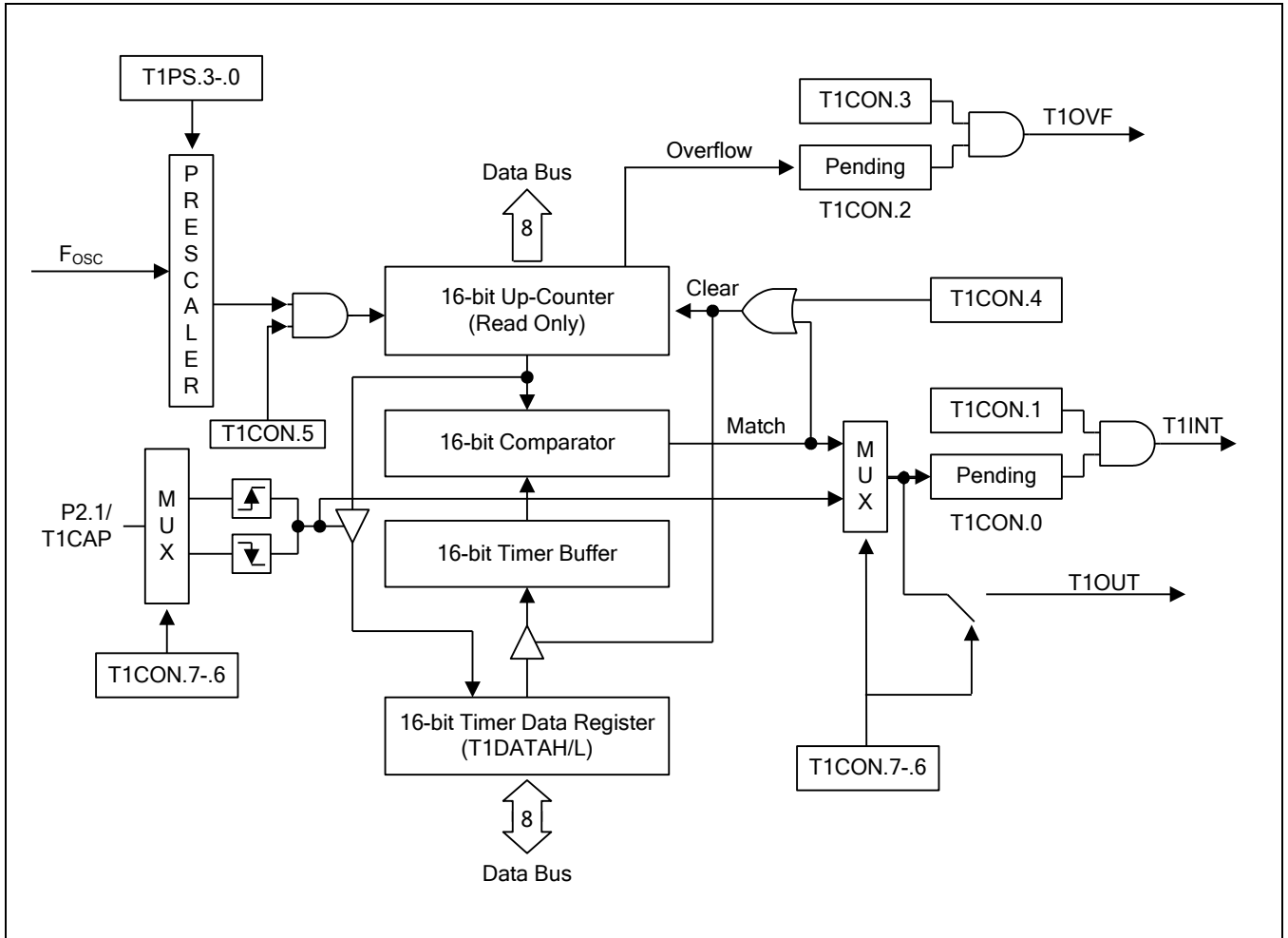


Figure 11-7 Timer 1 Functional Block Diagram



Example 11-1 Using the Timer 1

```

ORG    003Ch
VECTOR0ECh, INT_Timer1_match
ORG    0100h

INITIAL:
    LD    SYM,#00h           ; Disable Global/Fast interrupt
    LD    SP,#0COH         ; Set stack area
    LD    BTCON, #10100011b ; Disable Watch-dog
    LD    T1DATAH, #00H
    LD    T1DATAL, #0F0H
    LD    T1CON, #00100010b ; Interval, timer start run; clear counter, Enable interrupt
    LD    T1PS, #00000100b ; Timer 1 clock = FOSC/16
    EI

MAIN:
    •
    •
    •
    MAIN ROUTINE
    •
    •
    •

    JR    T, MAIN

INT_Timer1_match:
    •
    •
    •
    Interrupt service routine
    •
    •
    •

    IRET

    END
  
```

# 12 Watchdog Timer

## 12.1 Overview

The S3F8S28/S3F8S24 has an free running enhanced 11-bit Watchdog Timer (WDT), the main features are:

- Selectable clock: system clock or free running ring oscillator (by ROSCCON.6)
- Interrupt generation if INTEN is enable
- Overflow reset generation if RSTEN is enabled
- Selectable clock divider

The Watchdog Timer (WDT) is a timer counting cycles of the Low Power Ring Oscillator or system clock. The WDT gives an pending interrupt when the counter reach to 0x3FF (half of the full counter value) or a system reset when the counter overflow. In normal operation mode, it is required that the system writing "1" to bit 4 of WDTCON to clear bit 10 of the counter before the time-out value is reached. If the system doesn't clear the counter, a system reset will be issued if the overflow Reset (RSTEN) is enabled.

## 12.2 Function Description

### 12.2.1 Watchdog Interrupt

The WDT can work as a general system timer that gives an interrupt when the 11-bit counter reach to 0x3FF (the half of the full counter value). One example is to limit the maximum time allowed for certain operations, giving an interrupt when the operation has run longer than expected.

### 12.2.2 Release Stop

The Watchdog Timer interrupt can be used to wake up the device from Stop Mode when the watchdog timer is clocked by free running Ring Oscillator.

### 12.2.3 System Reset

The WDT gives a reset when the 11-bit counter expires if the RSTEN is set. This is typically used to prevent system hang-up in case of runaway code.

In order to prevent an internal reset (if RSTEN bit is set), the software must clear the half of the counter before it reaches 0x7FF by writing "1" to bit 4 of Watchdog Control Register (WDTCON).

There is a possibility to set a pending window where users can restart the watchdog counter within this window. When the interrupt occurred, User can clear the counter to prevent the internal reset. If the reset is needed, User still can save some critical parameters before a system reset. This is useful for allowing a safe shutdown.

## 12.3 Watchdog Timer Control Register (WDTCON)

You use the Watchdog Timer Control Register, WDTCON, to

- Enable/Disable Watchdog Timer
- Interrupt Enable/Disable
- Overflow Reset Enable/Disable
- Counter clear
- Program the clock prescaler for watchdog timer

WDTCON is located at address E6H, Bank 0, Set 1, and is read/write addressable using Register addressing mode. A reset clears WDTCON to "00H". This disable Watchdog Timer, disable Watchdog interrupt and Reset.

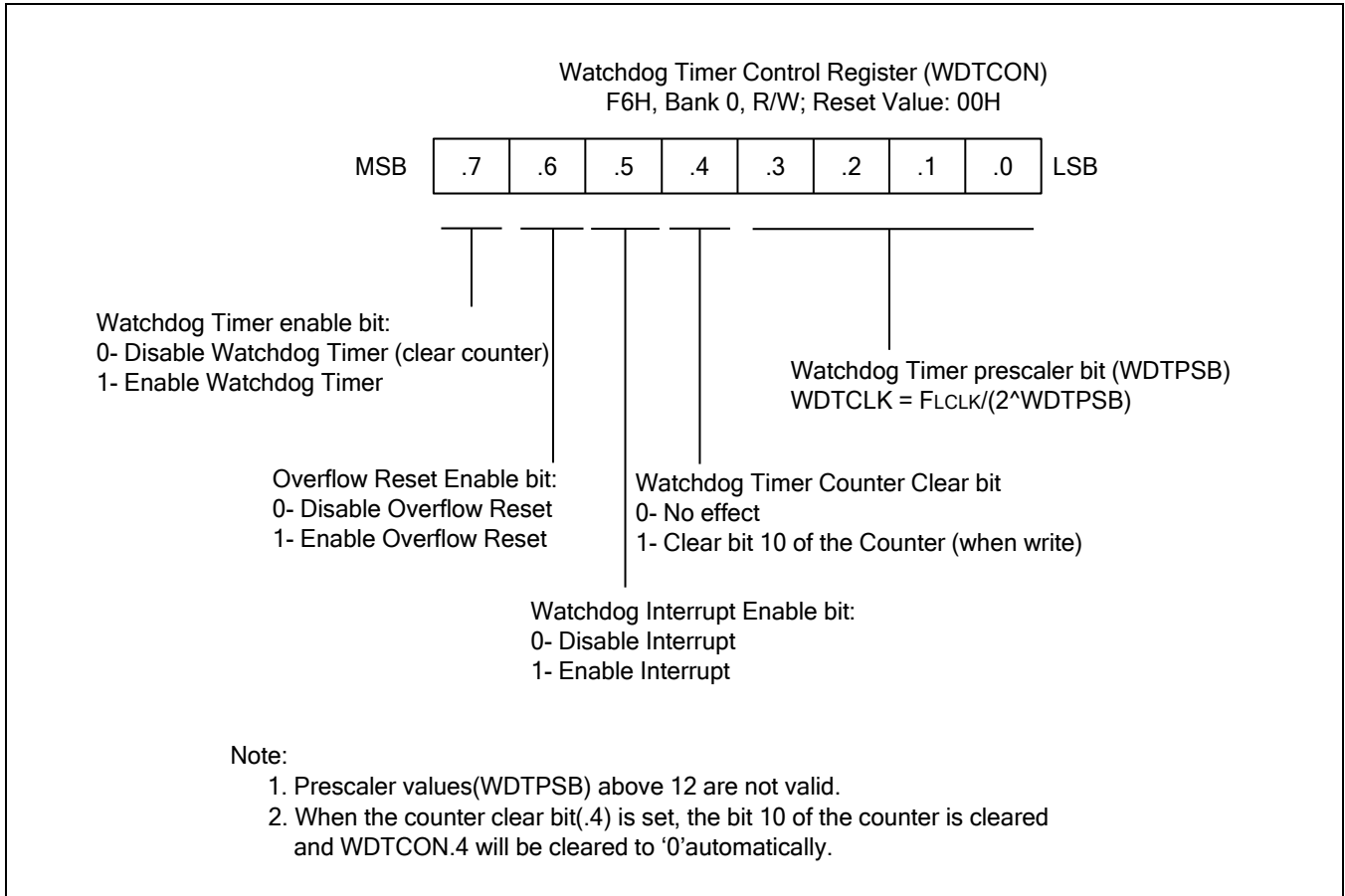
You can enable Watchdog Timer by writing a "1" to WDTCON.7. Writing a "1" to the enable bit clear the counter and restart to counting at any time.

Watchdog Overflow Reset is controlled by WDTCON.6, if it is set, when the counter overflow, the system reset will occur. Watchdog Timer's interrupt is controlled by WDTCON.5, if it is set, the watchdog timer interrupt will generated when counter reach to 0x3FF.

You can clear the Watchdog Timer counter bit 10 of the counter at any time during normal operation by writing a "1" to WDTCON.4.

WDTCON.3-0 are used to set select the clock prescaler of Watchdog Timer. The clock prescaler value of WDTPS should be kept to not larger than 12, the values larger than 12 is not valid.

The clock source of Watchdog Timer is selected by ROSCCON.6 to choose system clock or Ring Oscillator as the clock for Watchdog Timer.



**Figure 12-1 Watchdog Timer Control Register (WDTCN)**

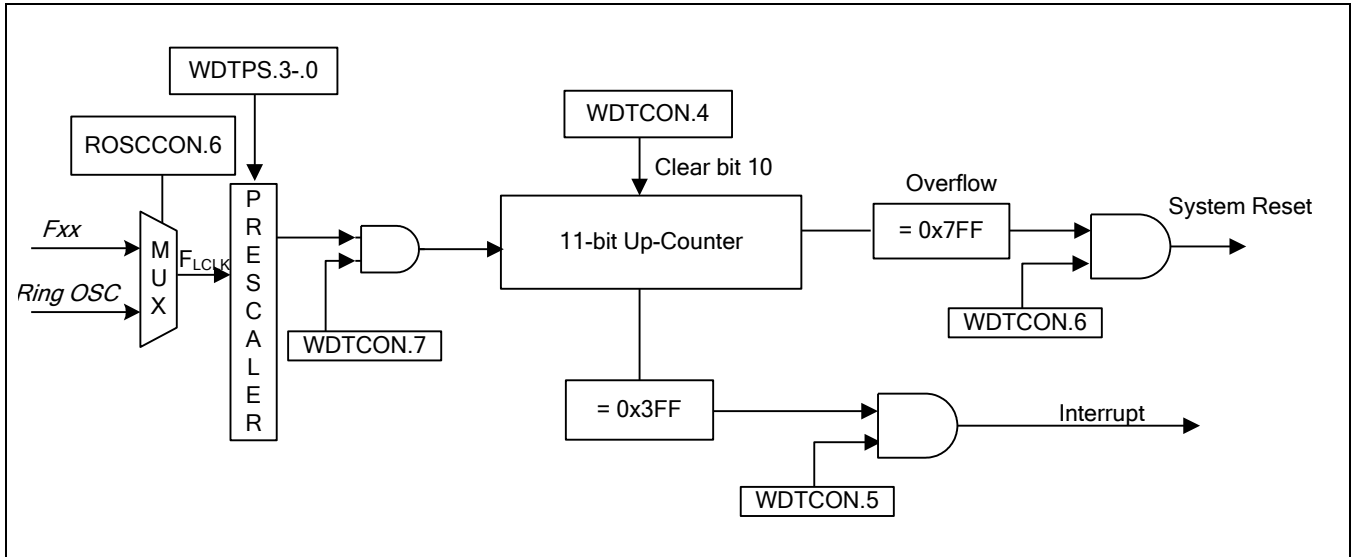


Figure 12-2 Watchdog Timer Functional Block Diagram

Table 12-1 Watchdog Timer Presaler Select

WDTCON.3	WDTCON.2	WDTCON.1	WDTCON.0	Number of 32kHz Ring OSC Cycles	Typical Time-out
0	0	0	0	2048	32ms
0	0	0	1	4096	64ms
0	0	1	0	8109	128ms
0	0	1	1	16384	256ms
0	1	0	0	32768	512ms
0	1	0	1	65536	1024ms
0	1	1	0	131072	2048ms
0	1	1	1	262144	4096ms
1	0	0	0	524288	8192ms
1	0	0	1	1048576	16384ms
1	0	1	0	2097152	32768ms
1	0	1	1	10485760	65536ms
1	1	0	0	20971520	131072ms

## 12.4 Interrupt

User can use Watchdog Timer as general timer/counter to generate interval interrupt with programmable period.

Enable the interrupt by setting the WDTCON.5 to "1", If the global interrupt is enabled, after you enable the Watchdog (Set WDTCON.7 to "1"), the 11-bit counter start to counting, when the counter reach to 0x3FF, the interrupt will be generated and the counter will keep counting. If the Overflow Reset is disabled, when the counter overflow, it counter value reset to "0" and continues to up count. This is to be used to generate periodic interrupts. Watchdog Timer interrupt pending bit will be cleared automatically by hardware when the interrupt request is served.

One example is to limit the maximum time allowed for certain operations, giving an interrupt when the operation has run longer than expected.

It can be used to Release Stop Mode with setting the clock source as Ring Oscillator. After enter Stop Mode, the system clock (External crystal or Internal RC OSC) is stopped, but the Ring Oscillator can be set to run to provide clock for Watchdog Timer, when the 11-bit counter of Watchdog Timer interrupt generated, the interrupt will release Stop Mode.

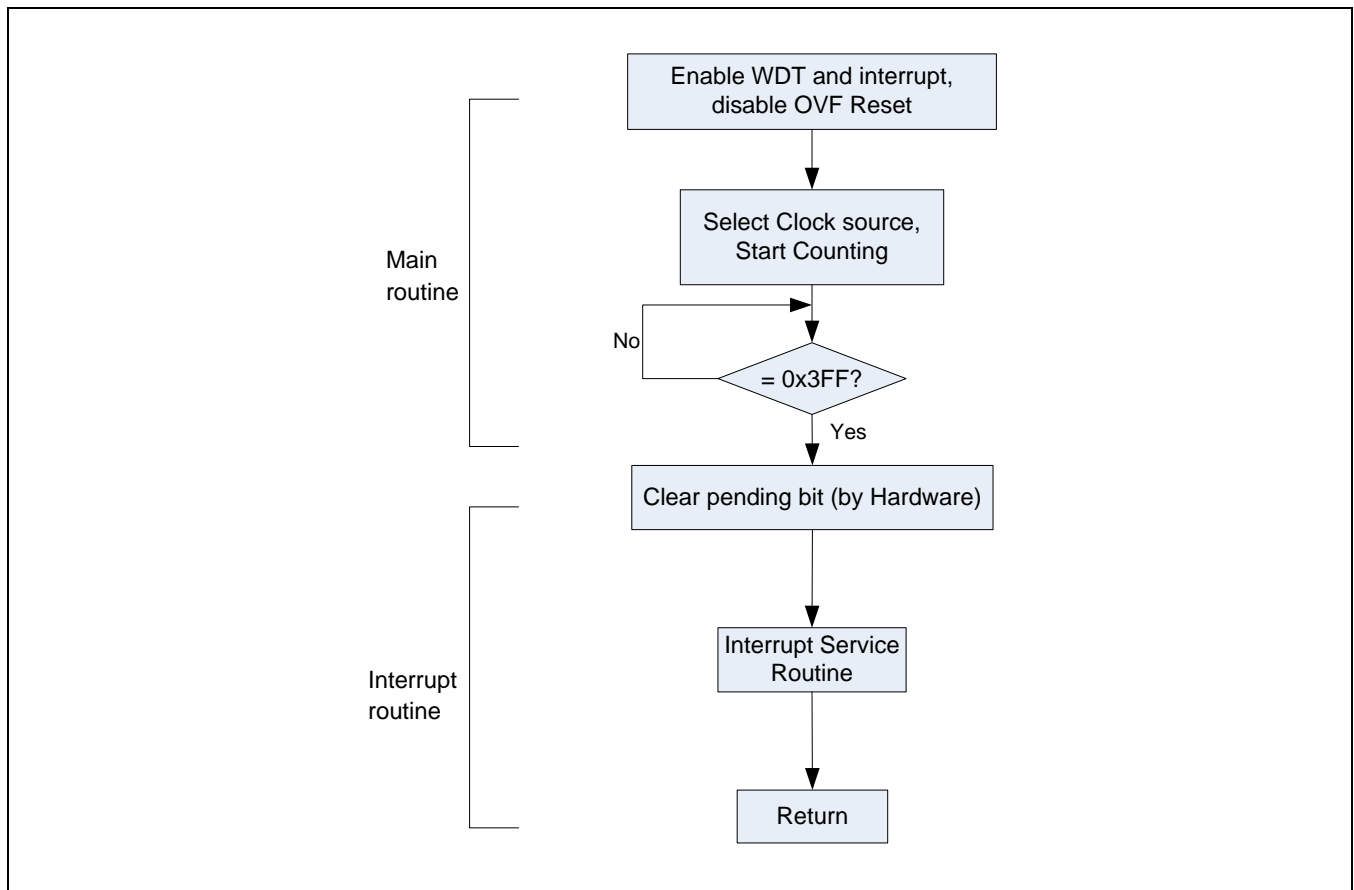


Figure 12-3 Interrupt Operation Sequence

## 12.5 System Reset

Watchdog Timer is typically used to prevent system hang-up in case of runaway code.

That's similar with the basic timer overflow reset, but, as the clock source of Watchdog Timer is selectable, so when it is clocked by Ring Oscillator, it can be used in Stop Mode to reset chip, it is useful when the main system is stopped and the basic timer overflow reset is not available.

The typical time-out period of the Watchdog Timer is listed as [Table 12-1](#). It similar with a 10-bit Basic Timer's watchdog function, user should clear the counter (write "1" to WDTCON.4) for prevent system reset.

When the WDTCON.6 is set to "1", after you enable the Watchdog Timer to start counting, the chip will be reset immediately at the 11-bit counter overflow.

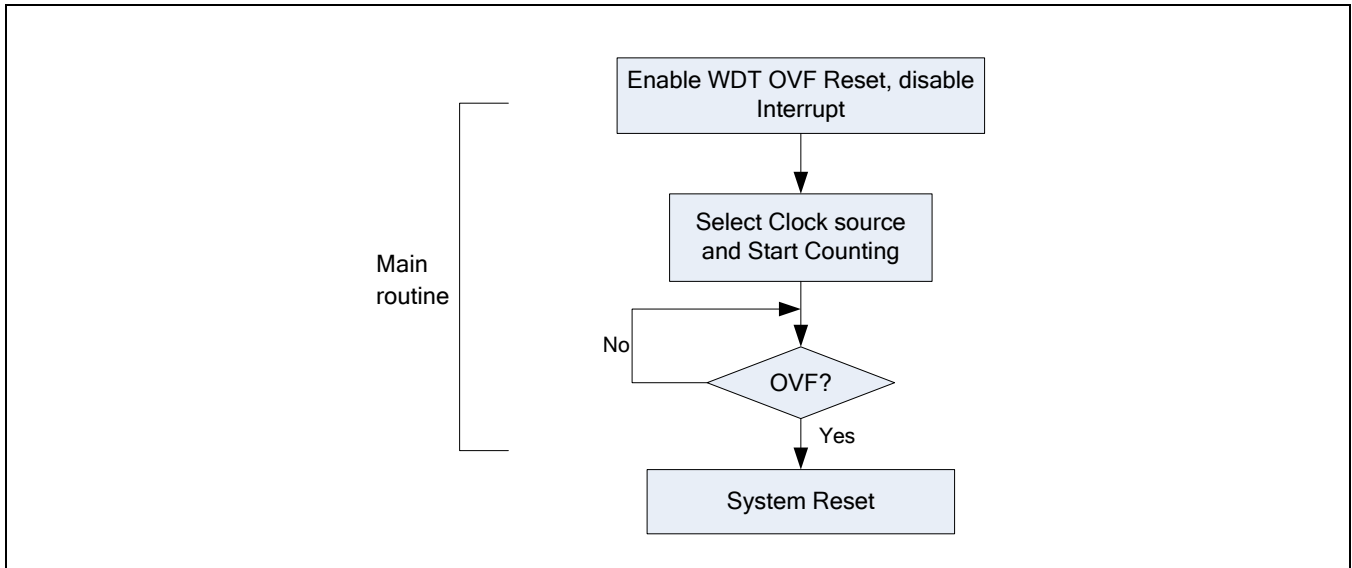


Figure 12-4 System Reset Operation Sequence



## 12.6 Interrupt & System Reset

If all the watchdog interrupt and overflow reset are enabled, and when you enable the watchdog, the counter start to counting, the interrupt will be generated at the counter reach to 0x3FF, and the counter continuous to counting, if the counter overflow, the overflow reset will generated.

This operating mechanism combines the two events by first giving an interrupt and then giving a reset. This will for instance allow a safe shutdown by saving critical parameters before a system reset.

There is a possibility to set a pending window where users can restart the watchdog counter within this window. When the interrupt occurred, user can clear the counter to prevent the internal reset.

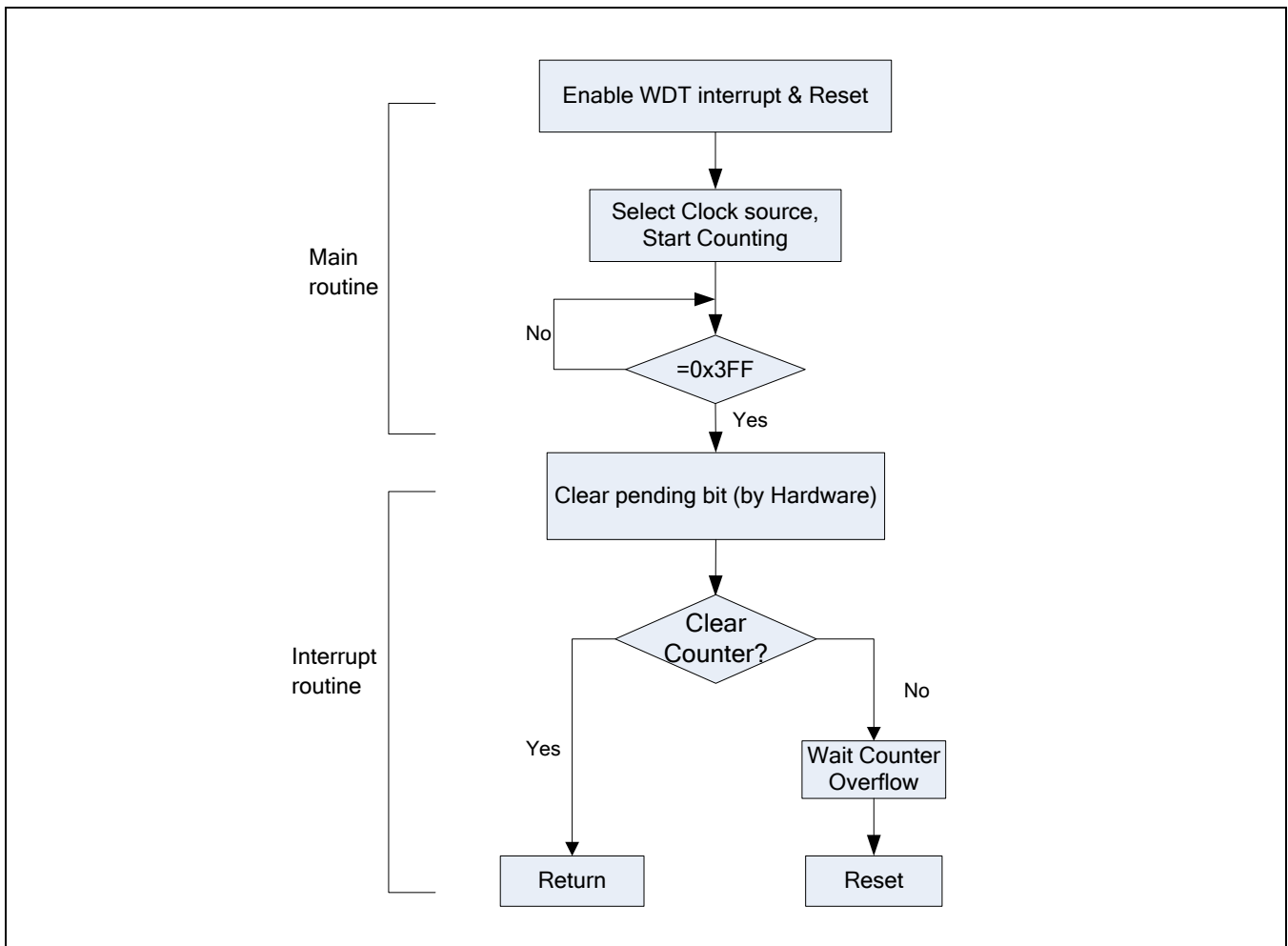


Figure 12-5 Interrupt & System Reset Operation Sequence

# 13 PWM (Pulse Width Modulation)

## 13.1 Overview

This microcontroller has 2 PWM modules: PWM0 & PWM1. These two PWM modules are totally same, the output pin of PWM0 is P0.6, the output pin of PWM1 is P0.5.

The PWM can be configured as one of these three resolutions:

- 12-bit resolution: 6-bit base + 6-bit extension
- 8-bit resolution: 6-bit base + 2-bit extension
- 14-bit resolution: 8-bit base + 6-bit extension

These three resolutions are mutually exclusive; only one resolution can work at any time. And which resolution is used is selected via PWM extension register, PWM0EX.1–0. PWM1EX.1–0.

The operation of all PWM circuits is controlled by the control register, PWM0CON, PWM1CON.

The PWM counter is an incrementing counter. It is used by the PWM circuits. To start the counter and enable the PWM circuits, you set PWM start bit (PWM0CON.2, PWM1CON.2) to "1". If the counter is stopped, it retains its current count value; when restarted, it resumes counting from the retained count value. When there is a need to clear the counter you set the counter clear bit (PWM0CON.3, PWM1CON.3) to "1".

You can select a clock for the PWM counter by set PWM0CON.7–6, PWM1CON.7–6. Clocks which you can select are  $f_{osc}/64$ ,  $f_{osc}/8$ ,  $f_{osc}/2$ ,  $f_{osc}/1$ .

## 13.2 Function Description

### 13.2.1 PWM

The PWM circuits have the following components:

- PWM mode selection (PWM0EX.1–0, PWM1EX.1–0)
- Base comparator and extension cycle circuit
- Base reference data registers (PWM0DATA, PWM1DATA)
- Extension data registers (PWM0EX.7–2, PWM1EX.7–2)
- PWM output pins (P0.6/PWM0; P0.5/PWM1)

### 13.2.2 PWM Counter

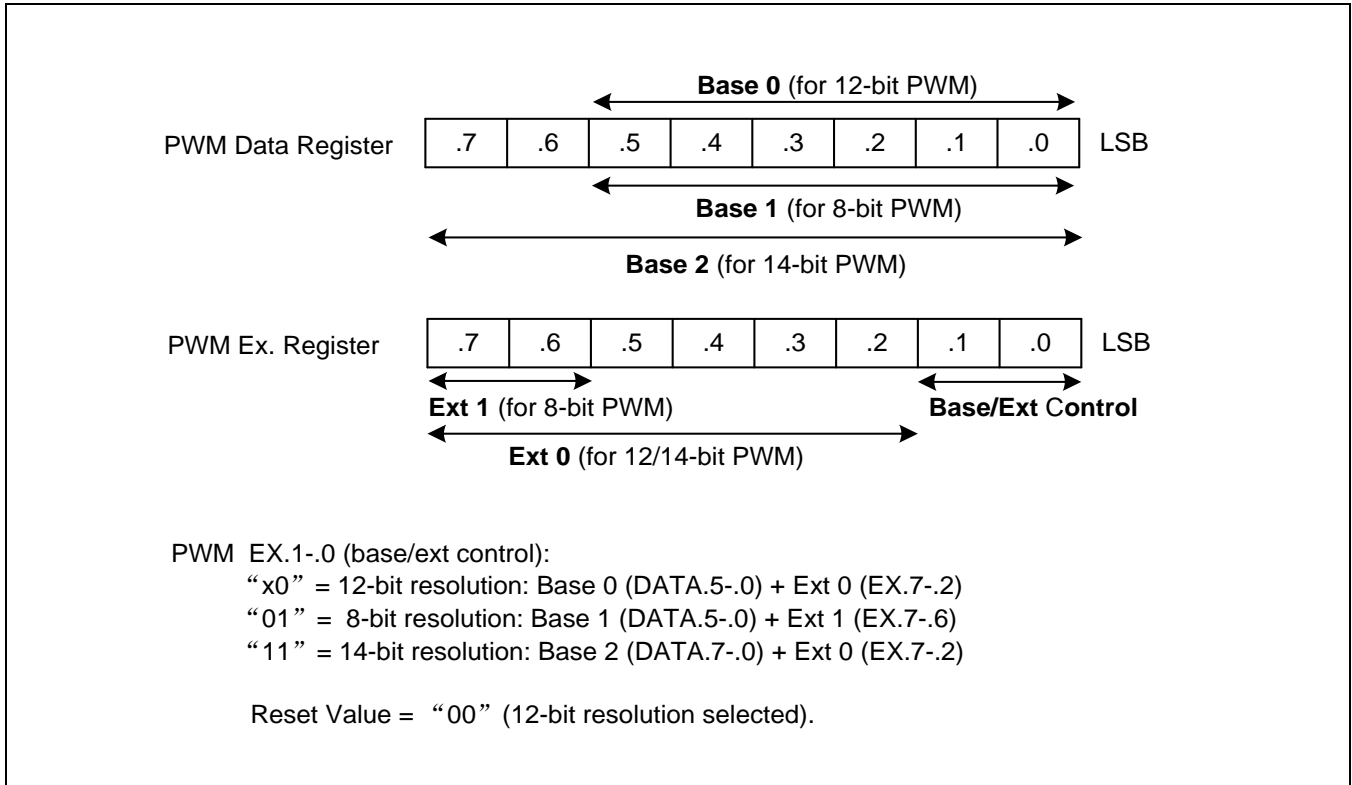
The PWM counter is an incrementing counter comprised of a lower base counter and an upper extension counter.

To determine the PWM module's base operating frequency, the lower base counter is compared to the PWM base data register value. In order to achieve higher resolutions, the extension 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 extended counter value is compared with the value that you write to the module's extension bits.

### 13.2.3 PWM Data and Extension Registers

PWM (duty) data consist of base data bits and extension data bits; determine the output value generated by the PWM circuit. For each PWM resolution, the location of base data bits and extension data bits are different combination of PWM data register and PWM extension register:

- 12-bit resolution, 6-bit base + 6-bit extension:
  - Base 6 data bits: PWM0DATA.5–0, PWM1DATA.5–0
  - Extension 6 bits: PWM0EX.7–2, PWM1EX.7–2
- 8-bit resolution, 6-bit base + 2-bit extension:
  - Base 6 data bits: PWM0DATA.5–0, PWM0DATA.5–0
  - Extension 2 bits: PWM1EX.7–6, PWM1EX.7–6
- 14-bit resolution, 8-bit base + 6-bit extension:
  - Base 8 data bits: PWM0DATA.7–0, PWM0DATA.7–0
  - Extension 6 bits: PWM1EX.7–2, PWM1EX.7–2



**Figure 13-1 PWM Data and Extension Registers**

To program the required PWM output, you load the appropriate initialization values into the data registers (PWM0DATA, PWM1DATA) and the extension registers (PWM0EX, PWM1EX). To start the PWM counter, or to resume counting, you set the PWM counter enable bit (PWM0CON.2, PWM1CON.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.

### 13.2.4 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 input clock setting bits: PWM0CON.6–7, PWM1CON.6–7.

**Table 13-1 PWM Control and Data Registers**

Register Name	Mnemonic	Address	Function
PWM data registers	PWM0DATA	F2H, Bank 0	PWM waveform Ia output setting registers.
	PWM1DATA	E7H, Bank 1	
PWM control registers	PWM0EX	F1H, Bank 0	PWM counter stop/start (resume), and $f_{OSC}$ clock settings
	PWM1EX	E6H, Bank 1	
	PWM0CON	F3H, Bank 0	
	PWM1CON	E8H, Bank 1	

### 13.2.5 PWM Function Description

The PWM output signal toggles to Low level whenever the lower base counter matches the reference value stored in the module's data register (PWM0DATA, PWM1DATA). If the value in the data 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 extension counter is compared with the extension settings in the extension data bits. This extension counter value, together with extension logic and the PWM module's extension bits, 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 13-2](#)).

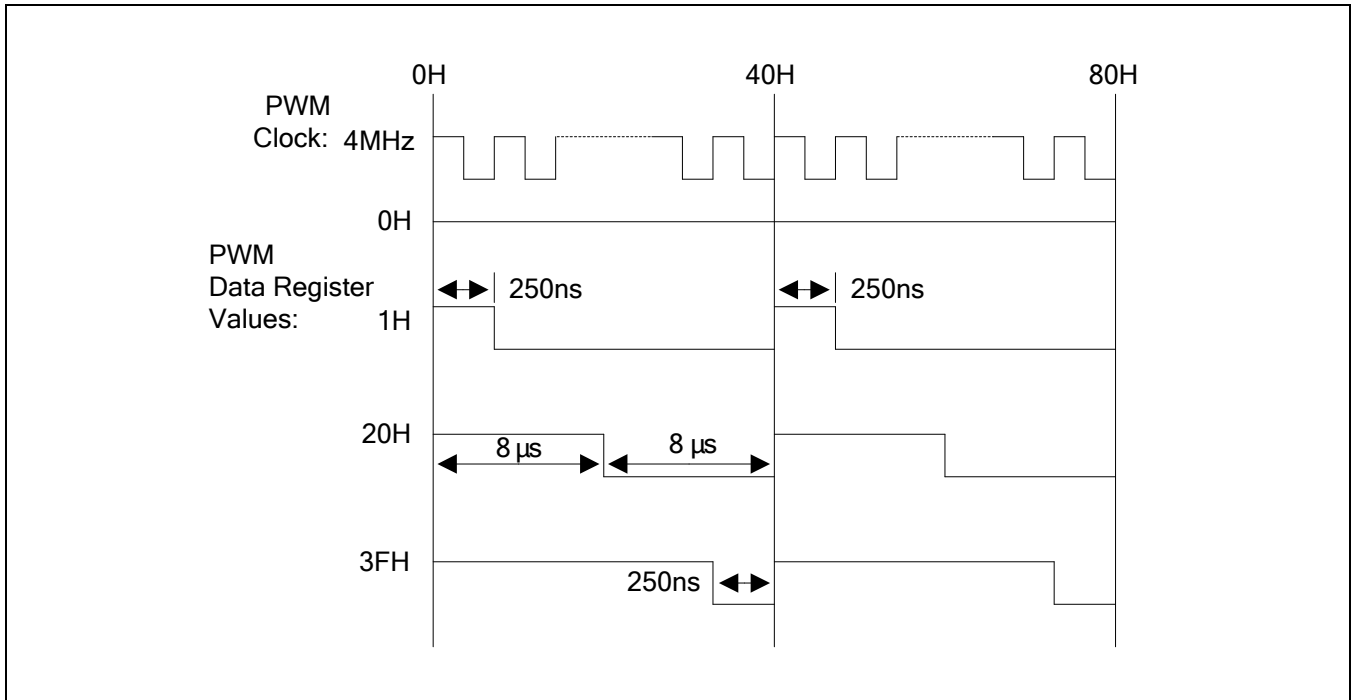
If, for example, in 8-bit base + 6-bit extension mode, the value in the extension register is "04H", the 32nd cycle will be one pulse longer than the other 63 cycles. If the base duty cycle is 50%, the duty of the 32nd cycle will therefore be "stretched" to approximately 51% duty. For example, if you write 80H to the extension register, all odd-numbered cycles will be one pulse longer. If you write FCH to the extension register, all cycles will be stretched by one pulse except the 64th cycle. 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.

