

# **S3P7588X**

## **4-BIT RISC MICROPROCESSOR USER'S MANUAL**

**Revision 0**



**ELECTRONICS**

# Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

## **S3P7588X 4-Bit CMOS Microcontroller User's Manual, Revision 0 Publication Number:**

© 2002 Samsung Electronics

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electric or mechanical, by photocopying, recording, or otherwise, without the prior written consent of Samsung Electronics.

*Samsung Electronics' microcontroller business has been awarded full ISO-14001 certification (BVQI Certificate No. 9330). All semiconductor products are designed and manufactured in accordance with the highest quality standards and objectives.*

Samsung Electronics Co., Ltd.  
San #24 Nongseo-Ri, Giheung-Eup  
Yongin-City Gyeonggi-Do, Korea  
C.P.O. Box #37, Suwon 449-900

TEL: (82)-(31)-209-1999  
FAX: (82)-(31)-209-1899  
Home-Page URL: [Http://www.samsungsemi.com/](http://www.samsungsemi.com/)

# Preface

The S3P7588X *Microcontroller User's Manual* is designed for application designers and programmers who are using the S3P7588X microcontroller for application development. It is organized in two parts:

Part I Programming Model

Part II Hardware Descriptions

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

Chapter 1	Product Overview	Chapter 4	Memory Map
Chapter 2	Address Spaces	Chapter 5	Instruction Set
Chapter 3	Addressing Modes	Chapter 6	Oscillator Circuits

Chapter 1, "Product Overview," is a high-level introduction to the S3P7588X with a general product description, and detailed information about individual pin characteristics and pin circuit types.

Chapter 2, "Address Spaces," explains the S3P7588X program and data memory, internal register file, and mapped control registers, and explains how to address them. Chapter 2 also describes working register addressing, as well as system and user-defined stack operations.

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

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

Chapter 5, "Instruction Set," describes the S3P7588X interrupt structure in detail and further prepares you for additional information presented in the individual hardware module descriptions in Part II.

Chapter 6, "Oscillator Circuits," describes the features and conventions of the instruction set used for all S3P7-series microcontrollers. Several summary tables are presented for orientation and reference. Detailed descriptions of each instruction are presented in a standard format. Each instruction description includes one or more practical examples of how to use the instruction when writing an application program.

A basic familiarity with the information in Part I will help you to understand the hardware module descriptions in Part II. If you are not yet familiar with the SAM88RCRI product family and are reading this manual for the first time, we recommend that you first read Chapters 1–3 carefully. Then, briefly look over the detailed information in Chapters 4, 5, and 6. Later, you can reference the information in Part I as necessary.

Part II contains detailed information about the peripheral components of the S3P7588X microcontrollers. Also included in Part II are electrical, mechanical, OTP, development tools and errata.

It has ten Chapters:

Chapter 7	Caller ID	Chapter 13	DTMF Generator
Chapter 8	Interrupts	Chapter 14	Electrical Data
Chapter 9	Power Down	Chapter 15	Mechanical Data
Chapter 10	RESET	Chapter 16	OTP
Chapter 11	I/O Ports	Chapter 17	Development Tools
Chapter 12	Timers and Timer/Counters	Chapter 18	Errata

Two order forms are included at the back of this manual to facilitate customer order for S3P7588X microcontrollers: the Mask ROM Order Form, and the Mask Option Selection Form. You can photocopy these forms, fill them out, and then forward them to your local Samsung Sales Representative.



# Table of Contents

## Part I — Programming Model

### Chapter 1 Product Overview

OTP.....	1-1
Features Summary .....	1-2
Block Diagram .....	1-3
Pin Assignments .....	1-4
Pin Descriptions.....	1-6
Pin Circuit Diagrams.....	1-8

### Chapter 2 Address Spaces

Overview .....	2-1
General-Purpose Program Memory (ROM).....	2-1
Data Memory (RAM) .....	2-7
Stack Operations.....	2-15

### Chapter 3 Addressing Modes

Overview .....	3-1
Enable Memory Bank Settings.....	3-4
Select Bank Register (SB) .....	3-5
Direct and Indirect Addressing.....	3-6

### Chapter 4 Memory Map

Overview .....	4-1
Register Descriptions.....	4-4

### Chapter 5 Instruction Set

Overview .....	5-1
Instruction Set Features.....	5-1
High-Level Summary.....	5-9
Binary Code Summary .....	5-14
Instruction Descriptions .....	5-24

### Chapter 6 Oscillator Circuits

Overview .....	6-1
Clock Control Registers .....	6-1
Clock Output Mode Register (CLMOD).....	6-5
Clock Output Circuit .....	6-6

## Table of Contents (Continued)

### Part II — Hardware Descriptions

#### Chapter 7 Caller ID

Overview .....	7-1
Application.....	7-2
Functional Descriptions of Caller ID Block.....	7-7
Functional Block Diagram.....	7-7
Analog Input and Preprocessor.....	7-8
CAS Tone Detection.....	7-9
FSK Reception .....	7-10
Stutter DIAL Tone (SDT) Detector .....	7-12
Bit Transfer.....	7-16
Register Maps of Caller ID Block .....	7-21
Mode Register (MODE) .....	7-22
Function Register (FUNC) .....	7-22
Dtmf Tone Select Register (DTMFT) .....	7-23
Guard Time Register (GTIME).....	7-23
Interrupt Register (INTR) .....	7-23
Status Register (STAT).....	7-24
FSK Data Register (FSKDT).....	7-24
Dtmf Output Gain Control Register (DTMFG) .....	7-24
Special Control Register (CONT1).....	7-25
Special Control Register (CONT2).....	7-25

#### Chapter 8 Interrupts

Overview .....	8-1
Vectored Interrupts .....	8-2
Interrupt Priority Register (IPR).....	8-8
External Interrupt 0 and 1 Mode Registers (Continued).....	8-10
External Interrupt 2 Mode Register (IMOD2).....	8-11
Interrupt Flags .....	8-14

#### Chapter 9 Power — Down

Overview .....	9-1
Idle Mode Timing Diagrams.....	9-2
Stop Mode Timing Diagrams .....	9-3
Port Pin Configuration for Power-Down.....	9-4
Recommended Connections for Unused Pins .....	9-5

## Table of Contents (Continued)

### Chapter 10 RESET

Overview .....	10-1
Caller ID Reset Signal .....	10-1
Hardware Reset Values After Reset.....	10-2

### Chapter 11 I/O Ports

Overview .....	11-1
Port Mode Flags (PM FLAGS).....	11-3
Pull-Up Resistor Mode Register (PUMOD) .....	11-4
N-Channel Open-Drain Mode Register (PNE).....	11-4
Port 1 Circuit Diagram.....	11-5
Port 2, 3, 6, 7, 8, and 9 Circuit Diagram.....	11-6
Port 4, 5 Circuit Diagram .....	11-7

### Chapter 12 Timers and Timer/Counters

Overview .....	12-1
Basic Timer (BT) .....	12-2
Overview.....	12-2
Basic Timer Mode Register (BMOD).....	12-5
Basic Timer Counter (BCNT).....	12-6
Basic Timer Output Enable Flag (BOE).....	12-6
Basic Timer Operation Sequence .....	12-6
Watchdog Timer Mode Register (WDMOD).....	12-8
Watchdog Timer Counter (WDCNT).....	12-8
Watchdog Timer Counter Clear Flag (WDTCF).....	12-8
8-Bit Timer/Counters 0 and 1 (TC0, TC1) .....	12-10
Overview.....	12-10
TC Function Summary .....	12-10
TC Component Summary.....	12-11
TC Enable/Disable Procedure .....	12-12
TC Programmable Timer/Counter Function .....	12-13
TC Operation Sequence .....	12-13
TC Event Counter Function .....	12-14
TC Clock Frequency Output .....	12-15
TC External Input Signal Divider .....	12-16
TC Mode Register (TMODN) .....	12-17
TC Counter Register (TCNTN) .....	12-19
TC Reference Register (TREFN).....	12-20
TC Output Latch (TOLN) .....	12-20
Watch Timer.....	12-22
Overview.....	12-22
Watch Timer Mode Register (WMOD).....	12-24

## Table of Contents (Continued)

### Chapter 13 DTMF Generator

Overview .....	13-1
Dtmf Mode Register.....	13-2
Dtmf Gain Register.....	13-3
RC Filtering .....	13-3

### Chapter 14 Electrical Data

Overview .....	14-1
----------------	------

### Chapter 15 Mechanical Data

Overview .....	15-1
----------------	------

### Chapter 16 OTP

Overview .....	16-1
Operating Mode Characteristics.....	16-3

### Chapter 17 Development Tools

Overview .....	17-1
Otps .....	17-1

### Chapter 18 Errata

Revision 1.0.....	18-1
Errata List .....	18-1



# List of Figures

Figure Number	Title	Page Number
1-1	S3P7588X Simplified Block Diagram.....	1-3
1-2	S3P7588X Pin Assignment Diagrams (100-TQFP-1414) .....	1-4
1-3	Pin Diagram of Pellet Type.....	1-5
1-4	Pin Circuit Type A .....	1-8
1-5	Pin Circuit Type B .....	1-8
1-6	Pin Circuit Type B-1 .....	1-8
1-7	Pin Circuit Type A-4 .....	1-8
1-8	Pin Circuit Type C .....	1-8
1-9	Pin Circuit Type D-2 .....	1-9
1-10	Pin Circuit Type E-2 .....	1-9
1-11	Pin Circuit Type D-4 .....	1-9
2-1	ROM Address Structure.....	2-3
2-2	Vector Address Map .....	2-3
2-3	Data Memory (RAM) Map.....	2-7
2-4	Working Register Map.....	2-10
2-5	Register Pair Configuration.....	2-11
2-6	1-Bit, 4-Bit, and 8-Bit Accumulator.....	2-12
2-7	Push-Type Stack Operations .....	2-16
2-8	Pop-Type Stack Operations.....	2-17
3-1	RAM Address Structure .....	3-2
3-2	SMB and SRB Values in the SB Register .....	3-5
4-1	Register Description Format.....	4-5
6-1	Clock Circuit Diagram.....	6-2
6-2	Crystal/Ceramic Oscillator .....	6-3
6-3	External Oscillator .....	6-3
6-4	CLO Output Pin Circuit Diagram.....	6-6
7-1	Application Diagram for S3P7588X Development System with KS57C5208 SMDS .....	7-4
7-2	Recommended Diagram for Typical Application .....	7-5
7-3	Block Diagram of CID Module .....	7-7
7-4	Differential Input Buffer of S3P7588X.....	7-8
7-5	Single Ended Buffer of S3P7588X.....	7-9
7-6	CASdetect, CASint and INT Related to the CAS Tone.....	7-9
7-7	Sequence to Receive an FSK Data Byte .....	7-10
7-8	Interrupt Behavior of the FSK Receiver with BOMDC = 1 .....	7-11
7-9	Interrupt Behavior of the FSK Receiver with BOMDC = 0 .....	7-11
7-10	SDT Detector Operation .....	7-12
7-11	External Component to Generate LRin .....	7-13
7-12	Behavior of Signals on a Line Reversal .....	7-13
7-13	Behavior of Signals During Ring.....	7-14
7-14	Start and Stop Conditions.....	7-16
7-15	Bit Transfer Timing.....	7-16
7-16	Byte Transmission and Acknowledge.....	7-17
7-17	Write Sequence of the Serial Interface .....	7-18

## List of Figures (Continued)

Figure Number	Title	Page Number
7-18	(a) Read Sequence of the Serial Interface when new Register Start Address is Programmed .....	7-19
7-18	(b) Read Sequence of the Serial Interface when no Register Start Address is Programmed .....	7-19
8-1	Interrupt Execution Flowchart .....	8-3
8-2	Interrupt Control Circuit Diagram .....	8-4
8-3	Interrupt Control Circuit Diagram .....	8-5
8-4	Two-Level Interrupt Handling.....	8-6
8-5	Multi-Level Interrupt Handling.....	8-7
8-6	Circuit Diagram for INT0 and INT1 Pins.....	8-10
8-7	Circuit Diagram for INT2 and KS0–KS7 Pins .....	8-12
9-1	Timing When Idle Mode is Released by RESET .....	9-2
9-2	Timing When Idle Mode is Released by an Interrupt.....	9-3
9-3	Timing When Stop Mode is Released by RESET.....	9-3
9-4	Timing When Stop Mode is Release by an Interrupt .....	9-3
10-1	Timing for Oscillation Stabilization After RESET .....	10-1
11-1	Port 1 Circuit Diagram .....	11-5
11-2	Port 2, 3, 6, 7, 8, and 9 Circuit Diagram.....	11-6
11-3	Port 4 and 5 Circuit Diagram .....	11-7
12-1	Basic Timer Circuit Diagram.....	12-4
12-2	TC Circuit Diagram.....	12-12
12-3	TC Timing Diagram .....	12-19
12-4	Watch Timer Circuit Diagram .....	12-23
13-1	Block Diagram of DTMF Generator .....	13-1
14-1	Stop Mode Release Timing When Initiated By RESET .....	14-8
14-2	Stop Mode Release Timing When Initiated By Interrupt Request.....	14-8
14-3	A.C. Timing Measurement Points (Except for Xin) .....	14-9
14-4	Clock Timing Measurement at Xin .....	14-9
14-5	TCL Timing .....	14-9
14-6	Input Timing for RESET Signal.....	14-10
14-7	Input Timing for External Interrupts and Quasi-Interrupts.....	14-10
14-8	Waveform for CAS Timing Characteristics .....	14-12
14-9	Waveform for SDT Timing Characteristics .....	14-13
14-10	Timing Constraints of Start and Stop Condition .....	14-14
14-11	Timing of SCK and SDT During Byte Transmission .....	14-14

## List of Figures (Concluded)

<b>Figure Number</b>	<b>Title</b>	<b>Page Number</b>
15-1	Pin Diagram of Pellet Type.....	15-1
15-2	100-TQFP-1414 Package Dimensions.....	15-2
16-1	S3P7588X Pin Assignments (100-TQFP-1414).....	16-2
16-2	OTP Programming Algorithm .....	16-4
17-1	S3P7588X Development System Configuration.....	17-2
17-1	S3P7588X Development System Configuration (Continued).....	17-3
17-2	S3P7588X Target Board Diagram .....	17-4
17-3	Pin Assignment of 50-Pin DIP Connector .....	17-8



# List of Tables

Table Number	Title	Page Number
1-1	S3P7588X Pin Descriptions .....	1-6
1-1	S3P7588X Pin Descriptions (Continued).....	1-7
2-1	Program Memory Address Ranges .....	2-2
2-2	Data Memory Organization and Addressing.....	2-9
2-3	Working Register Organization and Addressing.....	2-11
2-4	BSC Register Organization.....	2-18
2-6	Interrupt Status Flag Bit Settings .....	2-20
2-7	Valid Carry Flag Manipulation Instructions.....	2-23
3-1	RAM Addressing Not Affected by the EMB Value .....	3-4
3-2	1-Bit Direct and Indirect RAM Addressing.....	3-6
3-3	4-Bit Direct and Indirect RAM Addressing.....	3-8
3-4	8-Bit Direct and Indirect RAM Addressing.....	3-11
4-1	I/O Map for Memory Bank 15 .....	4-2
4-1	I/O Map for Memory Bank 15 (Continued) .....	4-3
4-1	I/O Map for Memory Bank 15 (Continued) .....	4-4
5-1	Valid 1-Byte Instruction Combinations for REF Look-Ups .....	5-2
5-2	Bit Addressing Modes and Parameters .....	5-5
5-3	Skip Conditions for ADC and SBC Instructions .....	5-6
5-4	Data Type Symbols .....	5-7
5-5	Register Identifiers .....	5-7
5-6	Instruction Operand Notation .....	5-7
5-7	Opcode Definitions (Direct) .....	5-8
5-8	Opcode Definitions (Indirect) .....	5-8
5-9	CPU Control Instructions — High-Level Summary.....	5-10
5-10	Program Control Instructions — High-Level Summary.....	5-10
5-11	Data Transfer Instructions — High-Level Summary .....	5-11
5-12	Logic Instructions — High-Level Summary .....	5-12
5-13	Arithmetic Instructions — High-Level Summary.....	5-12
5-14	Bit Manipulation Instructions — High-Level Summary .....	5-13
5-15	CPU Control Instructions — Binary Code Summary .....	5-15
5-16	Program Control Instructions — Binary Code Summary .....	5-16
5-16	Program Control Instructions — Binary Code Summary (Continued) .....	5-17
5-17	Data Transfer Instructions — Binary Code Summary.....	5-17
5-17	Data Transfer Instructions — Binary Code Summary (Continued).....	5-18
5-17	Data Transfer Instructions — Binary Code Summary (Concluded).....	5-19
5-18	Logic Instructions — Binary Code Summary.....	5-19
5-19	Arithmetic Instructions — Binary Code Summary .....	5-20
5-20	Bit Manipulation Instructions — Binary Code Summary .....	5-21
5-20	Bit Manipulation Instructions — Binary Code Summary (Continued).....	5-22
5-20	Bit Manipulation Instructions — Binary Code Summary (Concluded) .....	5-23

## List of Tables (Continued)

Table Number	Title	Page Number
6-1	Power Control Register (PCON) Organization.....	6-4
6-2	Instruction Cycle Times for CPU Clock Rates .....	6-5
6-3	Clock Output Mode Register (CLMOD) Organization .....	6-5
7-1	Interconnections Between Internal MCU and Caller ID.....	7-2
7-2	Pin Assignment in Caller ID Mode .....	7-3
7-3	Pin Configurations for Selecting Operation Modes.....	7-3
7-4	Recommended External Component Values for Typical Application .....	7-6
7-5	CAS Detector Parameters .....	7-9
7-6	FSK Receiver Parameters.....	7-10
7-7	Stutter Dial Tone Parameters .....	7-12
7-8	DTMF Frequencies Code Table.....	7-15
7-9	Bit Specification of the Address Field .....	7-17
7-10	Interrupt Sources of the CID Block.....	7-20
7-11	Register Overview .....	7-21
8-1	Interrupt Types and Corresponding Port Pin(s) .....	8-1
8-2	IS1 and IS0 Bit Manipulation for Multi-Level Interrupt Handling .....	7
8-3	Standard Interrupt Priorities.....	8
8-4	Interrupt Priority Register Settings .....	8
8-5	IMOD0 and IMOD1 Register Organization .....	9
8-6	IMOD2 Register Bit Settings.....	11
8-7	Interrupt Enable and Interrupt Request Flag Addresses .....	14
8-8	Interrupt Request Flag Conditions and Priorities .....	15
9-1	Hardware Operation During Power-Down Modes .....	9-2
9-2	Unused Pin Connections for Reduced Power Consumption .....	9-5
10-1	Hardware Register Values After RESET .....	10-2
10-1	Hardware Register Values After RESET (Continued).....	10-3
11-1	I/O Port Overview.....	11-1
11-1	I/O Port Overview (Continued).....	11-2
11-2	Port Pin Status During Instruction Execution.....	11-2
11-3	Port Mode Group Flags .....	11-3
11-4	Pull-Up Resistor Mode Register (PUMOD) Organization.....	11-4
12-1	Basic Timer Register Overview .....	12-3
12-2	Basic Timer Mode Register (BMOD) Organization.....	12-5
12-3	Watchdog Timer Interval Time .....	12-8
12-4	TC Register Overview .....	12-11
12-5	TMODn Settings for TCLn Edge Detection .....	12-14
12-6	TC Mode Register (TMODn) Organization .....	12-17
12-7	TMODn.6, TMODn.5, and TMODn.4 Bit Settings.....	12-18
12-8	Watch Timer Mode Register (WMOD) Organization .....	12-24

## List of Tables (Continued)

Table Page Number Number	Title	
13-1	Keyboard Arrangement.....	13-2
13-2	Tone Output Frequencies.....	13-2
13-3	DTMF Mode Register (DTMR) Organization.....	13-2
13-4	DTMR.7–DTMR.4 key Input Control Settings.....	13-3
13-5	DTMF Gain Register (DTGR) Organization.....	13-3
14-1	Absolute Maximum Ratings.....	14-2
14-2	D.C. Electrical Characteristics.....	14-2
14-2	D.C. Electrical Characteristics (Continued).....	14-3
14-2	D.C. Electrical Characteristics (Continued).....	14-4
14-3	Main System Clock Oscillator Characteristics.....	14-5
14-4	Input/Output Capacitance.....	14-6
14-5	A.C. Electrical Characteristics.....	14-6
14-6	RAM Data Retention Supply Voltage in Stop Mode.....	14-7
14-7	Electrical Characteristics of CID Block.....	14-11
14-8	CAS Timing Characteristics.....	14-12
14-9	SDT Timing Characteristics.....	14-12
14-10	Serial Interface Timing Characteristics.....	14-13
16-1	S3P7588X Pin Descriptions Used to Read/Write the EPROM.....	15-3
16-2	S3P7588X Features.....	15-3
16-3	Operating Mode Selection Criteria.....	15-3
17-1	Switch Settings for Power Configuration.....	17-5
17-1	Switch Settings for Power Configuration (Continued).....	17-6
17-2	Switch Settings for User Clock Selection.....	17-7
17-3	Switch Settings for Reset Signal of Caller ID.....	17-7





# List of Programming Tips

Description	Page Number
<b>Chapter 2: Address Spaces</b>	
Defining Vectored Interrupts .....	2-4
Using the REF Look-Up Table .....	2-6
Clearing Data Memory Banks 0 and 1 .....	2-9
Selecting the Working Register Area .....	2-13
Selecting the Working Register Area .....	2-14
Initializing the Stack Pointer.....	2-15
Using the BSC Register to Output 16-Bit Data .....	2-18
Setting ISx Flags for Interrupt Processing .....	2-20
Using the EMB Flag to Select Memory Banks.....	2-21
Using the ERB Flag to Select Register Banks.....	2-22
Using the Carry Flag as a 1-Bit Accumulator.....	2-24
<b>Chapter 3: Addressing Mode</b>	
Initializing the EMB and ERB Flags .....	3-3
1-Bit Addressing Modes .....	3-7
4-Bit Addressing Modes .....	3-8
4-Bit Addressing Modes (Continued).....	3-9
4-Bit Addressing Modes (Continued).....	3-10
8-Bit Addressing Modes .....	3-12

# 1

## PRODUCT OVERVIEW

The S3P7588X single-chip CMOS microcontroller has been designed for high-performance using SAM 47 (Samsung Arrangeable Microcontrollers). SAM 47, Samsung's newest 4-bit CPU core is notable for its low energy consumption and low operating voltage.

With its DTMF generator, watchdog timer function, versatile 8-bit timer/counters, and Caller ID module the S3P7588X offers an excellent design solution for a wide variety of telecommunication applications.

Up to 25 pins of the available 100-pin TQFP package can be assign to I/O. Six vectored interrupts provide fast response to internal and external events. In addition, the S3P7588X's advanced CMOS technology provides for low power consumption and a wide operating voltage range.

The S3P7588X has two package options, one is TQFP type and the other is pellet type. Only pellet type can be ordered for mass production. The TQFP type is provided only for system software development.

### OTP

The S3P7588X microcontroller has an on-chip 8K-byte one-time-programable EPROM instead of masked ROM, which provides comfortable environments for new application development.

## FEATURES SUMMARY

### Memory

- 768 × 4-bit RAM
- 8,192 × 8-bit EPROM

### 28 I/O Pins

- Input only: 3 pins
- I/O: 25 pins
- N-channel open-drain I/O: 8 pins

### Memory-Mapped I/O Structure

- Data memory bank 15

### Caller Id

- 1200 baud FSK (Frequency Shift Keying) demodulator with sensitivity -38dBm (600Ω) confirms to Bell 202 and CCITT V.23 standards
- Receive sensitivity of -32dBm (in 600Ω) for CAS (CPE Alerting Signal)
- Stutter Dial Tone (SDT) detector with sensitivity of -36dBm
- Ring or line reversal detector
- On-hook and off-hook applications according to Bellcore TR-NWT-000030 and SR-TSV-002476 specifications
- Compatible with ETSI standards ETS 300 659-1 and ETS 3000 659-2

### DTMF Generator

- 16 dual-tone frequencies for tone dialing

### 8-Bit Basic Timer

- Programmable interval timer
- Watchdog timer

### Two 8-Bit Timer/Counters

- Programmable 8-bit timer
- External event counter function
- Arbitrary clock frequency output

### Watch Timer

- Real-time and interval time measurement
- Four frequency outputs to the BUZ pin
- Bit Sequential Carrier
- Supports 8-bit serial data transfer in arbitrary format

### Interrupts

- 2 external interrupt vectors
- 4 internal interrupt vectors
- 2 quasi-interrupts

### Power-Down Modes

- Idle: Only CPU clock stops
- Stop: System clock stops

### Oscillation Sources

- Crystal, or ceramic for main system clock
- Main system clock frequency: 0.4–6.0MHz. 3.579545MHz is mandatory for caller id application
- CPU clock divider circuit (by 4, 8, or 64)

### Instruction Execution Times

- 0.95, 1.91, and 15.3μs at 4.19MHz
- 1.12, 2.23, 17.88μs at 3.58MHz
- 0.67, 1.33, 10.7μs at 6.0MHz

### Operating Temperature

- 0°C to 70°C

### Operating Voltage Range

- 3.0V to 5.5V

### Package Types

- 100-TQFP-1414 (only for system development)
- Pellet version is available (for mass production)