**13.2.6 PWM Output Waveform**

6-bit base + 6-bit extension mode:

**Table 13-2 PWM Output "Stretch" Values for Extension Data Bits Ext1 (PWM0EX.7–2, PWM1EX.7–2)**

PWM EX. 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



**Figure 13-2 PWM Basic Waveform (6-Bit Base)**

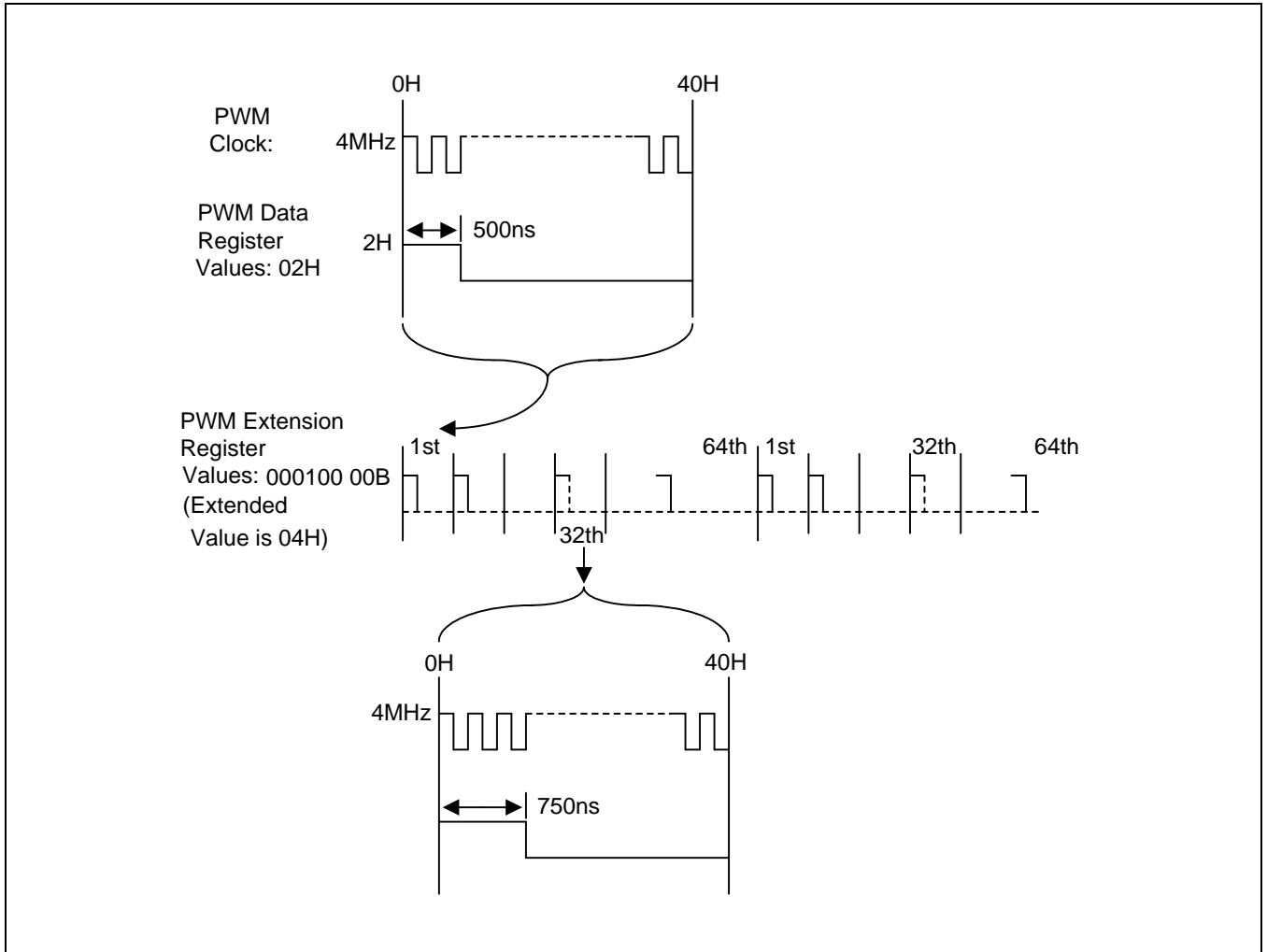
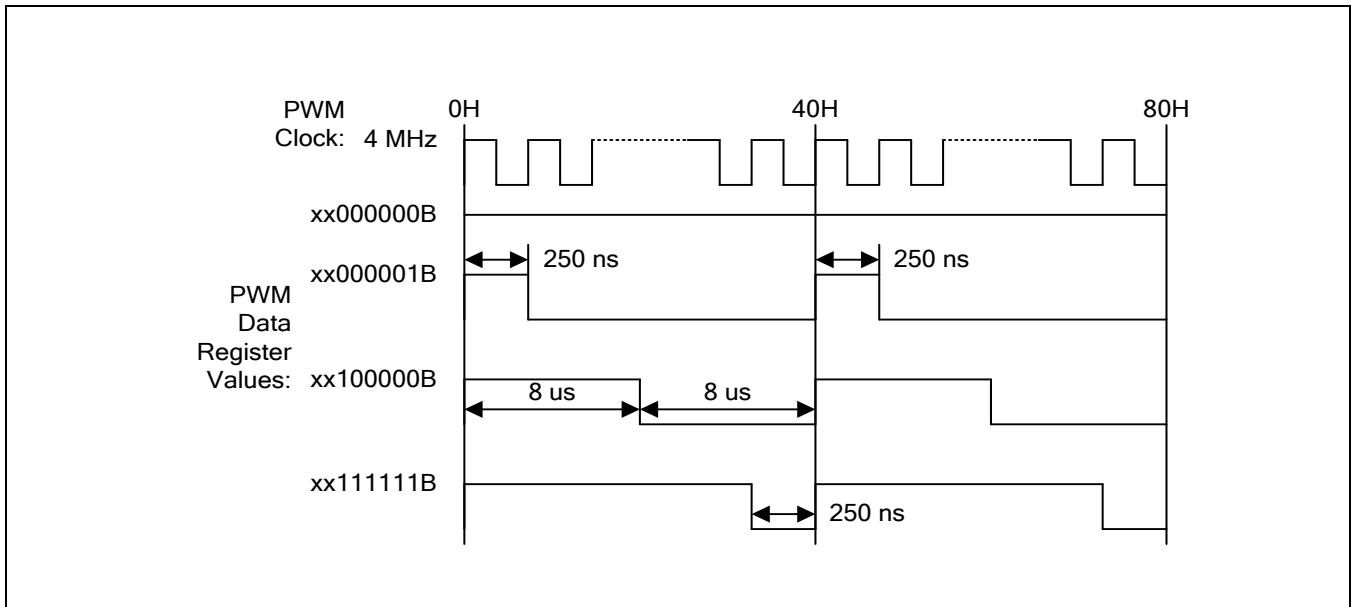


Figure 13-3 Extended PWM Waveform (6-Bit Base + 6-Bit Extension)

6-bit base + 2-bit extension mode:

**Table 13-3 PWM Output "Stretch" Values for Extension Data Bits Ext0 (PWM0EX.7–6, PWM1EX.7–6)**

PWM Ex. Bit (7-Bit 6-Bit)	"Stretched" Cycle Number
00	–
01	2
10	1, 3
11	1, 2, 3



**Figure 13-4 PWM Basic Waveform (6-Bit Base)**



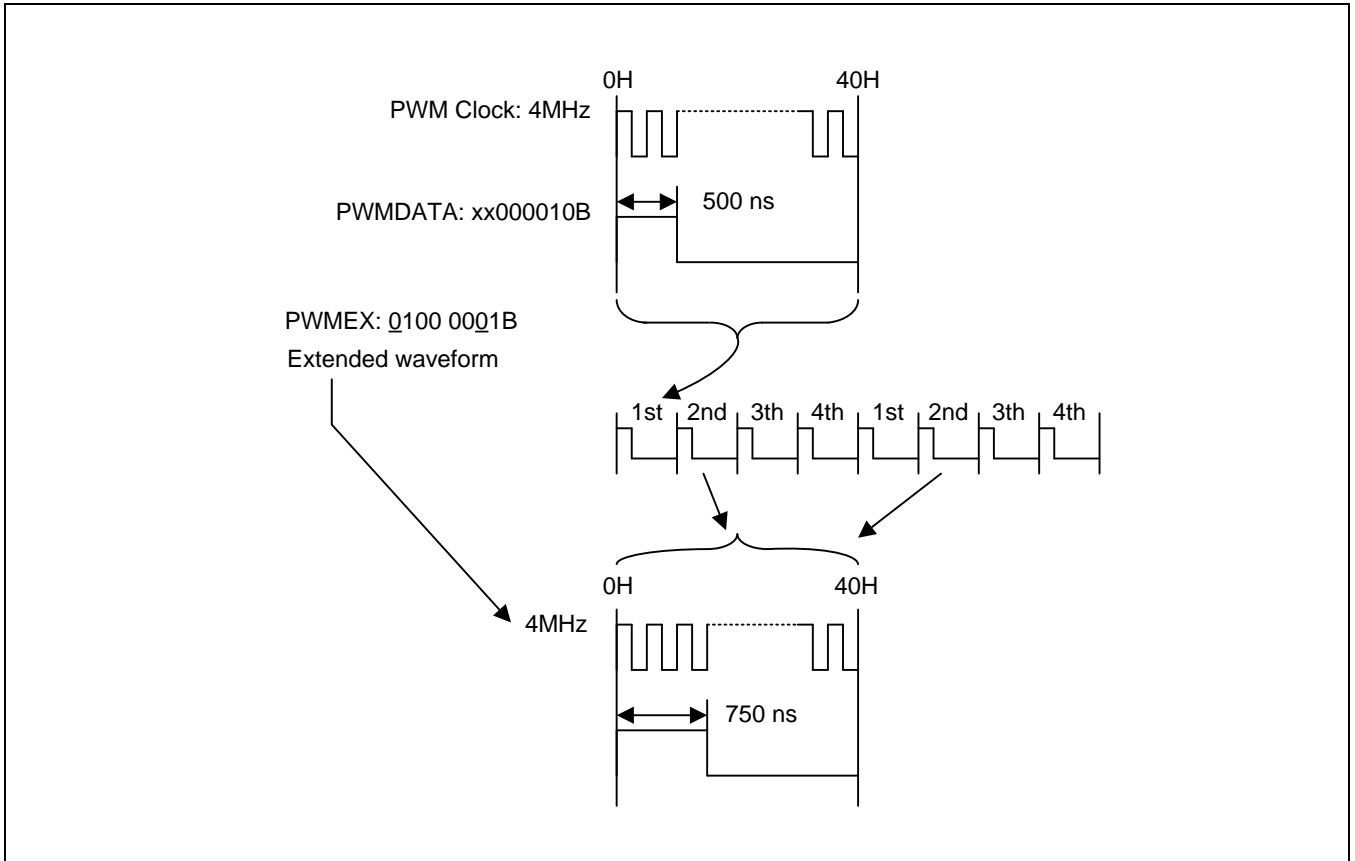
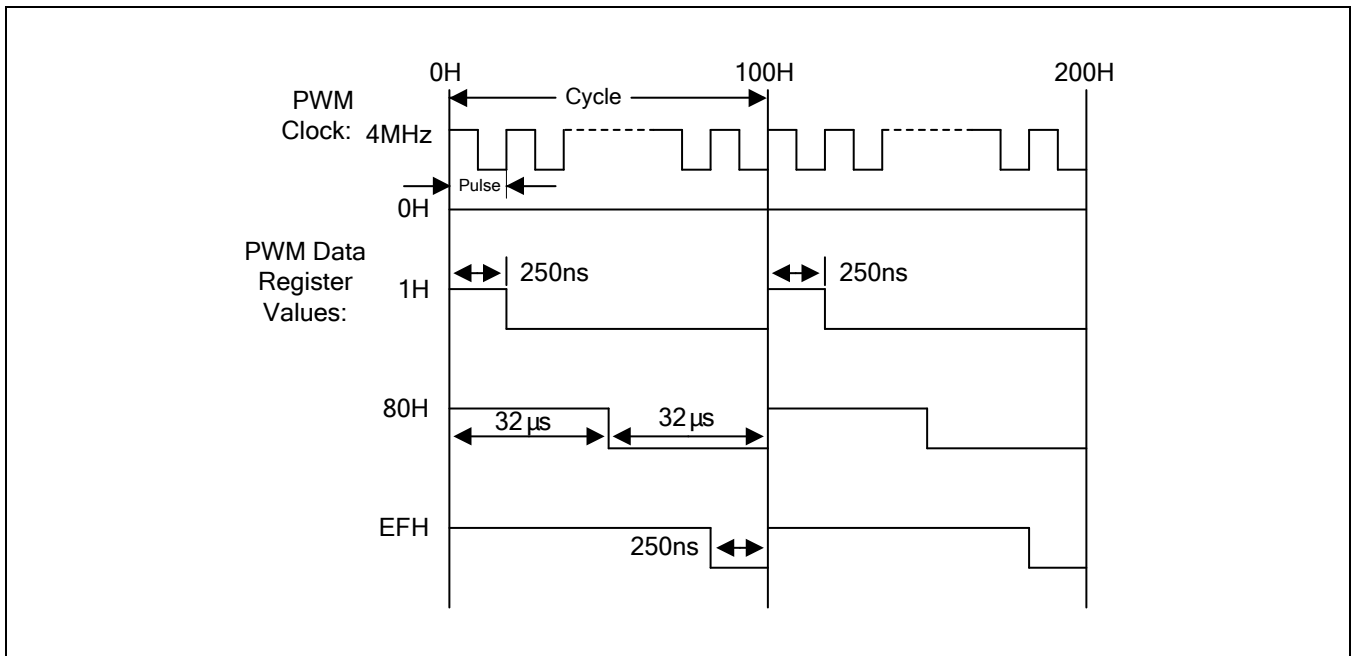


Figure 13-5 Extended PWM Waveform (6-Bit Base + 2-Bit Extension)

8-bit base + 6-bit extension mode:

**Table 13-4 PWM Output "Stretch" Values for Extension Data Bits Ext1 (PWM0EX.7–2, PWM1EX.7–2)**

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



**Figure 13-6 PWM Basic Waveform (8-Bit Base)**

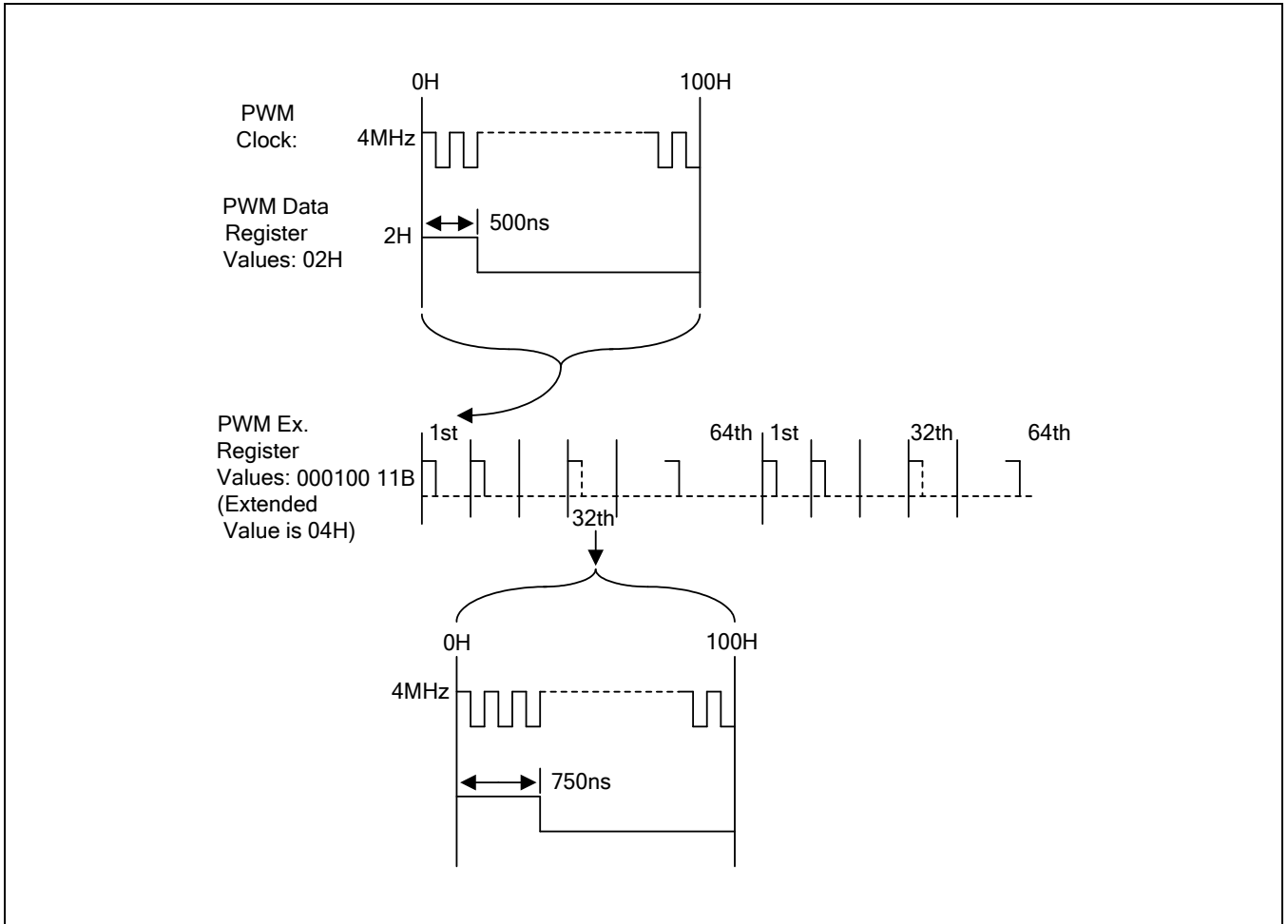


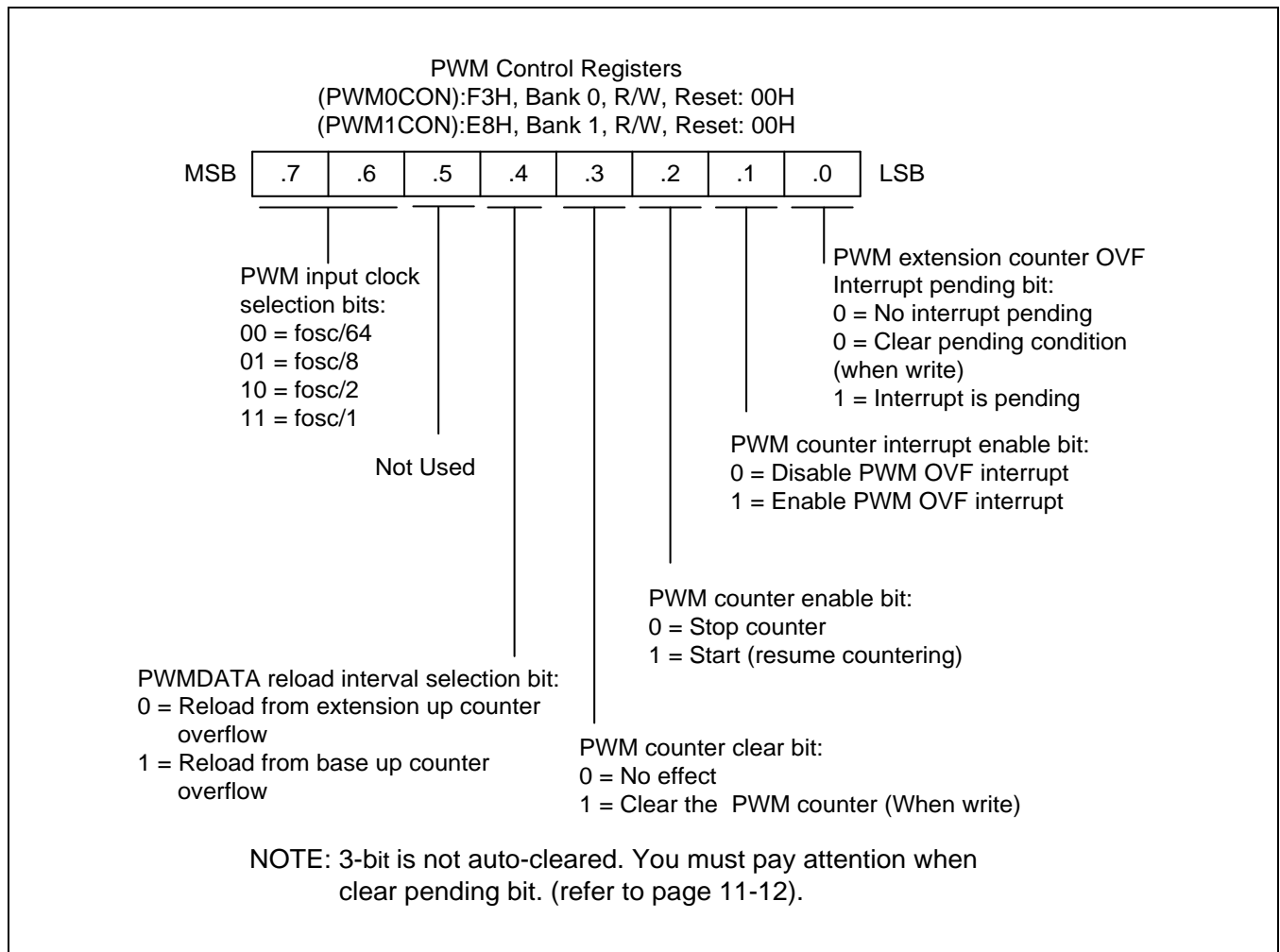
Figure 13-7 PWM Basic Waveform (8-Bit Base + 6-Bit Extension)

### 13.3 PWM Control Register (PWM0CON/PWM1CON)

The control register for the PWM modules, PWM0CON and PWM1CON, are located at register address F3H, Bank 0 and E8H, Bank 1. The control register (PWM0CON, PWM1CON) is used for all three PWM resolutions. Bit settings in the 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 counter overflow) interrupt control

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

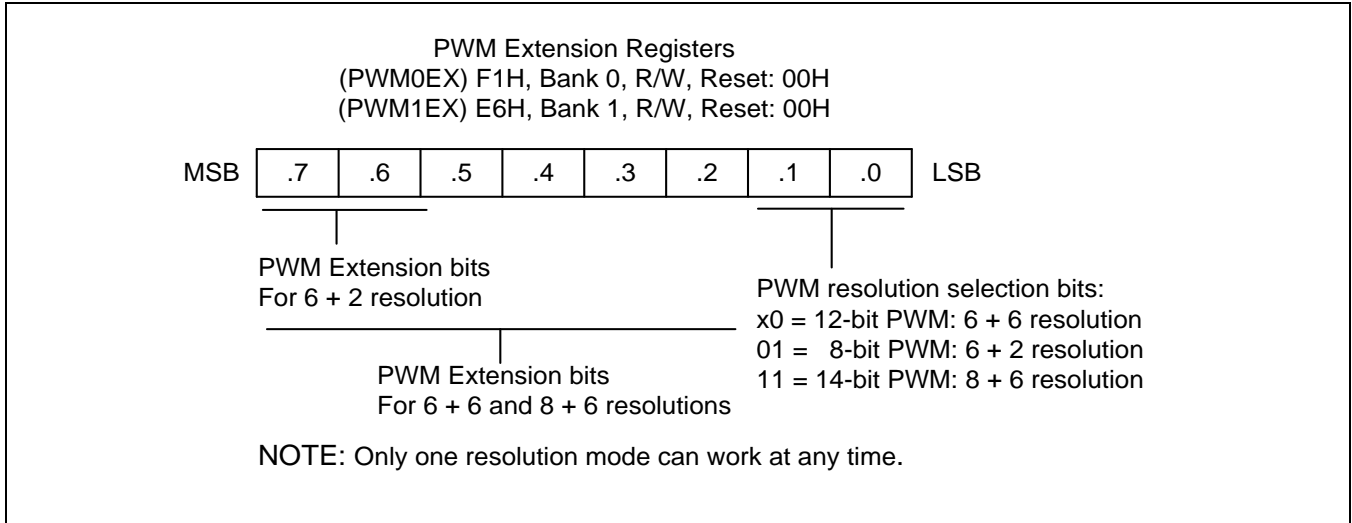


**Figure 13-8 PWM Control Register (PWM0CON, PWM1CON)**

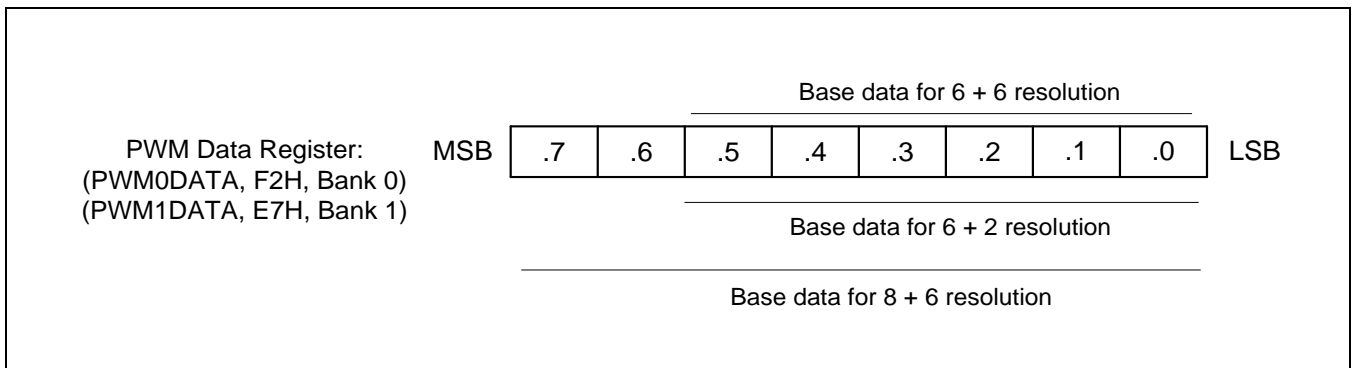
### 13.4 PWM Extension Register (PWM0EX/PWM1EX)

The extension register for the PWM module, PWM0EX and PWM1EX, are located at register address F1H, Bank 0 and E6H, Bank 1. PWM extension register is used for resolution selection and extension bits of PWM waveform. Bit settings in the PWM extension register (PWM0EX, PWM1EX) control the following functions:

- PWM extension bits
- PWM resolution selection
- A reset clears all PWM extension register's to "00H", choose 6 + 2 as default resolution, no extension.



**Figure 13-9 PWM Extension Register (PWM0EX, PWM0EX)**



**Figure 13-10 PWM Data Register (PWM0DATA PWM1DATA)**



**Example 13-1 Programming the PWM Module to Sample Specifications**

```

;-----<< Smart Option >>
      ORG    003CH
      DB     000H           ; 003CH, must be initialized to 1.
      DB     000H           ; 003DH, must be initialized to 1.
      DB     0FFH           ; 003EH, Enable LVR (2.3)
      DB     000H           ; 003FH, External Crystal oscillator

;-----<< Interrupt Vector Address >>
      VECTOR F2H, PWM0_INT      ; S3F8S28/S3F8S24 interrupt vector

;-----<< Initialize System and Peripherals >>
      ORG    0100H
RESET:  DI           ; disable interrupt
      LD     BTCON, #10100011B ; Watchdog disable
      .
      .
      LD     PWM0EX, #00000000B ; Configure PWM0 as 6-bit base +6-bit extension
      LD     POCONH, #10011010B ; Configure P0.6 PWM output
      LD     PWM0CON, #00000110B ; fosc/64, counter/interrupt enable
      AND    PWM0EX, #00000011B ; set extension bits as 00(basic output)

      LD     PWM0DATA, #80H      ;
      .
      .
      EI           ; Enable interrupt

;-----<< Main loop >>
MAIN:
      .
      .
      .
      .
      JR     t, MAIN           ;

PWM0_INT:
      .
      .
      .
      AND    PWM0CON, #11110110B ; pending bit clear
      IRET          ;
      .
      .
      END
  
```

# 14 A/D Converter

## 14.1 Overview

The 12-bit A/D converter (ADC) module uses successive approximation logic to convert analog levels entering at one of the nine input channels to equivalent 12-bit digital values. The analog input level must lie between the VDD and VSS values. The A/D converter has the following components:

- Analog comparator with successive approximation logic
- Sample and Hold circuit
- D/A converter logic
- ADC control register (ADCON)
- Thirteen multiplexed analog data input pins (ADC0 to ADC12)
- 12-bit A/D conversion data output register (ADDATAH/L)

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion.

To enable and 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 (ADC<sub>n</sub>, n = 0–12) and set the conversion start 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 800H (the approximate half-way point of an 12-bit register). This register is then updated automatically during each conversion step. The successive approximation block performs 12-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 the enable bit, ADCON.0. When a conversion is completed, ADCON.3, the end-of-conversion (EOC) bit is automatically set to 1 and the result is dumped into the ADDATA register where it can be read. The A/D converter then enters an idle state. Remember to read the contents of ADDATA before another conversion starts. Otherwise, the previous result will be overwritten by the next conversion result.

**NOTE:** Normally, when a conversion is completed, the A/D converter then enters an idle state and will still work with power consumption. For power saving, when a conversion is completed, you can set the channel selection bit value (ADCON.7–4) to "1111B" to disable the ADC module, then the ADC module will be stopped and without any power consumption.



## 14.2 Using A/D Pins for Standard digital Input

The ADC module's input pins are alternatively used as digital input in port 0, P2.6–P2.4 and P3.0–P3.1.

## 14.3 A/D Converter Control Register (ADCON)

The A/D converter control register, ADCON, is located at address F7H. ADCON has four functions:

- Bits 7 to 4 select an analog input pin (ADC0 to ADC12) and enable/disable ADC module meanwhile.
- Bit 3 indicates the status of the A/D conversion.
- Bits 2 to 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 to ADC12) by manipulating the 4-bit value for ADCON.7 to ADCON.4.

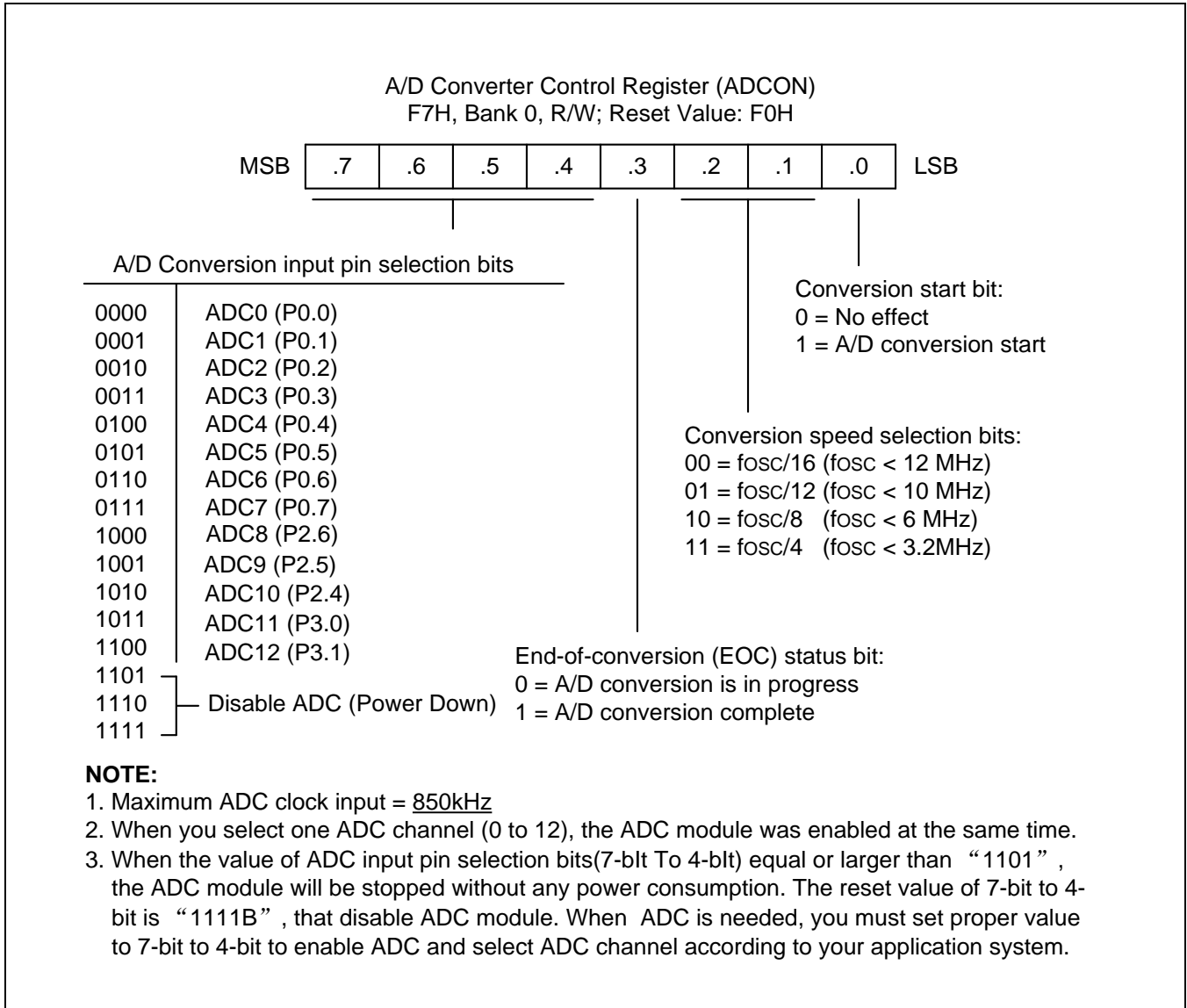


Figure 14-1 A/D Converter Control Register (ADCON)

### 14.4 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  $1/2V_{DD}$ .

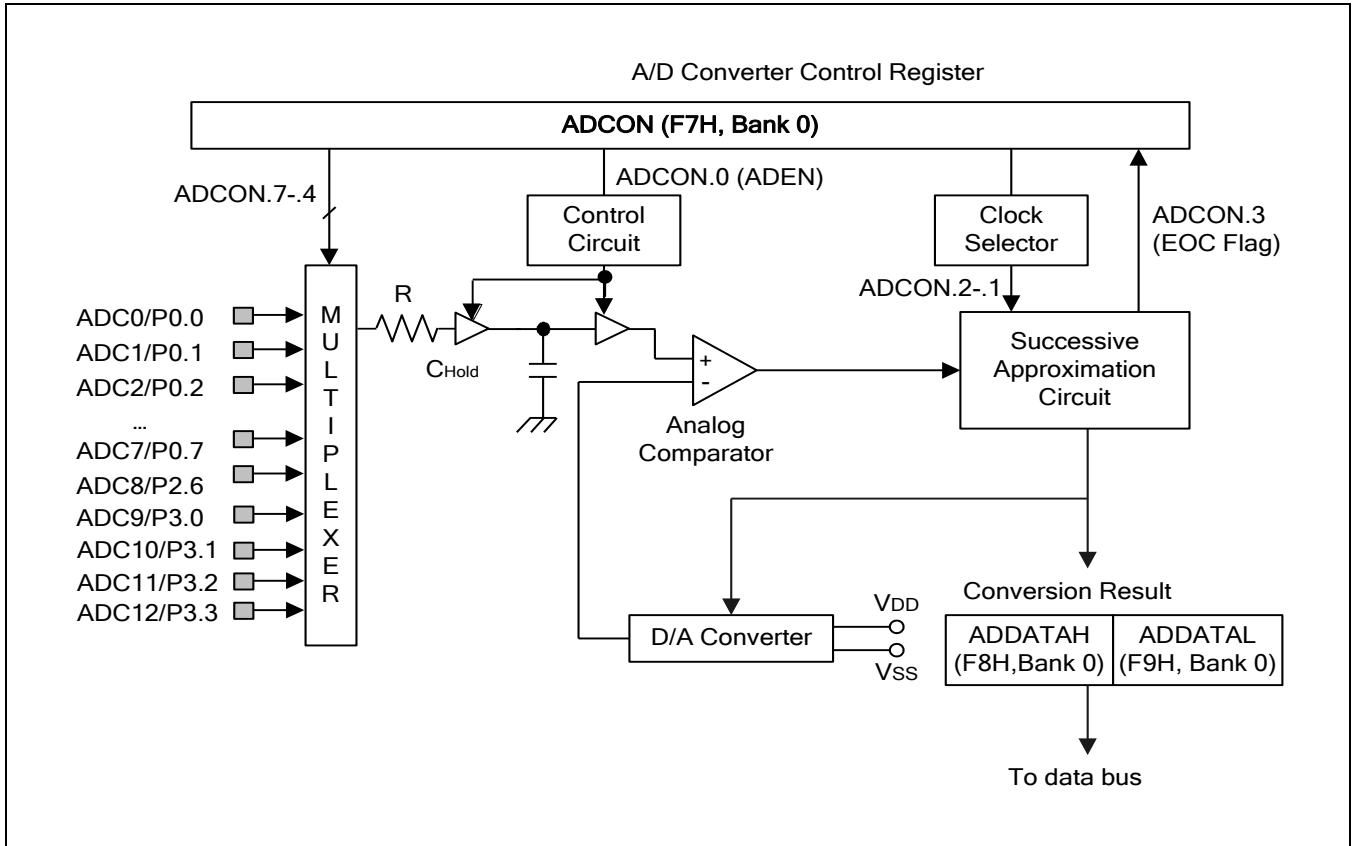


Figure 14-2 A/D Converter Circuit Diagram

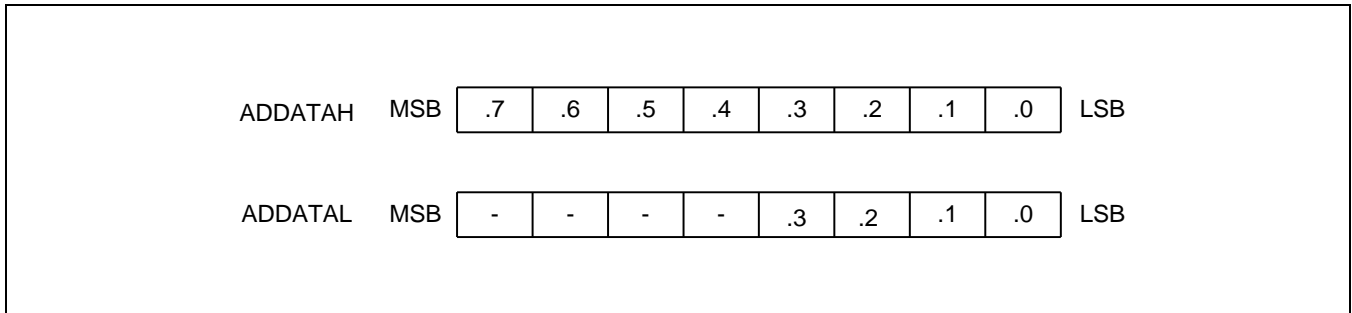


Figure 14-3 A/D Converter Data Register (ADDATAH/L)

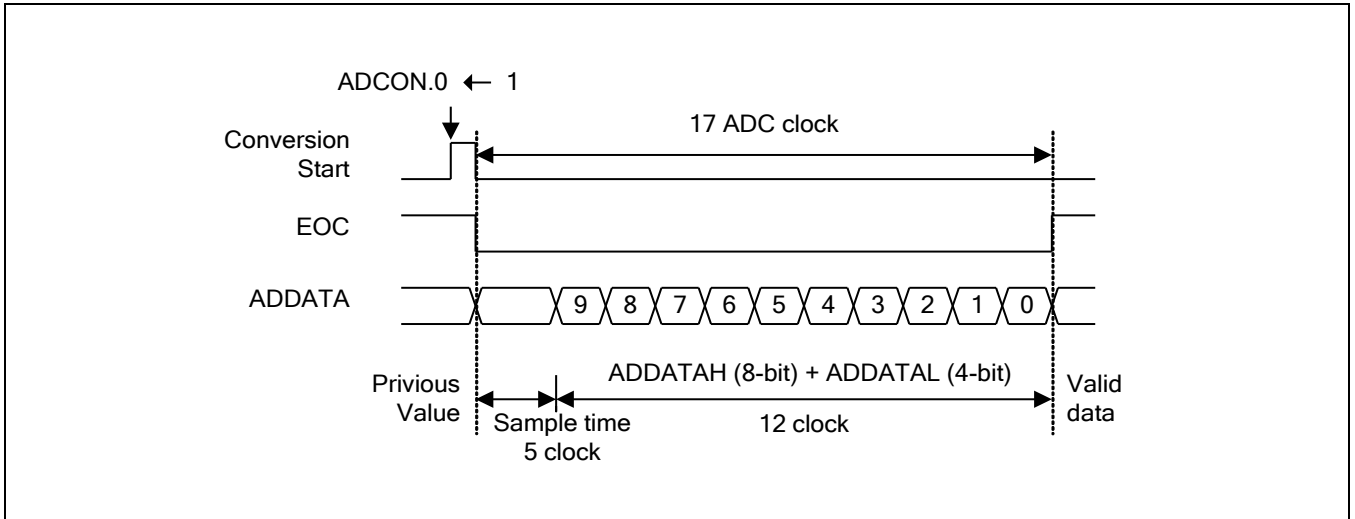


Figure 14-4 A/D Converter Timing Diagram

## 14.5 Conversion timing

The A/D conversion process requires 1 step (1 clock edge) to convert each bit and 5 clocks to step-up A/D conversion. Therefore, total of 17 clocks are required to complete a 12-bit conversion: With an 10MHz CPU clock frequency, one clock cycle is  $1.2\mu\text{s}$  ( $12/f_{\text{OSC}}$ ). If each bit conversion requires 1 clock, the conversion rate is calculated as follows:

- $1 \text{ clock/bit} \times 12\text{-bits} + \text{Sample time (5 clock)} = 17 \text{ clocks}$
- $17 \text{ clock} \times 1.2\mu\text{s} = 20.4\mu\text{s}$  at 10MHz, 1 clock time =  $12/f_{\text{OSC}}$  (assuming  $\text{ADCON.2-1} = 01$ )

## 14.6 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 thirteen input pins (ADC0 to ADC12) by writing the appropriate value to the ADCON register.
4. When conversion has been completed, (17 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 (4-bit), then the ADC module enters an idle state.
6. The digital conversion result can now be read from the ADDATAH and ADDATAL register.

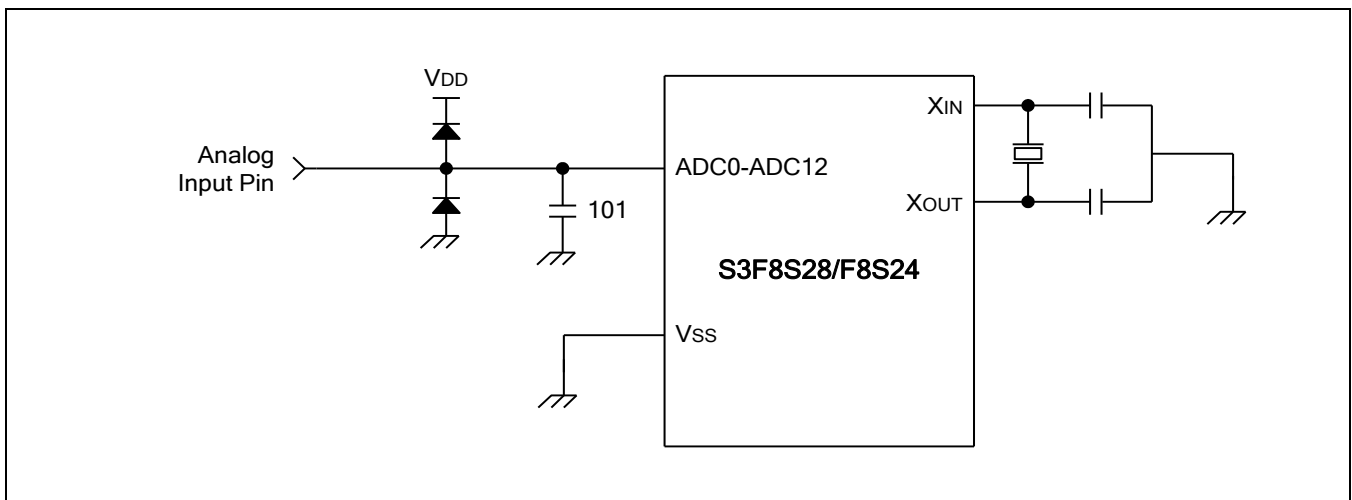


Figure 14-5 Recommended A/D Converter Circuit for Highest Absolute Accuracy

**Example 14-1 Configuring A/D Converter**

```

    ORG    003CH
    DB    0FFH           ; 003CH, must be initialized to 0
    DB    0FFH           ; 003DH, must be initialized to 0
    DB    0FFH           ; 003EH, enable LVR
    DB    0FEH           ; 003FH, external RC oscillator

    VECTOR 0F6H,INT_TIMER0 ; Timer 0 interrupt vector

    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, and enable ADC.
        ; 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

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 4 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
        .