BLOCK DIAGRAM

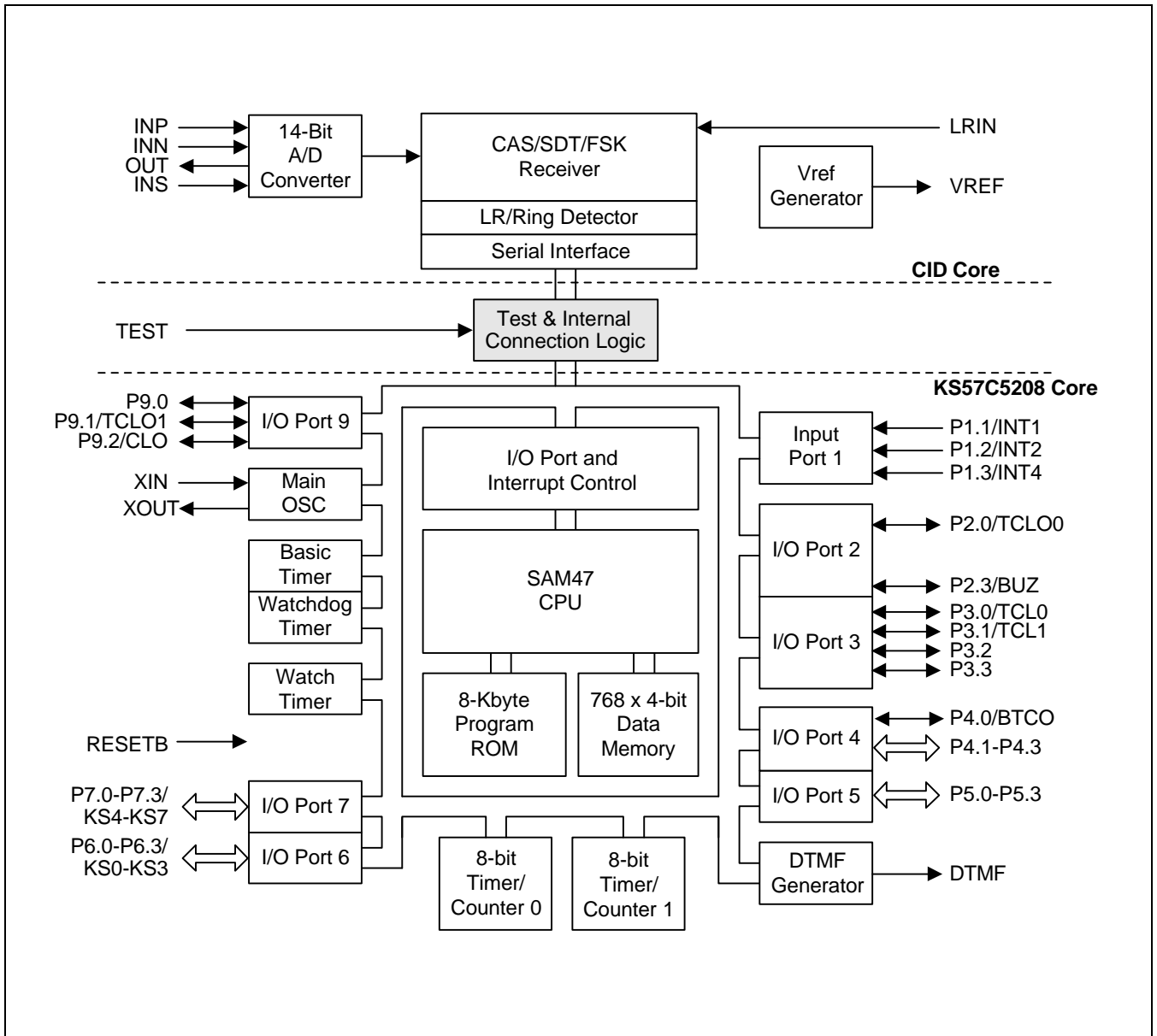


Figure 1-1. S3P7588X Simplified Block Diagram

PIN ASSIGNMENTS

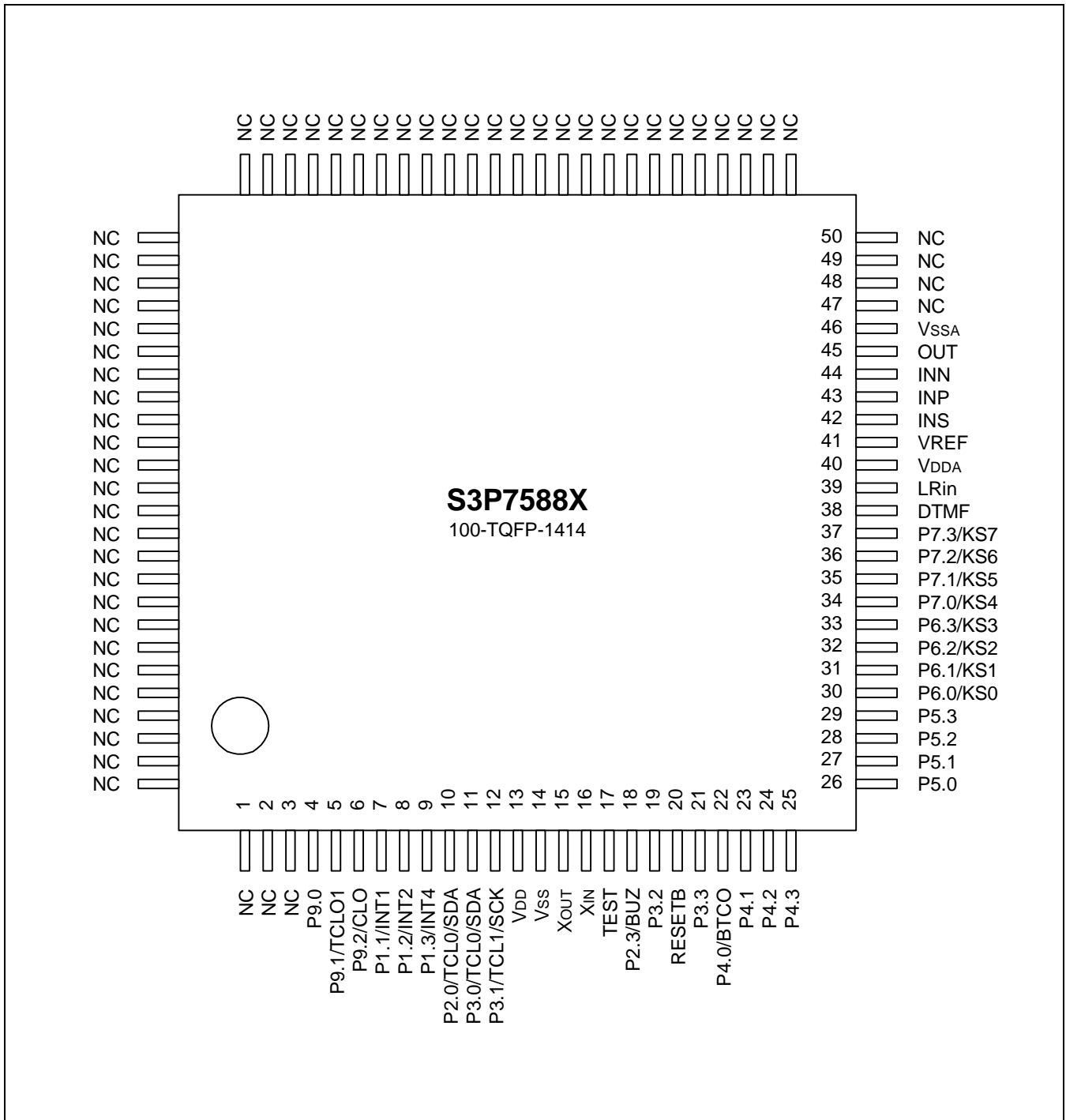


Figure 1-2. S3P7588X Pin Assignment Diagrams (100-TQFP-1414)

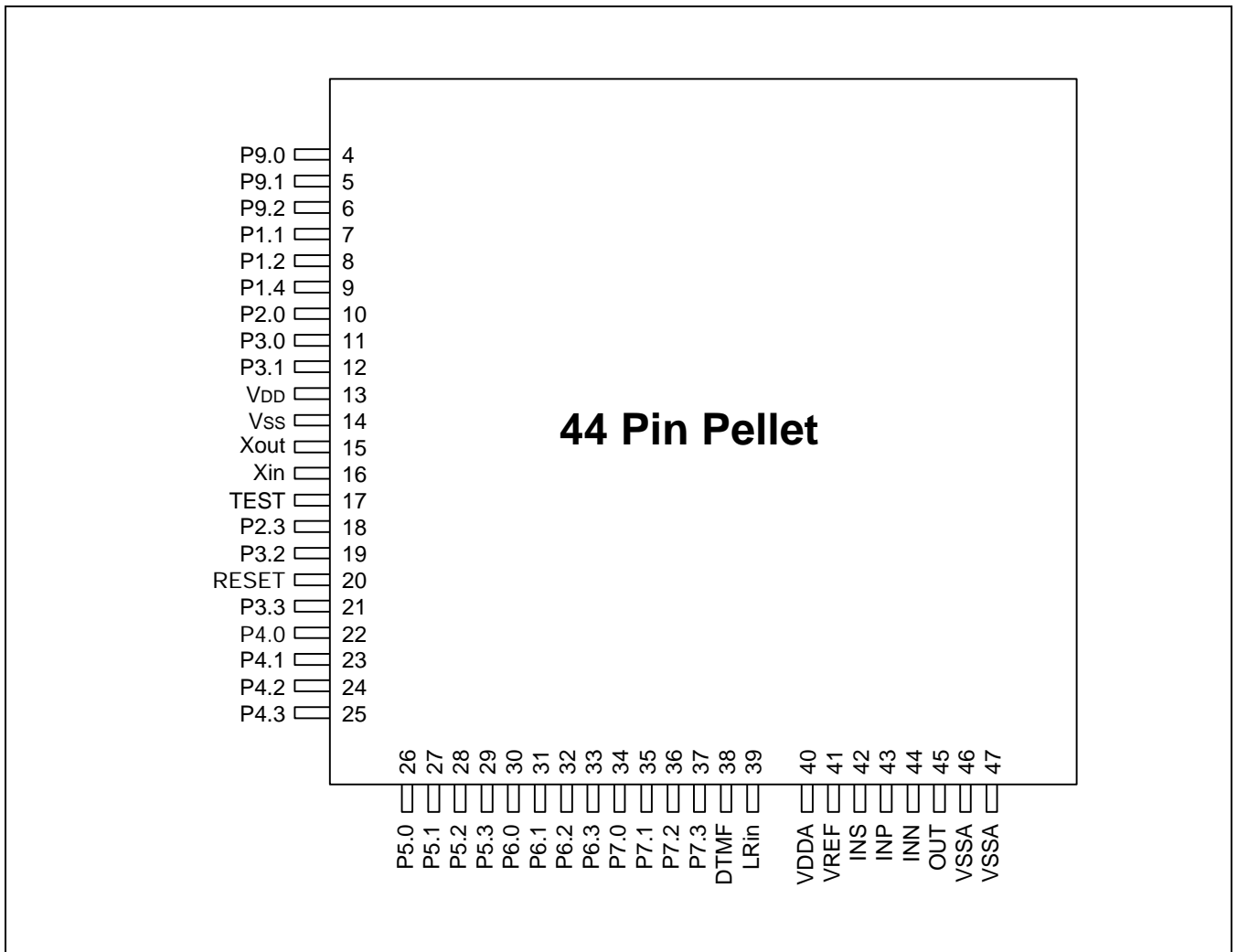


Figure 1-3. Pin Diagram of Pellet Type

## PIN DESCRIPTIONS

Table 1-1. S3P7588X Pin Descriptions

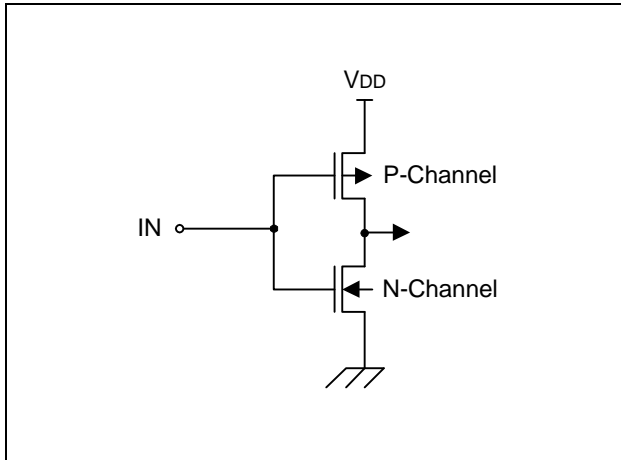
Pin Name	Pin Type	Reset Value	Description	Pin Number	Circuit Type
P1.1/INT1 P1.2/INT2 P1.3/INT4	I	I	3-bit Input port of Schmitt triggered type. 1-bit and 4-bit read and test is possible. Each port has software assignable pull-up resistor. P1.1-P1.3 are alternatively used as external interrupt input pins.	7 8 9	A-4
P2.0/TCLO0 P2.3/BUZ	I/O	I	2-bit I/O ports. 1-bit and 4-bit read/write and test is possible. Each individual pin is software configurable as input or output. 2-bit pull-up resistors are software assignable to input pins and are automatically disabled for output pins. Port 2, port 3 can be paired to enable 6-bit data transfer. P2.0 is alternatively used as the clock outputs of timer/counter 0. P2.3 is alternatively used as 2kHz, 4kHz, 8kHz, 16kHz frequency output at the watch timer clock frequency of 4.19MHz.	10 18	D-2
P3.0/TCL0 P3.1/TCL1 P3.2 P3.3	I/O	I	4-bit pull-up resistors are software assignable to input pins and are automatically disabled for output pins. Ports 2 and 3 can be paired to enable 8-bit data transfer. P3.0, P3.1 are alternatively used as external clock input for timer/counter 0, 1.	11 12 19 21	D-4
P4.0/BTCO P4.1-P4.3 P5.0-P5.3	I/O	I	4-bit I/O ports. 1-bit and 4-bit read/write and test is possible. Individual pins are software configurable as input or output. 4-bit pull-up resistors are software assignable to input pins and are automatically disabled for output pins. N-channel open-drain or push-pull output can be selected by software (1-bit unit) Ports 4 and 5 can be paired to support 8-bit data transfer.	22 23-25 26-29	E-2

Table 1-1. S3P7588X Pin Descriptions (Continued)

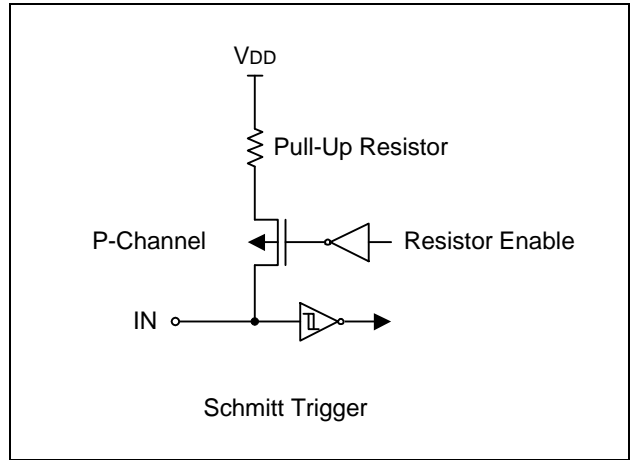
Pin Name	Pin Type	Reset Value	Description	Pin Number	Circuit Type
P6.0–P6.3 /KS0-KS3 P7.0–P7.3 /KS4-KS7	I/O	I	4-bit I/O ports. 1-bit and 4-bit read/write and test is possible. Each individual pin can be assignable as input or output. 4-bit pull-up resistors are software assignable to input pins and are automatically disabled for output pins. Port 6, port 7 can be paired to enable 8-bit data transfer. Port 6, port 7 are alternatively used as two quasi-interrupt inputs with falling edge detection.	30-33  34-37	D-4
P9.0 P9.1/TCLO1 P9.2/CLO	I/O	I	4-bit I/O port. 1-bit or 4-bit read/write and test is possible. Individual pins are software configurable as input or output. 3-bit pull-up resistors are software assignable to input pins and are automatically disabled for output pins. P9.2 is alternatively used as a clock output, and P9.1 is alternatively used as the clock output of timer/counter 1.	4 5 6	D-2
DTMF	O	–	DTMF output.	38	C
INP	I	–	Op-amp positive signal input for CAS, FSK and SDT	43	
INN	I	–	Op-amp negative signal input for CAS, FSK and SDT	44	
INS	I	–	Op-amp single-ended signal input for CAS, FSK and SDT	42	
OUT	O	–	Op-amp output signal for CAS, FSK and SDT	45	
VREF	O	–	Reference voltage for Op-amp signals	41	
LRin	I	–	Input for line reversal or ring detection	39	B-1
VDD	–	–	Digital Power supply	13	–
VSS	–	–	Digital ground	14	–
VDDA	–	–	Analog power supply	40	–
VSSA	–	–	Analog ground	46	–
RESETB	–	–	RESET signal (low active)	20	B
X <sub>in</sub> X <sub>out</sub>	–	–	Crystal, or ceramic oscillator signal for main system clock. (For external clock input, use X <sub>in</sub> and input X <sub>in</sub> 's reverse phase to X <sub>out</sub> )	16 15	–
TEST	–	–	Test signal input (high active)	17	–
NC	–	–	No connection	–	–



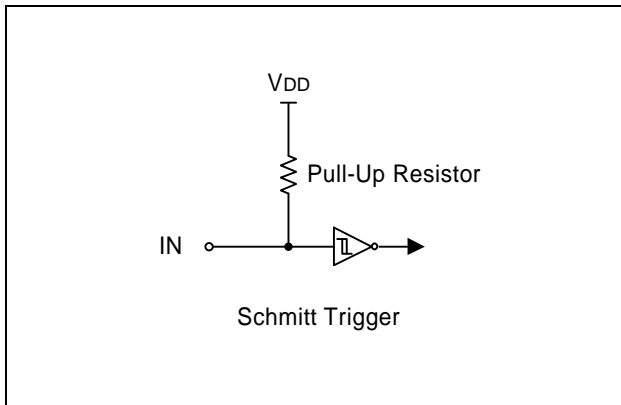
**PIN CIRCUIT DIAGRAMS**



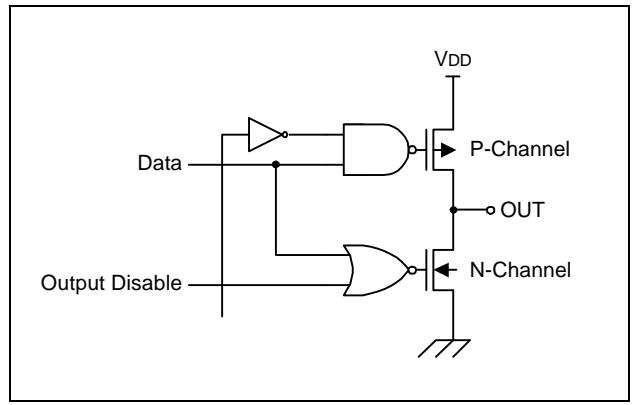
**Figure 1-4. Pin Circuit Type A**



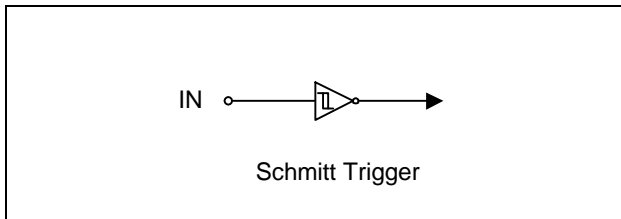
**Figure 1-7. Pin Circuit Type A-4**



**Figure 1-5. Pin Circuit Type B**



**Figure 1-8. Pin Circuit Type C**



**Figure 1-6. Pin Circuit Type B-1**

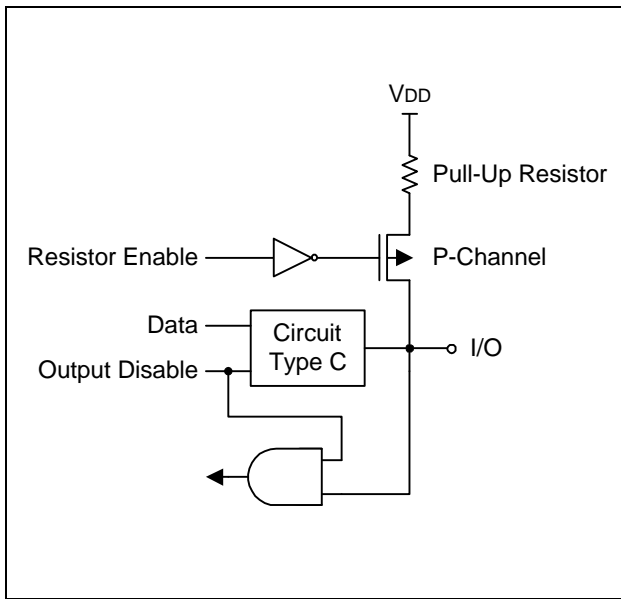


Figure 1-9. Pin Circuit Type D-2

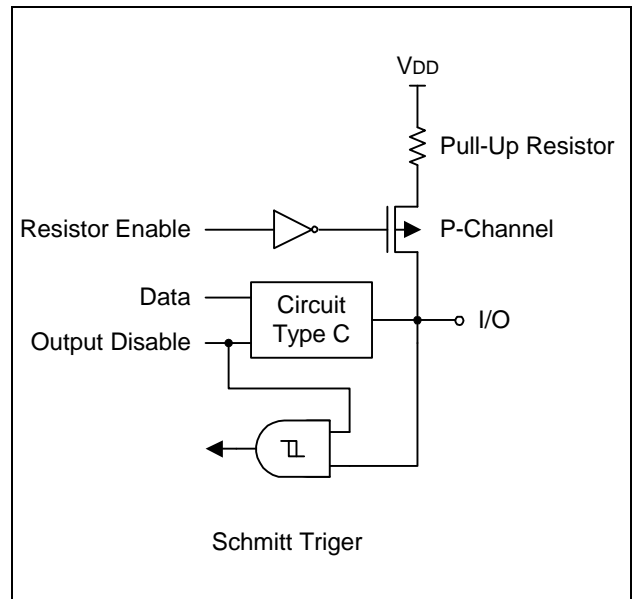


Figure 1-11. Pin Circuit Type D-4

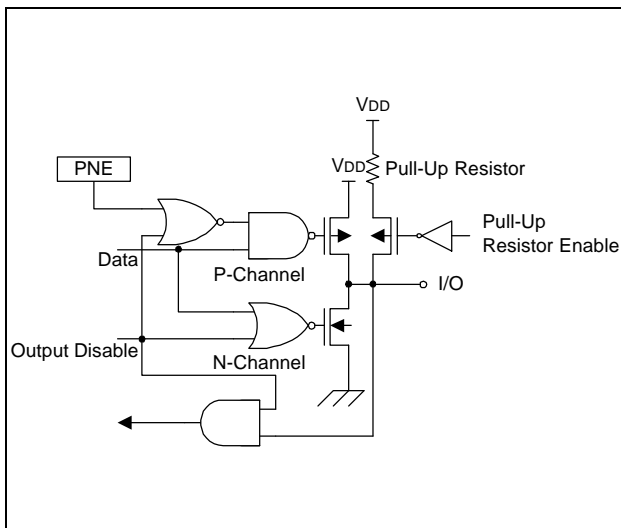


Figure 1-10. Pin Circuit Type E-2

## NOTES

# 2 ADDRESS SPACES

## OVERVIEW

Program ROM maps for the S3P7588X are one time programmable at the application field. In its standard configuration, the device's  $8,192 \times 8$ -bit program memory have three areas that are directly addressable by the program counter (PC):

- 16-byte area for vector addresses
- 16-byte general-purpose area
- 96-byte instruction reference area
- 8,064-byte general-purpose area

## GENERAL-PURPOSE PROGRAM MEMORY (ROM)

Two program memory areas are allocated for general-purpose use: One area is 16 bytes in size and the other is 8,064 bytes.

### Vector Addresses

A 16-byte vector address area is used to store the vector addresses required to execute system resets and interrupts. Start addresses for interrupt service routines are stored in this area, along with the values of the enable memory bank (EMB) and enable register bank (ERB) flags that are used to initialize the corresponding service routines. The 16-byte area can be used alternately as general-purpose ROM.

### REF Instructions

Locations 0020H–007FH are used as a reference area (look-up table) for 1-byte REF instructions. The REF instruction reduces the byte size of instruction operands. REF can reference one 2-byte instruction, two 1-byte instructions, and three-byte instructions which are stored in the look-up table. Unused look-up table addresses can be used as general-purpose ROM.

Table 2-1. Program Memory Address Ranges

ROM Area Function	Address Ranges	Area Size (in Bytes)
Vector address area	0000H–000FH	16
General-purpose program memory	0010H–001FH	16
REF instruction look-up table area	0020H–007FH	96
General-purpose program memory	0080H–1FFFFH	8,064

### General-Purpose Memory Areas

The 16-byte area at ROM locations 0010H–001FH and the 8,064-byte area at ROM locations 0080H–1FFFFH are used as general-purpose program memory. Unused locations in the vector address area and REF instruction look-up table areas can be used as general-purpose program memory. However, care must be taken not to overwrite live data when writing programs that use special-purpose areas of the ROM.