```

```
INT_TIMER0:
    •
    •
    RET                                ;
    •                                ; Interrupt enable bit and pending bit check
    •                                ;
    •                                ; Pending bit clear
    IRET                              ;
    •
    •
    END
```



# 15 UART

## 15.1 Overview

The UART block has a full-duplex serial port with programmable operating modes: There is one synchronous mode and three UART (Universal Asynchronous Receiver/Transmitter) modes:

- Shift Register I/O with baud rate of  $f_{xx}/(16 \times (8\text{-bit BRDATA} + 1))$
- 8-bit UART Mode; variable baud rate,  $f_{xx}/(16 \times (8\text{-bit BRDATA} + 1))$
- 9-bit UART Mode;  $f_{xx}/16$
- 9-bit UART Mode; variable baud rate,  $f_{xx}/(16 \times (8\text{-bit BRDATA} + 1))$

UART receive and transmit buffers are both accessed via the data register, UDATA, is at address F8H, Set 1, Bank 1. Writing to the UART data register loads the transmit buffer; reading the UART data register accesses a physically separate receive buffer.

When accessing a receive data buffer (shift register), reception of the next byte can begin before the previously received byte has been read from the receive register. However, if the first byte has not been read by the time the next byte has been completely received, the first data byte will be lost (Overrun error).

In all operating modes, transmission is started when any instruction (usually a write operation) uses the UDATA register as its destination address. In mode 0, serial data reception starts when the receive interrupt pending bit (UARTPND.1) is "0" and the receive enable bit (UARTCON.4) is "1". In mode 1 and 2, reception starts whenever an incoming start bit ("0") is received and the receive enable bit (UARTCON.4) is set to "1".

### 15.1.1 Programming Procedure

To program the UART modules, follow these basic steps:

1. Configure P2.2 and P2.3 to alternative function (RxD (P2.2), TxD (P2.3)) for UART module by setting the P2CONL register to appropriately value.
2. Load an 8-bit value to the UARTCON control register to properly configure the UART I/O module.
3. For interrupt generation, set the UART interrupt enable bit (UARTCON.1 or UARTCON.0) to "1".
4. When you transmit data to the UART buffer, write transmit data to UDATA, the shift operation starts.
5. When the shift operation (transmit/receive) is completed, UART pending bit (UARTPND.1 or UARTPND.0) is set to "1" and an UART interrupt request is generated

•

### 15.1.2 UART Control Register (UARTCON)

The control register for the UART is called UARTCON at address F5H, Set1 Bank1. It has the following control functions:

- Operating mode and baud rate selection
- Multiprocessor communication and interrupt control
- Serial receive enable/disable control
- 9th data bit location for transmit and receive operations (mode 2)
- UART transmit and receive interrupt control

A reset clears the UARTCON value to "00H". So, if you want to use UART module, you must write appropriate value to UARTCON.

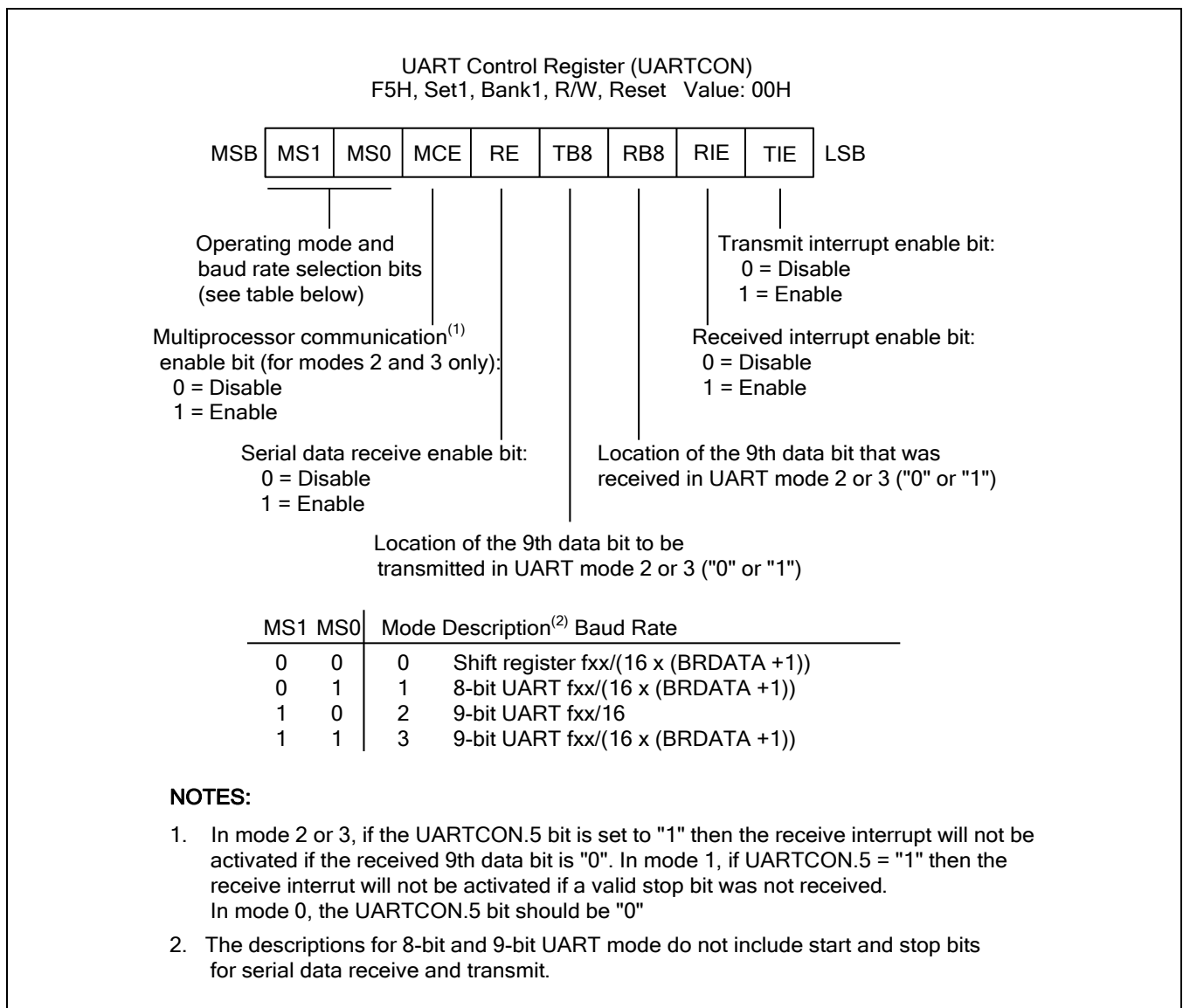


Figure 15-1 UART Control Register (UARTCON)

### 15.1.3 UART Interrupt Pending Register (UARTPND)

The UART interrupt pending register, UARTPND is located at address F6H, Set1 Bank1. It contains the UART data transmit interrupt pending bit (UARTPND.0) and the receive interrupt pending bit (UARTPND.1).

In mode 0 of the UART module, the receive interrupt pending flag UARTPND.1 is set to "1" when the 8th receive data bit has been shifted. In mode 1 or 2, the UARTPND.1 bit is set to "1" at the halfway point of the stop bit's shift time. When the CPU has acknowledged the receive interrupt pending condition, the UARTPND.1 flag must be cleared by software in the interrupt service routine.

In mode 0 of the UART module, the transmit interrupt pending flag UARTPND.0 is set to "1" when the 8th transmit data bit has been shifted. In mode 1 or 2, the UARTPND.0 bit is set at the start of the stop bit. When the CPU has acknowledged the transmit interrupt pending condition, the UARTPND.0 flag must be cleared by software in the interrupt service routine.

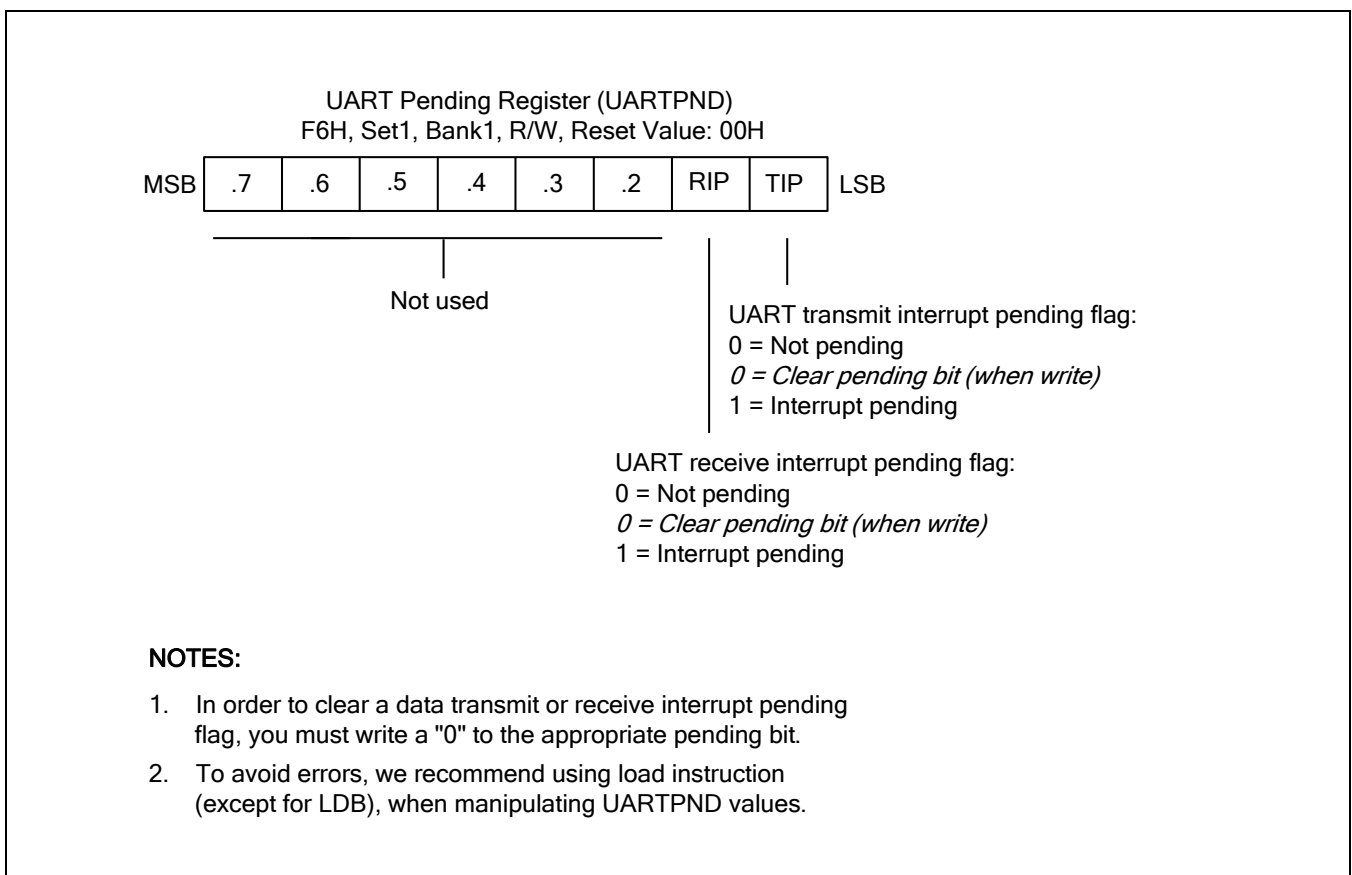
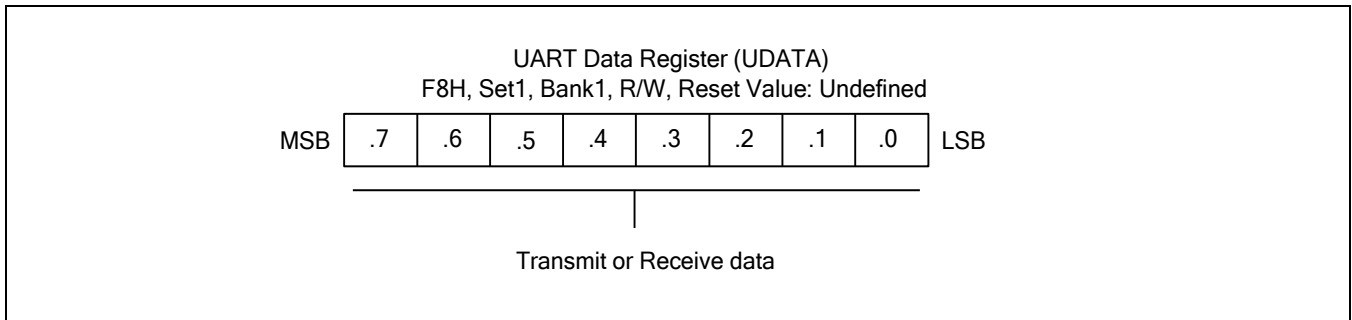


Figure 15-2 UART Interrupt Pending Register (UARTPND)

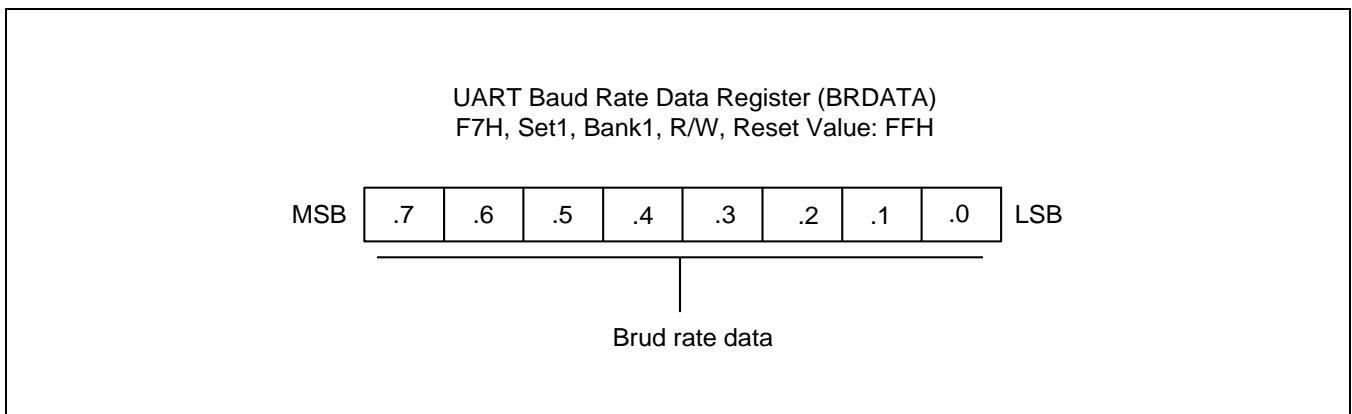
### 15.1.4 UART Data Register (UDATA)



**Figure 15-3 UART Data Register (UDATA)**

### 15.1.5 UART Baud Rate Data Register (BRDATA)

The value stored in the UART baud rate register, (BRDATA), lets you determine the UART clock rate (baud rate).



**Figure 15-4 UART Baud Rate Data Register (BRDATA)**

### 15.1.6 Baud Rate Calculations

The baud rate is determined by the baud rate data register, 8-bit BRDATA

- Mode 0 baud rate =  $f_{xx}/(16 \times (8\text{-bit BRDATA} + 1))$
- Mode 1 baud rate =  $f_{xx}/(16 \times (8\text{-bit BRDATA} + 1))$
- Mode 2 baud rate =  $f_{xx}/16$
- Mode 3 baud rate =  $f_{xx}/(16 \times (8\text{-bit BRDATA} + 1))$

**Table 15-1 Commonly Used Baud Rates Generated by 8-Bit BRDATA**

Mode	Baud Rate	Oscillation Clock	BRDATA	
			Decimal	Hex
Mode 2	0.5MHz	8MHz	x	x
Mode 0	62,500Hz	10MHz	09	09H
Mode 1	9,615Hz	10MHz	64	40H
Mode 3	38,461Hz	8MHz	12	0CH
–	12,500Hz	8MHz	39	27H
–	19,230Hz	4MHz	12	0CH
–	9,615Hz	4MHz	25	19H

## 15.2 Block Diagram

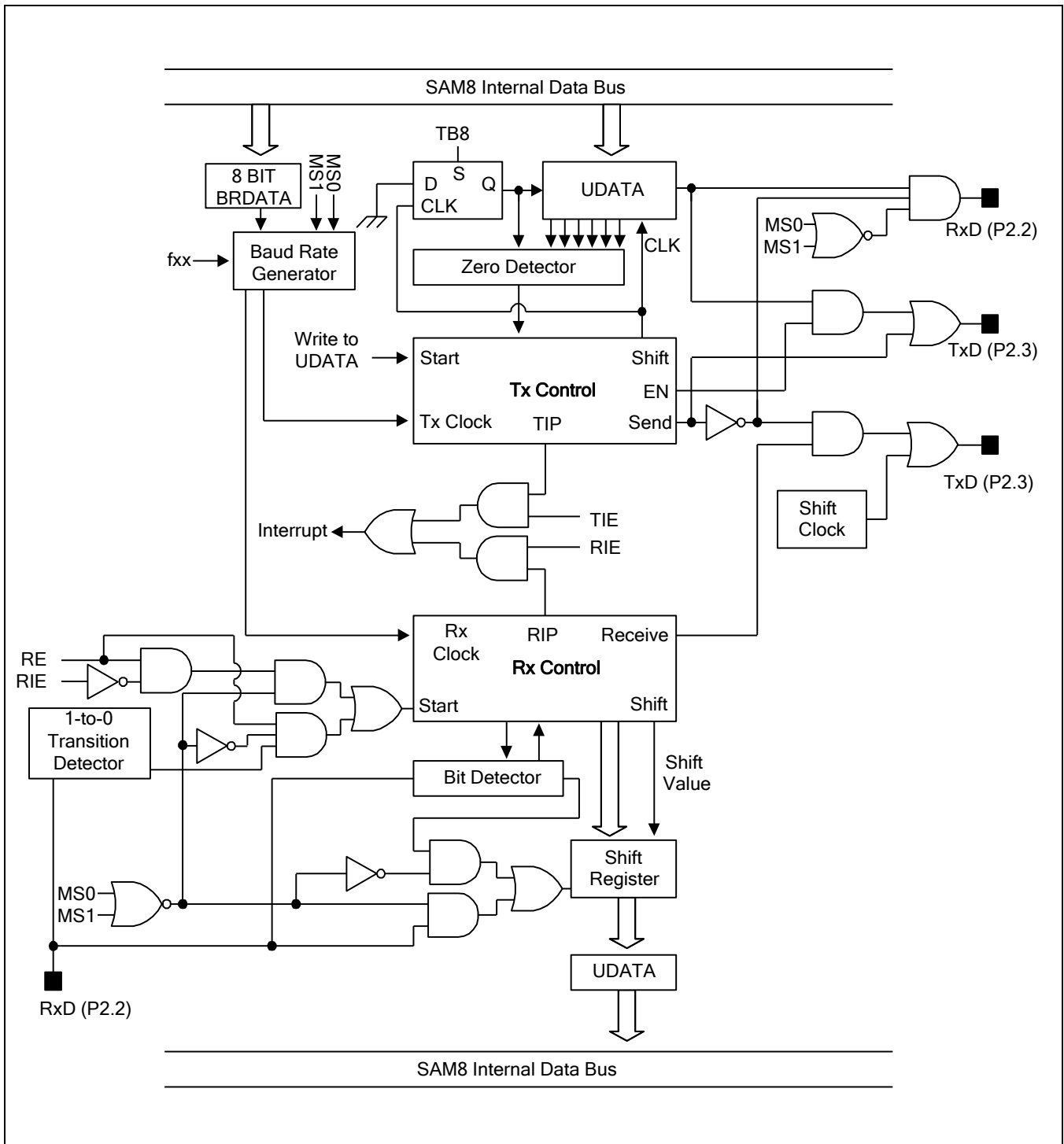


Figure 15-5 UART Functional Block Diagram

### 15.2.1 UART Mode 0 Function Description

In mode 0, UART is input and output through the RxD (P2.2) pin and TxD (P2.3) pin outputs the shift clock. Data is transmitted or received in 8-bit units only. The LSB of the 8-bit value is transmitted (or received) first.

#### 15.2.1.1 Mode 0 Transmit Procedure

1. Select mode 0 by setting UARTCON.6 and .7 to "00B".
2. Write transmission data to the shift register UDATA (F8H, set1, bank 1) to start the transmission operation.

#### 15.2.1.2 Mode 0 Receive Procedure

1. Select mode 0 by setting UARTCON.6 and .7 to "00B".
2. Clear the receive interrupt pending bit (UARTPND.1) by writing a "0" to UARTPND.1.
3. Set the UART receive enable bit (UARTCON.4) to "1".
4. The shift clock will now be output to the TxD (P2.3) pin and will read the data at the RxD (P2.2) pin. A UART receive interrupt (vector FAH) occurs when UARTCON.1 is set to "1".

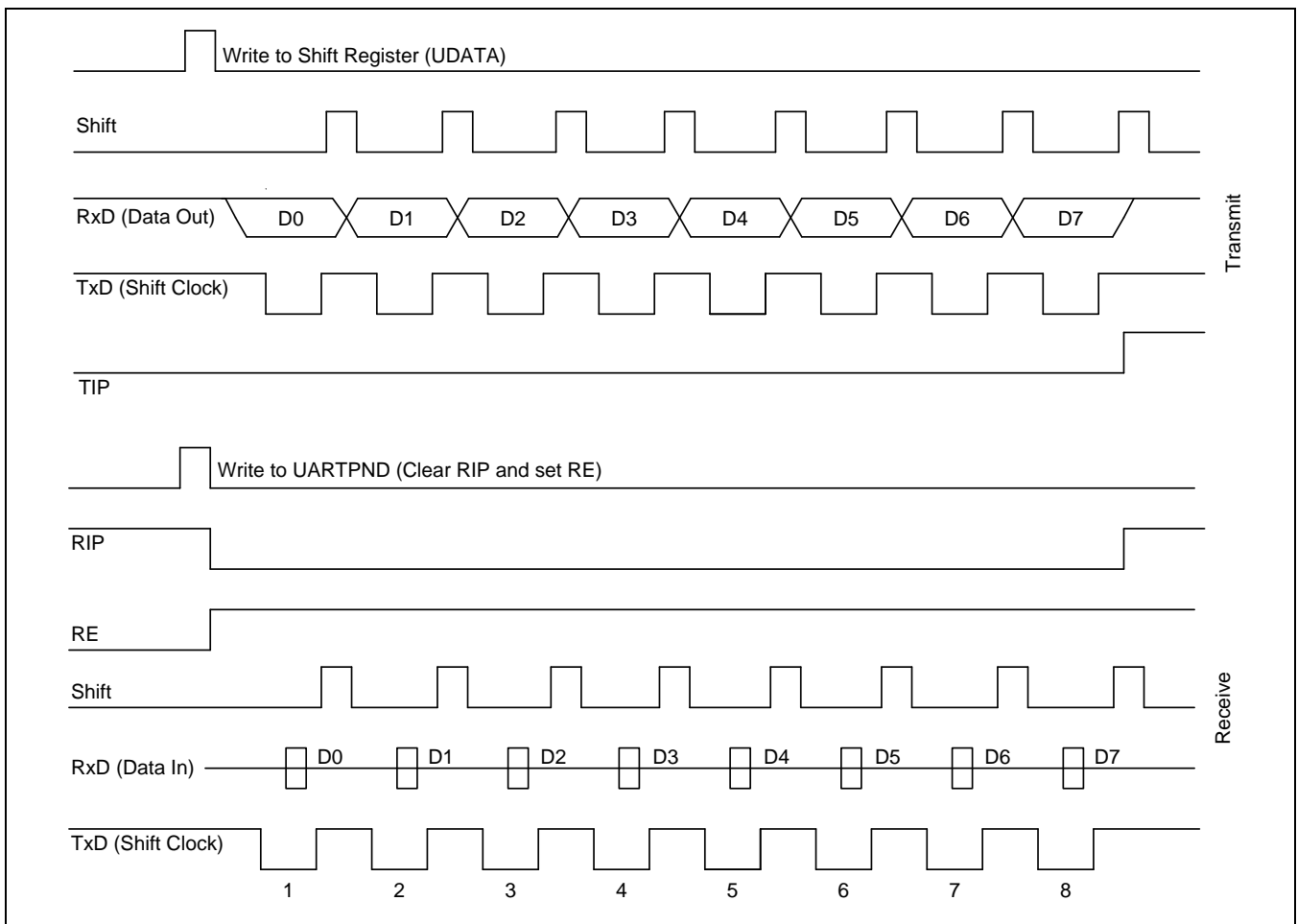


Figure 15-6 Timing Diagram for UART Mode 0 Operation

### 15.2.2 UART Mode 1 Function Description

In mode 1, 10 bits are transmitted (through the TxD (P2.3) pin) or received (through the RxD (P2.2) pin). Each data frame has three components:

- Start bit ("0")
- 8 data bits (LSB first)
- Stop bit ("1")

When receiving, the stop bit is written to the RB8 bit in the UARTCON register. The baud rate for mode 1 is variable.

#### 15.2.2.1 Mode 1 Transmit Procedure

1. Select the baud rate generated by 8-bit BRDATA.
2. Select mode 1 (8-bit UART) by setting UARTCON bits 7 and 6 to "01B".
3. Write transmission data to the shift register UDATA (F8H, Set1, and Bank 1). The start and stop bits are generated automatically by hardware.

#### 15.2.2.2 Mode 1 Receive Procedure

1. Select the baud rate to be generated by 8-bit BRDATA.
2. Select mode 1 and set the RE (Receive Enable) bit in the UARTCON register to "1".
3. The start bit low ("0") condition at the RxD (P2.2) pin will cause the UART module to start the serial data receive operation.

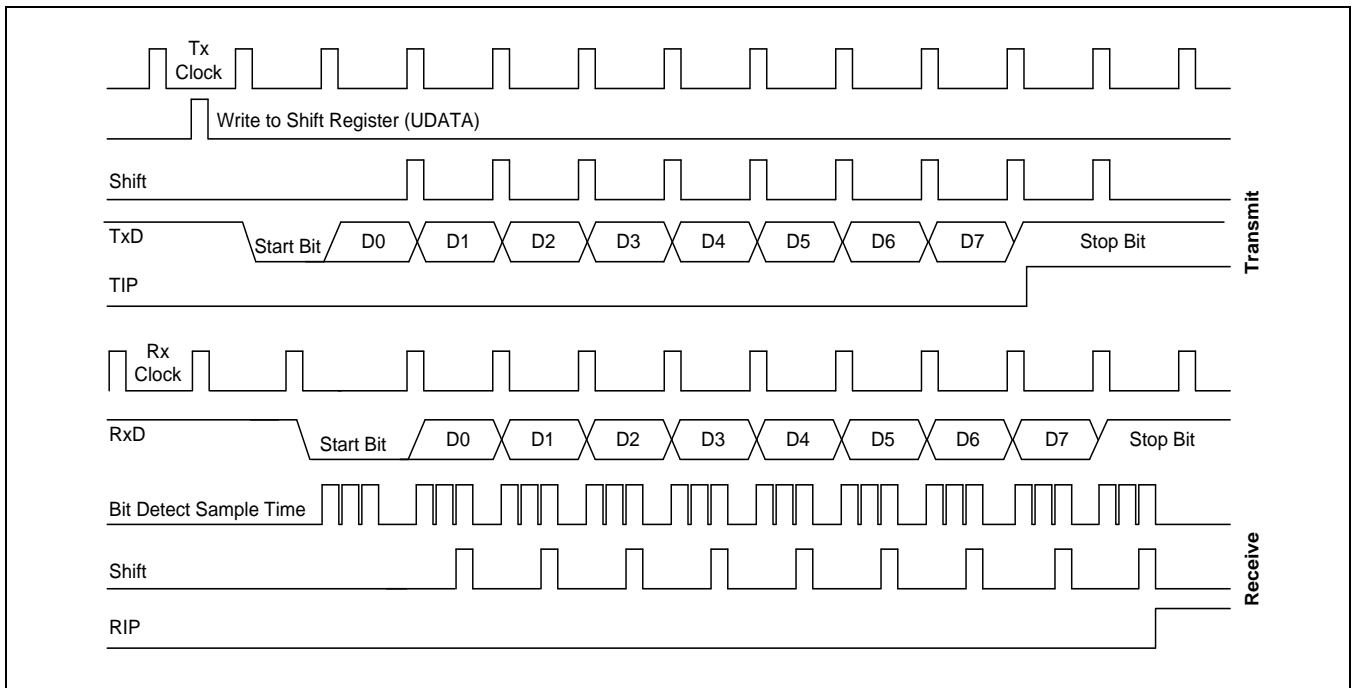


Figure 15-7 Timing Diagram for UART Mode 1 Operation



### 15.2.3 UART Mode 2 Function Description

In mode 2, 11-bit are transmitted through the TxD pin or received through the RxD pin. In mode 2, the baud rate is fixed at  $f_{xx}/16$ .

Each data frame has three components:

- Start bit ("0")
- 8 data bits (LSB first)
- Programmable 9th data bit
- Stop bit ("1")

The 9th data bit to be transmitted can be assigned a value of "0" or "1" by writing the TB8 bit (UARTCON0.3). When receiving, the 9th data bit that is received is written to the RB8 bit (UARTCON0.2), while the stop bit is ignored. The baud rate for mode 2 is  $f_{osc}/16$  clock frequency.

#### 15.2.3.1 Mode 2 Transmit Procedure

1. Select mode 2 (9-bit UART0) by setting UARTCON bits 6 and 7 to "10B". Also, select the 9th data bit to be transmitted by writing TB8 to "0" or "1".
2. Write transmission data to the shift register, UDATA (F8H, Set1, Bank 1), to start the transmit operation.

#### 15.2.3.2 Mode 2 Receive Procedure

1. Select mode 2 and set the receive enable bit (RE) in the UARTCON register to "1".
2. The receive operation starts when the signal at the RxD pin goes to low level.

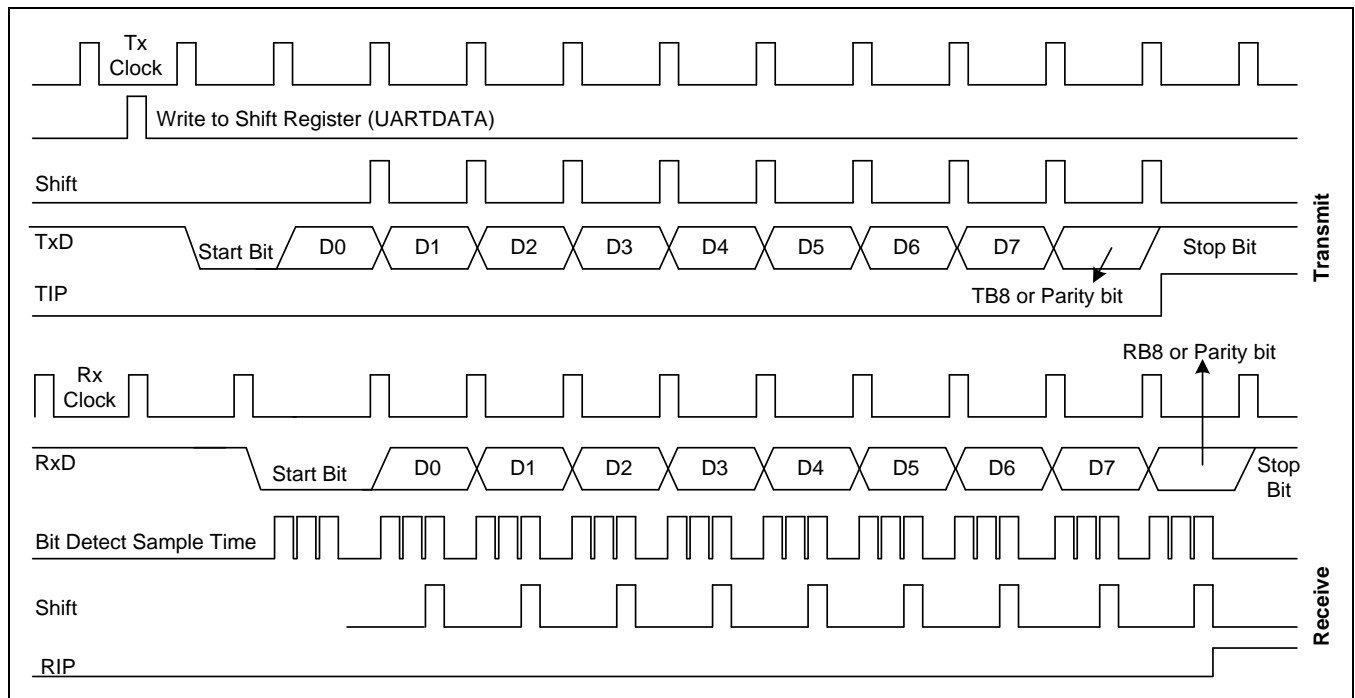


Figure 15-8 Timing Diagram for UART Mode 2 Operation

### 15.2.4 UART Mode 3 Function Description

In mode 3, 11 bits are transmitted (through the TxD) or received (through the RxD). Mode 3 is identical to mode 2 but can be configured to variable baud rate. Each data frame has four components:

- Start bit ("0")
- 8 data bits (LSB first)
- Programmable 9th data bit
- Stop bit ("1")

#### 15.2.4.1 Mode 3 Transmit Procedure

1. Select the baud rate generated by setting BRDATA.
2. Select mode 3 (9-bit UART) by setting UARTCON bits 6 and 7 to "11B". Also, select the 9th data bit to be transmitted by writing TB8 to "0" or "1"
3. Write transmission data to the shift register, UDATA (F8H, Set 1, Bank 1), to start the transmit operation.

#### 15.2.4.2 Mode 3 Receive Procedure

1. Select the baud rate to be generated by setting BRDATA.
2. Select mode 3 and set the receive enable bit (RE) in the UARTCON register to "1".
3. The receive operation starts when the signal at the RxD pin goes to low level.

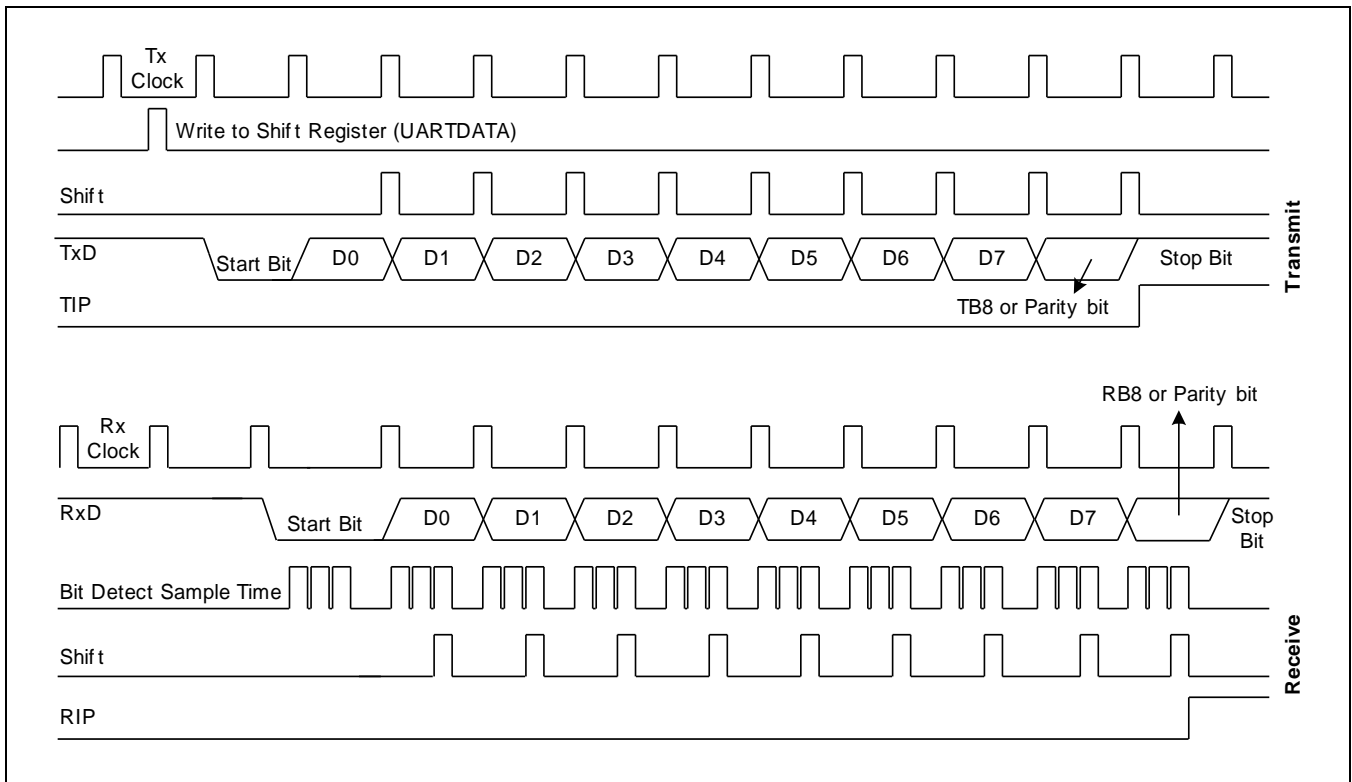


Figure 15-9 Timing Diagram for UART Mode 3 Operation

### 15.2.5 Serial Communication for Multiprocessor Configurations

The S3F8 Series multiprocessor communication features let a "master" S3F8S28/S3F8S24 send a multiple-frame serial message to a "slave" device in a multi S3F8S28/S3F8S24 configuration. It does this without interrupting other slave devices that may be on the same serial line.

This feature can be used only in UART Mode 2 or 3 with the parity disable mode. In mode 2 and 3, 9 data bits are received. The 9th bit value is written to RB8 (UARTCON.2). The data receive operation is concluded with a stop bit. You can program this function so that when the stop bit is received, the serial interrupt will be generated only if RB8 = "1".

To enable this feature, you set the MCE bit in the UARTCON registers. When the MCE bit is "1", serial data frames that are received with the 9th bit = "0" do not generate an interrupt. In this case, the 9th bit simply separates the address from the serial data.

#### 15.2.5.1 Sample Protocol for Master/Slave Interaction

When the master device wants to transmit a block of data to one of several slaves on a serial line, it first sends out an address byte to identify the target slave. Note that in this case, an address byte differs from a data byte: In an address byte, the 9th bit is "1" and in a data byte, it is "0".

The address byte interrupts all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave then clears its MCE bit and prepares to receive incoming data bytes.

The MCE bits of slaves that were not addressed remain set, and they continue operating normally while ignoring the incoming data bytes.

While the MCE bit setting has no effect in mode 0, it can be used in mode 1 to check the validity of the stop bit. For mode 1 reception, if MCE is "1", the receive interrupt will be issue unless a valid stop bit is received.

#### 15.2.5.2 Setup Procedure for Multiprocessor Communications

Follow these steps to configure multiprocessor communications:

1. Set all S3F8S28/S3F8S24 devices (masters and slaves) to UART Mode 2 or 3
2. Write the MCE bit of all the slave devices to "1".
3. The master device's transmission protocol is:
  - First byte: the address identifying the target slave device (9th bit = "1")
  - Next bytes: data (9th bit = "0")
4. When the target slave receives the first byte, all of the slaves are interrupted because the 9th data bit is "1". The targeted slave compares the address byte to its own address and then clears its MCE bit in order to receive incoming data. The other slaves continue operating normally.

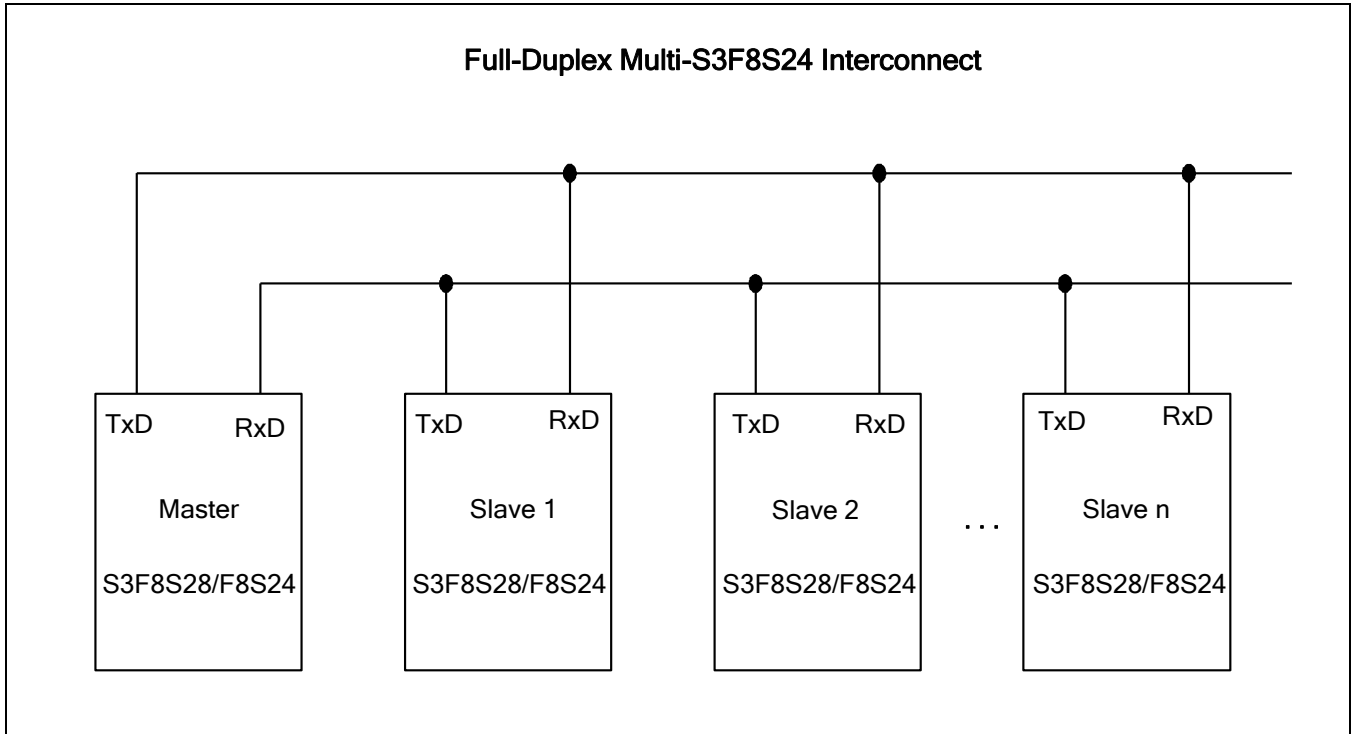


Figure 15-10 Connection Example for Multiprocessor Serial Data Communications

# 16 IIC Bus Interface

## 16.1 Overview

The S3F8S28/S3F8S24 microcontrollers support a multi-master IIC-bus serial interface. A dedicated serial data line (SDA) and a serial clock line (SCL) carry information between bus masters and peripheral devices which are connected to the IIC-bus. The SDA and SCL lines are bi-directional.

In multi-master IIC-bus mode, multiple S3F8S28/S3F8S24 microcontrollers can receive or transmit serial data to or from slave devices. The master S3F8S28/S3F8S24 which initiates a data transfer over the IIC-bus is responsible for terminating the transfer. Standard bus arbitration functions are supported.

To control multi-master IIC-bus operations, you write values to the following registers:

- IIC-bus control register, ICCR
- IIC-bus control/status register, ICSR
- IIC-bus Tx/Rx data shift register, IDSR
- IIC-bus address register, IAR

When the IIC-bus is free, the SDA and SCL lines are both at High level. A High-to-Low transition of SDA initiates a Start condition. A Low-to-High transition of SDA while SCL remains steady at High level initiates a Stop condition.

Start and Stop conditions are always generated by the bus master. A 7-bit address value in the first data byte that is put onto the bus after the Start condition is initiated determines which slave device the bus master selects. The 8th bit determines the direction of the transfer (read or write).

Every data byte that is put onto the SDA line must total eight bits. The number of bytes which can be sent or received per bus transfer operation is unlimited. Data is always sent most-significant bit (MSB) first and every byte must be immediately followed by an acknowledge (ACK) bit.

### 16.1.1 Multi-Master IIC Bus Control Register (ICCR)

The multi-master IIC-bus control register, ICCR, is located at address F0H, bank 1. It is read/write addressable. ICCR settings control the following IIC-bus functions:

- CPU acknowledge signal (ACK) enable or suppress
- IIC-bus clock source selection ( $f_{osc}/16$  or  $f_{osc}/512$ )
- Transmit/receive interrupt enable or disable
- Transmit/receive interrupt pending control
- 4-bit prescaler for the serial transmit clock (SCL)

In the S3F8S28/S3F8S24 interrupt structure, the IIC-bus Tx/Rx interrupt is assigned level IRQ6, vector E0H. To enable this interrupt, you set ICCR.5 to "1". Program software can then poll the IIC-bus Tx/Rx interrupt pending bit (ICCR.4) to detect IIC-bus receive or transmit requests. When the CPU acknowledges the interrupt request from the IIC-bus, the interrupt service routine must clear the interrupt pending condition by writing a "0" to ICCR.4.

The SCL frequency is determined by the IIC-bus clock source selection ( $f_{osc}/16$  or  $f_{osc}/512$ ) and the 4-bit prescaler value in the ICCR register (see [Figure 16-1](#)).

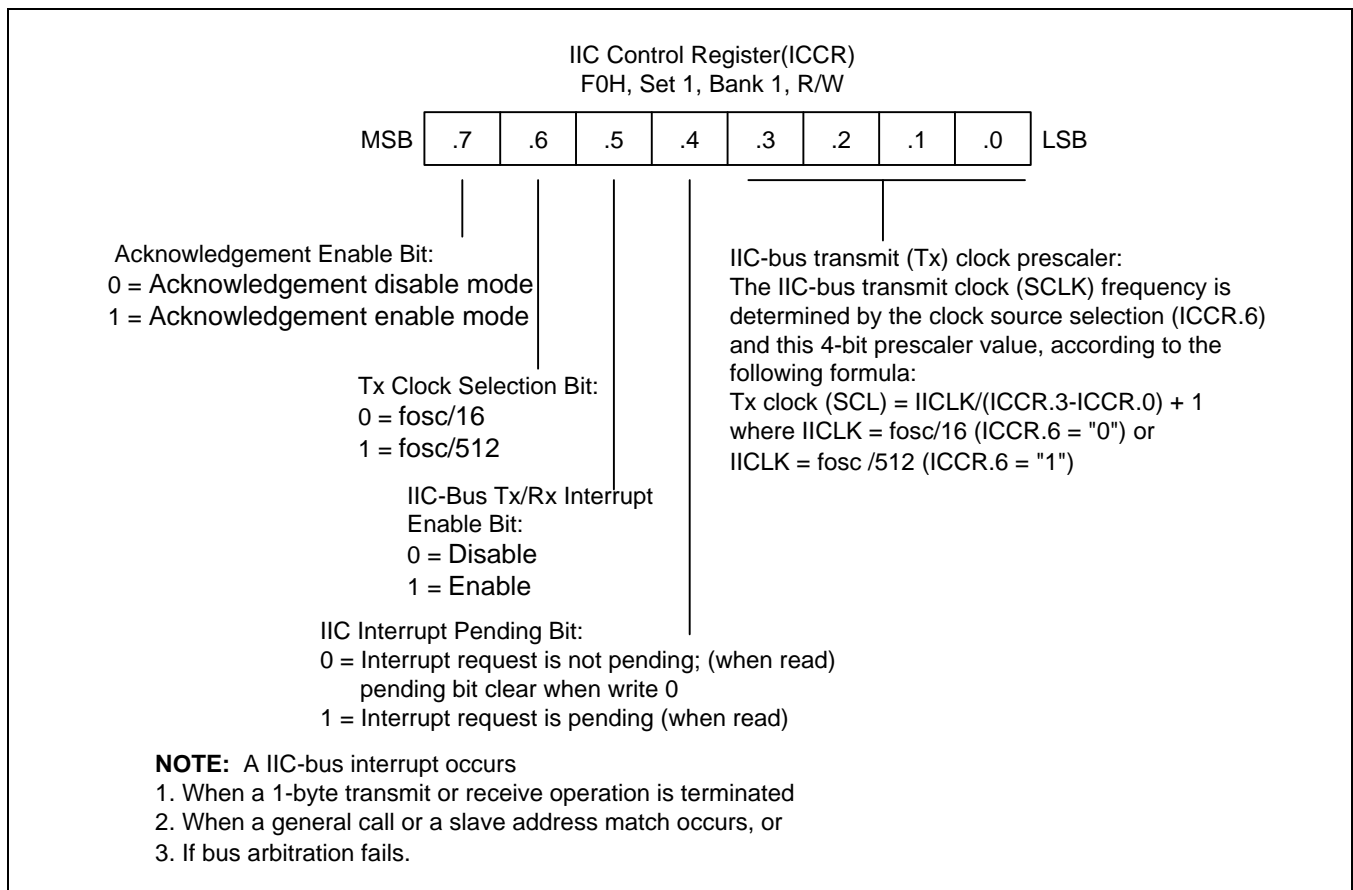


Figure 16-1 Multi-Master IIC Bus Control Register (ICCR)

**Table 16-1 Sample Timing Calculations for the IIC Bus Transmit Clock (SCL)**

ICCR.3–ICCR.0 Value	IICLK (ICCR.3–ICCR.0, Settings + 1)	(f <sub>osc</sub> = 8MHz) ICCR.6 = 0 (f <sub>osc</sub> /16) IICLK = 500kHz	(f <sub>osc</sub> = 8MHz) ICCR.6 = 1 (f <sub>osc</sub> /512) IICLK = 15.625kHz
0000	IICLK/1	400kHz (Note)	15.625kHz
0001	IICLK/2	250kHz	7.1825kHz
0010	IICLK/3	16.7kHz	5.2038kHz
0011	IICLK/4	125kHz	3.9063kHz
0100	IICLK/5	100kHz	3.1250kHz
0101	IICLK/6	83.3kHz	2.6042kHz
0110	IICLK/7	71.4kHz	2.2321kHz
0111	IICLK/8	62.5kHz	1.9531kHz
1000	IICLK/9	55.6kHz	1.7361kHz
1001	IICLK/10	50kHz	1.5625kHz
1010	IICLK/11	45.5kHz	1.4205kHz
1011	IICLK/12	41.7kHz	1.3021kHz
1100	IICLK/13	38.5kHz	1.2019kHz
1101	IICLK/14	35.7kHz	1.1160kHz
1110	IICLK/15	33.3kHz	1.0417kHz
1111	IICLK/16	31.25kHz	0.9766kHz

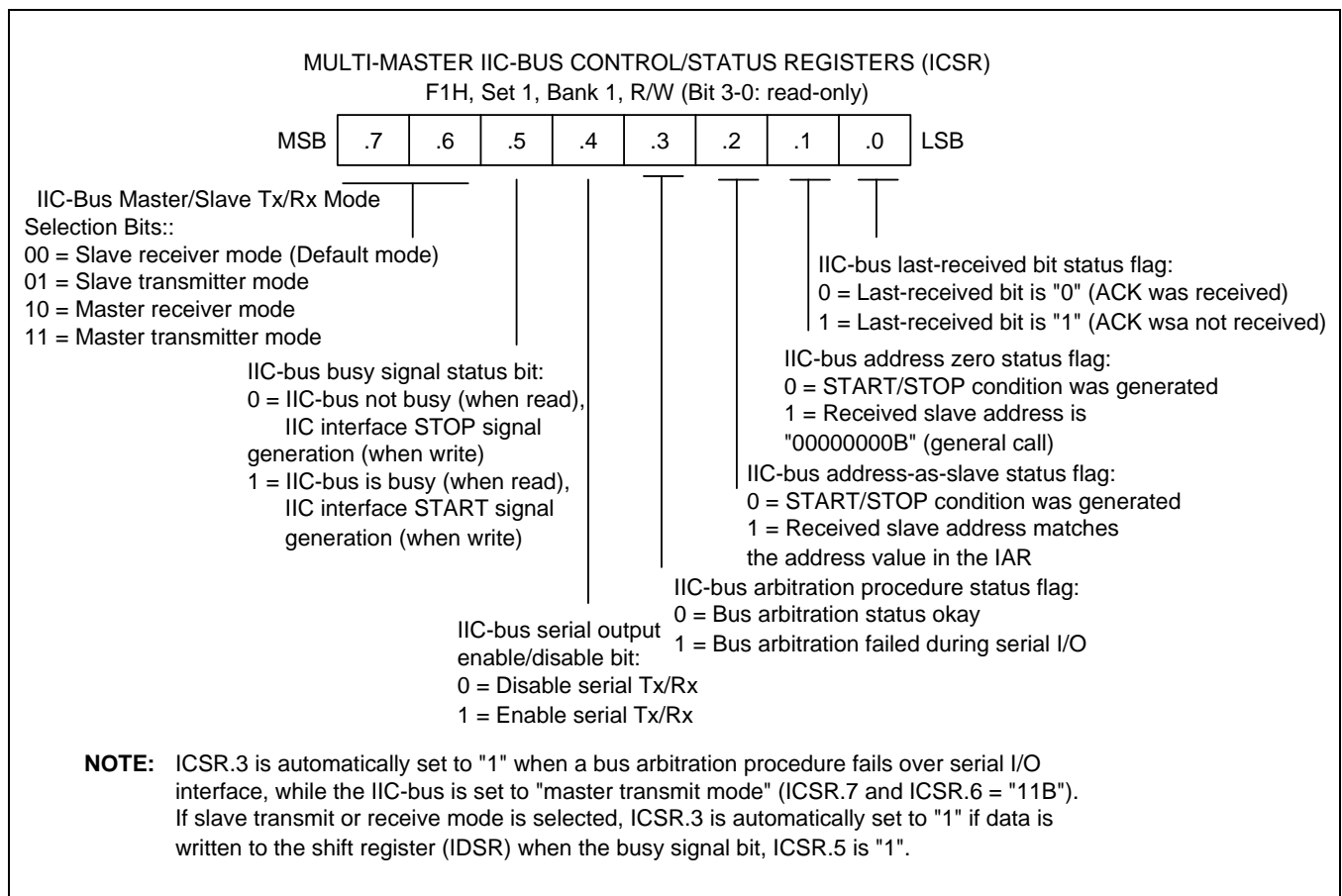
**NOTE:** Max. IICLK = 400kHz.

### 16.1.2 Multi-Master IIC Bus Control/Status Register (ICSR)

The multi-master IIC-bus control/status register, ICSR, is located at address F1H, BANK1. Four bits in this register, ICSR.3 to ICSR.0, are read-only status flags.

ICSR register settings are used to control or monitor the following IIC-bus functions (see [Figure 16-2](#)):

- Master/slave transmit or receive mode selection
- IIC-bus busy status flag
- Serial output enable/disable
- Failed bus arbitration procedure status flag
- Slave address/address register match or general call received status flag
- Slave address 0000000B (general call) received status flag
- Last received bit status flag (not ACK = "1", ACK = "0")



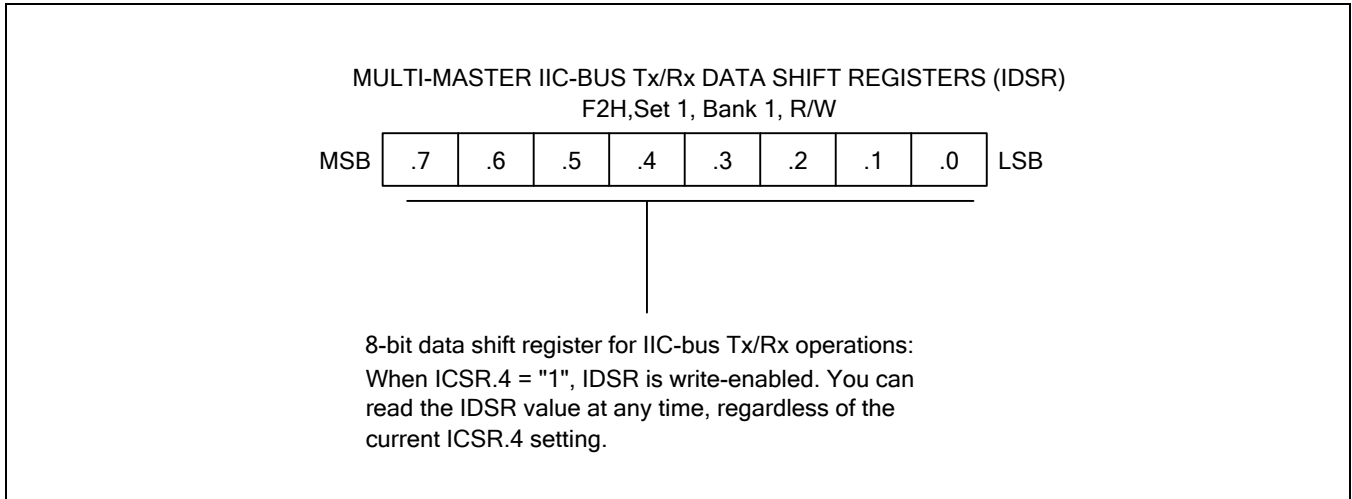
**Figure 16-2 Multi-Master IIC Bus Control/Status Register (ICSR)**



### 16.1.3 Multi-Master IIC Bus Transmit/Receive Data Shift Register (IDSR)

The IIC-bus data shift register, IDSR, is located at address F2H, Bank 1. In a transmit operation, data that is written to the IDSR is transmitted serially, MSB first. (For receive operations, the input data is written into the IDSR register LSB first.)

The ICSR.4 setting enables or disables serial transmit/receive operations. When ICSR.4 = "1", data can be written to the shift register. The IIC-bus shift register can, however, be read at any time, regardless of the current ICSR.4 setting.

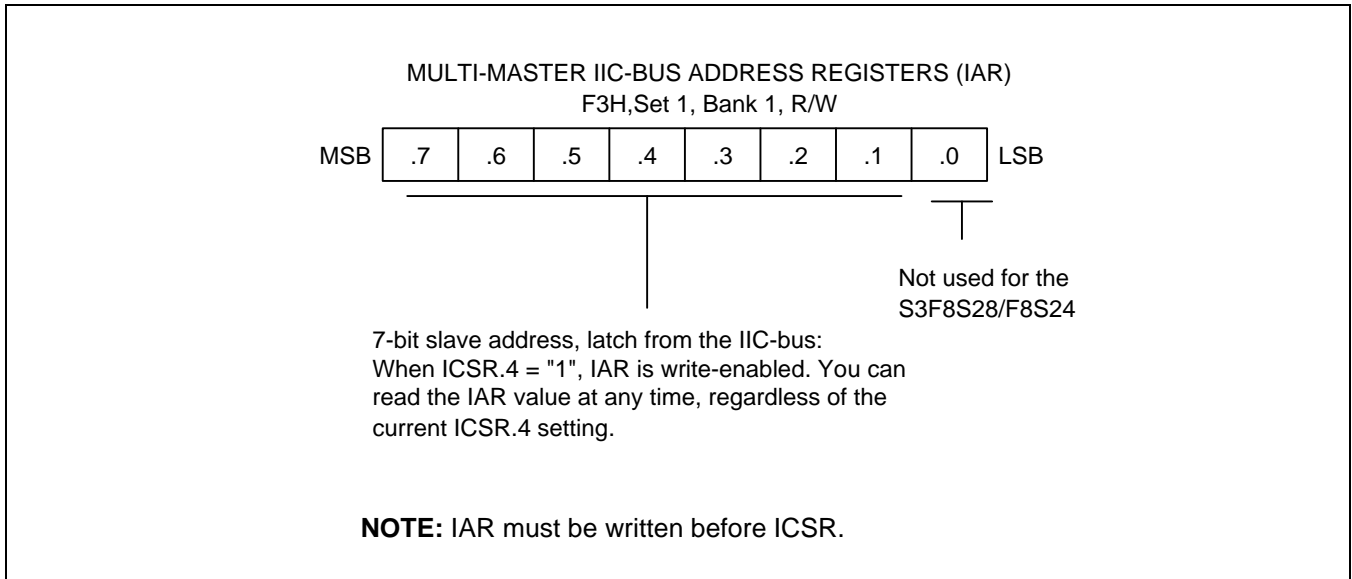


**Figure 16-3 Multi-Master IIC Bus Tx/Rx Data Shift Register (IDSR)**

### 16.1.4 Multi-Master IIC Bus Address Register (IAR)

The address register for the IIC-bus interface, IAR, is located at address F3H, Bank 1. It is used to store a latched 7-bit slave address. This address is mapped to IAR.7 to IAR.1; bit 0 is not used (see [Figure 16-4](#)).

The latched slave address is compared to the next received slave address. If a match condition is detected, and if the latched value is 00000000B, a general call status is detected.



**Figure 16-4 Multi-Master IIC Bus Address Register (IAR)**

## 16.2 Block Diagram

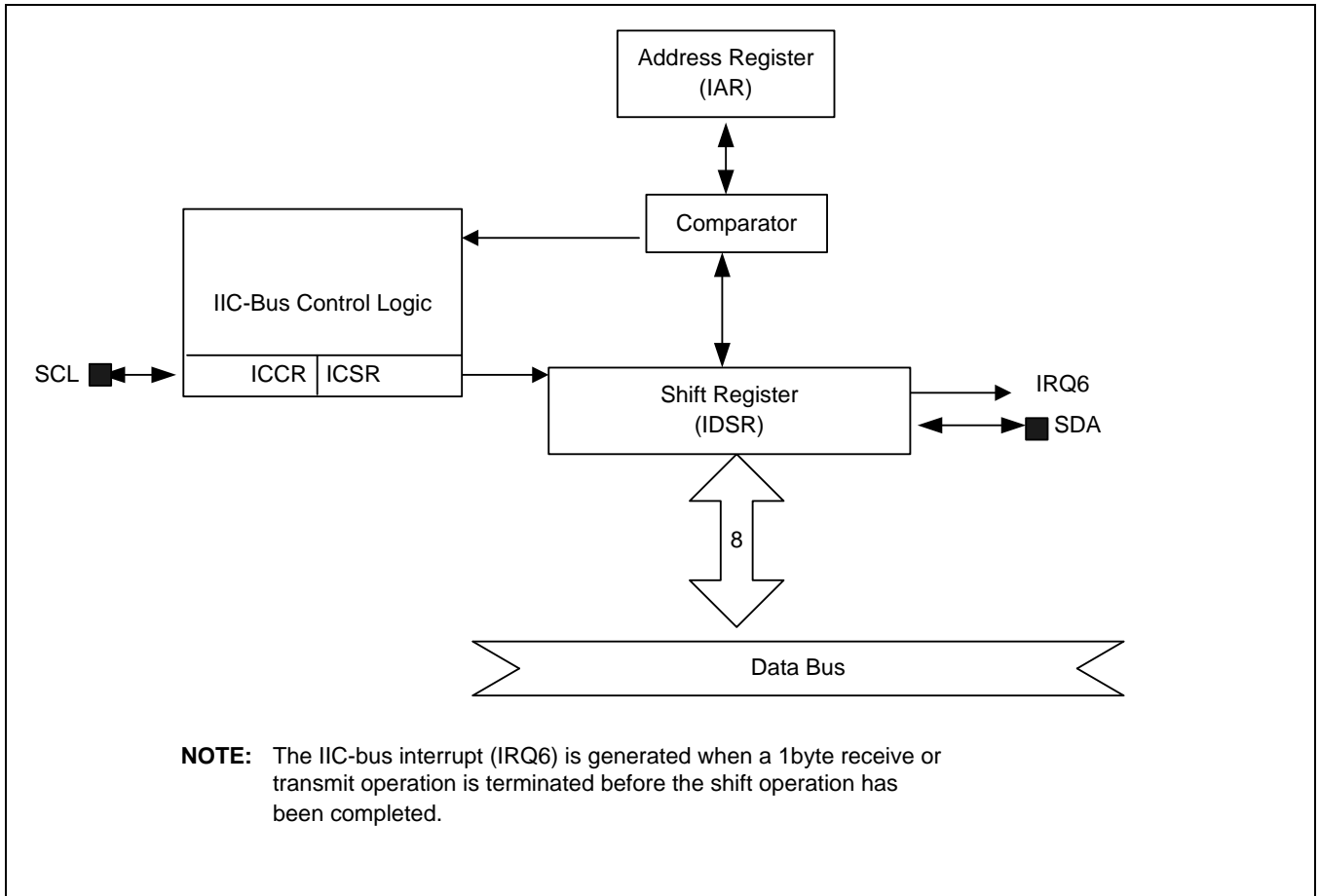


Figure 16-5 IIC Bus Block Diagram

## 16.3 The IIC Bus Interface

The S3F8S28/S3F8S24 IIC-bus interface has four operating modes:

- Master transmitter mode
- Master receive mode
- Slave transmitter mode
- Slave receive mode

Functional relationships between these operating modes are described below.

## 16.4 Start and Stop Conditions

When the IIC-bus interface is inactive, it is in slave mode. The interface is therefore always in slave mode when a start condition is detected on the SDA line. (A start condition is a High-to-Low transition of the SDA line while the clock signal, SCL, is high level.) When the interface enters master mode, it initiates a data transfer and generates the SCL signal.

A start condition initiates a one-byte serial data transfer over the SDA line and a stop condition ends the transfer. (A stop condition is a Low-to-High transition of the SDA line while SCL is High level.) Start and stop conditions are always generated by the master. The IIC-bus is "busy" when a start condition is generated. A few clocks after a stop condition is generated, the IIC-bus is again "free".

When a master initiates a start condition, it sends its slave address onto the bus. The address byte consists of a 7-bit address and a 1-bit transfer direction indicator (that is, write or read). If bit 8 is "0", a transmit operation (write) is indicated; if bit 8 is "1", a request for data (read) is indicated.

The master ends the indicated transfer operation by transmitting a stop condition. If the master wants to continue sending data over the bus, it can generate another slave address and another start condition. In this way, read write operations can be performed in various formats.

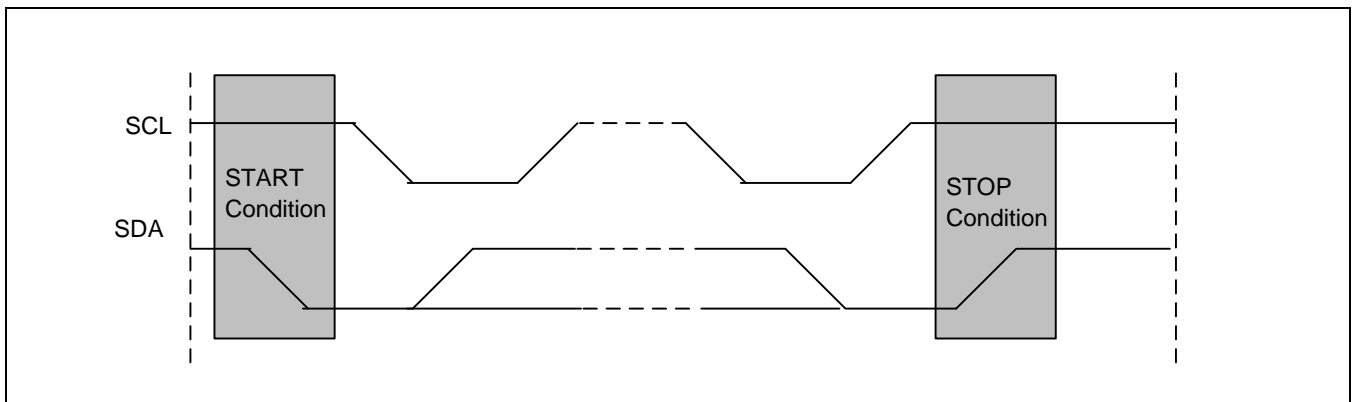


Figure 16-6 Start and Stop Conditions

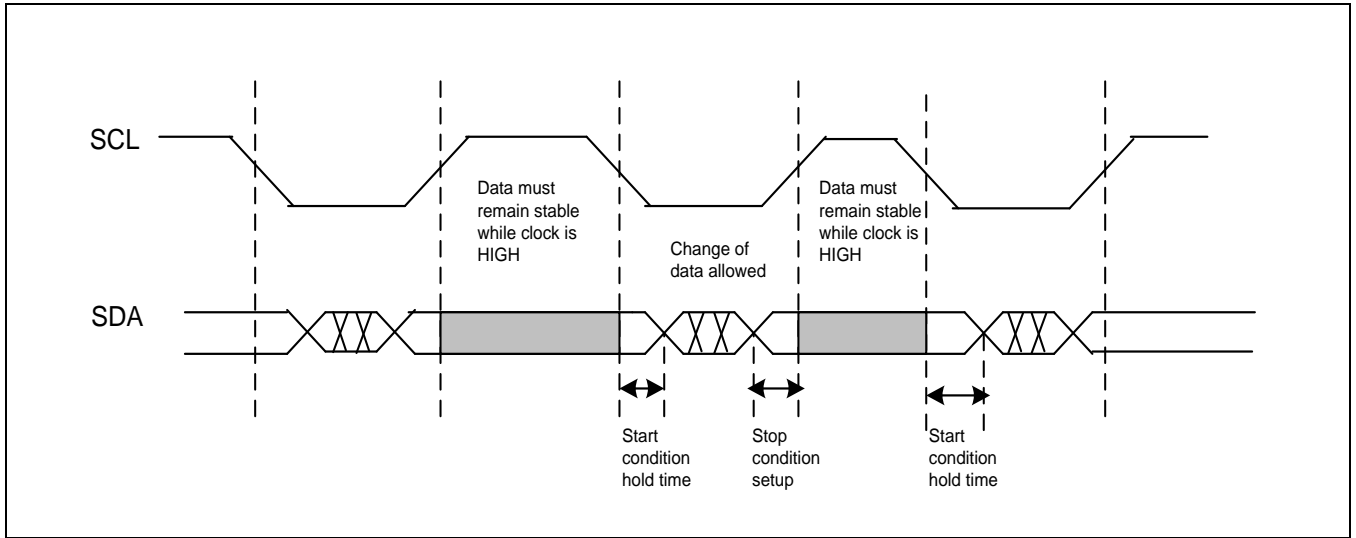
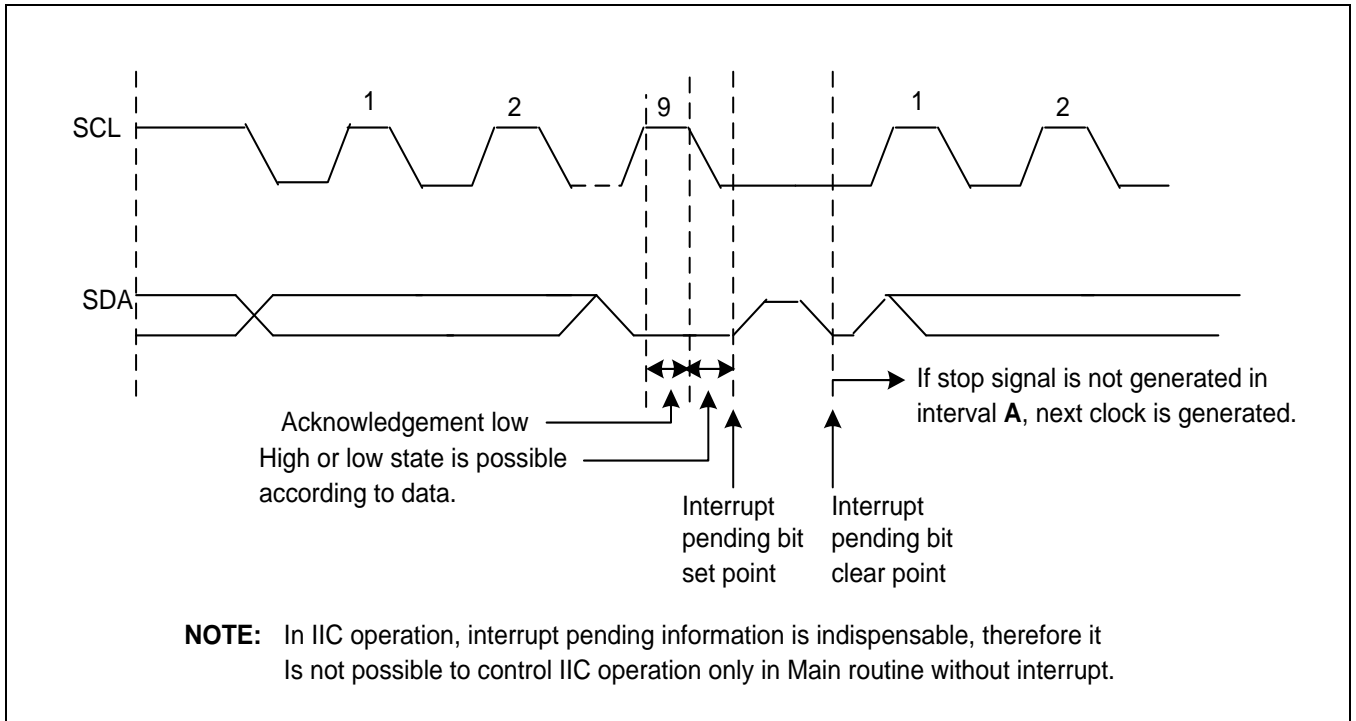


Figure 16-7 Input Data Protocol



**NOTE:** In IIC operation, interrupt pending information is indispensable, therefore it is not possible to control IIC operation only in Main routine without interrupt.

Figure 16-8 Interrupt Pending Information

## 16.5 Data Transfer Formats

Every byte put on the SDA line must be eight bits in length. The number of bytes which can be transmitted per transfer is unlimited. The first byte following a start condition is the address byte. This address byte is transmitted by the master when the IIC-bus is operating in master mode. Each byte must be followed by an acknowledge (ACK) bit. Serial data and addresses are always sent MSB first.

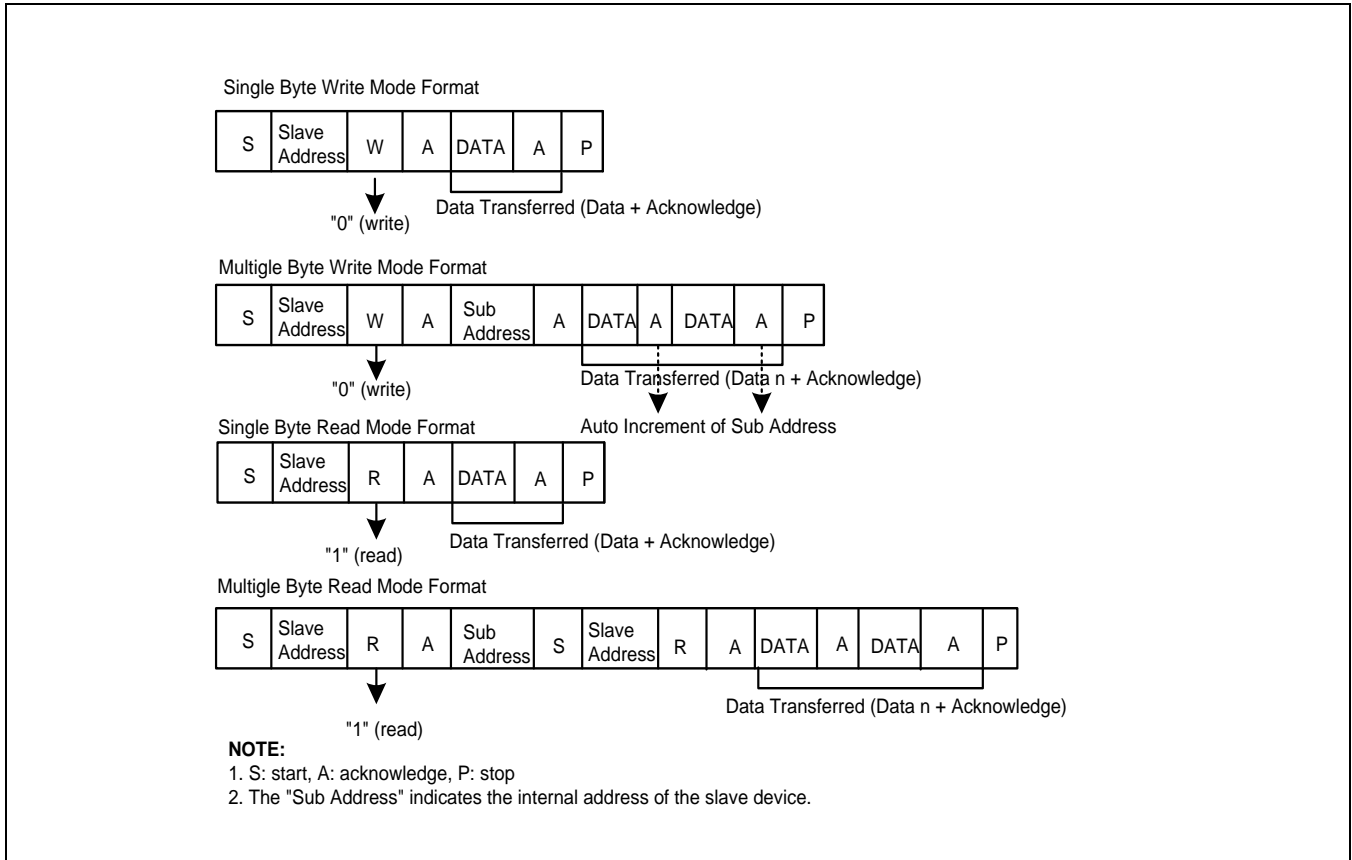


Figure 16-9 IIC Bus Interface Data Formats

## 16.6 ACK Signal Transmission

To complete a one-byte transfer operation, the receiver must send an ACK bit to the transmitter. The ACK pulse occurs at the ninth clock of the SCL line (eight clocks are required to complete the one-byte transfer). The clock pulse required for the transmission of the ACK bit is always generated by the master.

The transmitter releases the SDA line (that is, it sends the SDA line High) when the ACK clock pulse is received. The receiver must drive the SDA line Low during the ACK clock pulse so that SDA is Low during the High period of the ninth SCL pulse.

The ACK bit transmit function can be enabled and disabled by software (ICCR.7). However, the ACK pulse on the ninth clock of SCL is required to complete a one-byte data transfer operation.