### Vector Address Area

The 16-byte vector address area of the ROM is used to store the vector addresses for executing system resets and interrupts. The starting addresses of interrupt service routines are stored in this area, along with the enable memory bank (EMB) and enable register bank (ERB) flag values that are needed to initialize the service routines. 16-byte vector addresses are organized as follows:

EMB	ERB	0	PC12	PC11	PC10	PC9	PC8
PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

To set up the vector address area for specific programs, use the instruction VENTn. The programming tips on the next page explain how to do this.

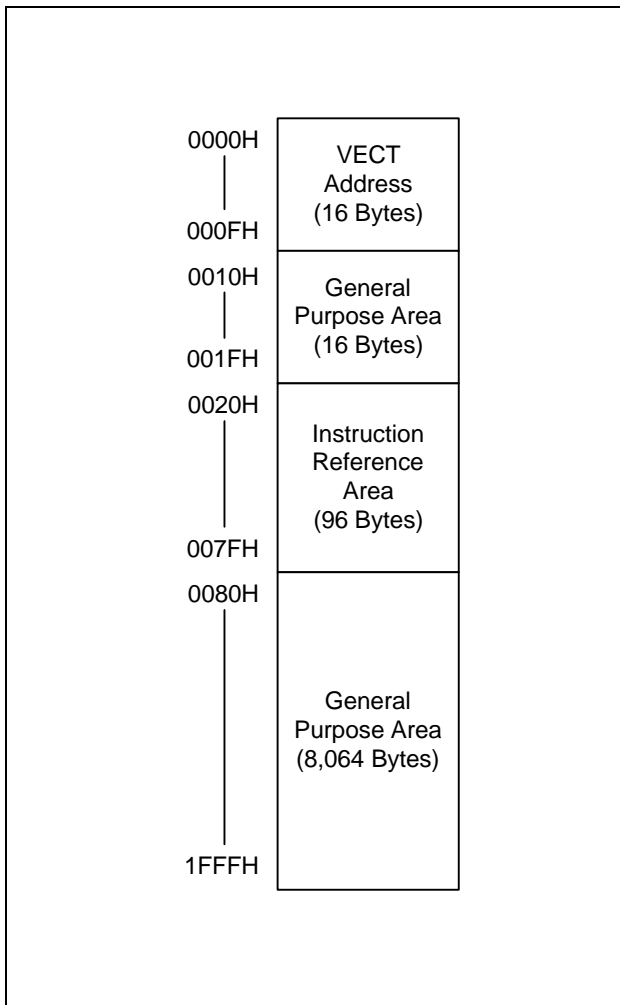


Figure 2-1. ROM Address Structure

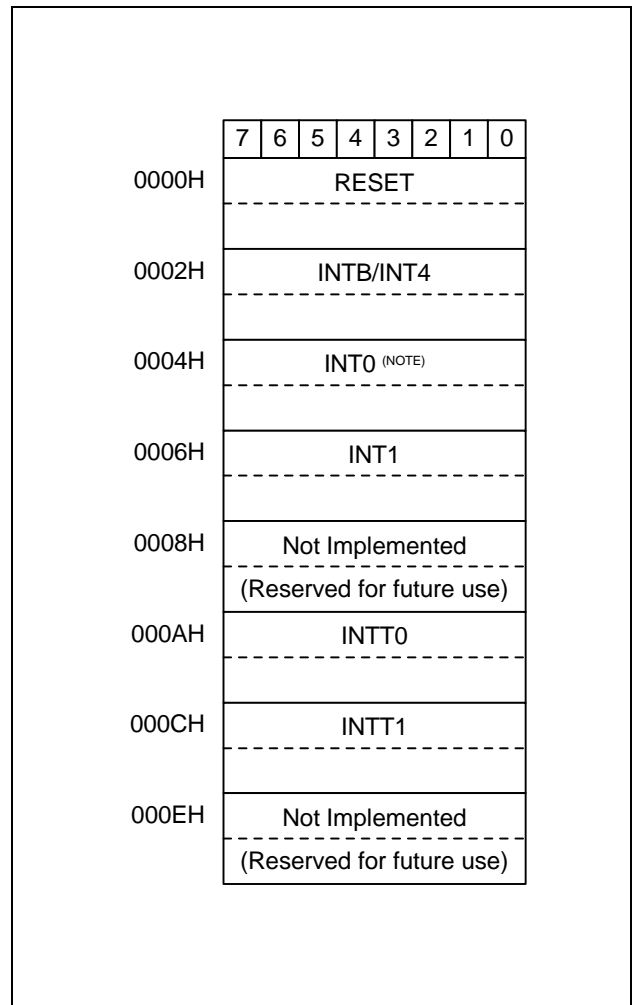


Figure 2-2. Vector Address Map

**NOTE:** INT0 is dedicated to caller id interrupt

### PROGRAMMING TIP — Defining Vectored Interrupts

The following examples show you several ways you can define the vectored interrupt and instruction reference areas in program memory:

1. When all vector interrupts are used:

```

ORG      0000H

VENT0    1,0,RESET      ; EMB ← 1, ERB ← 0; Jump to RESET address
VENT1    0,0,INTB       ; EMB ← 0, ERB ← 0; Jump to INTB address
VENT2    0,0,INT0       ; EMB ← 0, ERB ← 0; Jump to INT0 address
VENT3    0,0,INT1       ; EMB ← 0, ERB ← 0; Jump to INT1 address
NOP
NOP
VENT5    0,0,INTT0      ; EMB ← 0, ERB ← 0; Jump to INTT0 address
VENT6    0,0,INTT1      ; EMB ← 0, ERB ← 0; Jump to INTT1 address

```

2. When a specific vectored interrupt such as INT0, and INTT0 is not used, the unused vector interrupt locations must be skipped with the assembly instruction ORG so that jumps will address the correct locations:

```

ORG      0000H

VENT0    1,0,RESET      ; EMB ← 1, ERB ← 0; Jump to RESET address
VENT1    0,0,INTB       ; EMB ← 0, ERB ← 0; Jump to INTB address

ORG      0006H          ; INT0 interrupt not used

VENT3    0,0,INT1       ; EMB ← 0, ERB ← 0; Jump to INT1 address

ORG      000CH          ; INTT0 interrupt not used

VENT6    0,0,INTT1      ; EMB ← 0, ERB ← 0; Jump to INTT1 address
ORG      0010H

```

3. If an INT0 interrupt is not used and if its corresponding vector interrupt area is not fully utilized, or if it is not written by a ORG instruction as in Example 2, a CPU malfunction will occur:

```

ORG      0000H

VENT0    1,0,RESET      ; EMB ← 1, ERB ← 0; Jump to RESET address
VENT1    0,0,INTB       ; EMB ← 0, ERB ← 0; Jump to INTB address
VENT3    0,0,INT1       ; EMB ← 0, ERB ← 0; Jump to INT0 address
NOP
NOP
VENT5    0,0,INTT0      ; EMB ← 0, ERB ← 0; Jump to INT1 address
VENT6    0,0,INTT1      ; EMB ← 0, ERB ← 0; Jump to INTT0 address

ORG      0010H

```

**General-Purpose ROM Area**

In this example, when an INTT0 interrupt is generated, the corresponding vector area is not VENT5 INTT0, but VENT6 INTT1. This causes an INTT0 interrupt to jump incorrectly to the INTT1 address and causes a CPU malfunction to occur.

**Instruction Reference Area**

Using 1-byte REF instructions, you can easily reference instructions with larger byte sizes that are stored in addresses 0020H–007FH of program memory. This 96-byte area is called the REF instruction reference area, or look-up table. Locations in the REF look-up table may contain two one-byte instructions, a single two-byte instruction, or three-byte instruction such as a JP (jump) or CALL. The starting address of the instruction you are referencing must always be an even number. To reference a JP or CALL instruction, it must be written to the reference area in a two-byte format: for JP, this format is TJP; for CALL, it is TCALL.

You can use REF instructions to execute instructions larger than one byte. There are three ways you can use REF instruction:

- Using the 1-byte REF instruction to execute one 2-byte or two 1-byte instructions,
- Branching to any location by referencing a branch instruction stored in the look-up table,
- Calling subroutines at any location by referencing a call instruction stored in the look-up table.



 **Programming Tip — Using the REF Look-Up Table**

Here is one example of how to use the REF instruction look-up table:

```

                ORG      0020H

JMAIN          TJP      MAIN          ; 0, MAIN
KEYCK          BTSF    KEYFG         ; 1, KEYFG check
WATCH         TCALL   CLOCK         ; 2, call CLOCK
INCHL         LD      @HL,A         ; 3, (HL) ← A
                INCS   HL
                .
                .
                .
ABC           LD      EA,#00H       ; 47, EA ← #00H
                ORG      0080

MAIN          NOP
                NOP
                .
                .
                .
                REF    KEYCK         ; BTSF KEYFG (1-byte instruction)
                REF    JMAIN        ; KEYFG = 1, jump to MAIN (1-byte instruction)
                REF    WATCH       ; KEYFG = 0, call CLOCK (1-byte instruction)
                REF    INCHL        ; LD @HL,A
                REF    INCS HL      ; INCS HL
                REF    ABC          ; LD EA,#00H (1-byte instruction)
                .
                .
                .

```

## DATA MEMORY (RAM)

### Overview

In its standard configuration, the  $896 \times 4$ -bit data memory has five areas:

- $32 \times 4$ -bit working register area
- $224 \times 4$ -bit general-purpose area (also used as stack area)
- $2 \times 256 \times 4$ -bit general-purpose area
- $128 \times 4$ -bit area for peripheral hardware

To make it easier to reference, the data memory area has four memory banks — bank 0, bank 1, bank 2, and bank 15. The select memory bank instruction (SMB) is used to select the bank you want to select as working data memory. Data stored in RAM locations are 1-, 4-, and 8-bit addressable.

Initialization values for the data memory area are not defined by hardware and must therefore be initialized by program software following RESETB. However, when RESETB signal is generated in power-down mode, the data memory contents are held.

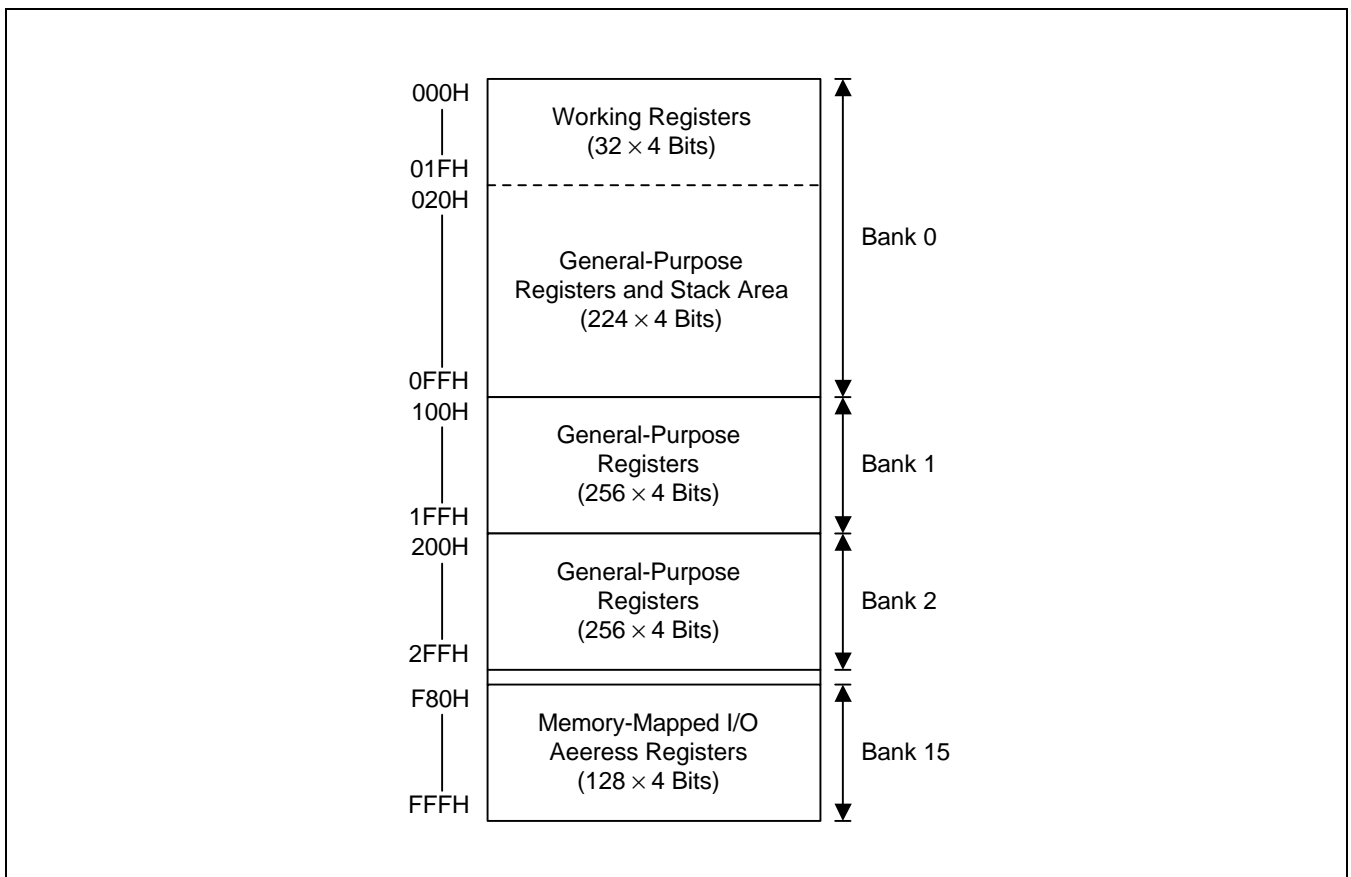


Figure 2-3. Data Memory (RAM) Map

## Memory Banks 0, 1, 2, and 15

Bank 0	(000H–0FFH)	The lowest 32 nibbles of bank 0 (000H–01FH) are used as working registers; the next 224 nibbles (020H–0FFH) can be used both as stack area and as general-purpose data memory. Use the stack area for implementing subroutine calls and returns, and for interrupt processing.
Bank 1	(100H–1FFH)	The 256 nibbles of bank 1 (100H–1FFH) are for general-purpose use.
Bank 2	(200H–2FFH)	The 256 nibbles of bank 2 (200H–2FFH) are for general-purpose use.
Bank 15	(F80H–FFFH)	The microcontroller uses bank 15 for memory-mapped peripheral I/O. Fixed RAM locations for each peripheral hardware register: the port latches, timers, peripherals controls, etc. are mapped into this area.

**Data Memory Addressing Modes**

The enable memory bank (EMB) flag controls the addressing mode for data memory banks 0, 1, 2 or 15. When the EMB flag is logic zero, the addressable area is restricted to specific locations, depending on whether direct or indirect addressing is used. With direct addressing, you can access locations 000H–07FH of bank 0 and bank 15. With indirect addressing, only bank 0 (000H–0FFH) can be accessed. When the EMB flag is set to logic one, all four data memory banks can be accessed according to the current SMB value.

For 8-bit addressing, two 4-bit registers are addressed as a register pair. Also, when using 8-bit instructions to address RAM locations, remember to use the even-numbered register address as the instruction operand.

**Working Registers**

The RAM working register area in data memory bank 0 is further divided into four register banks (bank 0, 1, 2, and 3). Each register bank has eight 4-bit registers and paired 4-bit registers are 8-bit addressable.

Register A is used as a 4-bit accumulator and register pair EA as an 8-bit extended accumulator. The carry flag bit can also be used as a 1-bit accumulator. Register pairs WX, WL, and HL are used as address pointers for indirect addressing. To limit the possibility of data corruption due to incorrect register addressing, it is advisable to use register bank 0 for the main program and banks 1, 2, and 3 for interrupt service routines.

Table 2-2. Data Memory Organization and Addressing

Addresses	Register Areas	Bank	EMB Value	SMB Value
000H–01FH	Working registers	0	0, 1	0
020H–0FFH	Stack and general-purpose registers			
100H–1FFH	General-purpose registers	1	1	1
200H–2FFH	General-purpose registers	2	1	2
F80H–FFFH	Peripheral hardware registers	15	0, 1	15

 **PROGRAMMING TIP — Clearing Data Memory Banks 0 and 1**

Clear banks 0 and 1 of the data memory area:

```

RAMCLR  SMB    1                ; RAM (100H–1FFH) clear
         LD     HL,#00H
         LD     A,#0H
RMCL1   LD     @HL,A
         INCS  HL
         JR     RMCL1

         SMB    0                ; RAM (010H–0FFH) clear
RMCL0   LD     HL,#10H
         LD     @HL,A
         INCS  HL
         JR     RMCL0

```

## Working Registers

Working registers, mapped to RAM address 000H–01FH in data memory bank 0, are used to temporarily store intermediate results during program execution, as well as pointer values used for indirect addressing. Unused registers may be used as general-purpose memory. Working register data can be manipulated as 1-bit units, 4-bit units or, using paired registers, as 8-bit units.

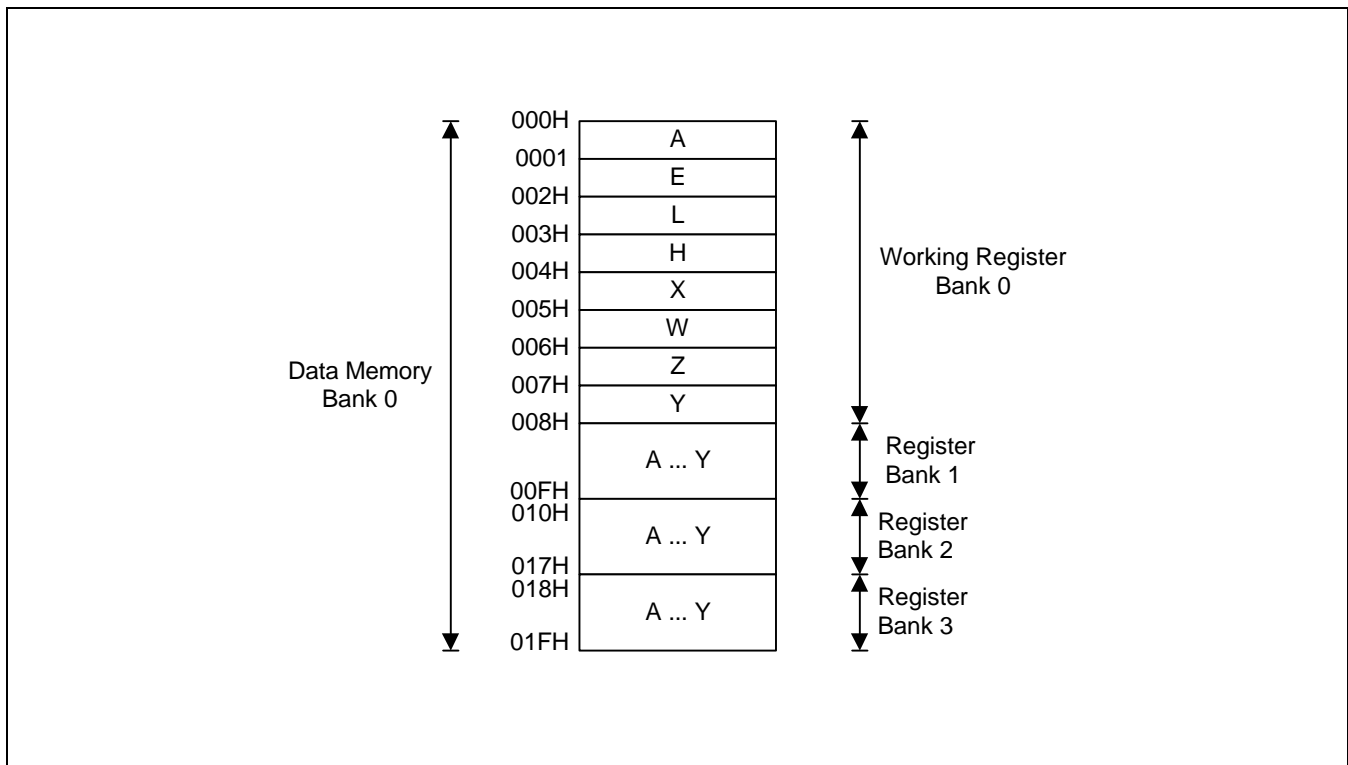


Figure 2-4. Working Register Map

### Working Register Banks

For addressing purposes, the working register area is divided into four register banks — bank 0, bank 1, bank 2, and bank 3. Any one of these banks can be selected as the working register bank by the register bank selection instruction (SRB n) and by setting the status of the register bank enable flag (ERB).

Generally, working register bank 0 is used for the main program, and banks 1, 2, and 3 for interrupt service routines. Following this convention helps to prevent possible data corruption during program execution due to contention in register bank addressing.

**Table 2-3. Working Register Organization and Addressing**

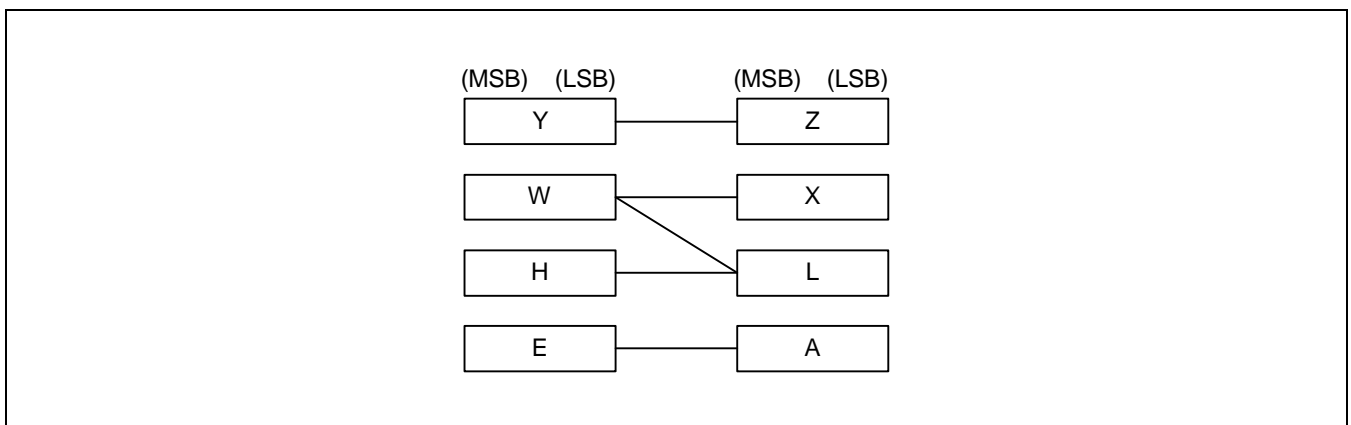
ERB Setting	SRB Settings				Selected Register Bank
	3	2	1	0	
0	0	0	x	x	Always set to bank 0
1	0	0	0	0	Bank 0
			0	1	Bank 1
			1	0	Bank 2
			1	1	Bank 3

**NOTE:** x = not applicable.

### Paired Working Registers

Each of the register banks is subdivided into eight 4-bit registers. These registers, named Y, Z, W, X, H, L, E and A, can either be manipulated individually using 4-bit instructions, or together as register pairs for 8-bit data manipulation.

The names of the 8-bit register pairs in each register bank are EA, HL, WX, YZ and WL. Registers A, L, X and Z always become the lower nibble when registers are addressed as 8-bit pairs. This makes a total of eight 4-bit registers or four 8-bit double registers in each of the four working register banks.

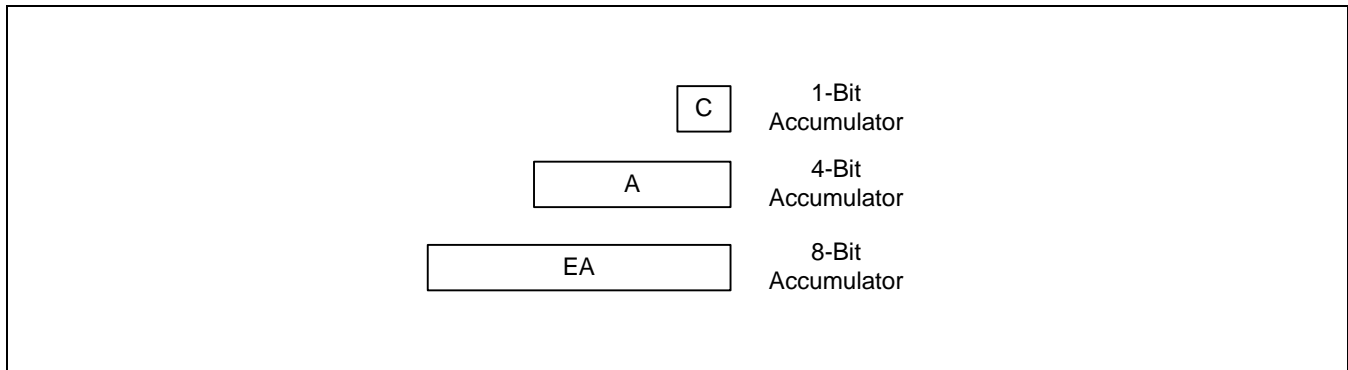


**Figure 2-5. Register Pair Configuration**

### Special-Purpose Working Registers

Register A is used as a 4-bit accumulator and double register EA as an 8-bit accumulator. The carry flag can also be used as a 1-bit accumulator.

8-bit double registers WX, WL and HL are used as data pointers for indirect addressing. When the HL register serves as a data pointer, the instructions LDI, LDD, XCHI, and XCHD can make very efficient use of working registers as program loop counters by letting you transfer a value to the L register and increment or decrement it using a single instruction.