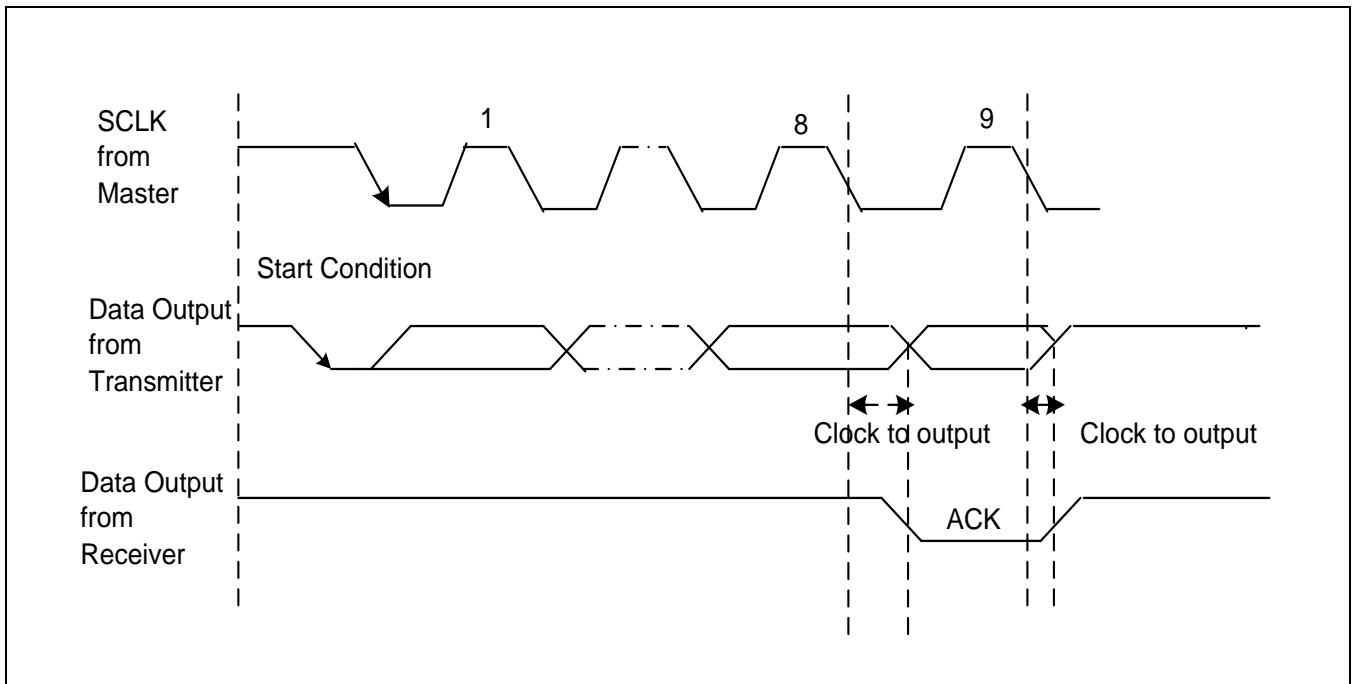


Figure 16-10 Acknowledge Response from Receiver

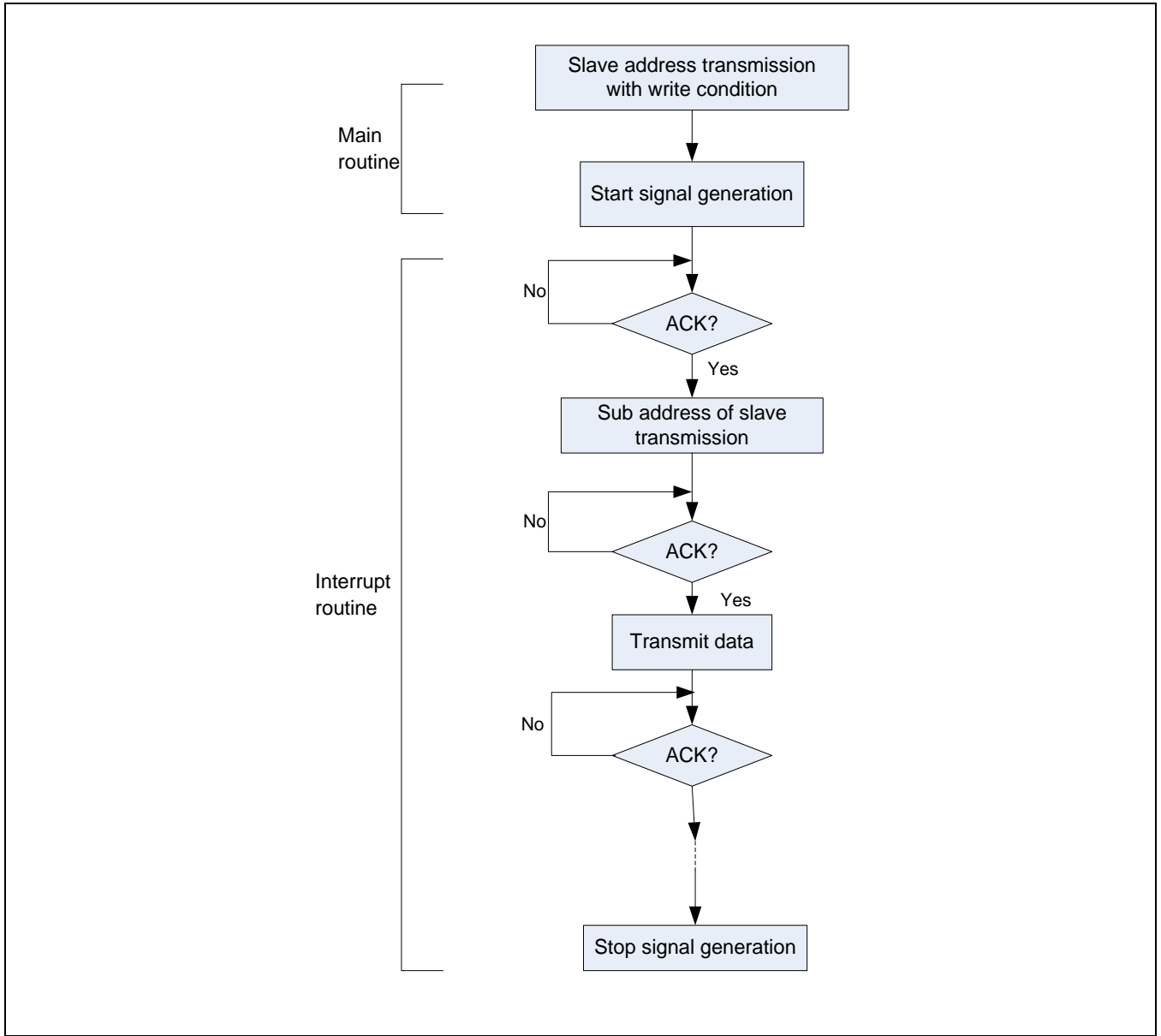


Figure 16-11 Write Operation Sequence



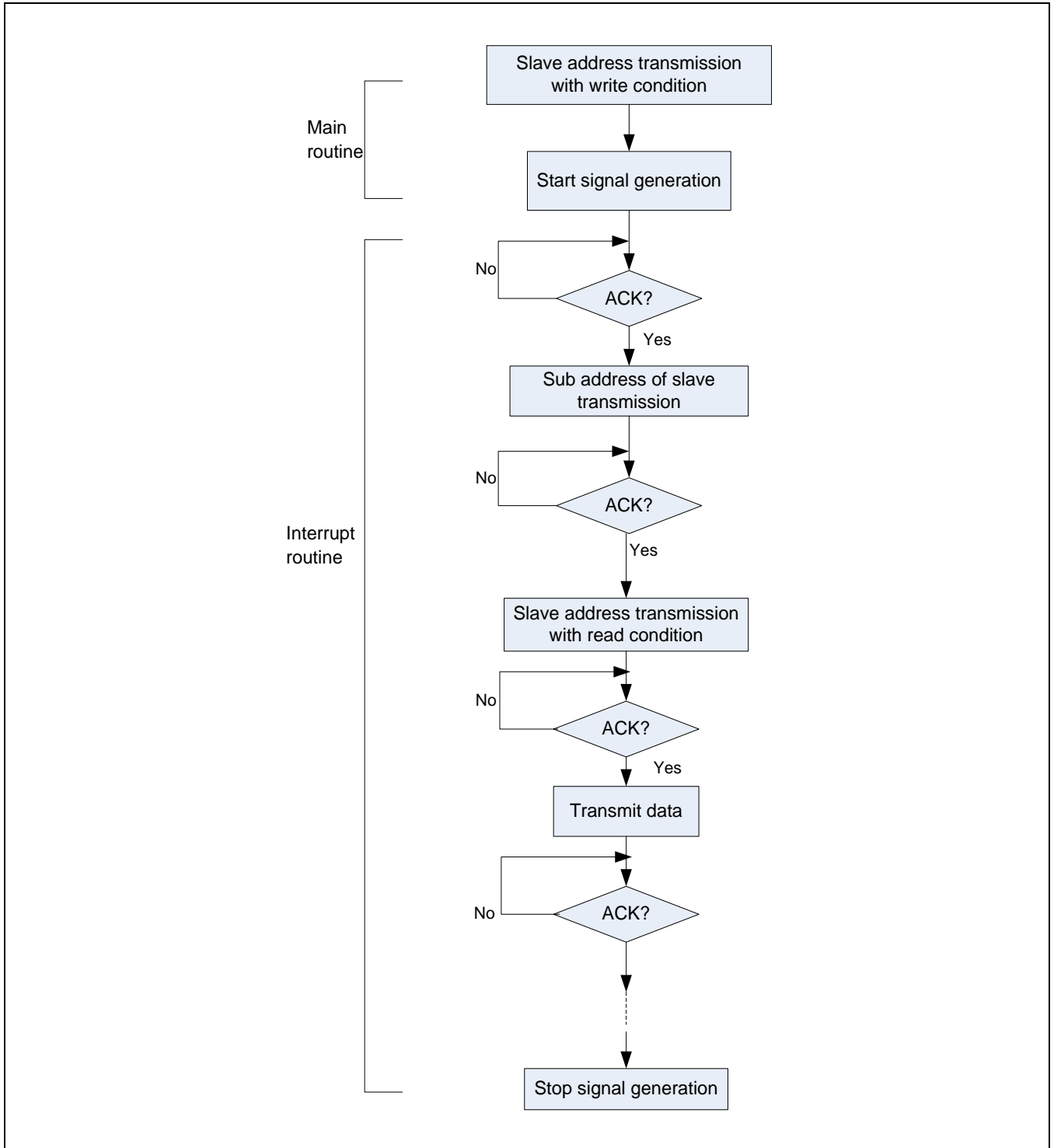


Figure 16-12 Read Operation Sequence

## 16.7 Read/Write Operations

When operating in transmitter mode, the IIC-bus interface interrupt routine waits for the master (the S3F8S28/S3F8S24) to write a data byte into the IIC-bus data shift register (IDSR). To do this, it holds the SCL line Low prior to transmission.

In receive mode, the IIC-bus interface waits for the master to read the byte from the IIC-bus data shift register (IDSR). It does this by holding the SCL line Low following the complete reception of a data byte.

## 16.8 Bus Arbitration Procedures

Arbitration takes place on the SDA line to prevent contention on the bus between two masters. If a master with a SDA High level detects another master with an SDA active Low level, it will not initiate a data transfer because the current level on the bus does not correspond to its own. The master which loses the arbitration can generate SCL pulses only until the end of the last-transmitted data byte. The arbitration procedure can continue while data continues to be transferred over the bus.

The first stage of arbitration is the comparison of address bits. If a master loses the arbitration during the addressing stage of a data transfer, it is possible that the master which won the arbitration is attempting to address the master which lost. In this case, the losing master must immediately switch to slave receiver mode.

## 16.9 Abort Conditions

If a slave receiver does not acknowledge the slave address, it must hold the level of the SDA line High. This signals the master to generate a stop condition and to abort the transfer.

If a master receiver is involved in the aborted transfer, it must also signal the end of the slave transmit operation. It does this by not generating an ACK after the last data byte received from the slave. The slave transmitter must then release the SDA to allow a master to generate a stop condition.

## 16.10 Configuring the IIC-Bus

To control the frequency of the serial clock (SCL), you program the 4-bit prescaler value in the ICCR register. The IIC-bus interface address is stored in IIC-bus address register, IAR. (By default, the IIC-bus interface address is an unknown value).

# 17

## Low Voltage Detector

### 17.1 Overview

The S3F8S28/S3F8S24 micro-controller has a built-in LVD (Low Voltage Detector) circuit which allows detection of power voltage drop to generate flag:

- Generate flag when VDD less than one selected level from 4.1, 3.2, 2.5 or 2.1V

Low voltage detector circuits have following functional components:

- Enable or disable LVD module
- LVD Flag when detector setting level.

Turning the LVD operation on and off can be controlled by software. Because the IC consumes a large amount of current during LVD operation, it is recommended that the LVD operation should be kept OFF unless it is necessary.

Also the LVD criteria voltage can be set by the software. The LVD flag criteria voltage can be set by matching to one of the 4 kinds of voltage 2.1V, 2.5V, 3.2V, 4.1V (VDD reference voltage).

The LVD block works only when LVDCON.7 is set. If VDD level is lower than the reference voltage selected with LVDCON.1-0, LVDCON.5 will be set. If VDD level is higher, LVDCON.5 will be cleared.

## 17.2 Low Voltage Detector Control Register (LVDCON)

You use the Low Voltage Detector control register, LVDCON, to

- Enable low voltage detector circuit
- Check LVD flag
- Set low voltage detector flag level

LVDCON is located at address F4H, Set1, Bank 1, and is read/write addressable using register addressing mode.

A reset clears LVDCON to "00H". This disable Low Voltage Detector Circuit, set Low voltage detector level as 4.1V.

You can disable LVD at any time during normal operation by writing a "0" to LVDCON.7 for lower power consumption. Write specific value to LVDCON.1–0 to select LVD flag level.

To check a voltage detector result the application program should poll the Flag bit LVDCON.5. When a "1" is detected, VDD level has drop below LVD level.

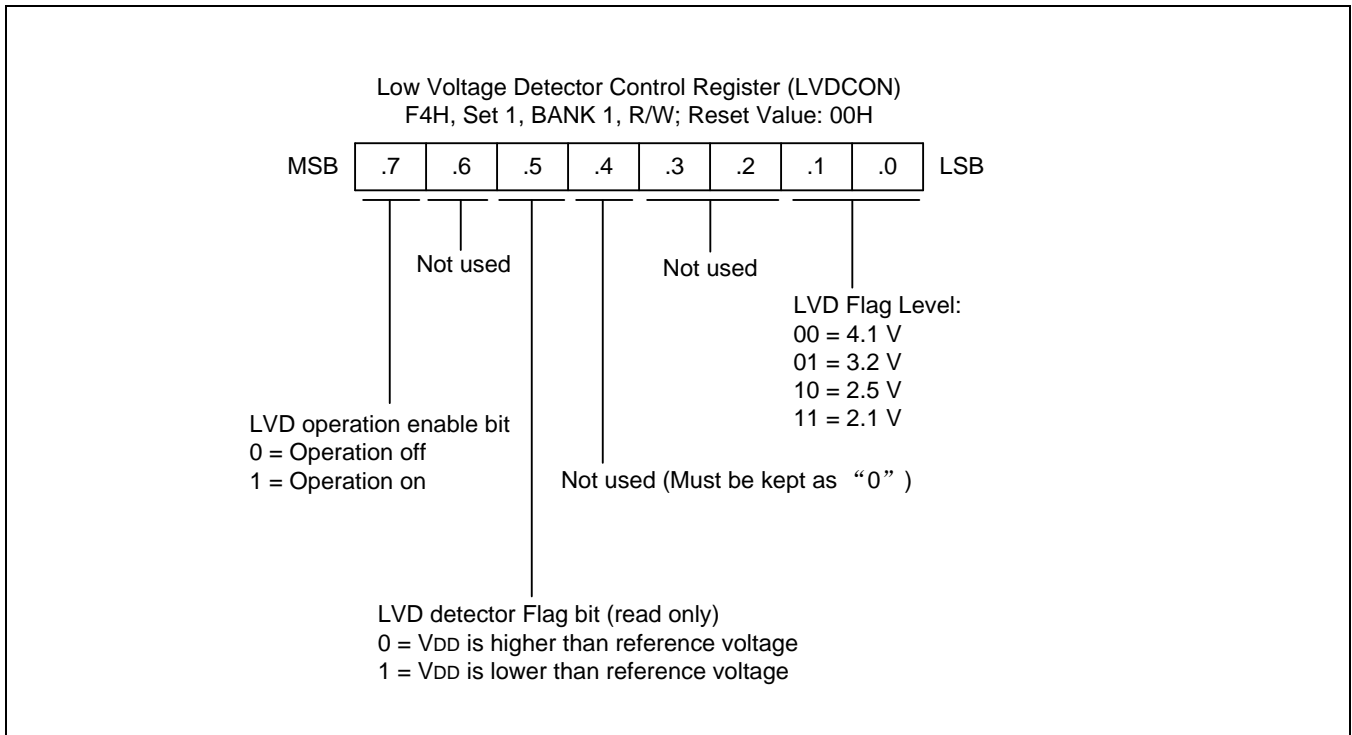


Figure 17-1 LVD Control Register (LVDCON)

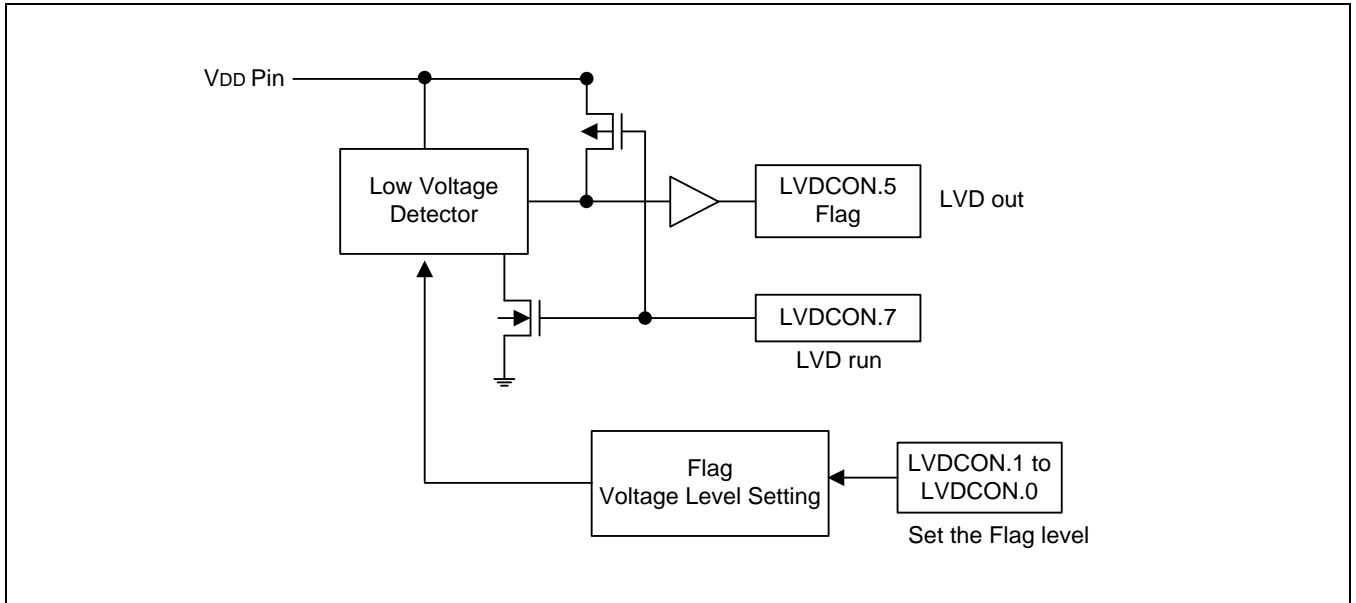


Figure 17-2 Block Diagram for Low Voltage Detector

### 17.3 Voltage (VDD) Level Detection Sequence-LVD Usage

- STEP 0: Don't make LVD on in normal conditions for small current consumption.
- STEP 1: For initializing analog comparator, write #80h to LVDCON. (Comparator initialization, LVD enable)
- STEP 2: Write value to reference voltage setting bits in LVDCON. (Voltage setting, LVD enable)
- STEP 3: Wait 10 to 20usec for comparator operation time (Wait compare time)
- STEP 4: Check result by loading voltage level set bit in LVDCON. (Check result)
- STEP 5: For another measurement, repeat above steps.

#### Example 17-1 LVD Using Method

```

LD    LVDCON, #80H           ; Comparator initialization, LVD enable (STEP 1)

LD    LVDCON, #10000011B    ; 2.1V detection voltage setting, LVD enable (STEP 2)

NOP
NOP
NOP
.           ; Wait 10 to 20usec (STEP 3)
.
.

LD    R0, LVDCON            ; Load LVDCON to R0 (STEP 4)
TM    R0, #00010000B        ; Check bit 5 of R0. If bit 5 is "H", VDD is lower than 2.1V.
JP    NZ, LOW_VDD           ; If not zero (bit 5 is "H"), jump to "LOW_VDD" routine.

LOW_VDD:                    ; Take action when Low VDD detected.
.
.
.

```

# 18 Embedded Flash Memory Interface

## 18.1 Overview

The S3F8S28/S3F8S24 has an on-chip Flash memory internally instead of masked ROM. The Flash memory is accessed by instruction "LDC". This is a sector erasable and a byte programmable Flash. User can program the data in a Flash memory area any time you want. The S3F8S28/S3F8S24 embedded 8K / 4Kbyte memory has two operating features as below:

- Tool Program Mode: Refer to the chapter 18. S3F8S28/S3F8S24 Flash MCU
- User Program Mode

### 18.1.1 Flash ROM Configuration

The S3F8S28/S3F8S24 Flash memory consists of 64 sectors. Each sector consists of 128bytes. So, the total size of Flash memory is  $64 \times 128$  (8KB) or  $32 \times 128$ bytes (4KB). User can erase the Flash memory by a sector unit at a time and write the data into the Flash memory by a byte unit at a time.

- 8K/4Kbyte Internal Flash memory
- Sector size: 128 bytes
- 10years data retention
  - Fast programming Time:
  - Sector Erase: 8ms (min.)
  - Byte Program: 25us (min.)
- Byte programmable
- User programmable by "LDC" instruction
- Sector (128 bytes) erase available
- Endurance: 100,000 Erase/Program cycles (min.)

### 18.1.2 Tool Program Mode

This mode is for erasing and programming full area of Flash memory by external programming tools. The 5 pins of S3F8S28/S3F8S24 are connected to a programming tool and then internal Flash memory of S3F8S28/S3F8S24 can be programmed by serial OTP/MTP Tools, SPW2 plus single programmer or GW-PRO2 gang programmer and so on. The other modules except Flash memory module are at a reset state. This mode doesn't support the sector erase but chip erase (all Flash memory erased at a time) and two protection modes (Hard lock protection/Read protection). The read protection mode is available only in tool program mode. So in order to make a chip into read protection, you need to select a read protection option when you write a program code to a chip in tool program mode by using a programming tool. After read protect, all data of Flash memory read "00". This protection is released by chip erase execution in the tool program mode.

**Table 18-1 Descriptions of Pins Used to Read/Write the Flash in Tool Program Mode**

Main Chip Pin Name	During Programming			
	Pin Name	Pin No.	I/O	Function
P0.1	SDAT	22 (24-pin), 18 (20-pin)	I/O	Serial data pin (output when reading, Input when writing) Input and push-pull output port can be assigned
P0.0	SCLK	23 (24-pin), 19 (20-pin)	I	Serial clock pin (input only pin)
RESET/P1.2	VPP	4	I	Power supply pin for Tool mode entering (indicates that MTP enters into the Tool mode). When 11V is applied, MTP is in Tool mode.
V <sub>DD</sub> /V <sub>SS</sub>	V <sub>DD</sub> /V <sub>SS</sub>	24 (24-pin), 20 (20-pin), 1 (24-pin), 1 (20-pin),	I	Logic power supply pin.

### 18.1.3 User Program Mode

This mode supports sector erase, byte programming, byte read and one protection mode (Hard Lock Protection). The S3F8S28/S3F8S24 has the internal pumping circuit to generate high voltage. To program a Flash memory in this mode several control registers will be used.

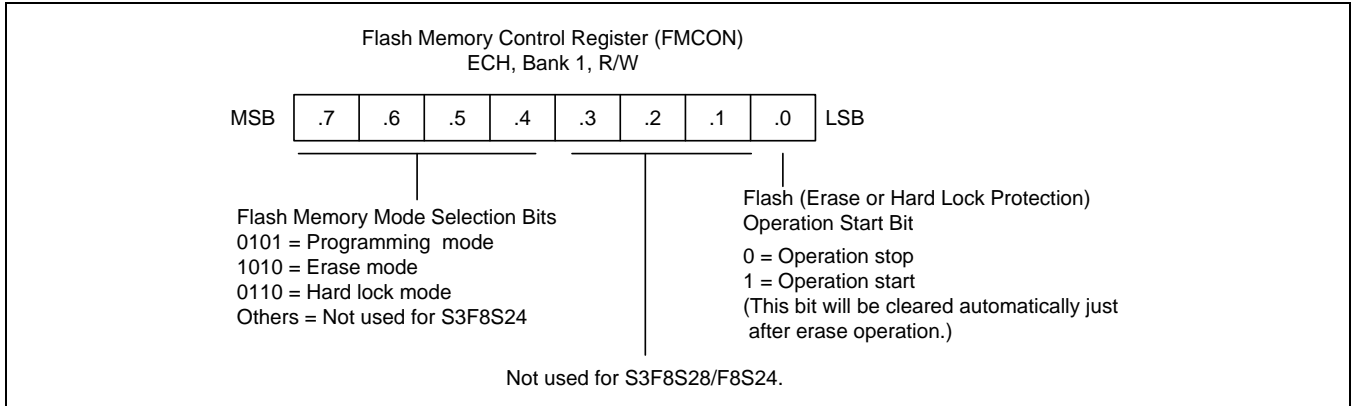
There are four kind functions in user program mode-programming, reading, sector erase, and one protection mode (Hard lock protection).



## 18.2 Flash Memory Control Registers (User Program Mode)

### 18.2.1 Flash Memory Control Register (FMCON)

FMCON register is available only in user program mode to select the Flash memory operation mode; sector erase, byte programming, and to make the Flash memory into a hard lock protection.

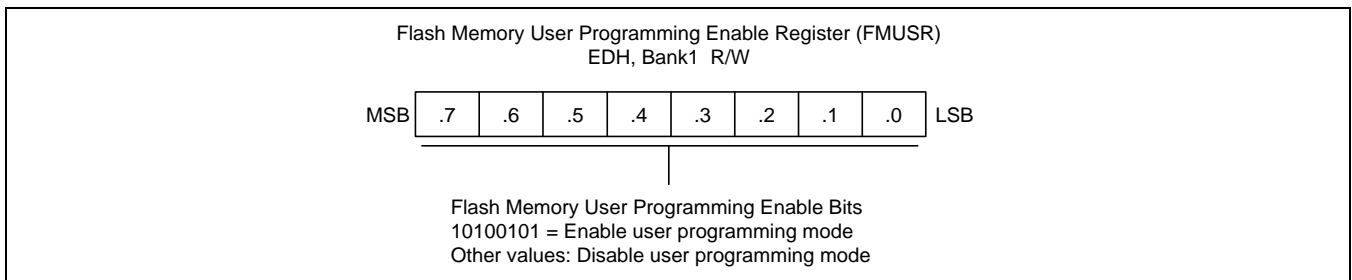


**Figure 18-1 Flash Memory Control Register (FMCON)**

The bit 0 of FMCON register (FMCON.0) is a bit for the operation start of Erase and Hard Lock Protection. Therefore, operation of Erase and Hard Lock Protection is activated when you set FMCON.0 to "1". If you write FMCON.0 to 1 for erasing, CPU is stopped automatically for erasing time (min.4ms). After erasing time, CPU is restarted automatically. When you read or program a byte data from or into Flash memory, this bit is not needed to manipulate.

### 18.2.2 Flash Memory User Programming Enable Register (FMUSR)

The FMUSR register is used for a safe operation of the Flash memory. This register will protect undesired erase or program operation from malfunctioning of CPU caused by an electrical noise. After reset, the user-programming mode is disabled, because the value of FMUSR is "00000000B" by reset operation. If necessary to operate the Flash memory, you can use the user programming mode by setting the value of FMUSR to "10100101B". The other value of "10100101B", user program mode is disabled.



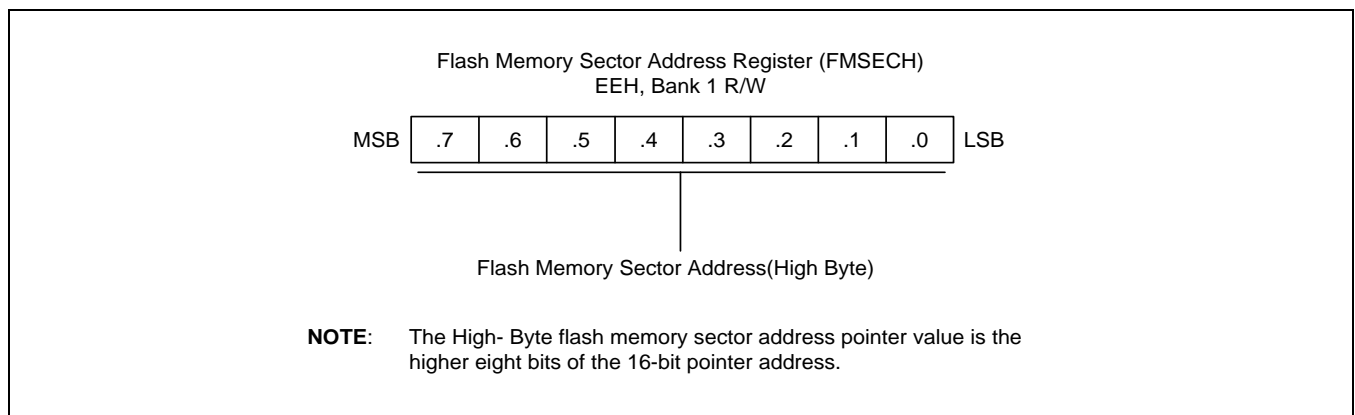
**Figure 18-2 Flash Memory User Programming Enable Register (FMUSR)**

### 18.2.3 Flash Memory Sector Address Registers

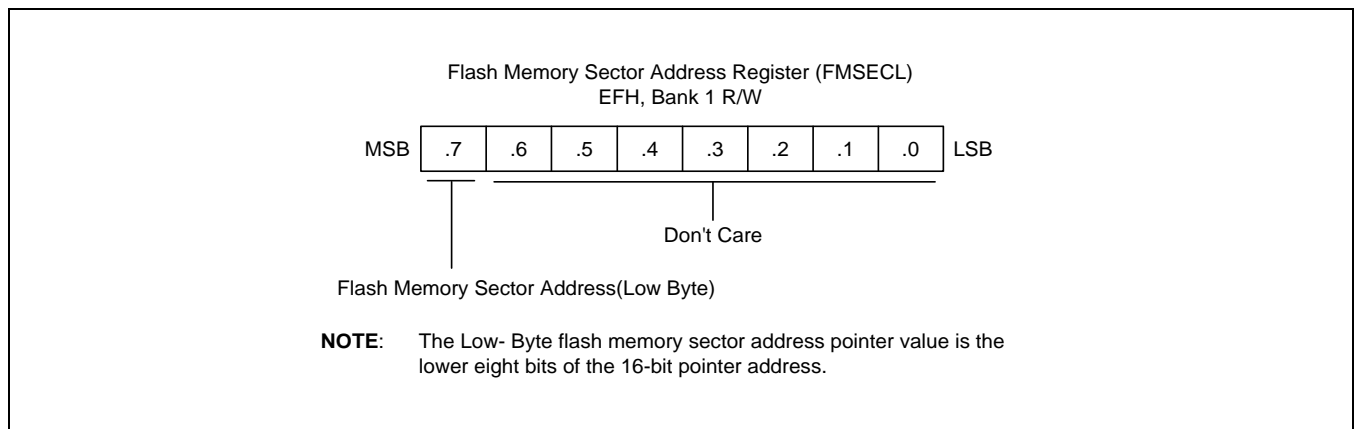
There are two sector address registers for the erase or programming Flash memory. The FMSECL (Flash Memory Sector Address Register Low Byte) indicates the low byte of sector address and FMSECH (Flash Memory Sector Address Register High Byte) indicates the high byte of sector address. The FMSECH is needed for S3F8S28/F8S24 because it has 32 sectors.

One sector consists of 128bytes. Each sector's address starts XX00H or XX80H, that is, a base address of sector is XX00H or XX80H. So bit.6–0 of FMSECL don't mean whether the value is "1" or "0". We recommend that it is the simplest way to load the sector base address into FMSECH and FMSECL register. When programming the Flash memory, user should program after loading a sector base address, which is located in the destination address to write data into FMSECH and FMSECL register. If the next operation is also to write one byte data, user should check whether next destination address is located in the same sector or not. In case of other sectors, user should load sector address to FMSECH and FMSECL Register according to the sector.

(Refer to [Example 18-2 Programming the PWM Module to Sample Specifications](#)).



**Figure 18-3 Flash Memory Sector Address Register (FMSECH)**

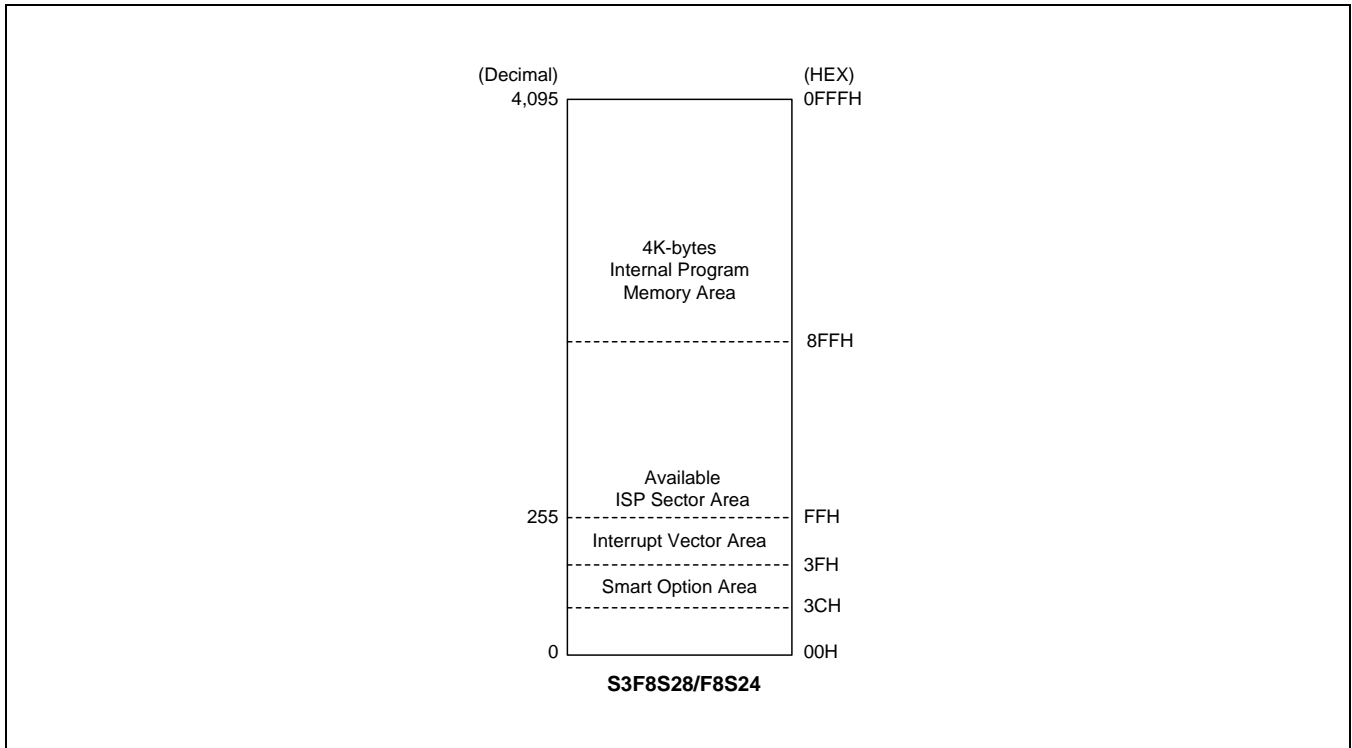


**Figure 18-4 Flash Memory Sector Address Register (FMSECL)**

### 18.3 ISP™ (On-Board Programming) Sector

ISP™ sectors located in program memory area can store on board program software (boot program code for upgrading application code by interfacing with I/O pin). The ISP™ sectors cannot be erased or programmed by LDC instruction for the safety of On Board Program software.

The ISP sectors are available only when the ISP enable/disable bit is set 0, that is, enable ISP at the Smart Option. If you don't like to use ISP sector, this area can be used as a normal program memory (can be erased or programmed by LDC instruction) by setting ISP disable bit ("1") at the Smart Option. Even if ISP sector is selected, ISP sector can be erased or programmed in the Tool Program mode, by Serial programming tools. The size of ISP sector can be varied by settings of Smart Option. You can choose appropriate ISP sector size according to the size of On Board Program software.



**Figure 18-5 Program Memory Address Space**

**Table 18-2 ISP Sector Size**

Smart Option (003EH) ISP Size Selection Bit			Area of ISP Sector	ISP Sector Size
Bit 2	Bit 1	Bit 0		
1	x	x	–	0
0	0	0	100H to 1FFH (256byte)	256bytes
0	0	1	100H to 2FFH (512byte)	512bytes
0	1	0	100H to 4FFH (1024byte)	1024bytes
0	1	1	100H to 8FFH (2048byte)	2048bytes

**NOTE:** The area of the ISP sector selected by Smart Option bit (003EH.2 to 003EH.0) cannot be erased and programmed by LDC instruction in user program mode.

### 18.3.1 ISP Reset Vector and ISP Sector Size

If you use ISP sectors by setting the ISP enable/disable bit to "0" and the Reset Vector Selection bit to "0" at the Smart Option, you can choose the reset vector address of CPU as shown in [Table 18-3](#) by setting the ISP Reset Vector Address Selection bits.

**Table 18-3 Reset Vector Address**

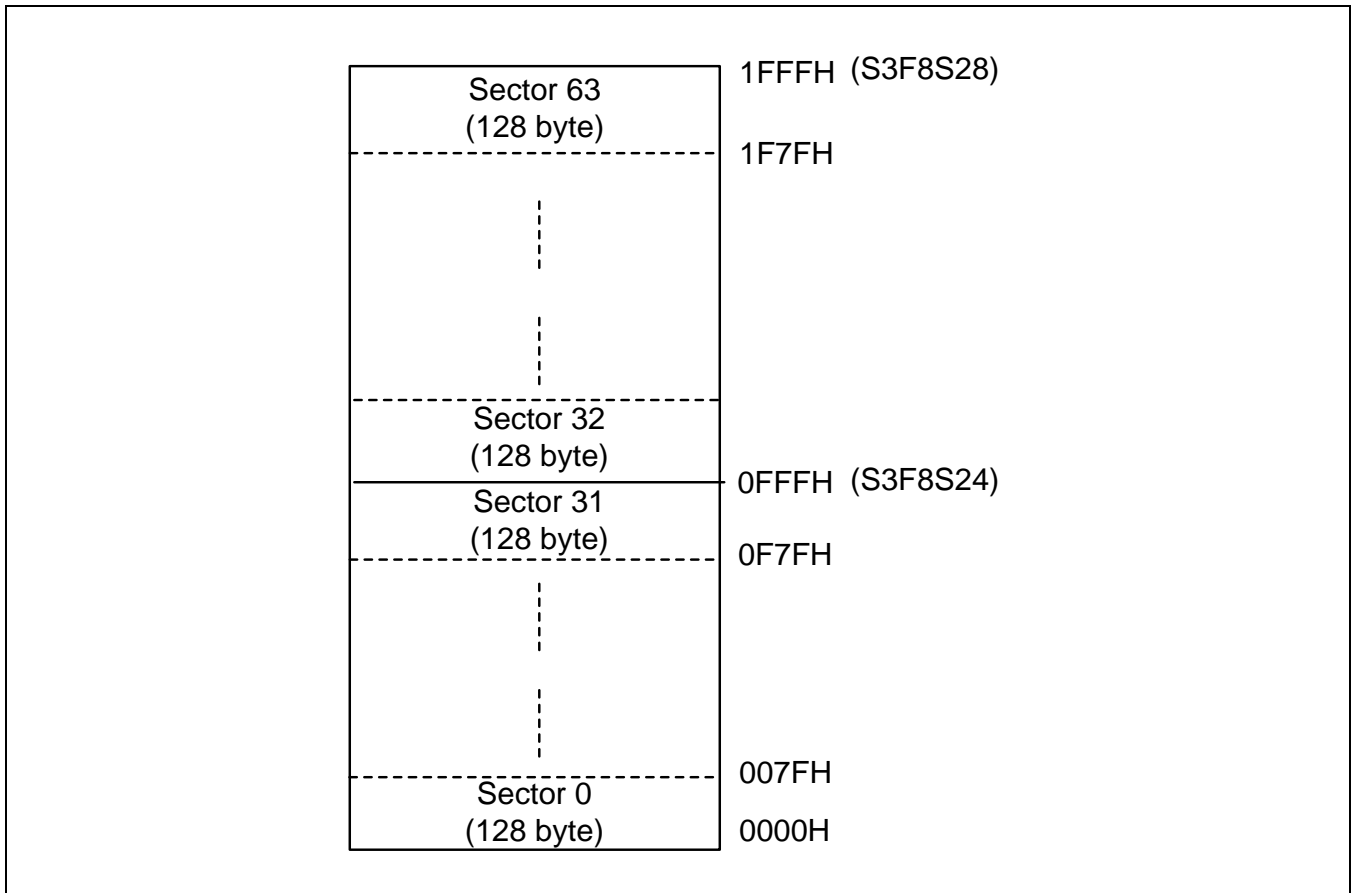
Smart Option (003EH) ISP Reset Vector Address Selection Bit			Reset Vector Address After POR	Usable Area for ISP Sector	ISP Sector Size
Bit 7	Bit 6	Bit 5			
1	x	x	0100H	–	–
0	0	0	0200H	100H to 1FFH	256 bytes
0	0	1	0300H	100H to 2FFH	512 bytes
0	1	0	0500H	100H to 4FFH	1024 bytes
0	1	1	0900H	100H to 8FFH	2048 bytes

**NOTE:** The selection of the ISP reset vector address by Smart Option (003EH.7 to 003EH.5) is not dependent of the selection of ISP sector size by Smart Option (003EH.2 to 003EH.0).

## 18.4 Sector Erase

User can erase a Flash memory partially by using sector erase function only in user program mode. The only unit of Flash memory to be erased in the user program mode is a sector.

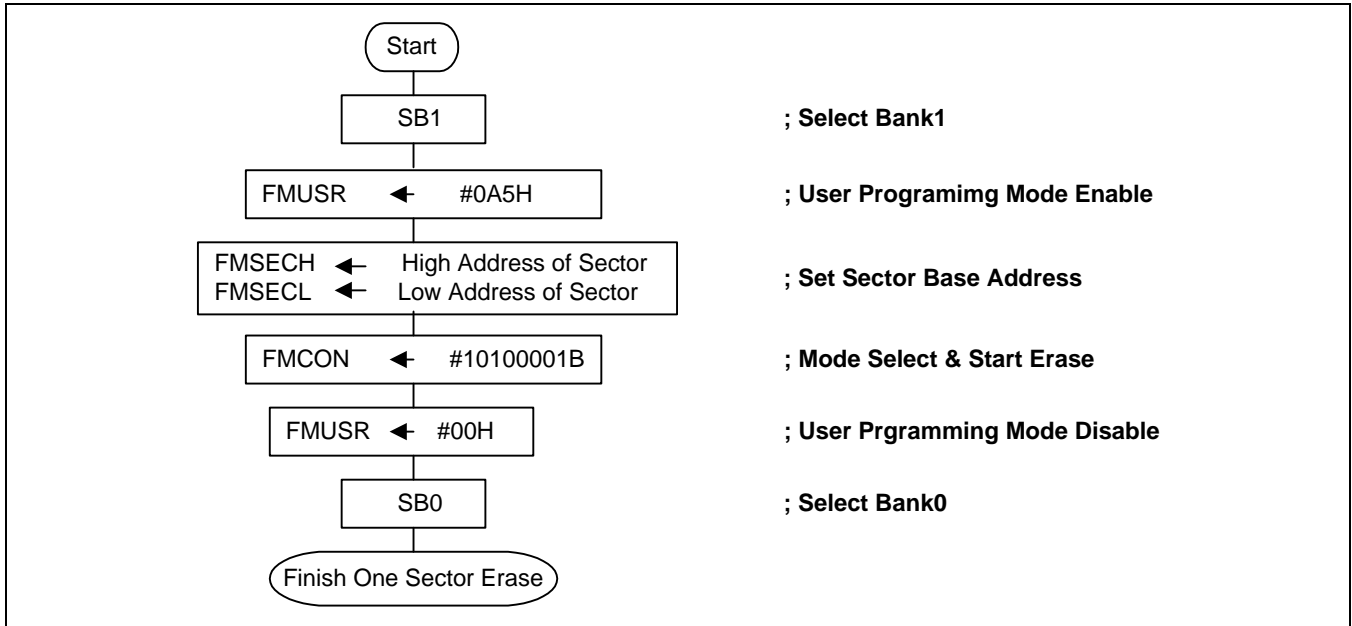
The program memory of S3F8S28/S3F8S24 8K/4Kbytes Flash memory is divided into 64/32 sectors. Every sector has all 128byte sizes. So the sector to be located destination address should be erased first to program a new data (one byte) into Flash memory. Minimum 4ms delay time for the erase is required after setting sector address and triggering erase start bit (FMCON.0).Sector erase is not supported in tool program modes (MDS mode tool or programming tool).



**Figure 18-6 Sector configurations in User Program Mode**

The sector erase procedure in user program mode

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



**Figure 18-7 Sector Erase Flowchart in User Program Mode**

**NOTE:**

1. If user erases a sector selected by Flash Memory Sector Address Register FMSECH and FMSECL, FMUSR should be enabled just before starting sector erase operation. And to erase a sector, Flash Operation Start Bit of FMCON register is written from operation stop "0" to operation start "1". That bit will be cleared automatically just after the corresponding operation completed. In other words, when S3F8S28/S3F8S24 is in the condition that Flash memory user programming enable bits is enabled and executes start operation of sector erase, it will get the result of erasing selected sector as user's a purpose and Flash Operation Start Bit of FMCON register is also clear automatically.
2. If user executes sector erase operation with FMUSR disabled, FMCON.0 bit, Flash Operation Start Bit, remains "High", which means start operation, and is not cleared even though next instruction is executed. So user should be careful to set FMUSR when executing sector erase, for no effect on other Flash sectors.

**Example 18-1 Sector Erase**

```

Case1. Erase one sector
    •
    •
ERASE_ONESECTOR:
    SB1
    LD    FMUSR,#0A5H           ; User program mode enable
    LD    FMSECH,#04H          ; Set sector address 0400H, sector8
    LD    FMSECL,#00H          ; among sector 0 to 32
    LD    FMCON,#10100001B     ; Select erase mode enable & Start sector erase

ERASE_STOP:
    LD    FMUSR,#00H           ; User program mode disable
    SB0
  
```

## 18.5 Programming

A Flash memory is programmed in one-byte unit after sector erase. The write operation of programming starts by "LDC" instruction.

The program procedure in user program mode:

1. Must erase target sectors before programming.
2. Set Flash Memory User Programming Enable Register (FMUSR) to "10100101B".
3. Set Flash Memory Control Register (FMCON) to "0101000XB".
4. Set Flash Memory Sector Address Register (FMSECH and FMSECL) to the sector base address of destination address to write data.
5. Load a transmission data into a working register.
6. Load a Flash memory upper address into upper register of pair working register.
7. Load a Flash memory lower address into lower register of pair working register.
8. Load transmission data to Flash memory location area on "LDC" instruction by indirectly addressing mode
9. Set Flash Memory User Programming Enable Register (FMUSR) to "00000000B".

**NOTE:** In programming mode, it doesn't care whether FMCON.0's value is "0" or "1".

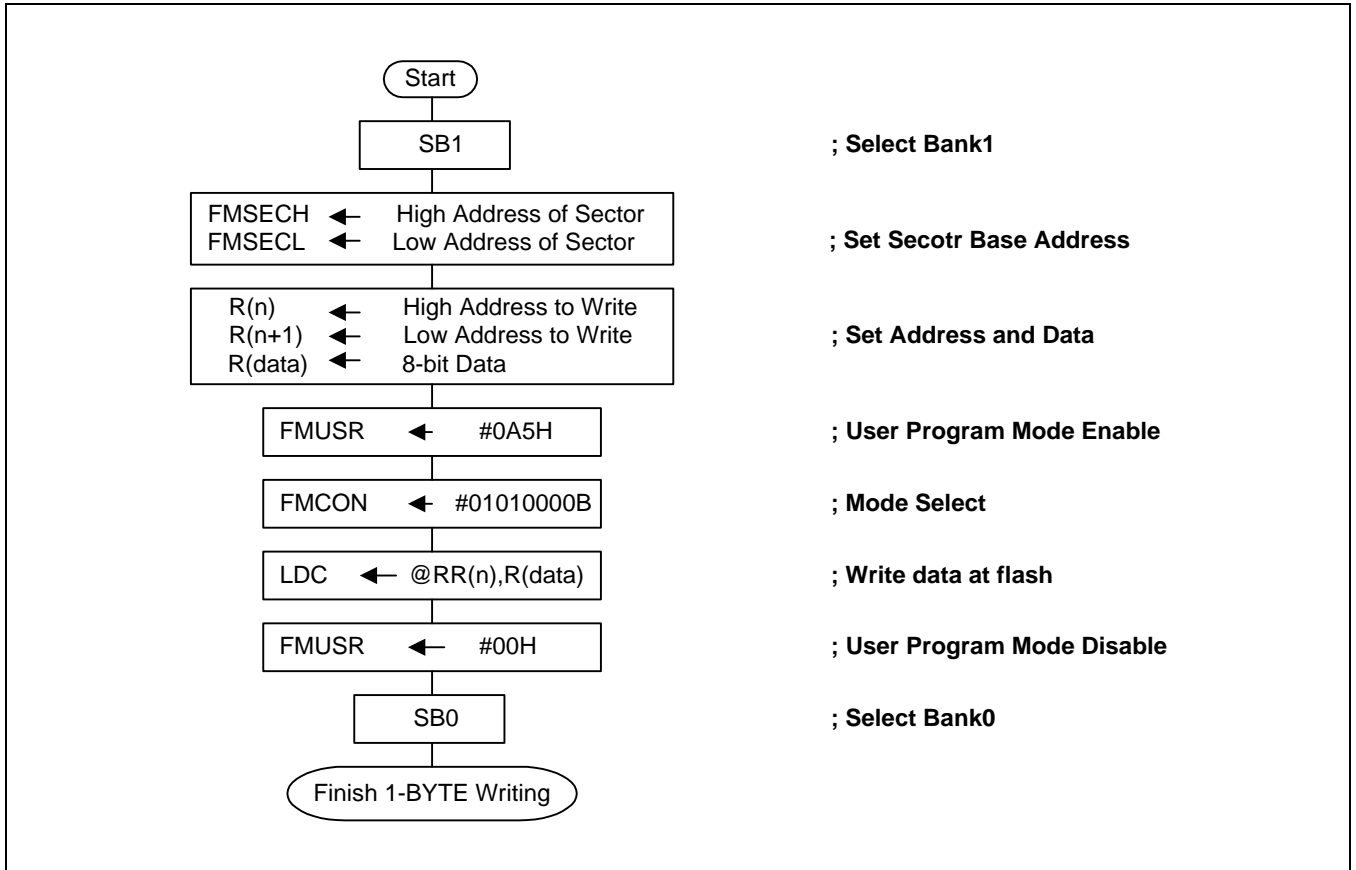


Figure 18-8 Byte Program Flowchart in a User Program Mode



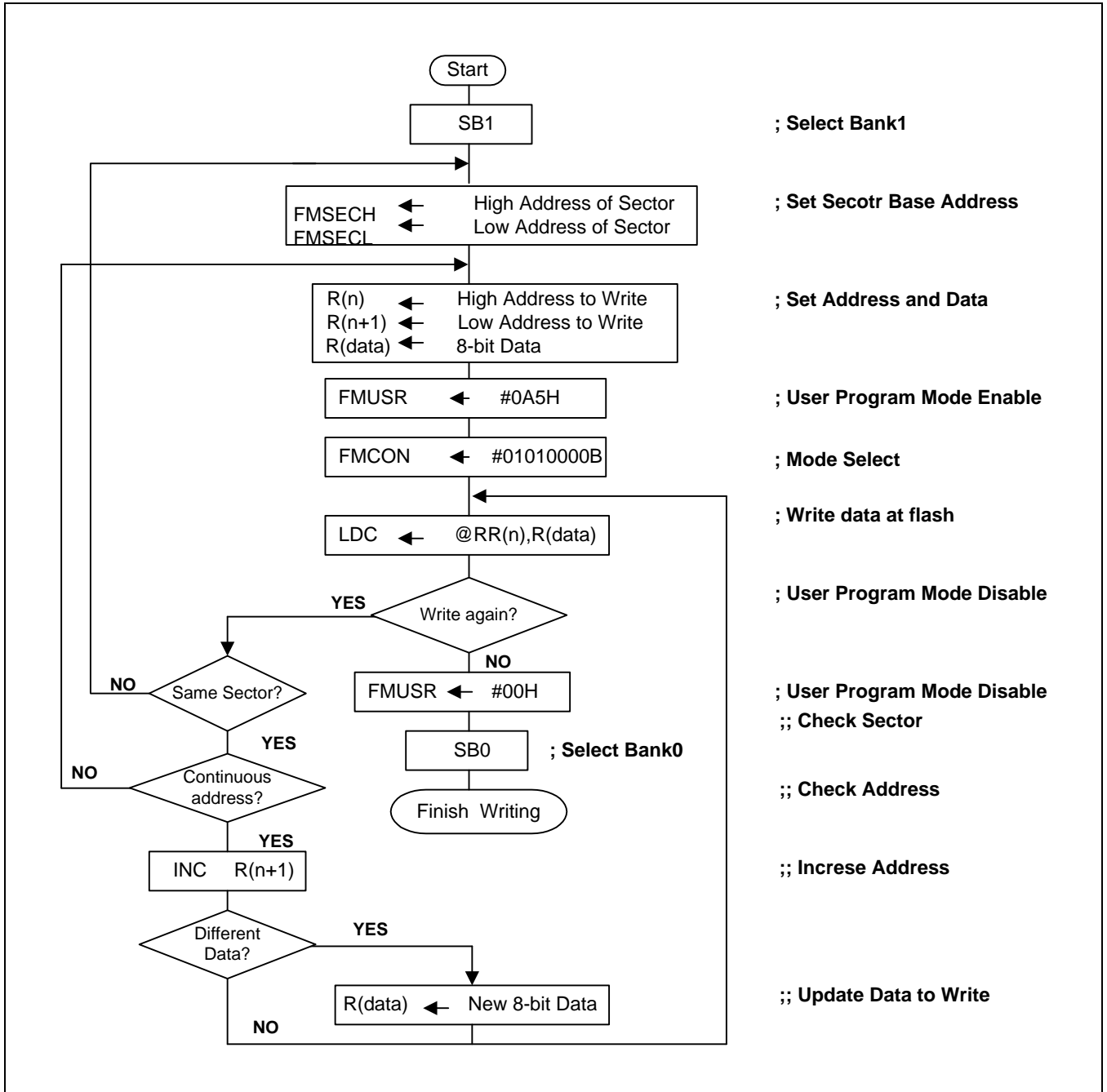


Figure 18-9 Program Flowchart in a User Program Mode

**Example 18-2 1Byte Programming**

**Case1. 1-Byte Programming**

```

•
•
WR_BYTE:                                ; Write data "AAH" to destination address 0310H
    SB1
    LD    FMUSR,#0A5H                    ; User program mode enable
    LD    FMCON,#01010000B              ; Selection programming mode
    LD    FMSECH,#03H                   ; Set the base address of sector (0300H)
    LD    FMSECL,#00H
    LD    R9,#0AAH                      ; Load data "AA" to write
    LD    R10,#03H                      ; Load Flash memory upper address into upper register of
    ; pair working; register
    LD    R11,#10H                      ; Load Flash memory lower address into lower register of
    ; pair working; register
    LDC   @RR10,R9                      ; Write data "AAH" at Flash memory location (0310H)
    LD    FMUSR,#00H                    ; User program mode disable
    SB0

```

**Case2. Programming in the same sector**

```

•
•
WR_INSECTOR:                            ; RR10 → Address copy (R10-high address, R11-low address)
    LD    R0, #40H
    SB1
    LD    FMUSR,#0A5H                    ; User program mode enable
    LD    FMCON,#0101000                ; Selection programming mode and Start programming
    LD    FMSECH,#06H                   ; Set the base address of sector located in target address
    ; to write data
    LD    FMSECL,#00H                   ; The sector 12's base address is 0600H.
    LD    R9,#33H                       ; Load data "33H" to write
    LD    R10,#06H                      ; Load Flash memory upper address into upper register of
    ; pair working register
    LD    R11, #00H                     ; Load Flash memory lower address into lower register of
    ; pair working register

WR_BYTE:
    LDC   @RR10,R9                      ; Write data "33H" at Flash memory location
    INC   R11                            ; Reset address in the same sector by INC instruction
    DEC   R0
    JP    NZ, WR_BYTE                   ; Check whether the end address for programming reach 0640H
    ; or not.
    LD    FMUSR,#00H                    ; User Program mode disable
    SB0

```

**Case3. Programming to the Flash memory space located in other sectors**

```

•
•
WR_INSECTOR2:
    LD    R0, #40H
    LD    R1, #40H
    SBI
    LD    FMUSR, #0A5H           ; User program mode enable
    LD    FMCON, #01010000B     ; Selection programming mode and Start programming
    LD    FMSECH, #01H          ; Set the base address of sector located in target address
                                ; to write data
    LD    FMSECL, #00H          ; The sector 2's base address is 100H
    LD    R9, #0CCH              ; Load data "CCH" to write
    LD    R10, #01H              ; Load Flash memory upper address into upper register of
                                ; pair working register
    LD    R11, #40H              ; Load Flash memory lower address into lower register of
                                ; pair working register
    CALL  WR_BYTE
    LD    R0, #40H
WR_INSECTOR5:
    LD    FMSECH, #02H          ; Set the base address of sector located in target address
to write data
    LD    FMSECL, #80H          ; The sector 5's base address is 0280H
    LD    R9, # 55H              ; Load data "55H" to write
    LD    R10, #02H              ; Load Flash memory upper address into upper
register of
pair working register
    LD    R11, #90H              ; Load Flash memory lower address into lower
register of
pair working register
    CALL  WR_BYTE
WR_INSECTOR12:
    LD    FMSECH, #06H          ; Set the base address of sector located in target address
                                ; to write data
    LD    FMSECL, #00H          ; The sector 12's base address is 0600H
    LD    R9, #0A3H              ; Load data "A3H" to write
    LD    R10, #06H              ; Load Flash memory upper address into upper
register of
                                ; pair working register
    LD    R11, #40H              ; Load Flash memory lower address into lower
register of
                                ; pair working register
WR_BYTE1:
    LDC   @RR10, R9              ; Write data "A3H" at Flash memory location
    INC   R11
    DEC   R1
    JP    NZ, WR_BYTE1
    LD    FMUSR, #00H           ; User Program mode disable
    SBI
•

```

•

```

WR_BYTE:
    LDC    @RR10,R9          ; Write data written by R9 at Flash memory location
    INC    R11
    DEC    R0
    JP     NZ, WR_BYTE
    RET
  
```

### 18.6 Reading

The read operation starts by "LDC" instruction.

The program procedure in user program mode:

1. Load a Flash memory upper address into upper register of pair working register.
2. Load a Flash memory lower address into lower register of pair working register.
3. Load receive data from Flash memory location area on "LDC" instruction by indirectly addressing mode

#### Example 18-3 Reading

```

•
•
LD    R2,#03H          ; Load Flash memory's upper address to upper register of pair working
                        ; register
LD    R3,#00H          ; Load Flash memory's lower address to lower register of pair working
                        ; register

LOOP: LDC    R0,@RR2    ; Read data from Flash memory location (Between 300H and 3FFH)
      INC    R3
      CP     R3,#0FFH
      JP     NZ,LOOP
•
•
•
•
  
```

## 18.7 Hard Lock Protection

User can set Hard Lock Protection by writing "0110B" in FMCON7–4. This function prevents the changes of data in a Flash memory area. If this function is enabled, the user cannot write or erase the data in a Flash memory area. This protection can be released by the chip erase execution in the tool program mode. In terms of user program mode, the procedure of setting Hard Lock Protection is following that. In tool mode, the manufacturer of serial tool writer could support Hardware Protection. Please refer to the manual of serial program writer tool provided by the manufacturer.

The program procedure in user program mode:

1. Set Flash Memory User Programming Enable Register (FMUSR) to "10100101B".
2. Set Flash Memory Control Register (FMCON) to "01100001B".
3. Set Flash Memory User Programming Enable Register (FMUSR) to "00000000B".

### Example 18-4 Hard Lock Protection

```

•
•
SB1
LD    FMUSR, #0A5H           ; User program mode enable
LD    FMCON, #01100001B     ; Select Hard Lock Mode and Start protection
LD    FMUSR, #00H           ; User program mode disable
SB0
•
•

```

# 19 Electrical Data

## 19.1 Overview

In this section, the following S3F8S28/S3F8S24 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
- UART Timing Characteristics
- A/D converter electrical characteristics
- LVD circuit characteristics
- LVR circuit characteristics
- LVR reset timing

**Table 19-1 Absolute Maximum Ratings**

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

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	$V_{DD}$	–	– 0.3 to + 6.5	V
Input voltage	$V_I$	All ports	– 0.3 to $V_{DD} + 0.3$	V
Output voltage	$V_O$	All output ports	– 0.3 to $V_{DD} + 0.3$	V
Output current high	$I_{OH}$	One I/O pin 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	$T_A$	–	– 40 to + 85	$^\circ\text{C}$
Storage temperature	$T_{STG}$	–	– 65 to + 150	$^\circ\text{C}$

**Table 19-2 DC Electrical Characteristics**

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

Parameter	Symbol	Conditions		Min.	Typ.	Max.	Unit
Operating voltage	$V_{DD}$	$f_{HGmain} = 0.4 - 4\text{MHz}$		1.8	–	5.5	V
		$f_{HGmain} = 0.4 - 12\text{MHz}$		2.7	–	5.5	
		$f_{LGmain} = 0.1 - 1\text{MHz}$		1.8	–	5.5	
HG main crystal or ceramic frequency	$f_{HGmain}$	$V_{DD} = 2.7\text{V}$ to $5.5\text{V}$		0.4	–	12	MHz
		$V_{DD} = 1.8\text{V}$ to $5.5\text{V}$		0.4	–	4	
LG Main crystal or ceramic frequency	$f_{LGmain}$	$V_{DD} = 1.8\text{V}$ to $5.5\text{V}$		0.4	–	1	
Input high voltage	$V_{IH1}$	Ports 0, 1, 2 and RESET	$V_{DD} = 1.8$ to $5.5\text{V}$	$0.8 V_{DD}$	–	$V_{DD}$	V
	$V_{IH2}$	$X_{IN}$ and $X_{OUT}$		$V_{DD} - 0.1$			
Input low voltage	$V_{IL1}$	Ports 0, 1, 2 and RESET	$V_{DD} = 1.8$ to $5.5\text{V}$	–	–	$0.2V_{DD}$	V
	$V_{IL2}$	$X_{IN}$ and $X_{OUT}$				0.1	
Output high voltage	$V_{OH}$	$I_{OH} = -10\text{mA}$ Ports 0,2,P1.0-P1.1	$V_{DD} = 4.5$ to $5.5\text{V}$	$V_{DD} - 1.5$	$V_{DD} - 0.4$	–	V
Output low voltage	$V_{OL}$	$I_{OL} = 25\text{mA}$ Ports 0,2,P1.0-P1.1	$V_{DD} = 4.5$ to $5.5\text{V}$	–	0.4	2.0	V
Input high leakage current	$I_{LIH1}$	All input except $I_{LIH2}$ , P1.21	$V_{IN} = V_{DD}$	–	–	1	$\mu\text{A}$
	$I_{LIH2}$	$X_{IN}$	$V_{IN} = V_{DD}$			20	
Input low leakage current	$I_{LIL1}$	All input except $I_{LIL2}$	$V_{IN} = 0\text{V}$	–	–	–1	$\mu\text{A}$
	$I_{LIL2}$	$X_{IN}$	$V_{IN} = 0\text{V}$			–20	
Output high leakage current	$I_{LOH}$	All output pins	$V_{OUT} = V_{DD}$	–	–	2	$\mu\text{A}$
Output low leakage current	$I_{LOL}$	All output pins	$V_{OUT} = 0\text{V}$	–	–	–2	$\mu\text{A}$
Pull-up resistors	$R_{P1}$	$V_{IN} = 0\text{V}$ , $T_A = 25^\circ\text{C}$ Ports 0, 1, 2	$V_{DD} = 5\text{V}$	25	50	100	k $\Omega$
Pull-down resistors	$R_{P2}$	$V_{IN} = 0\text{V}$ , $T_A = 25^\circ\text{C}$ P1.0-P1.11	$V_{DD} = 5\text{V}$	25	50	100	
Supply current (2)	$I_{DD1}$	Run mode 10MHz CPU clock HG oscillator mode	$V_{DD} = 4.5$ to $5.5\text{V}$	–	2	4	mA
		Run mode 0.5MHz CPU clock LG oscillator mode	$V_{DD} = 3.0\text{V}$	–	–	0.2	

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
	I <sub>DD2</sub>	Idle mode 10MHz clock HG oscillator mode	V <sub>DD</sub> = 4.5 to 5.5V	-	1.5	3.0
	V <sub>DD</sub> = 2.0V	-	1.2	2.4		
	Stop mode	V <sub>DD</sub> = 3.0 (LVR disable) T <sub>A</sub> = 25°C	-	0.3	1.0	
		V <sub>DD</sub> = 4.5 to 5.5V (LVR disable) T <sub>A</sub> = 25°C		0.3	2.0	
		V <sub>DD</sub> = 4.5 to 5.5V (LVR disable) T <sub>A</sub> = - 40°C to + 85°C		1.0	4.0	
		V <sub>DD</sub> = 4.5 to 5.5V (LVR enable) T <sub>A</sub> = - 40°C to + 85°C		40	80	

**NOTE:**

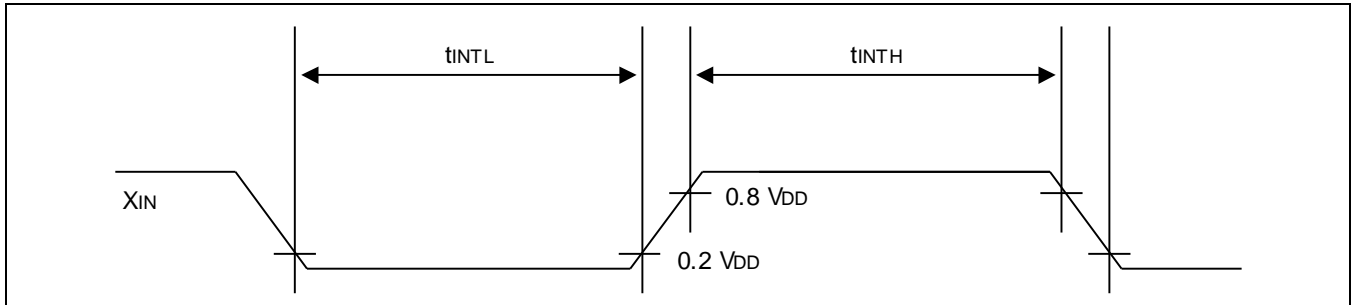
1. P1.2 have intrinsic internal pull-down resistor (Internal VPP circuit), the typical value is about 300Kohm.
2. Supply current does not include current drawn through internal pull-up resistors or external output current loads and ADC module.

**Table 19-3 AC Electrical Characteristics**

(T<sub>A</sub> = - 40°C to + 85°C, V<sub>DD</sub> = 1.8V to 5.5V)

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Interrupt input low width	t <sub>INTL</sub>	INT0, INT1 V <sub>DD</sub> = 5 V ± 10 %	500	-	-	ns
RESET input low width	t <sub>RSL</sub>	Input V <sub>DD</sub> = 5 V ± 10 %	10	-	-	us





**Figure 19-1 Input Timing Measurement Points**

**Table 19-4 Oscillator Characteristics**

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

Oscillator	Clock Circuit	Test Condition	Min.	Typ.	Max.	Unit
Main crystal or ceramic		$V_{DD} = 2.7$ to $5.5\text{V}$ HG oscillator mode	0.4	–	121	MHz
		$V_{DD} = 1.8$ to $5.5\text{V}$ HG oscillator mode	0.4	–	4	MHz
		$V_{DD} = 1.8$ to $5.5\text{V}$ LG oscillator mode	0.1	–	1	MHz
External clock (Main System)		$V_{DD} = 2.7$ to $5.5\text{V}$	0.4	–	12	MHz
		$V_{DD} = 1.8$ to $5.5\text{V}$	0.4	–	4	MHz
Internal RC oscillator	–	$V_{DD} = 5\text{V}$ $T_A = 25^\circ\text{C}$ , Tolerance:1%	7.92	8	8.08	MHz
			3.96	4	4.04	
			1.98	2	2.02	
			0.99	1	1.01	kHz
495	500	505				
Tolerance of internal RC	–	$V_{DD} = 1.8$ to $5.0\text{V}$ $T_A = 25^\circ\text{C}$	–	$\pm 0.5$	$\pm 1$	%
		$V_{DD} = 1.8$ to $5.5\text{V}$ $T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$	–	–	$\pm 3.5$	%
Internal ring OSC	–	$V_{DD} = 5\text{V}$ , $T_A = 25^\circ\text{C}$ , Run Mode (After trimming)	16.384	32.768	49.152	kHz
	–	$V_{DD} = 5\text{V}$ , $T_A = 25^\circ\text{C}$ , Stop Mode (After trimming)	–	24	40.2	

**NOTE:**

1. Please refer to the figure of Operating Voltage Range.
2. Ring OSC frequency will decrease in Stop Mode while VDD is not changed.

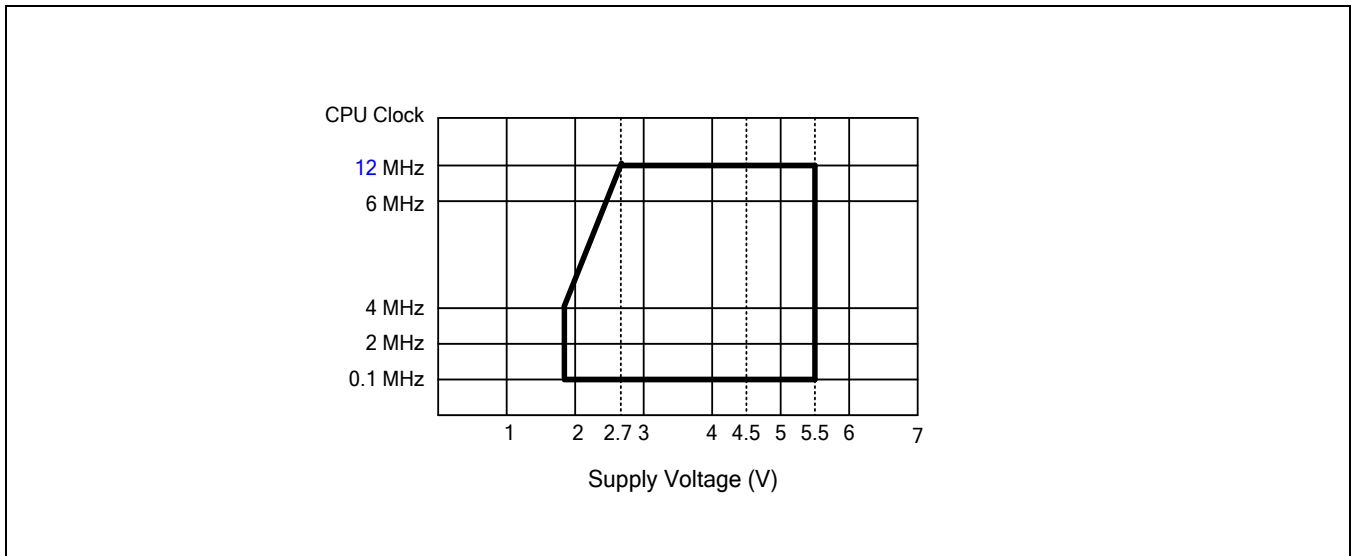
**Table 19-5 Oscillation Stabilization Time**

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

Oscillator	Test Condition	Min.	Typ.	Max.	Unit
Main crystal	$f_{\text{OSC}} > 1.0\text{MHz}$ Oscillation stabilization occurs when $V_{DD}$ is equal to the minimum oscillator voltage range.	–	–	20	ms
Main ceramic		–	–	10	ms
External clock (main system)	$X_{\text{IN}}$ input high and low width ( $t_{\text{XH}}$ , $t_{\text{XL}}$ )	25	–	500	ns
Oscillator stabilization wait time	$t_{\text{WAIT}}$ when released by a reset (1)	–	$2^{19}/f_{\text{OSC}}$	–	ms
	$t_{\text{WAIT}}$ when released by an interrupt (2)	–	–	–	ms

**NOTE:**

- $f_{\text{OSC}}$  is the oscillator frequency.
- The duration of the oscillator stabilization wait time,  $t_{\text{WAIT}}$ , when it is released by an interrupt is determined by the settings in the basic timer control register, BTCON.



**Figure 19-2 Operating Voltage Range**

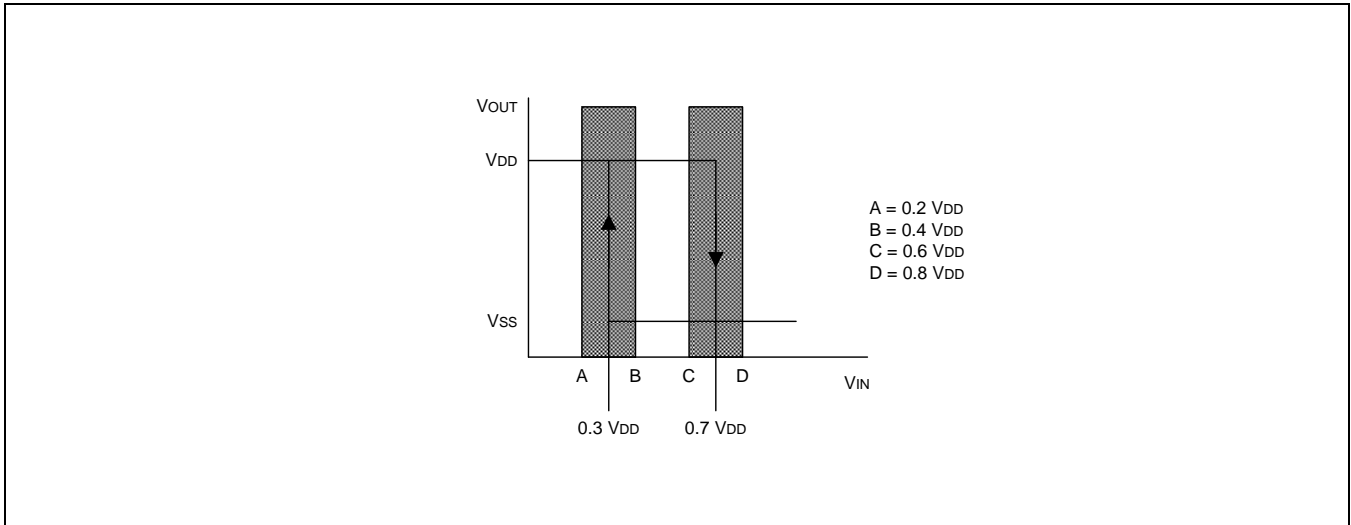


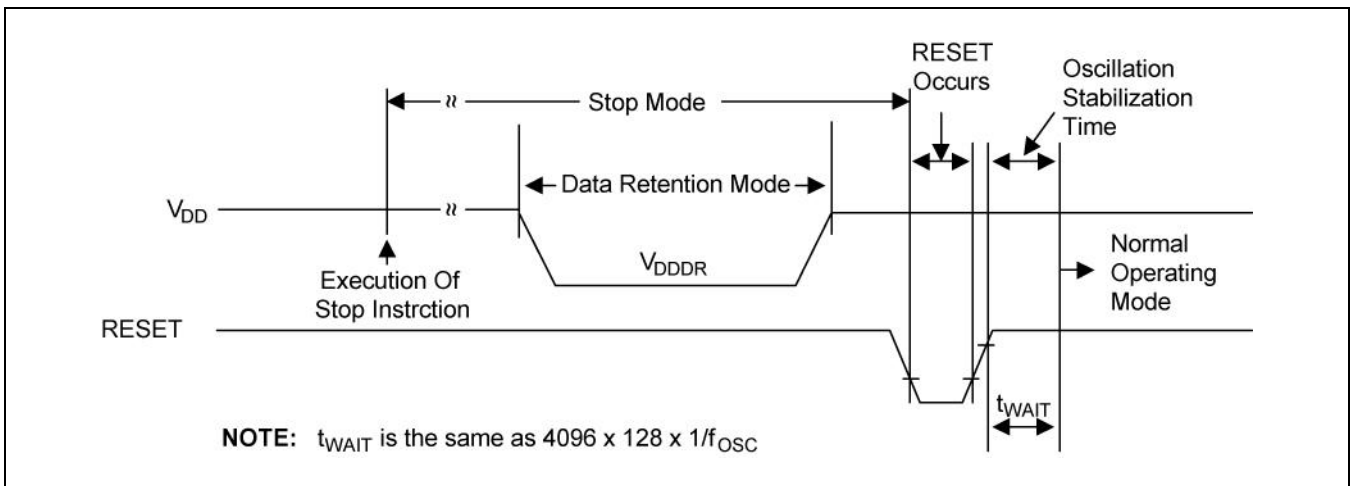
Figure 19-3 Schmitt Trigger Input Characteristics Diagram

Table 19-6 Data Retention Supply Voltage in Stop Mode

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

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; $V_{DDDR} = 2.0\text{V}$	–	0.1	5	$\mu\text{A}$

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



**NOTE:**  $t_{WAIT}$  is the same as  $4096 \times 128 \times 1/f_{OSC}$

Figure 19-4 Stop Mode Release Timing When Initiated by a RESET

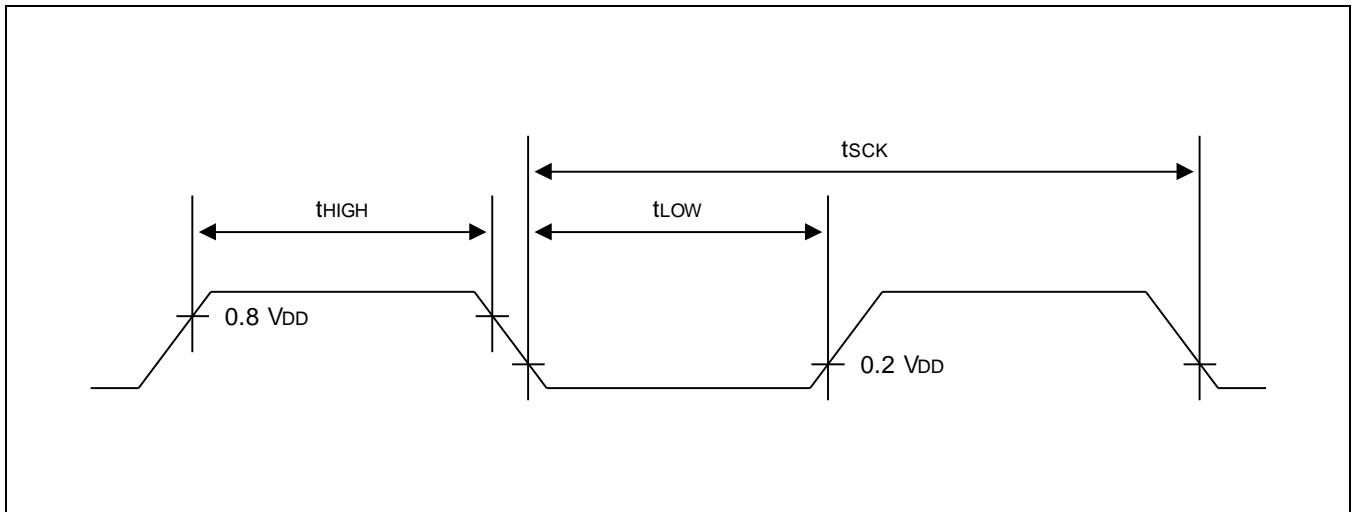
**Table 19-7 UART Timing Characteristics in Mode 0 (10MHz)**

( $T_A = -40^{\circ}\text{C}$  to  $+85^{\circ}\text{C}$ ,  $V_{DD} = 1.8\text{V}$  to  $5.5\text{V}$ , Load capacitance =  $80\text{pF}$ )

Parameter	Symbol	Min.	Typ.	Max.	Unit
Serial port clock cycle time	$t_{\text{SCK}}$	500	$t_{\text{CPU}} \times 6$	700	ns
Output data setup to clock rising edge	$t_{\text{S1}}$	300	$t_{\text{CPU}} \times 5$	–	
Clock rising edge to input data valid	$t_{\text{S2}}$	–	–	300	
Output data hold after clock rising edge	$t_{\text{H1}}$	$t_{\text{CPU}} - 50$	$t_{\text{CPU}}$	–	
Input data hold after clock rising edge	$t_{\text{H2}}$	0	–	–	
Serial port clock High, Low level width	$t_{\text{HIGH}}, t_{\text{LOW}}$	200	$t_{\text{CPU}} \times 3$	400	

**NOTE:**

1. All timings are in nanoseconds (ns) and assume a 10MHz CPU clock frequency
2. The unit  $t_{\text{CPU}}$  means one CPU clock period.