**Figure 2-6. 1-Bit, 4-Bit, and 8-Bit Accumulator**

### Recommendation for Multiple Interrupt Processing

If more than four interrupts are being processed at one time, you can avoid possible loss of working register data by using the PUSH RR instruction to save register contents to the stack before the service routines are executed in the same register bank. When the routines have executed successfully, you can restore the register contents from the stack to working memory using the POP instruction.

 **PROGRAMMING TIP — Selecting the Working Register Area**

The following examples show the correct programming method for selecting working register area:

1. When ERB = "0":

```

VENT2      1,0,INT0          ; EMB ← 1, ERB ← 0, Jump to INT0 address
;
INT0       PUSH      SB      ; PUSH current SMB, SRB
          SRB        2      ; Instruction does not execute because ERB = "0"
          PUSH      HL      ; PUSH HL register contents to stack
          PUSH      WX      ; PUSH WX register contents to stack
          PUSH      YZ      ; PUSH YZ register contents to stack
          PUSH      EA      ; PUSH EA register contents to stack
          SMB        0
          LD        EA,#00H
          LD        80H,EA
          LD        HL,#40H
          INCS      HL
          LD        WX,EA
          LD        YZ,EA
          POP       EA      ; POP EA register contents from stack
          POP       YZ      ; POP YZ register contents from stack
          POP       WX      ; POP WX register contents from stack
          POP       HL      ; POP HL register contents from stack
          POP       SB      ; POP current SMB, SRB
          IRET

```

The POP instructions execute alternately with the PUSH instructions. If an SMB n instruction is used in an interrupt service routine, a PUSH and POP SB instruction must be used to store and restore the current SMB and SRB values, as shown in Example 2 below.

2. When ERB = "1":

```

VENT2      1,1,INT0        ; EMB ← 1, ERB ← 1, Jump to INT0 address
;
INT0       PUSH      SB      ; Store current SMB, SRB
          SRB        2      ; Select register bank 2 because of ERB = "1"
          SMB        0
          LD        EA,#00H
          LD        80H,EA
          LD        HL,#40H
          INCS      HL
          LD        WX,EA
          LD        YZ,EA
          POP       SB      ; Restore SMB, SRB
          IRET

```



 **PROGRAMMING TIP — Selecting the Working Register Area**

The following examples show the correct programming method for selecting working register area:

1. When ERB = "0":

```

VENT2      1,0,INT0          ; EMB ← 1, ERB ← 0, Jump to INT0 address

INT0       PUSH      SB      ; PUSH current SMB, SRB
           SRB        2      ; Instruction does not execute because ERB = "0"
           PUSH      HL      ; PUSH HL register contents to stack
           PUSH      WX      ; PUSH WX register contents to stack
           PUSH      YZ      ; PUSH YZ register contents to stack
           PUSH      EA      ; PUSH EA register contents to stack
           SMB        0
           LD        EA,#00H
           LD        80H,EA
           LD        HL,#40H
           INCS      HL|
           LD        WX,EA
           LD        YZ,EA
           POP       EA      ; POP EA register contents from stack
           POP       YZ      ; POP YZ register contents from stack
           POP       WX      ; POP WX register contents from stack
           POP       HL      ; POP HL register contents from stack
           POP       SB      ; POP current SMB, SRB
           IRET

```

The POP instructions execute alternately with the PUSH instructions. If an SMB n instruction is used in an interrupt service routine, a PUSH and POP SB instruction must be used to store and restore the current SMB and SRB values, as shown in Example 2 below.

2. When ERB = "1":

```

VENT2      1,1,INT0        ; EMB ← 1, ERB ← 1, Jump to INT0 address

INT0       PUSH      SB      ; Store current SMB, SRB
           SRB        2      ; Select register bank 2 because of ERB = "1"
           SMB        0
           LD        EA,#00H
           LD        80H,EA
           LD        HL,#40H
           INCS      HL
           LD        WX,EA
           LD        YZ,EA
           POP       SB      ; Restore SMB, SRB
           IRET

```

## STACK OPERATIONS

### Stack Pointer (SP)

The stack pointer (SP) is an 8-bit register that stores the address used to access the stack, an area of data memory set aside for temporary storage of stack addresses. The SP can be read or written by 8-bit control instructions. When addressing the SP, bit 0 must always remain cleared to logic zero.

F80H	SP3	SP2	SP1	"0"
F81H	SP7	SP6	SP5	SP4

There are two basic stack operations: writing to the top of the stack (push), and reading from the top of the stack (pop). A push decrements the SP and a pop increments it so that the SP always points to the top address of the last data to be written to the stack.

The program counter contents and program status word are stored in the stack area prior to the execution of a CALL or a PUSH instruction, or during interrupt service routines. Stack operation is a LIFO (Last In-First Out) type. The stack area is located in general-purpose data memory bank 0.

During an interrupt or a subroutine, the PC value and the PSW are saved to the stack area. When the routine has completed, the stack pointer is referenced to restore the PC and PSW, and the next instruction is executed.

The SP can address stack registers in bank 0 (addresses 000H-0FFH) regardless of the current value of the enable memory bank (EMB) flag and the select memory bank (SMB) flag. Although general-purpose register areas can be used for stack operations, be careful to avoid data loss due to simultaneous use of the same register(s).

Since the reset value of the stack pointer is not defined in firmware, we recommend that you initialize the stack pointer by program code to location 00H. This sets the first register of the stack area to 0FFH.

### NOTE

A subroutine call occupies six nibbles in the stack; an interrupt requires six. When subroutine nesting or interrupt routines are used continuously, the stack area should be set in accordance with the maximum number of subroutine levels. To do this, estimate the number of nibbles that will be used for the subroutines or interrupts and set the stack area correspondingly.

### PROGRAMMING TIP — Initializing the Stack Pointer

To initialize the stack pointer (SP):

- When EMB = "1":

```
SMB      15          ; Select memory bank 15
LD       EA,#00H    ; Bit 0 of accumulator A is always cleared to "0"
LD       SP,EA      ; Stack area initial address (0FFH) ← (SP) – 1
```

- When EMB = "0":

```
LD       EA,#00H
LD       SP,EA      ; Memory addressing area (00H–7FH, F80H–FFFH)
```

**Push Operations**

Three kinds of push operations reference the stack pointer (SP) to write data from the source register to the stack: PUSH instructions, CALL instructions, and interrupts. In each case, the SP is *decreased* by a number determined by the type of push operation and then points to the next available stack location.

**Push Instructions**

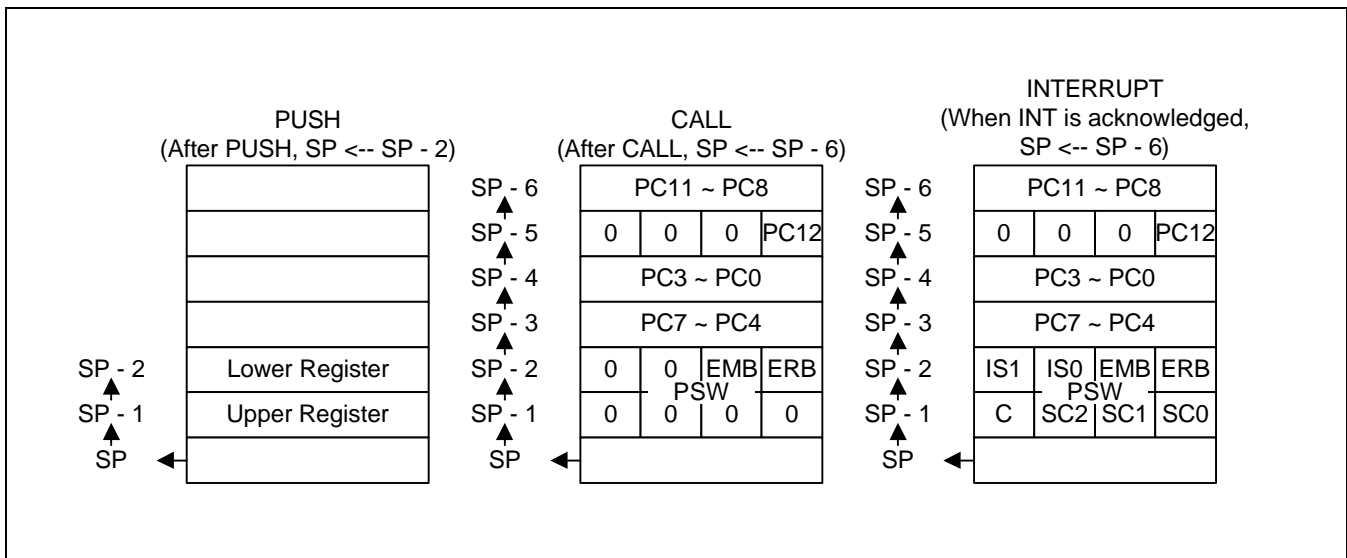
A PUSH instruction references the SP to write two 4-bit data nibbles to the stack. Two 4-bit stack addresses are referenced by the stack pointer: one for the upper register value and another for the lower register. After the PUSH has executed, the SP is decreased *by two* and points to the next available stack location.

**Call Instructions**

When a subroutine call is issued, the CALL instruction references the SP to write the PC's contents to six 4-bit stack locations. Current values for the enable memory bank (EMB) flag and the enable register bank (ERB) flag are also pushed to the stack. Since six 4-bit stack locations are used per CALL, you may nest subroutine calls up to the number of levels permitted in the stack.

**Interrupt Routines**

An interrupt routine references the SP to push the contents of the PC and the program status word (PSW) to the stack. Six 4-bit stack locations are used to store this data. After the interrupt has executed, the SP is decreased *by six* and points to the next available stack location. During an interrupt sequence, subroutines may be nested up to the number of levels which are permitted in the stack area.



**Figure 2-7. Push-Type Stack Operations**

**POP Operations**

For each push operation there is a corresponding pop operation to write data from the stack back to the source register or registers: for the PUSH instruction it is the POP instruction; for CALL, the instruction RET or SRET; for interrupts, the instruction IRET. When a pop operation occurs, the SP is *incremented* by a number determined by the type of operation and points to the next free stack location.

**POP Instructions**

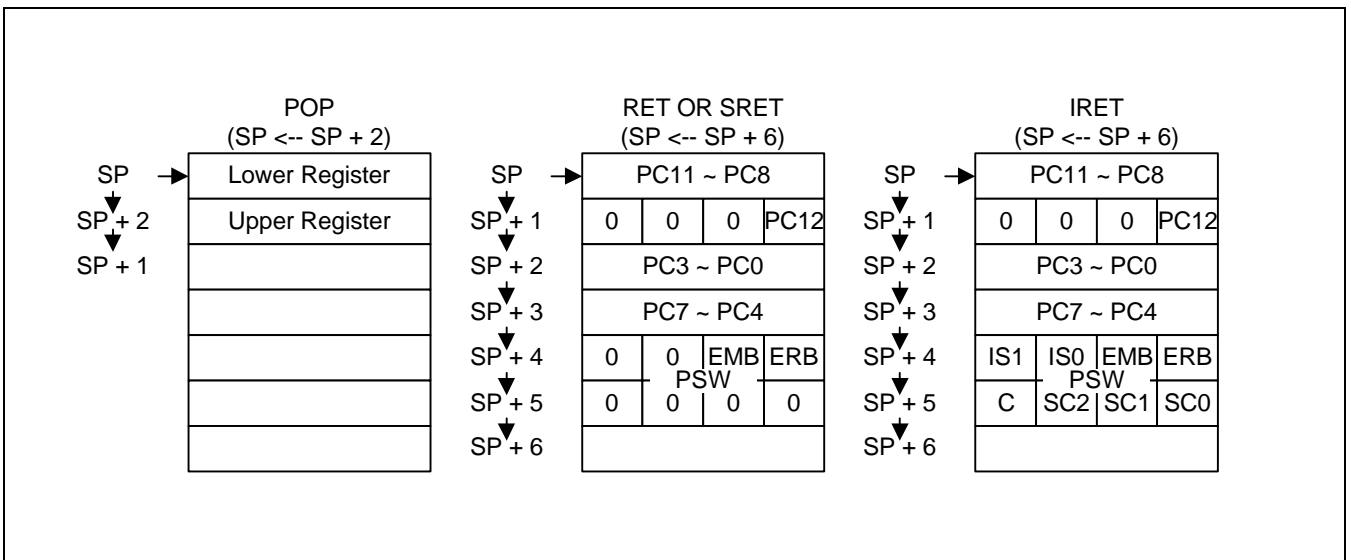
A POP instruction references the SP to write data stored in two 4-bit stack locations back to the register pairs and SB register. The value of the lower 4-bit register is popped first, followed by the value of the upper 4-bit register. After the POP has executed, the SP is incremented *by two* and points to the next free stack location.