**Figure 19-5 Waveform for UART Timing Characteristics**

**Table 19-8 A/D Converter Electrical Characteristics**

( $T_A = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ ,  $V_{DD} = 2.7\text{V}$  to  $5.5\text{V}$ ,  $V_{SS} = 0\text{V}$ )

Parameter	Symbol	Test Conditions	Min.	Typ.	Max.	Unit
Resolution	–	–	–	12	–	bit
Total accuracy	–	$V_{DD} = 5.12\text{V}$ $f_{ADC} = 850\text{kHz}$ $V_{SS} = 0\text{V}$	– 3	1	6	LSB
Integral linearity error	ILE	"	–	0.8	$\pm 2$	
Differential linearity error	DLE	"	–	0.6	$\pm 1$	
Offset error of top	EOT	"	– 3	1	6	
Offset error of bottom	EOB	"	– 3	1	6	
Conversion time (1)	$t_{CON}$	$f_{OSC} = 10\text{MHz}$ $f_{ADC} = f_{OSC} / 12 = 10\text{MHz}/12$	–	20.4	–	$\mu\text{s}$
Sampling time	$t_{SMP}$	$f_{OSC} = 10\text{MHz}$ $f_{ADC} = f_{OSC} / 12 = 10\text{MHz}/12$	–	5	–	$1/f_{ADC}$
ADC clock input	$f_{ADC}$	–	–	–	850	kHz
Analog input voltage	$V_{IAN}$	–	$V_{SS}$	–	$V_{DD}$	V
Analog input impedance	$R_{AN}$	–	2	–	–	$\text{M}\Omega$
Analog input current	$I_{ADIN}$	$V_{DD} = 5\text{V}$	–	–	10	$\mu\text{A}$
Analog block current (2)	$I_{ADC}$	$V_{DD} = 5\text{V}$	–	1	3	mA
		$V_{DD} = 3\text{V}$	–	0.5	1.5	–
		$V_{DD} = 5\text{V}$ power down mode	–	100	500	nA

**NOTE:**

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 19-9 LVD Circuit Characteristics**

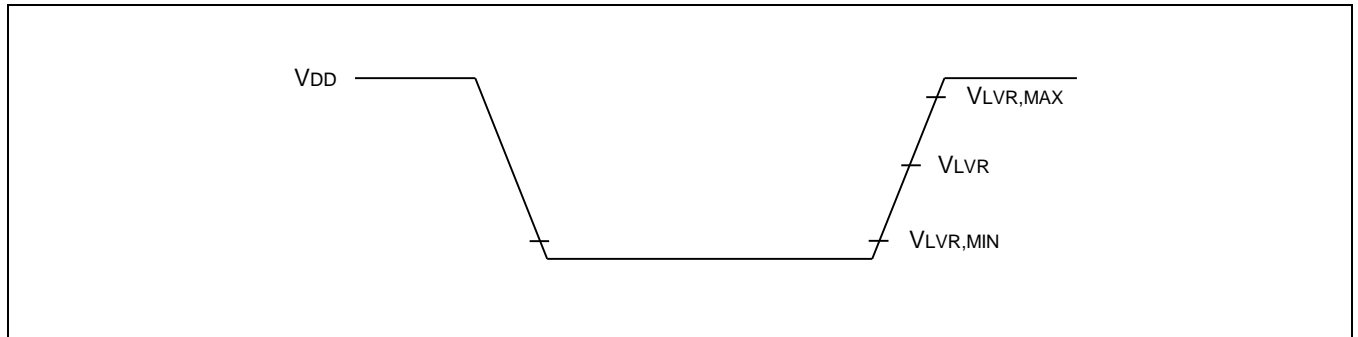
( $T_A = -40$  to  $85^\circ\text{C}$ ,  $V_{DD} = 1.8\text{V}$  to  $5.5\text{V}$ )

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
LVD Detect Voltage	$V_{LVD0}$	–	4.0	4.1	4.2	V
	$V_{LVD1}$	–	3.1	3.2	3.3	
	$V_{LVD2}$	–	2.4	2.5	2.6	
	$V_{LVD3}$	–	2.0	2.1	2.2	

**Table 19-10 LVR Circuit Characteristics**

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

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Low voltage reset	$V_{LVR}$	–	1.8	1.9	2.0	V
			2.2	2.3	2.4	
			2.9	3.0	3.1	
			3.8	3.9	4.0	



**Figure 19-6 LVR Reset Timing**

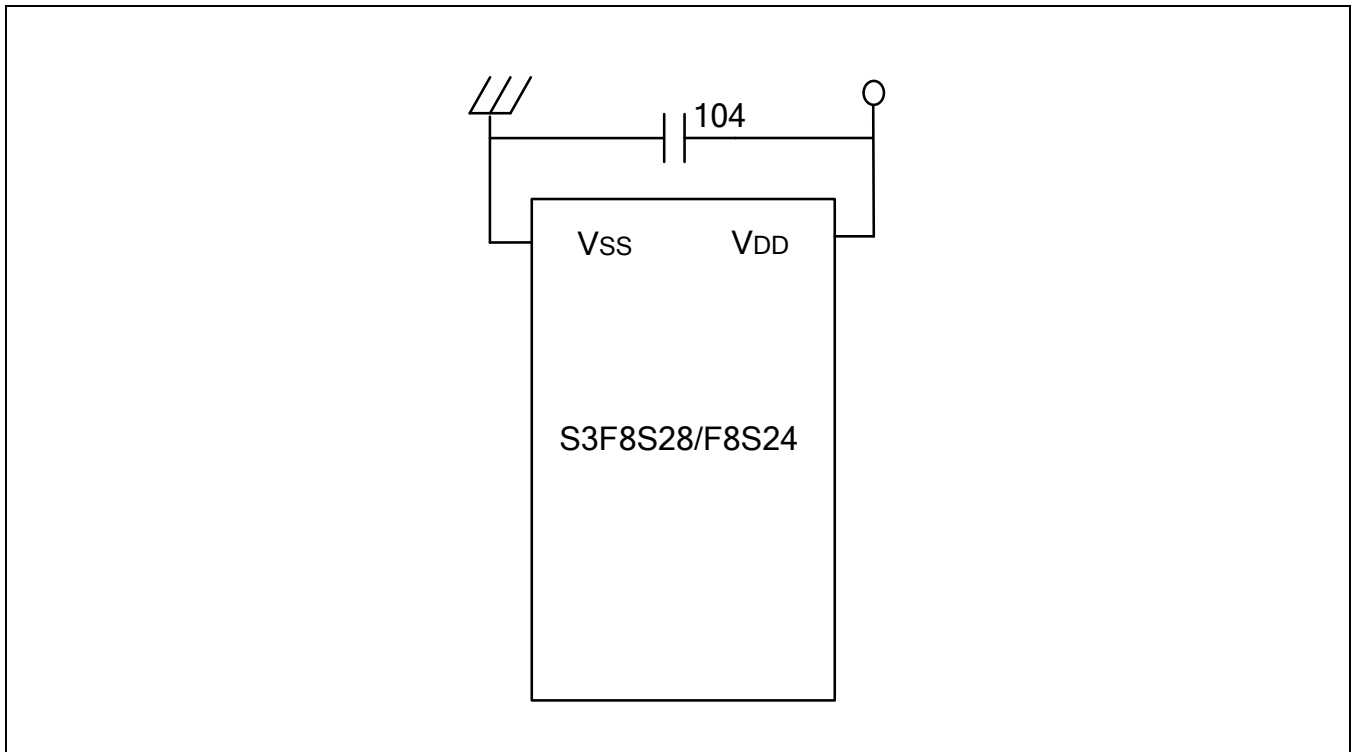
**Table 19-11 Flash Memory AC Electrical characteristics**

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

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Flash erase/write/read voltage	Fewrv	VDD	1.8	5.0	5.5	V
Programming time <sup>(1)</sup>	Ftp	-	20	-	30	$\mu\text{S}$
Chip erasing time <sup>(2)</sup>	Ftp1		32	-	70	mS
Sector erasing time <sup>(3)</sup>	Ftp2		4	-	12	mS
Data access time	FtRS	VDD = 2.0V	-	250	-	nS
Number of writing/erasing	FNwe	-	10,000	-	-	Times
Data retention	Ftdr	-	10	-	-	Years

**NOTE:**

1. The programming time is the time during which one byte (8-bit) is programmed.
2. The chip erasing time is the time during which entire program memory is erased.
3. The sector erasing time is the time during which all 128byte block is erased.
4. The chip erasing is available in tool program mode only.



**Figure 19-7 The Circuit Diagram to Improve EFT Characteristics**

**NOTE:** To improve EFT characteristics, we recommend using power capacitor near S3F8S28/F8S24 like [Figure 19-7](#).

**Table 19-12 ESD Characteristics**

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Electrostatic discharge	V <sub>ESD</sub>	HBM	2000	–	–	V
		MM	200	–	–	V
		CDM	500	–	–	V



# 20 Mechanical Data

## 20.1 Overview

The S3F8S28/S3F8S24 is available in a 24-pin SOP package (Zilog: 24-SOP-375), a 24-pin TSSOP package (Zilog: 24-TSSOP-BD44), a 20-pin DIP package (Zilog: 20-DIP-300A), a 20-pin SOP package (Zilog: 20-SOP-375), a 20-pin SSOP package (Zilog: 20-SSOP-225). Package dimensions are shown in [Figure 20-1](#), [Figure 20-2](#), [Figure 20-3](#), and [Figure 20-4](#).

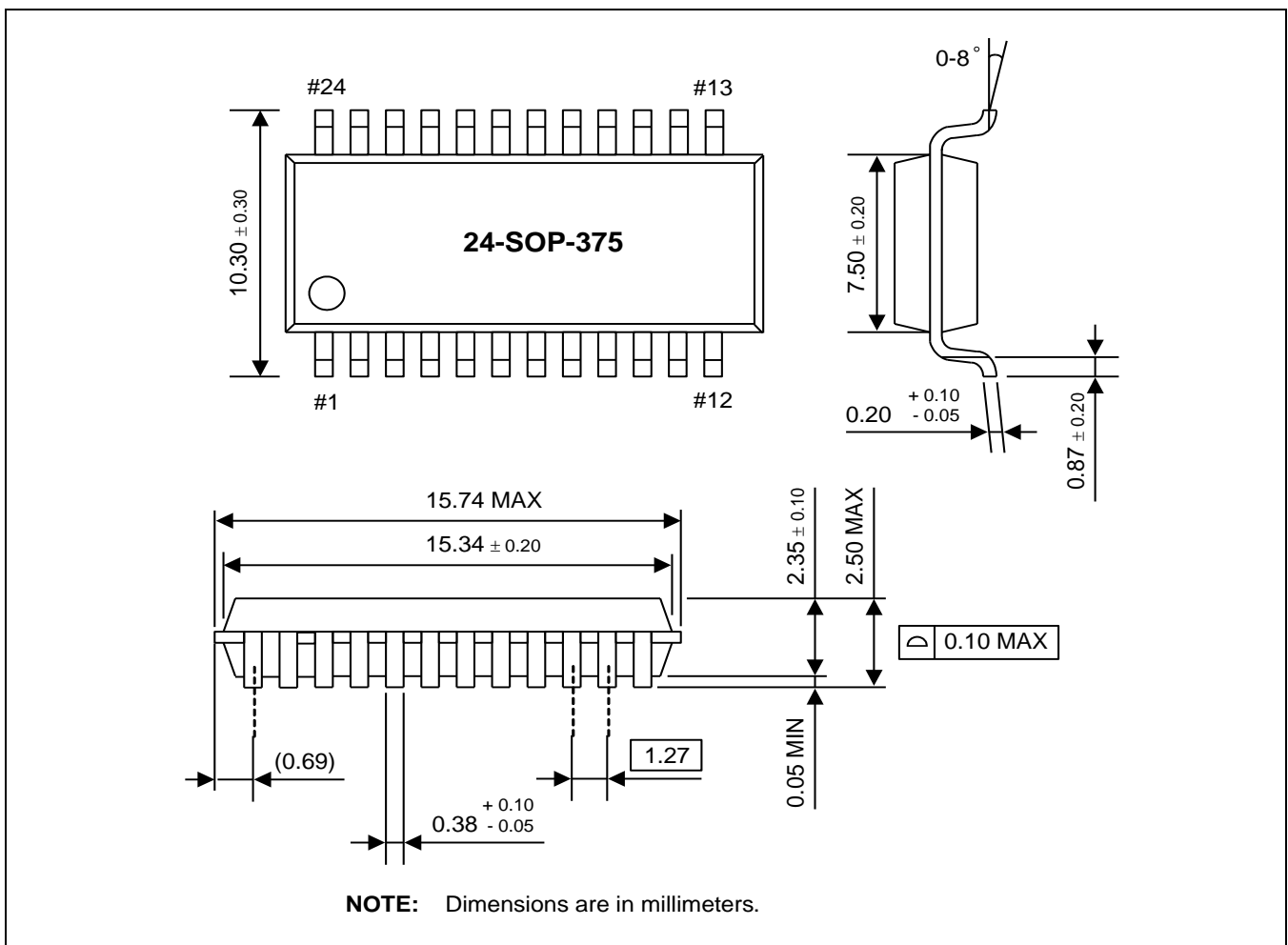


Figure 20-1 24-SOP-375 Package Dimensions

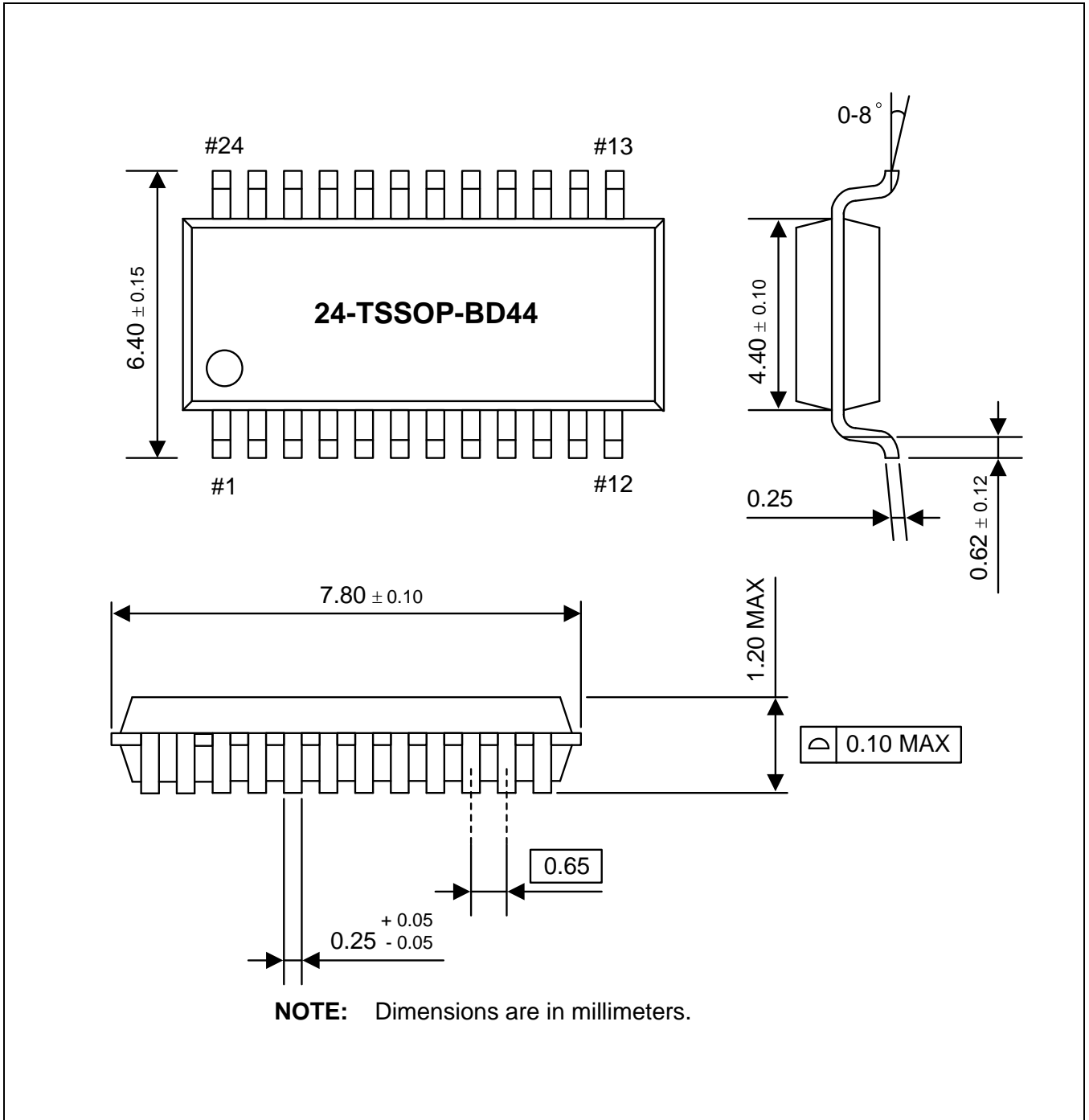


Figure 20-2 24-TSSOP-BD44 Package Dimensions

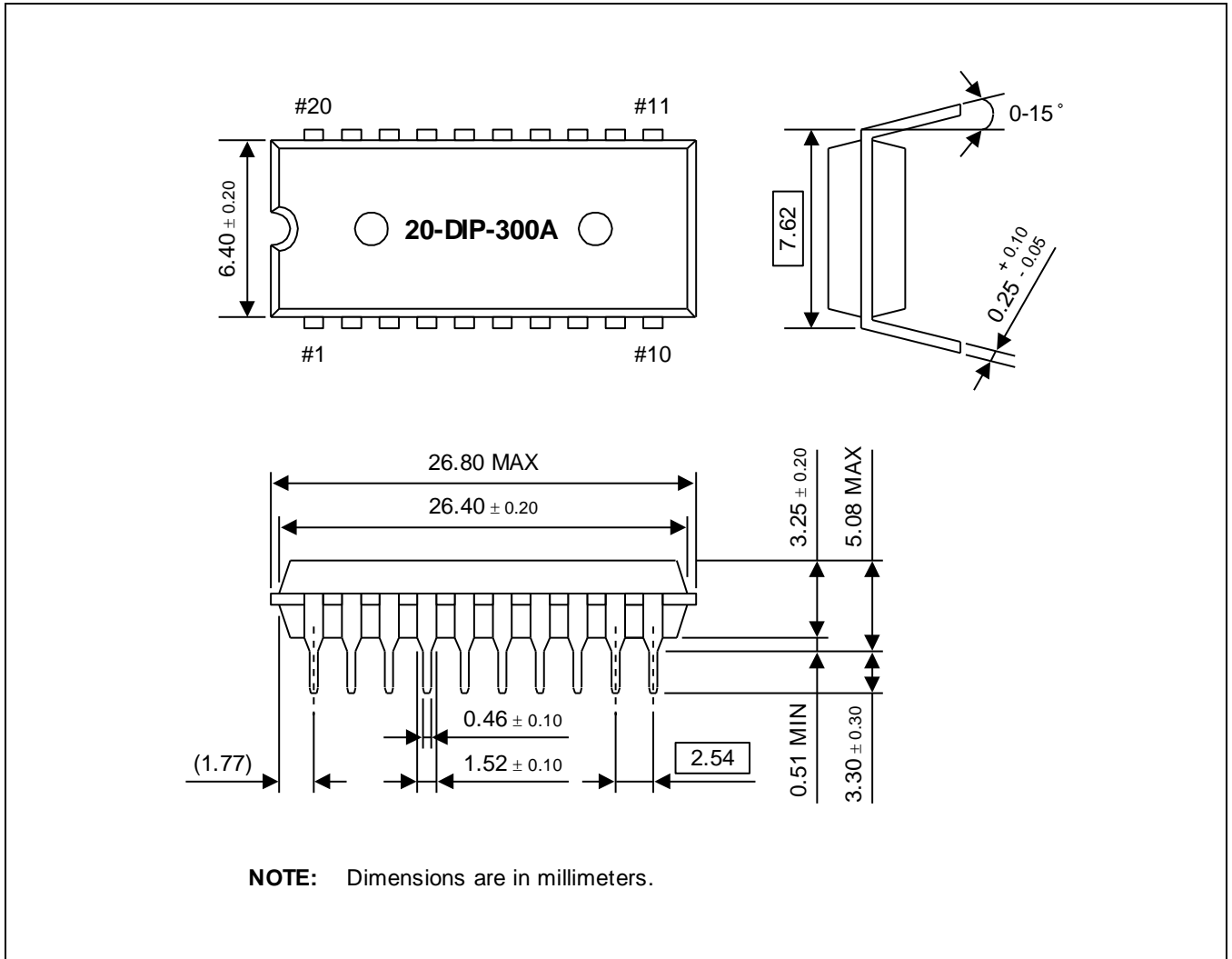


Figure 20-3 20-DIP-300A Package Dimensions

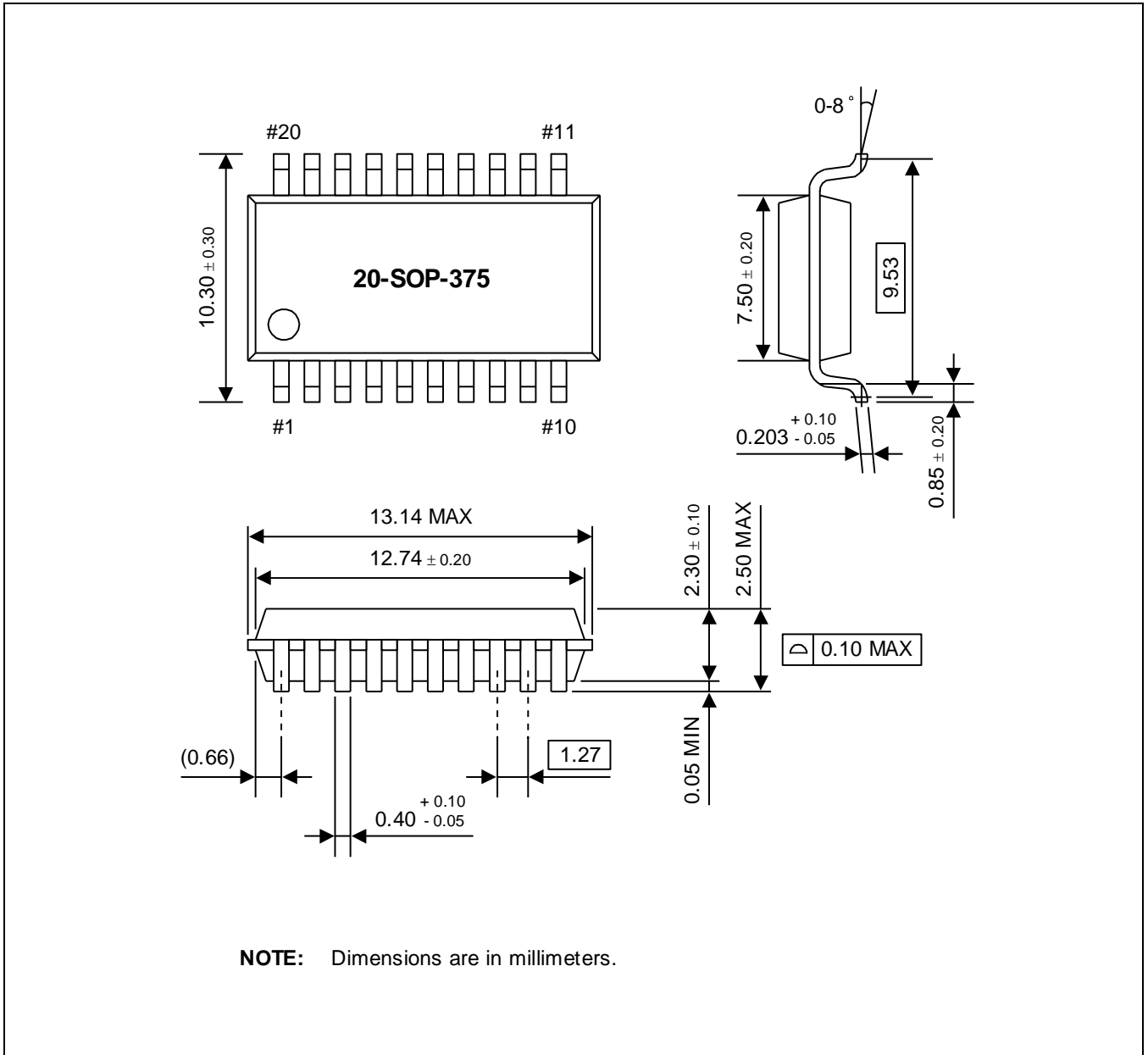


Figure 20-4 20-SOP-375 Package Dimensions

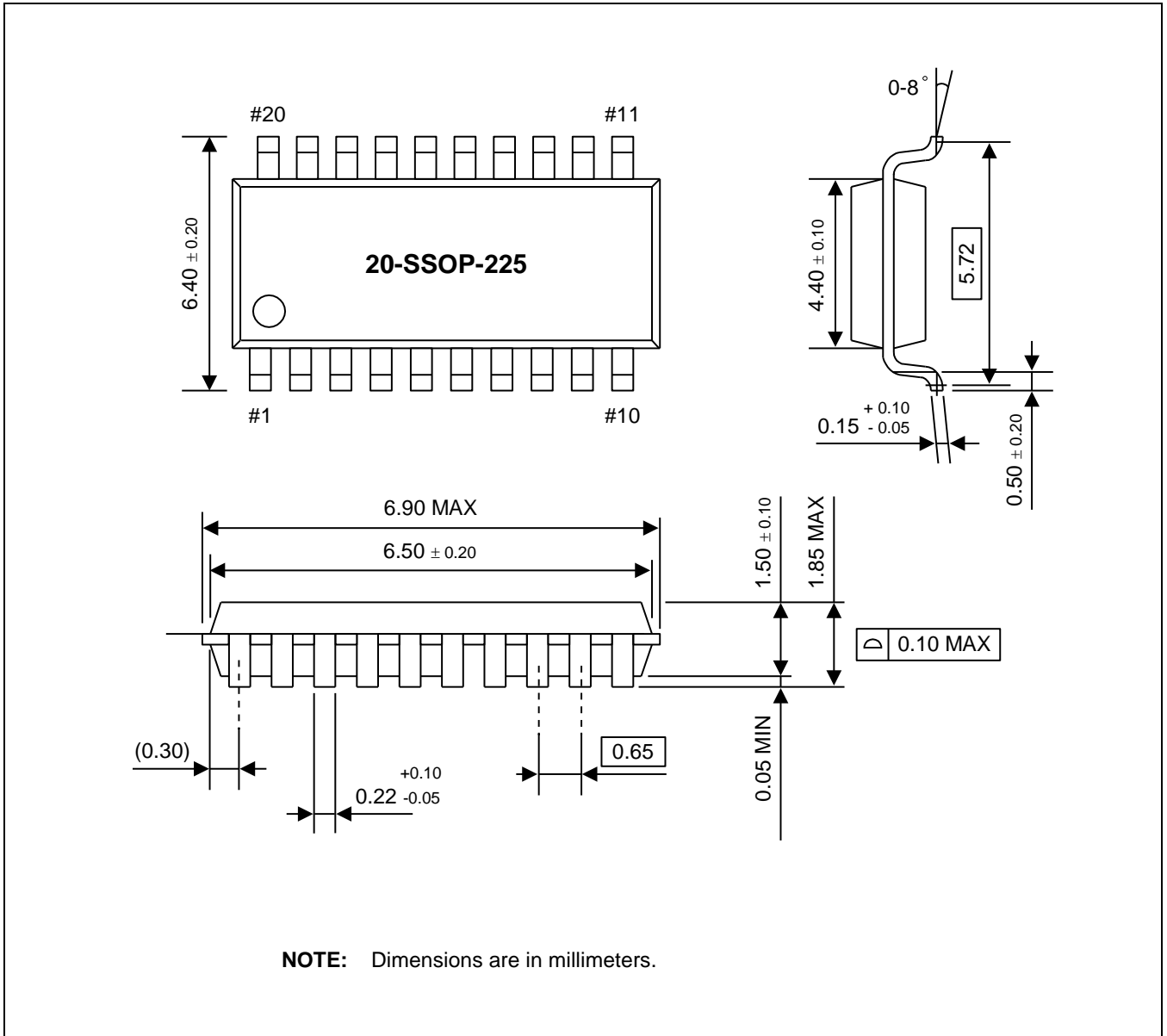


Figure 20-5 20-SSOP-225 Package Dimensions

# 21

## Flash MCU

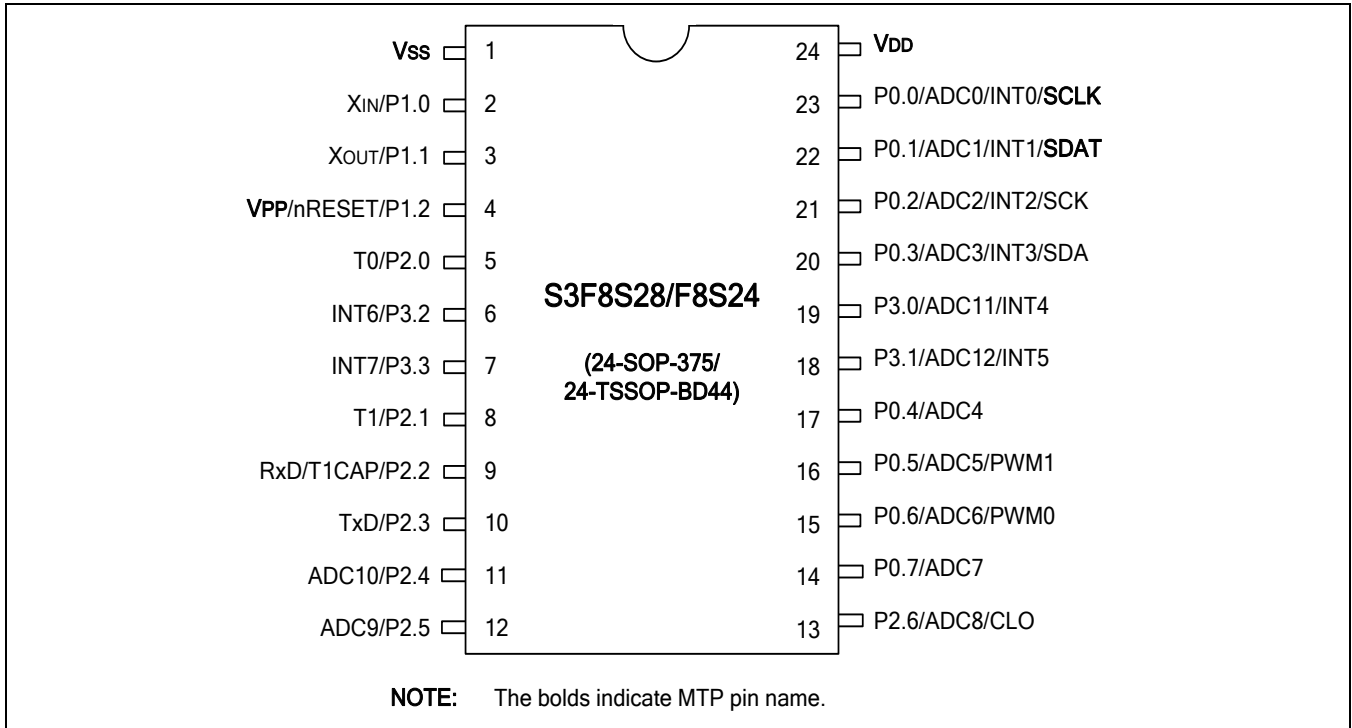
### 21.1 Overview

The S3F8S28/S3F8S24 single-chip CMOS microcontroller is the Flash MCU. It has an on-chip Flash MCU ROM of 8K/4Kbytes. The Flash ROM is accessed by serial data format.

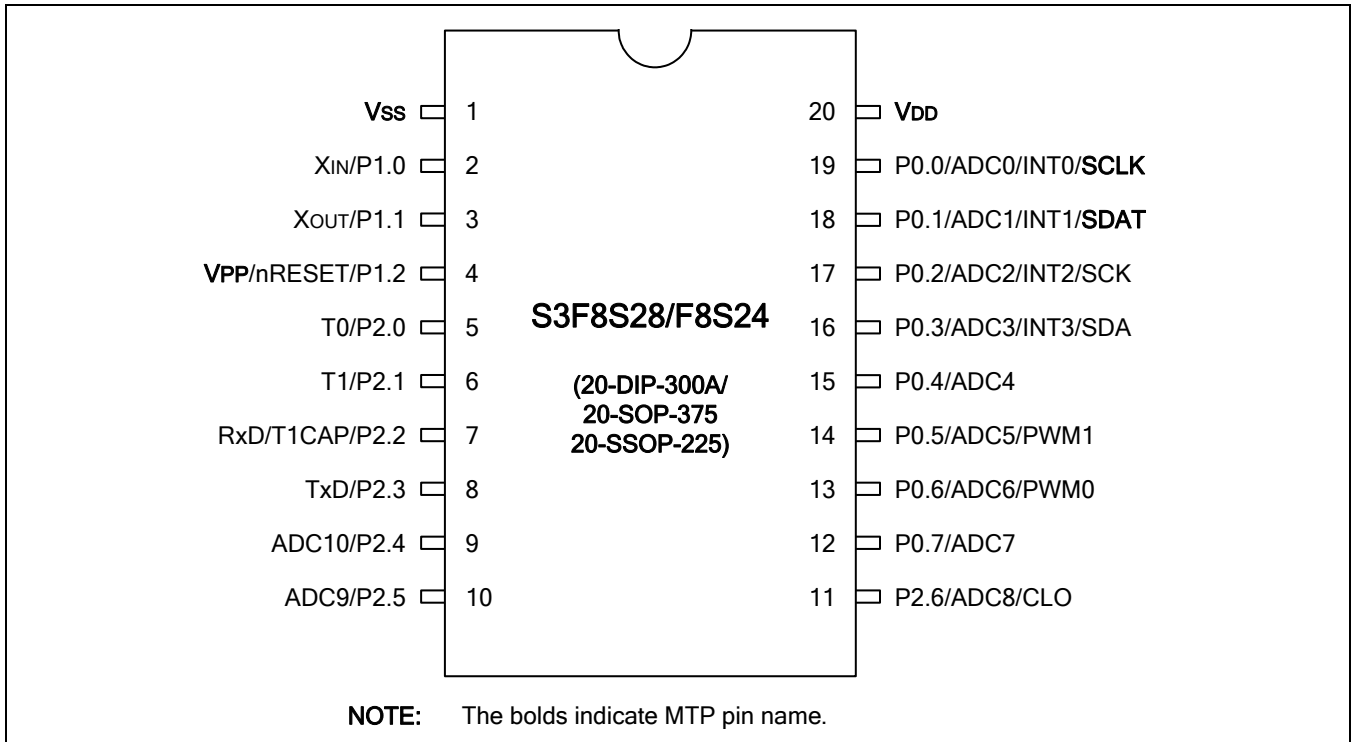
The serial data is transformed by two pins of the chip: SCLK and SDAT, SCLK is the synchronize signal, and the Flash Programmer Tool send data from the SDAT pin. The corresponding ports of SCLK and SDAT in S3F8S28/S3F8S24 are P0.0 and P1.1. And there also need power supply for chip to work and higher power for entering Flash tool mode. So the  $V_{DD}$ ,  $V_{SS}$  of chip must be connected to power and ground. The higher power supply for the Flash operation is named as  $V_{PP}$  port, the corresponding pin in S3F8S28/S3F8S24 is nRESET (P1.2) pin. The detail description of the pin functions are listed in the [Table 21-1](#). The pin assignments of the S3F8S28/S3F8S24 package types are shown in below figures.

**NOTE:**

1. This chapter is about the Tool Program Mode of Flash MCU. If you want to know the User Program Mode, refer to the chapter [18 Embedded Flash Memory Interface](#).
2. In S3F8S28/S3F8S24, there only 5 pins are used as Flash operation pins, the nRESET pin is used as  $V_{PP}$  input and without TEST pin that different with other Zilog MCU products.



**Figure 21-1 S3F8S28/S3F8S24 Pin Assignments (24-DIP/24-SOP)**



**Figure 21-2 S3F8S28/S3F8S24 Pin Assignments (20-DIP/20-SOP/20-SSOP)**

**Table 21-1 Descriptions of Pins Used to Read/Write the EPROM**

Main Chip	During Programming			
Pin Name	Pin Name	Pin No.	I/O	Function
P0.1	SDAT	22 (24-pin), 18 (20-pin)	I/O	Serial data pin (output when reading, Input when writing) Input and push-pull output port can be assigned
P0.0	SCLK	23 (24-pin), 19 (20-pin)	I	Serial clock pin (input only pin)
RESET/P1.2	V <sub>PP</sub>	4	I	Power supply pin for Tool mode entering (indicates that MTP enters into the Tool mode). When 11V is applied, MTP is in Tool mode.
V <sub>DD</sub> /V <sub>SS</sub>	V <sub>DD</sub> /V <sub>SS</sub>	24 (24-pin), 20 (20-pin), 1 (24-pin), 1 (20-pin),	I	Logic power supply pin.

**NOTE:** Parentheses indicate pin number for 20-DIP-300A package.

**Table 21-2 Comparison of S3F8S28/S3F8S24 Features**

Characteristic	S3F8S28/S3F8S24
Program memory	8K/4Kbyte Flash ROM
Operating voltage (V <sub>DD</sub> )	2.0V to 5.5V
Flash MCU programming mode	V <sub>DD</sub> = 5.0V, V <sub>PP</sub> (nRESET) = 11V
Pin configuration	24-SOP/24-TSSOP/20-DIP/20-SOP/20-SSOP
Programmability	User program multi time



## 21.2 On Board Writing

The S3F8S28/S3F8S24 needs only 5 signal lines including  $V_{DD}$  and GND pins for writing internal Flash memory with serial protocol. Therefore the on-board writing is possible if the writing signal lines are considered when the PCB of application board is designed.

### Circuit Design Guide:

At the Flash writing, the writing tool needs 5 signal lines that are GND,  $V_{DD}$ ,  $V_{PP}$ , SDAT and SCLK. When you design the PCB circuits, you should consider the usage of these signal lines for the on-board writing.

In case of  $V_{PP}$  (nRESET) pin, for the purpose of increase the noise effect, a capacitor should be inserted between the  $V_{PP}$  pin and GND.

Please be careful to design the related circuit of these signal pins because rising/falling timing of  $V_{PP}$ , SCLK and SDAT is very important for proper programming.

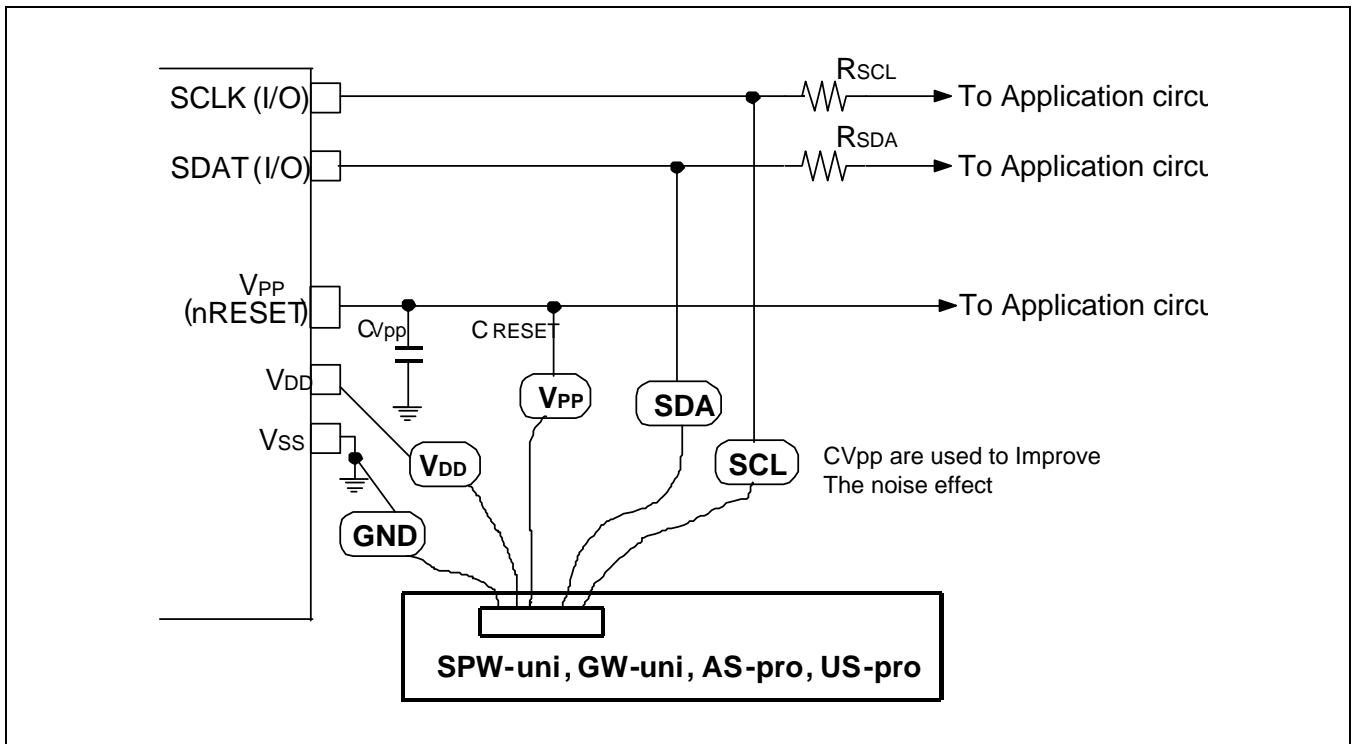


Figure 21-3 PCB Design Guide for on Board Programming

**Table 21-3 Reference Table for Connection**

Pin Name	I/O Mode in Applications	Resistor (Need)	Required Value
V <sub>PP</sub> (nRESET)	Input	Yes	CV <sub>pp</sub> is 0.01uF to 0.02uF.
SDAT (I/O)	Input	Yes	RSDAT is 2kΩ to 5kΩ.
	Output	No (NOTE)	–
SCLK (I/O)	Input	Yes	RSCLK is 2kΩ to 5kΩ.
	Output	No (NOTE)	–

**NOTE:**

1. In on-board writing mode, very high-speed signal will be provided to pin SCLK and SDAT. And it will cause some damages to the application circuits connected to SCLK or SDAT port if the application circuit is designed as high speed response such as relay control circuit. If possible, the I/O configuration of SDAT, SCLK pins had better be set to input mode.
2. The value of R, C in this table is recommended value. It varies with circuit of system.

# 22 Development Tools

## 22.1 Overview

Zilog offers software and hardware tools for S3 application development. Alternatively, a complete suite of 3<sup>rd</sup> party tools can be used. Applications targeting S3F8-series microcontrollers can use either the low-cost Zilog library-based Development Platform toolset or more sophisticated 3<sup>rd</sup> party emulator-based development tools. Applications targeting S3C8-series microcontrollers typically require the use of 3<sup>rd</sup> party emulator-based development tools.

Section 22.2 describes using 3<sup>rd</sup> party emulators (such as the OPENice i500 or i2000) to interface with a device-specific target board for application development on S3C8-series (or S3F8-series) microcontrollers. Section 22.3 describes the Zilog library-based Development Platform for Flash-based S3F8-series microcontrollers.

## 22.2 Emulator-based Development System

Figure 22-1 shows an emulator-based development system utilizing an emulator to interface with an application board through a Zilog-provided Target Board.

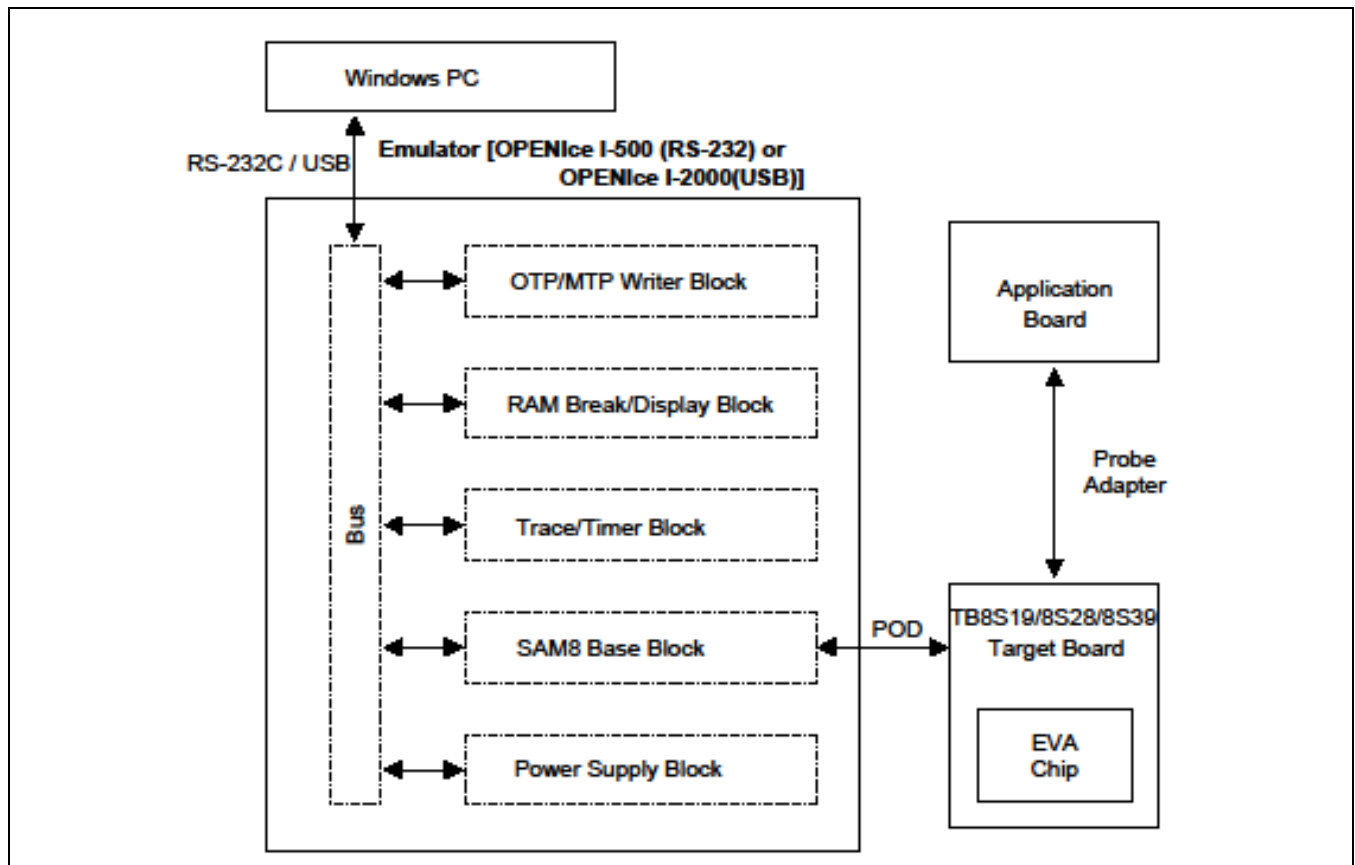


Figure 22-1 Emulator-based Development System Configuration

The S3 Emulator Based Development System includes the components listed in the following sections.

### 22.2.1 Host Software

Host software is required to create and debug S3 application programs in C or assembly language. The host software program converts the application source code into an executable format that is downloaded into the evaluation (EVA) chip on the target board for program execution/debugging. Optionally, the probe adapter cable(s) can be connected between the target board and the application board to debug program interaction with components on the application board.

Zilog provides the Zilog Developer Studio (ZDS) software suite host software package free of charge for any PC running a supported version of the Windows operating system. Alternatively, 3<sup>rd</sup> party host software packages (such as the IAR Embedded Workbench host software package) are available for purchase from vendor websites. The ZDS S3 software package is available for free download from the Zilog website.

### 22.2.2 Target Boards

Target boards are available for all S3C8/S3F8-series microcontrollers. Each target board includes the cables and adapters necessary to interface with an application board. The target board can be used with a 3<sup>rd</sup> party emulator to enable application debugging with or without an application board. Alternatively, the emulator can be used to program the target MCU on the application board using the supplied 10- circuit programming cable. The TB8S19/8S28/8S19 target board can be used with application boards targeting the S3F8S19, S3F8S28, and S3F8S39 MCUs.

Figure 22-2 shows how the TB8S19/8S28/8S19 Target Board is configured. The symbol “ ” marks the starting point of the jumper signals.

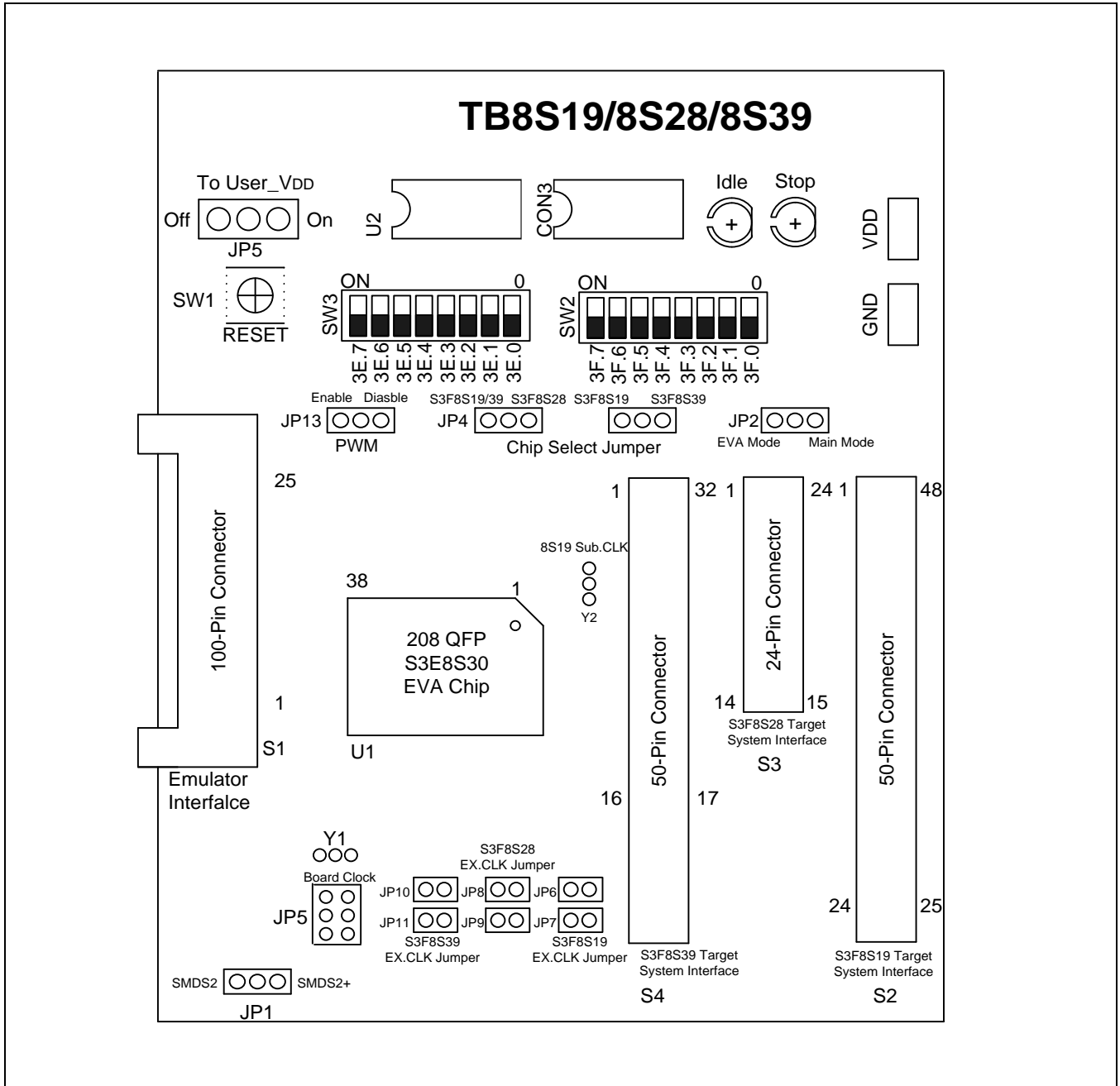



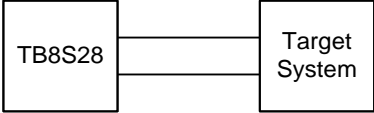


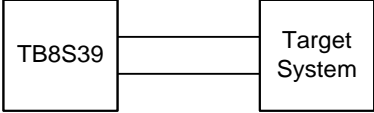


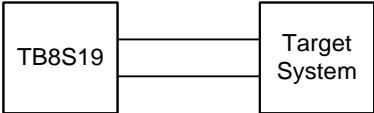
Figure 22-2 TB8S19/8S28/8S39 Target Board Configuration

**NOTE:** TB8S19/8S28/8S39 should be supplied 5V normally. Therefore, the power supply from Emulator should be set 5V for the target board operation.

**Table 22-1 Components of TB8S19/8S28/8S39**

Symbols	Usage	Description
JP3, JP4	Device Selection	Selection of device: S3F8S19, S3F8S28, S3F8S39
JP8, JP9	Ex.CLK selection	Set external clock connect to S3F8S28/S3F8S24 EVA-chip.
JP12	User's Power selection	Selection of Power to User.
JP2	MODE Selection	Selection of Eva/Main-chip mode of S3F8S28/S3F8S24 EVA-chip
JP1	Emulator selection	Selection of SMDS2/SMDS2+
JP5	Clock Source Selection	Selection of debug with internal/external clock
SW2, SW3	8-pin switch	Smart Option setting for S3F8S28/S3F8S24 EVA-chip
S1	100-pin connector	Connection between emulator and TB8S8S28 target board.
S3	24-pin connector	Connection between target board and user application system
RESET	Push button	Generation low active reset signal to S3F8S28/S3F8S24 EVA-chip
VCC, GND	POWER connector	External power connector for TB8S19/8S28/8S39
IDLE, STOP LED	STOP/IDLE Display	Indicate the status of STOP or IDLE of S3F8S28/F8S24 EVA-chip on TB8S19/8S28/8S39 target board
JP3	PWM selection	Selection of PWM enable/disable


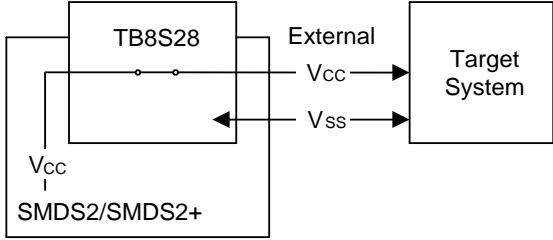

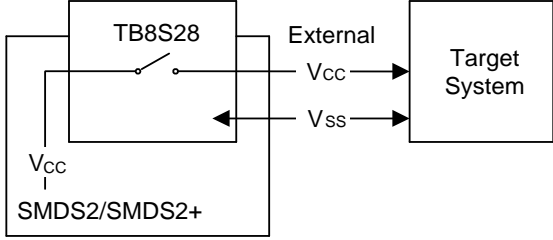
**Table 22-2 Device Selection Settings for TB8S19/8S28/8S39**

"Device Selection" Settings	Operating Mode	Comments
Device Selection:JP4 8S19/39  8S28		Operate with TB8S28
Device Selection JP4 8S19/8S39  8S28 JP3 8S19  8S39		Operate with TB8S39
Device Selection JP4 8S19/39  8S28 JP3 8S19  8S39		Operate with TB8S19

**NOTE:** The following symbol in the "8S28" Setting column indicates the electrical short (off) configuration:




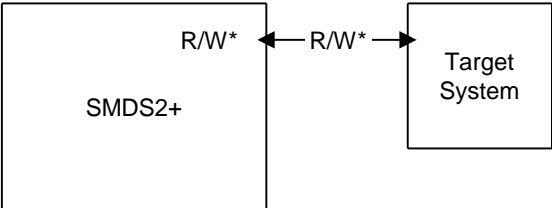
**Table 22-3 Power Selection Settings for TB8S19/8S28/8S39**

"To User_Vcc" Settings	Operating Mode	Comments
<p>Touser_VDD                      off  on</p>		<p>The SMDS2/SMDS2+ main board supplies VDD to the target board (evaluation chip) and the target system.</p>
<p>Touser_VDD                      off  on</p>		<p>The SMDS2/SMDS2+ main board supplies VDD only to the target board (evaluation chip). The target system must have its own power supply.</p>

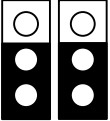
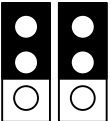





**22.2.3 SMDS2+ Selection (SAM8)**

In order to write data into program memory that is available in SMDS2+, the target board should be selected to be for SMDS2+ through a switch as follows. Otherwise, the program memory writing function is not available.

**Table 22-4 The SMDS2+ Tool Selection Setting**

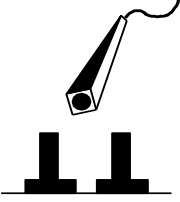
"JP4" Setting	Operating Mode
<p>SMDS2  SMDS2+</p>	

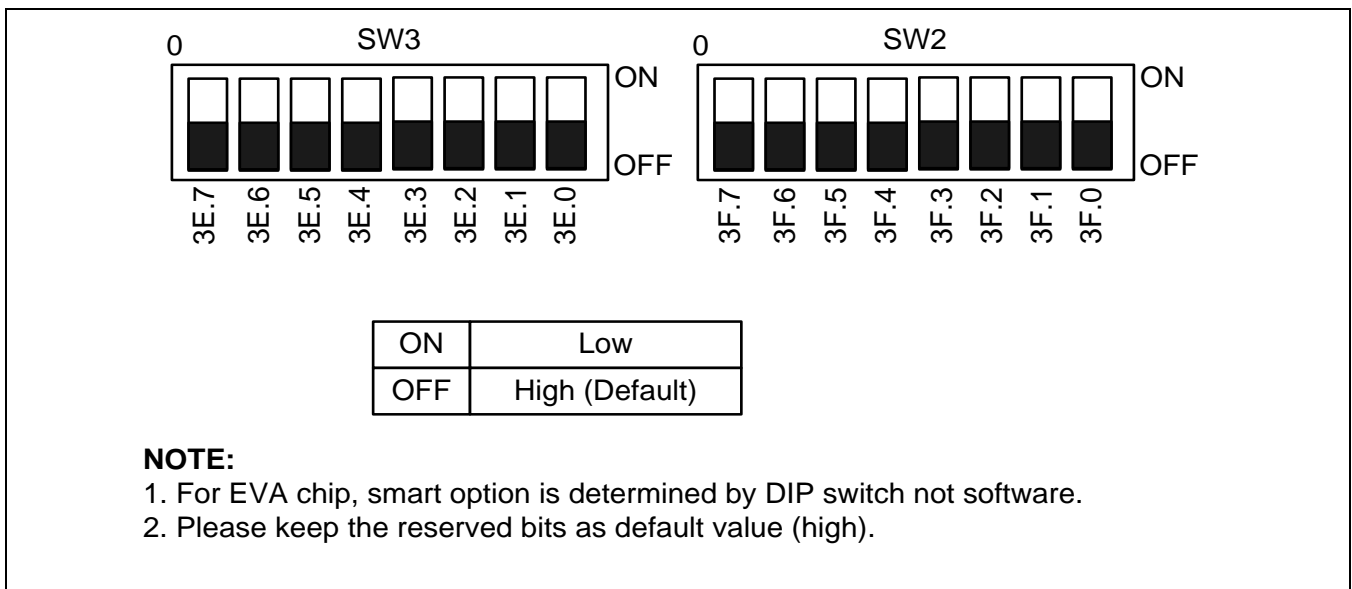
**Table 22-5 Using Single Header Pins to Select Clock Source/PWM/Operation Mode**

Target Board Part	Comments
<p>Board CLK</p>  <p>JP5 Clock Source</p> <p>Inner CLK</p>	<p>Use SMDS2/SMDS2+ internal clock source as the system clock. Default Setting</p>
<p>Board CLK</p>  <p>JP5 Clock Source</p> <p>Inner CLK</p>	<p>Use external crystal or ceramic oscillator as the system clock.</p>
 <p>JP8 JP9</p>	<p>Connect Clock to S3F8S28. (JP10 to 11, JP6 to 7 off)</p>
<p>PWM Enable</p>  <p>JP13</p> <p>PWM Disable</p>	<p>PWM function is DISABLED.</p>
<p>PWM Enable</p>  <p>JP13</p> <p>PWM Disable</p>	<p>PWM function is ENABLED. Default Setting</p>
<p>Main Mode</p>  <p>JP2</p> <p>EVA Mode</p>	<p>The S3E8S30 run in main mode, just same as S3F8S28/S3F8S24. The debug interface is not available.</p>
<p>Main Mode</p>  <p>JP2</p> <p>EVA Mode</p>	<p>The S3E8S30 run in EVA mode, available. When debug program, please set the jumper in this mode. Default Setting</p>



**Table 22-6 Using Single Header Pins as the Input Path for External Trigger Sources**

Target Board Part	Comments
<p>External Triggers</p> <p>○ Ch1(TP3)</p> <p>○ Ch2(TP4)</p>	<div style="text-align: center;">  </div> <p>Connector from External Trigger Sources of the Application System</p> <p>You can connect an external trigger source to one of the two external trigger channels (CH1 or CH2) for the SK-1000/SMDS2+ breakpoint and trace functions.</p>



**Figure 22-3 DIP Switch for Smart Option**

- IDLE LED  
This LED is ON when the evaluation chip (S3E8S30) is in idle mode.
- STOP LED  
This LED is ON when the evaluation chip (S3E8S30) is in stop mode.

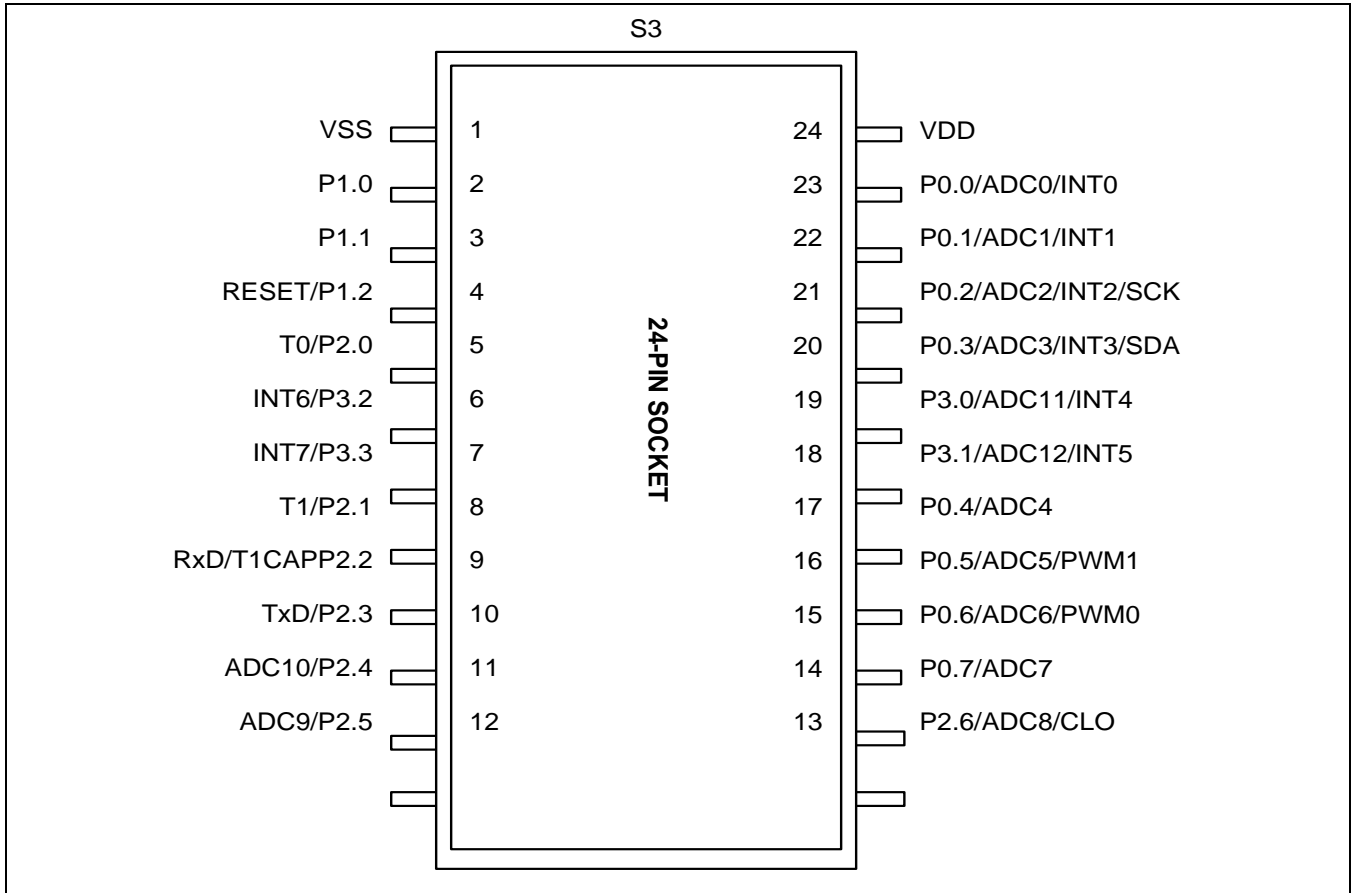


Figure 22-4 24-Pin Connector for TB8S19/8S28/8S39

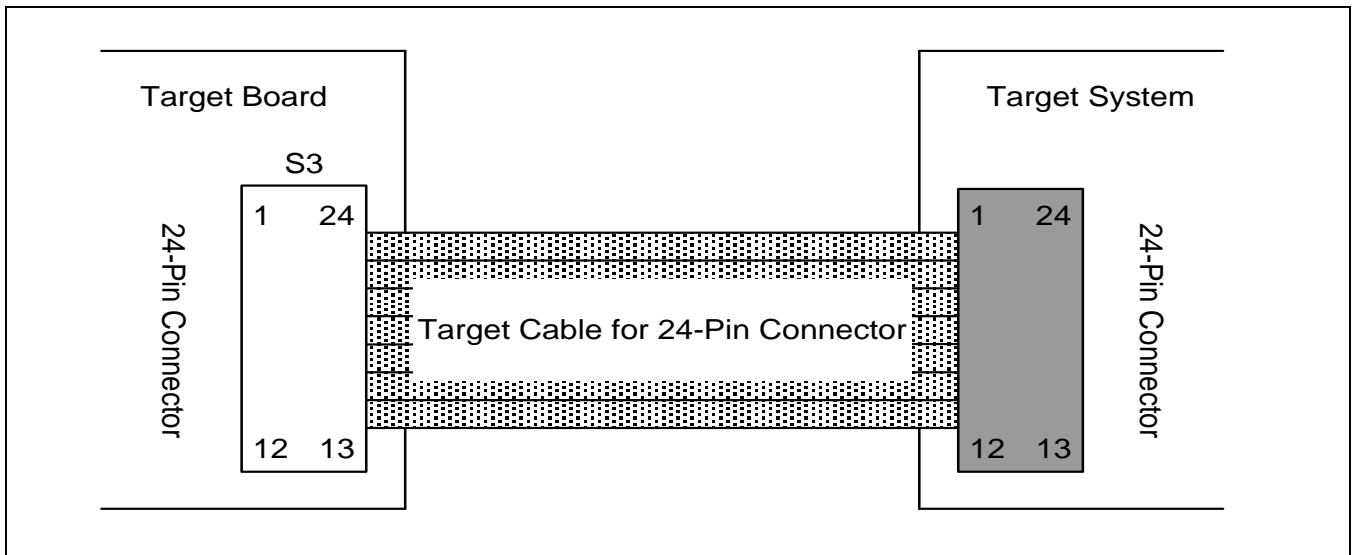


Figure 22-5 S3F8S28/S3F8S24 Probe Adapter for 24 Pin Package

## 22.3 Zilog Library-based Development Platform

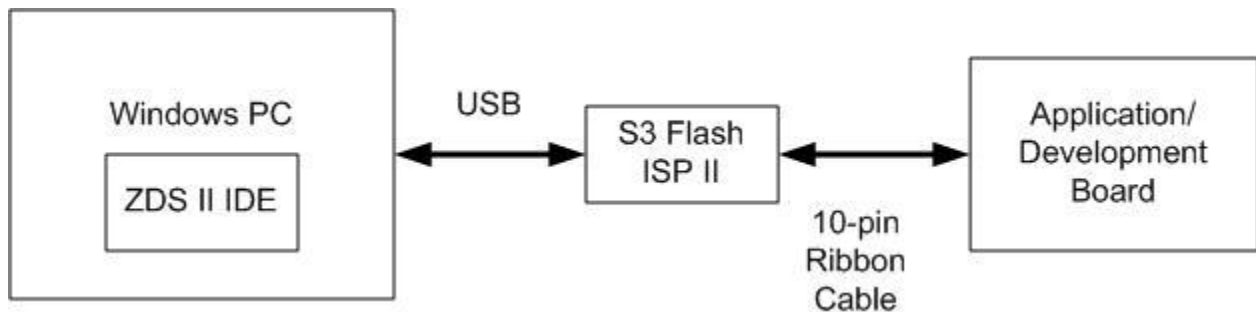
The Zilog developer platform is a suite of low-cost highly-integrated software and hardware tools for any PC running a supported version of Windows. The developer platform is composed of three components – the host Integrated Development Environment (IDE) software, the S3 Flash In-System Programmer (ISP) II USB interface, and a development board with a standard 10-pin ISP II connector. Together, these tools cost only a fraction of the price of most other 3<sup>rd</sup> party compilers, programmers/ emulators, or target boards.

Features include:

- Very low cost development tools
- Easy setup
- Source-level debugging using the application hardware board

### 22.3.1 Zilog Developer Platform Components

Figure 22-6 shows the simplicity of connecting all of the components of the Zilog developer platform.



**Figure 22-6 Zilog Development Platform**

#### 22.3.1.1 ZDS IDE

The Zilog Developer Studio (ZDS) Integrated Development Environment (IDE) is a suite of software tools that run on a Windows-based host PC. These tools include an editor used to create application programs in C or assembly, a compiler, assembler, a linker used to convert the application source code into an executable program image, and a debugger that allows the developer to single-step their application source code while it is executing on the actual target HW platform.

ZDS is completely free of charge and available from the Zilog website. For more information about the features of the ZDS IDE, please refer to the Zilog Developer Studio Help file integrated within the ZDS IDE by clicking the Help Topics item available through the IDE's Help menu, or by pressing F1 on the PC keyboard.

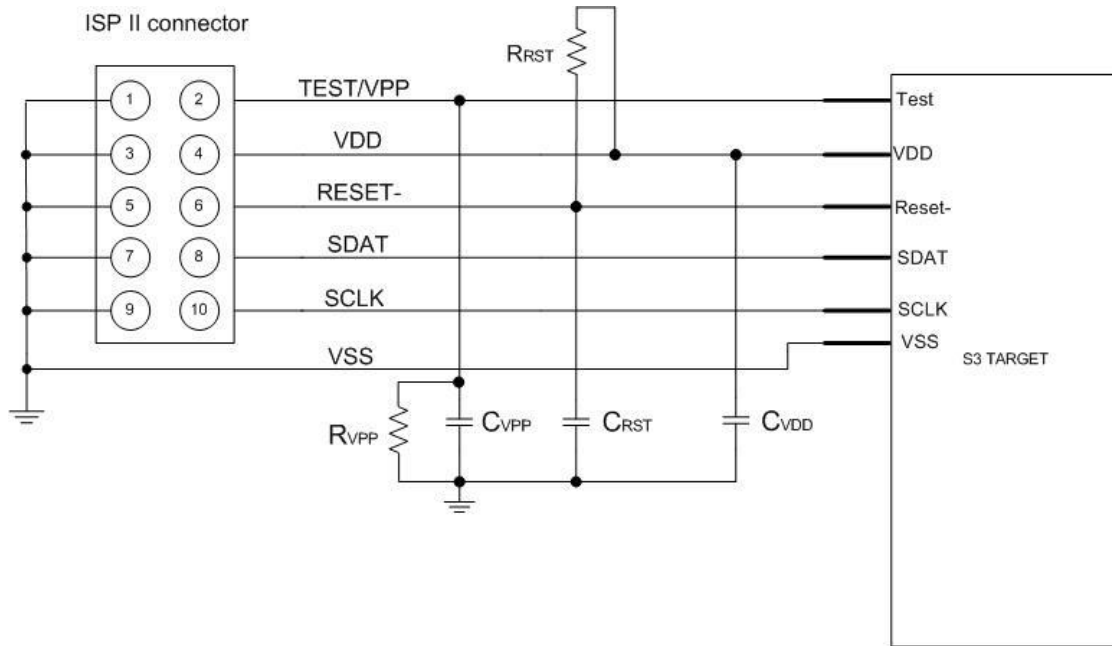
#### 22.3.1.2 S3 Flash ISP II

The Zilog S3 Flash ISP II is a low cost hardware interface between the PC and the application board or Zilog development board. The ISP II connects to the Windows PC through a USB cable and connects to the application or development board through a 10-pin ribbon cable. ZDS uses the ISP II to access Flash memory on the S3 target for read, erase, and program operations. Additionally, ZDS can use the S3 Flash ISP II to debug applications built with a Zilog-provided debug library.

#### 22.3.1.3 Application/Development Board

The S3 Flash ISP II communicates with the S3 microcontroller on a Zilog development board, or a customer application board, through a 10-pin ribbon cable. This requires the application or development board design to

include the components shown in Figure 22-7.



**Figure 22-7 PCB Design Guide for In System Programming**

Some S3 devices have a VPP/Test pin shared with a GPIO pin which can also be configured as the Reset pin. When designing a PCB that requires In-System Programming support for S3 devices with a shared VPP/ Reset pin, do not connect the Reset signal (pin 6) from the 10-pin ISP II connector to the S3 MCU. Instead, connect the MCU VPP/ Reset pin to the Test/ VPP signal (pin 2) of the ISP II connector with  $R_{RST}$  and  $C_{RST}$ . In this instance, it is not necessary to include  $R_{VPP}$  or  $C_{VPP}$ .

Table 22-7 shows the recommended values for the passive components in the ISP II circuit of Figure 22.6.

**Table 22-7 ISP II Circuit Recommended Values**

ISP Signal (Pin Number)	Passive Component	Notes
VPP/ Test (2)	$C_{VPP} = 0.1 \mu F$ $R_{VPP} = 10K$	If the S3 MCU has a shared VPP/Reset pin, connect the ISP II VPP/ Test pin to the MCU VPP/Test pin.
VDD (4)	$C_{VDD} = 0.1 \mu F$	
Reset (6)	$C_{RST} = 0.1 \mu F$ $R_{RST} = 40K$	
SDAT (8) SCLK (10)		The ZDS IDE and S3 Flash ISP II cannot be used to debug applications that use the GPIO pins associated with the SCLK & SDAT signals. In this instance, it is only possible to access Flash Memory in the target S3 MCU.
GND (1,3,5,7,9)		Connect all odd number pins of the ISP connector to GND on the target board and S3 MCU

Refer to the schematic diagram in the appropriate Zilog Development Kit User Manual for a complete reference design that includes an ISP II interface circuit applicable to a particular series of S3 devices. Zilog recommends keeping the traces connecting SCLK and SDAT to the ISP II connector as short as possible.

### 22.3.2 Compatibility with 3<sup>rd</sup> Party Tools

The Zilog IDE can also be used with 3<sup>rd</sup> party development tools. For example, the ZDS IDE can program a Hex file generated by a 3<sup>rd</sup> party compiler such as the IAR Embedded Workbench using the Zilog S3 Flash ISP II or a 3<sup>rd</sup> party programmer such as the OPENice-i2000 emulator. Information regarding 3<sup>rd</sup> party development tools can be found in section 22.4.

### 22.3.3 Benefits and Limitations of Zilog Development Tools

Zilog development tools provide a low cost turnkey solution capable of creating and debugging S3 applications on Zilog development boards or customer application boards. Debugging applications on a particular S3 target typically requires the application to be built with a Zilog-provided debug library that is capable of interfacing with the S3 Flash ISP II. The debug library consumes some amount of code space on the S3 target depending on the set of debugging features supported by the particular debug library linked to the application.


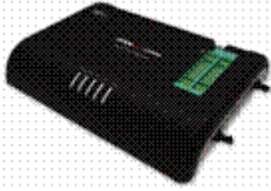
The ZDS IDE and S3 Flash ISP II can be used to program Flash memory on all Zilog S3 microcontrollers; however, single-step debugging support may not be available for every series of Zilog S3 microcontroller. For more information regarding the debugging features available on a particular S3 microcontroller, refer to the S3 ISP II Interface Debug Library chapter of the Zilog Developer Studio Help file available within the ZDS S3 IDE.

### 22.3.4 Development Tools

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

#### In-Circuit Emulators

- [YIC](#) – OPENice-i500/2000

<p style="text-align: center;"><b>OPENice-i500</b></p> 	<p>YIC System</p> <ul style="list-style-type: none"> <li>• TEL: 82-31-278-0461</li> <li>• FAX: 82-31-278-0463</li> <li>• E-mail: support@yicsystem.com</li> <li>• URL: <a href="http://www.yicsystem.com">http://www.yicsystem.com</a></li> </ul>
<p style="text-align: center;"><b>OPENice-i2000</b></p> 	<p>YIC System</p> <ul style="list-style-type: none"> <li>• TEL: 82-31-278-0461</li> <li>• FAX: 82-31-278-0463</li> <li>• E-mail : support@yicsystem.com</li> <li>• URL: <a href="http://www.yicsystem.com">http://www.yicsystem.com</a></li> </ul>


Zilog Library-based Development Tools



- [Zilog – S3USBISP000ZACG S3 Flash In-System Programmer \(ISP\) II](#)
- [Zilog – S3F8S280100ZCOG S3F8S28/8S24 Development Kit](#)


<p style="text-align: center;"><b>S3USBISP000ZACG</b></p> 	<p>Zilog</p> <ul style="list-style-type: none"> <li>• TEL: (408) 457-9000</li> <li>• FAX: (408) 416-0223</li> <li>• E-mail: <a href="mailto:s3sales@zilog.com">s3sales@zilog.com</a></li> <li>• URL: <a href="http://www.zilog.com">http://www.zilog.com</a></li> </ul>
<p style="text-align: center;"><b>S3F8S280100ZCOG</b></p> 	<p>Zilog</p> <ul style="list-style-type: none"> <li>• TEL: (408) 457-9000</li> <li>• FAX: (408) 416-0223</li> <li>• E-mail: <a href="mailto:s3sales@zilog.com">s3sales@zilog.com</a></li> <li>• URL: <a href="http://www.zilog.com">http://www.zilog.com</a></li> </ul>

**Programmiers (Writer)**

- [Seminix – GW-uni2](#)
- [C&A Tech – GW-Pro2](#)
- [Eltec – BeeHive series](#)
- [Zilog – S3 Flash ISP II](#)

	<p><b>GW-uni2</b> <b>Gang Programmer for OTP/MTP/FLASH MCU</b></p> <ul style="list-style-type: none"> <li>• Support all SAMSUNG OTP and MTP devices with SAMSUNG standard serial protocol format</li> <li>• Program up to 8 devices at one time</li> <li>• Operation mode: 1.PC base 2.Stand-alone (no PC)</li> <li>• Very fast programming speed: OTP(2 Kbps) MTP(10 Kbps)</li> <li>• Maximum buffer memory:100 Mbyte</li> <li>• Hex data file download via USB port from PC</li> <li>• Support simple GUI (Graphical User Interface)</li> <li>• Support data format: Intel hex, SAMSUNG hex, Binary</li> <li>• Device information can be set by a device part number</li> </ul>	<p>Seminix</p> <ul style="list-style-type: none"> <li>• TEL: 82-31-703-7891</li> <li>• FAX: 82-31-702-7869</li> <li>• E-mail: <a href="mailto:sales@seminix.com">sales@seminix.com</a></li> <li>• URL: <a href="http://www.seminix.com">http://www.seminix.com</a></li> </ul>
---	---	---

	<ul style="list-style-type: none"> <li>• LCD Display (Stand-alone mode operation)           <ul style="list-style-type: none"> <li>- Display an operation state</li> </ul> </li> <li>• Touch key (Stand-alone mode operation)</li> <li>• System upgradeable           <ul style="list-style-type: none"> <li>- The system firmware can be upgraded simply by user</li> </ul> </li> </ul>	
	<p><b>GW-Pro Gang Programmer</b></p> <ul style="list-style-type: none"> <li>• Programming of 8 MCUs at a time</li> <li>• Fast programming speed (2 Kbyte/sec)</li> <li>• Possible without PC (standalone)</li> <li>• Search operation based on a PC</li> <li>• Enough features to support Gang Programmer</li> <li>• Off data is also preserved</li> <li>• Key Lock function to prevent malfunction</li> <li>• Good and bad quantity counter</li> <li>• Program completion notification (sound)</li> <li>• Easy-to-use (PC) menu</li> </ul>	<p>C &amp; A Technology</p> <ul style="list-style-type: none"> <li>• TEL: 02-2612-9027</li> <li>• E-mail : <a href="mailto:jhc115@cnatech.com">jhc115@cnatech.com</a></li> <li>• URL: <a href="http://www.cnatech.com">http://www.cnatech.com</a></li> </ul>
	<p><b>Beehive204</b></p> <ul style="list-style-type: none"> <li>• Four independent universal programming sites</li> <li>• Two BeeHive 204 multiprogrammers can be attached to one PC to better utilize programming workplace</li> <li>• Extremely fast programming, one of the fastest programmers in this category. Sustainable programming speed greater than 5 Mbytes per second</li> <li>• Powerful independent pin driver circuit for each and every pin of the programmer</li> <li>• In-circuit programming capability through ISP connector</li> <li>• Very low voltage support for the latest Flash memory chips</li> <li>• ESD protection on each pin of the socket's USB (up to 480 Mbit/s) interface to PC</li> <li>• Comfortable and easy-to-use control program; works with all versions of MS Windows from Windows XP to Windows 10 (32-bit and 64-bit)</li> </ul>	<p>Elnec</p> <ul style="list-style-type: none"> <li>• TEL: +421-51-7734328</li> <li>• FAX: +421-51-7732797</li> <li>• E-mail: <a href="mailto:tech2@elnec.com">tech2@elnec.com</a></li> <li>• URL: <a href="http://www.elnec.com">http://www.elnec.com</a></li> </ul>

	<p><b>S3 Flash In-System Programmer II</b></p> <p>Zilog's S3 Flash ISP II provides an interface between any development or application board with an S3 microcontroller device to the high-speed USB port of a PC on which Zilog Developer Studio II for S3 Family devices (ZDS II – S3) is installed.</p> <p>The ISP II allows the Flash memory space on any S3 Family device to be programmed, and also offers limited debugging capabilities when used together with the Zilog Debug Library.</p> <p>The following features are available with the S3 Flash ISP II when using ZDS II for S3 Family devices:</p> <ul style="list-style-type: none"> <li>• Download code to Flash and begin to program execution</li> <li>• Break program execution arbitrarily</li> <li>• Single-step debugging of the application, view/edit memory and S3 special function registers. Resume normal program operation after a breakpoint</li> <li>• Insert multiple breakpoints in a program at compile/assembly time</li> </ul>	<p>Zilog</p>
		<ul style="list-style-type: none"> <li>• TEL: (408) 457-9000</li> <li>• FAX: (408) 416-0223</li> <li>• E-mail: <a href="mailto:s3sales@zilog.com">s3sales@zilog.com</a></li> <li>• URL: <a href="http://www.zilog.com">http://www.zilog.com</a></li> </ul>

To obtain the S3 Family development tools that will satisfy your S3F8S28/S3F8S24 development objectives, contact your local [Zilog Sales Office](#), or visit Zilog's [Third Party Tools page](#) to review our list of third party tool suppliers.