**RET and SRET Instructions**

The end of a subroutine call is signaled by the return instruction, RET or SRET. The RET or SRET uses the SP to reference the six 4-bit stack locations used for the CALL and to write this data back to the PC, the EMB, and the ERB. After the RET or SRET has executed, the SP is incremented *by six* and points to the next free stack location.

**IRET Instructions**

The end of an interrupt sequence is signaled by the instruction IRET. IRET references the SP to locate the six 4-bit stack addresses used for the interrupt and to write this data back to the PC and the PSW. After the IRET has executed, the SP is incremented *by six* and points to the next free stack location.



**Figure 2-8. Pop-Type Stack Operations**

### Bit Sequential Carrier (BSC)

The bit sequential carrier (BSC) is a 8-bit general register that can be manipulated using 1-, 4-, and 8-bit RAM control instructions. RESETB clears all BSC bit values to logic zero.

Using the BSC, you can specify sequential addresses and bit locations using 1-bit indirect addressing (memb.@L). (Bit addressing is independent of the current EMB value.) This way, programs can process 8-bit data by moving the bit location sequentially and then incrementing or decreasing the value of the L register.

BSC data can also be manipulated using direct addressing.

If the values of the L register are 0H at BSC2.@L, the address and bit location assignment is FC2H.0. If the L register content is 8H at BSC2.@L, the address and bit location assignment is FC3H.3.

**Table 2-4. BSC Register Organization**

Name	Address	Bit 3	Bit 2	Bit 1	Bit 0
BSC0	FC0H	BSC0.3	BSC0.2	BSC0.1	BSC0.0
BSC1	FC1H	BSC1.3	BSC1.2	BSC1.1	BSC1.0
BSC2	FC2H	BSC2.3	BSC2.2	BSC2.1	BSC2.0
BSC3	FC3H	BSC3.3	BSC3.2	BSC3.1	BSC3.0

#### PROGRAMMING TIP — Using the BSC Register to Output 16-Bit Data

To use the bit sequential carrier (BSC) register to output 8-bit data (59H) to the P2.3 pin:

```

        BITS      EMB
        SMB       15
        LD        EA,#59H          ;
        LD        BSC2,EA        ; BSC2 ← A, BSC3 ← E
        SMB       0
        LD        L,#8H          ;
AGN     LDB       C,BSC2.@L      ;
        LDB       P2.3,C         ; P2.3 ← C
        INCS     L
        JR        AGN
        RET
  
```

### Program Counter (PC)

A 13-bit program counter (PC) stores addresses for instruction fetches during program execution. Whenever a reset operation or an interrupt occurs, bits PC12 through PC0 are set to the vector address.

Usually, the PC is incremented by the number of bytes of the instruction being fetched. One exception is the 1-byte REF instruction which is used to reference instructions stored in the ROM.

### Program Status Word (PSW)

The program status word (PSW) is an 8-bit word that defines system status and program execution status and which permits an interrupted process to resume operation after an interrupt request has been serviced. PSW values are mapped as follows:

FB0H	IS1	IS0	EMB	ERB
FB1H	C	SC2	SC1	SC0

The PSW can be manipulated by 1-bit or 4-bit read/write and by 8-bit read instructions, depending on the specific bit or bits being addressed. The PSW can be addressed during program execution regardless of the current value of the enable memory bank (EMB) flag.

Part or all of the PSW is saved to stack prior to execution of a subroutine call or hardware interrupt. After the interrupt has been processed, the PSW values are popped from the stack back to the PSW address.

When a RESETB is generated, the EMB and ERB values are set according to the RESETB vector address, and the carry flag is left undefined (or the current value is retained). PSW bits IS0, IS1, SC0, SC1, and SC2 are all cleared to logical zero.

**Table 2-5. Program Status Word Bit Descriptions**

PSW Bit Identifier	Description	Bit Addressing	Read/Write
IS1, IS0	Interrupt status flags	1, 4	R/W
EMB	Enable memory bank flag	1	R/W
ERB	Enable register bank flag	1	R/W
C	Carry flag	1	R/W
SC2, SC1, SC0	Program skip flags	8	R

### Interrupt Status Flags (IS0, IS1)

PSW bits IS0 and IS1 contain the current interrupt execution status values. You can manipulate IS0 and IS1 flags directly using 1-bit RAM control instructions

By manipulating interrupt status flags in conjunction with the interrupt priority register (IPR), you can process multiple interrupts by anticipating the next interrupt in an execution sequence. The interrupt priority control circuit determines the IS0 and IS1 settings in order to control multiple interrupt processing. When both interrupt status flags are set to "0", all interrupts are allowed. The priority with which interrupts are processed is then determined by the IPR.

When an interrupt occurs, IS0 and IS1 are pushed to the stack as part of the PSW and are automatically incremented to the next status. Then, when the interrupt service routine ends with an IRET instruction, IS0 and IS1 values are restored to the PSW. Table 2-6 shows the effects of IS0 and IS1 flag settings.

**Table 2-6. Interrupt Status Flag Bit Settings**

IS1 Value	IS0 Value	Status of Currently Executing Process	Effect of IS0 and IS1 Settings on Interrupt Request Control
0	0	0	All interrupt requests are serviced
0	1	1	Only high-priority interrupt as determined in the interrupt priority register (IPR) is serviced
1	0	2	No more interrupt requests are serviced
1	1	–	Not applicable; these bit settings are undefined

Since interrupt status flags can be addressed by write instructions, programs can exert direct control over interrupt processing status. Before interrupt status flags can be addressed, however, you must first execute a DI instruction to inhibit additional interrupt routines. When the bit manipulation has been completed, execute an EI instruction to re-enable interrupt processing.

#### PROGRAMMING TIP — Setting ISx Flags for Interrupt Processing

The following instruction sequence shows how to use the IS0 and IS1 flags to control interrupt processing:

```
INTB      DI           ; Disable interrupt
          BITR        IS1      ; IS1 ← 0
          BITS        IS0      ; Allow interrupts according to IPR priority level
          EI           ; Enable interrupt
```

## EMB Flag (EMB)

The EMB flag is used to enable whether the memory bank selected by SMB register is to be valid or not. In this way, it controls the addressing mode for data memory banks 0, 1, or 15.

When the EMB flag is "0", the data memory address space is restricted to bank 15 and addresses 000H–07FH of memory bank 0, regardless of the SMB register contents. When the EMB flag is set to "1", the general-purpose areas of bank 0, 1, and 15 can be accessed by using the appropriate SMB value.

### PROGRAMMING TIP — Using the EMB Flag to Select Memory Banks

EMB flag settings for memory bank selection:

1. When EMB = "0":

SMB	1	; Non-essential instruction since EMB = "0"
LD	A,#9H	
LD	90H,A	; (F90H) ← A, bank 15 is selected
LD	34H,A	; (034H) ← A, bank 0 is selected
SMB	0	; Non-essential instruction since EMB = "0"
LD	90H,A	; (F90H) ← A, bank 15 is selected
LD	34H,A	; (034H) ← A, bank 0 is selected
SMB	15	; Non-essential instruction, since EMB = "0"
LD	20H,A	; (020H) ← A, bank 0 is selected
LD	90H,A	; (F90H) ← A, bank 15 is selected

2. When EMB = "1":

SMB	1	; Select memory bank 1
LD	A,#9H	
LD	90H,A	; (190H) ← A, bank 1 is selected
LD	34H,A	; (134H) ← A, bank 1 is selected
SMB	0	; Select memory bank 0
LD	90H,A	; (090H) ← A, bank 0 is selected
LD	34H,A	; (034H) ← A, bank 0 is selected
SMB	15	; Select memory bank 15
LD	20H,A	; Program error, but assembler does not detect it
LD	90H,A	; (F90H) ← A, bank 15 is selected



**ERB Flag (ERB)**

The 1-bit register bank enable flag (ERB) determines the range of addressable working register area. When the ERB flag is "1", the working register area from register banks 0 to 3 is selected according to the register bank selection register (SRB). When the ERB flag is "0", register bank 0 is the selected working register area, regardless of the current value of the register bank selection register (SRB).

When an internal RESETB is generated, bit 6 of program memory address 0000H is written to the ERB flag. This automatically initializes the flag. When a vectored interrupt is generated, bit 6 of the respective address table in program memory is written to the ERB flag, setting the correct flag status before the interrupt service routine is executed.

During the interrupt routine, the ERB value is automatically pushed to the stack area along with the other PSW bits. Afterwards, it is popped back to the FB0H.0 bit location in the PSW. The initial ERB flag settings for each vectored interrupt are defined using VENTn instructions.

 **PROGRAMMING TIP — Using the ERB Flag to Select Register Banks**

ERB flag settings for register bank selection:

1. When ERB = "0":

SRB	1	; Register bank 0 is selected (since ERB = "0", the
		; SRB is configured to bank 0)
LD	EA,#34H	; Bank 0 EA ← #34H
LD	HL,EA	; Bank 0 HL ← EA
SRB	2	; Register bank 0 is selected
LD	YZ,EA	; Bank 0 YZ ← EA
SRB	3	; Register bank 0 is selected
LD	WX,EA	; Bank 0 WX ← EA

2. When ERB = "1":

SRB	1	; Register bank 1 is selected
LD	EA,#34H	; Bank 1 EA ← #34H
LD	HL,EA	; Bank 1 HL ← Bank 1 EA
SRB	2	; Register bank 2 is selected
LD	YZ,EA	; Bank 2 YZ ← BANK2 EA
SRB	3	; Register bank 3 is selected
LD	WX,EA	; Bank 3 WX ← Bank 3 EA

### Skip Condition Flags (SC2, SC1, SC0)

The skip condition flags SC2, SC1, and SC0 indicate the current program skip conditions and are set and reset automatically during program execution. Skip condition flags can only be addressed by 8-bit read instructions. Direct manipulation of the SC2, SC1, and SC0 bits is not allowed.

### Carry Flag (C)

The carry flag is used to save the result of an overflow or borrow when executing arithmetic instructions involving a carry (ADC, SBC). The carry flag can also be used as a 1-bit accumulator for performing Boolean operations involving bit-addressed data memory.

If an overflow or borrow condition occurs when executing arithmetic instructions with carry (ADC, SBC), the carry flag is set to "1". Otherwise, its value is "0". When a RESETB occurs, the current value of the carry flag is retained during power-down mode, but when normal operating mode resumes, its value is undefined.

The carry flag can be directly manipulated by predefined set of 1-bit read/write instructions, independent of other bits in the PSW. Only the ADC and SBC instructions, and the instructions listed in Table 2-7, affect the carry flag.

**Table 2-7. Valid Carry Flag Manipulation Instructions**

Operation Type	Instructions	Carry Flag Manipulation
Direct manipulation	SCF	Set carry flag to "1"
	RCF	Clear carry flag to "0" (reset carry flag)
	CCF	Invert carry flag value (complement carry flag)
	BTST C	Test carry and skip if C = "1"
Bit transfer	LDB C (operand) <sup>(note 1)</sup>	Load carry flag value to the specified bit
	LDB C, (operand) <sup>(note 1)</sup>	Load contents of the specified bit to carry flag
Data transfer	RRC A	Rotate right through carry flag
Boolean manipulation	BAND C, (operand) <sup>(note 1)</sup>	AND the specified bit with contents of carry flag and save the result to the carry flag
	BOR C, (operand) <sup>(note 1)</sup>	OR the specified bit with contents of carry flag and save the result to the carry flag
	BXOR C, (operand) <sup>(note 1)</sup>	XOR the specified bit with contents of carry flag and save the result to the carry flag
Interrupt routine	INT <sub>n</sub> <sup>(note 2)</sup>	Save carry flag to stack with other PSW bits
Return from interrupt	IRET	Restore carry flag from stack with other PSW bits

#### NOTES:

1. The operand has three bit addressing formats: mema.a, memb.@L, and @H + DA.b.
2. 'INT<sub>n</sub>' refers to the specific interrupt being executed and is not an instruction.

 **PROGRAMMING TIP — Using the Carry Flag as a 1-Bit Accumulator**

1. Set the carry flag to logic one:

```
SCF                ; C ← 1
LD      EA,#0C3H   ; EA ← #0C3H
LD      HL,#0AAH   ; HL ← #0AAH
ADC     EA,HL      ; EA ← #0C3H + #0AAH + #1H, C ← 1
```

2. Logical-AND bit 3 of address 3FH with P3.3 and output the result to P5.0:

```
LD      H,#3H      ; Set the upper four bits of the address to the H register
                        value
LDB     C,@H+0FH.3 ; C ← bit 3 of 3FH
BAND   C,P3.3      ; C ← C AND P3.3
LDB     P5.0,C     ; Output result from carry flag to P5.0
```

# 3

## ADDRESSING MODES

### OVERVIEW

The enable memory bank flag, EMB, controls the two addressing modes for data memory. When the EMB flag is set to logic one, you can address the entire RAM area; when the EMB flag is cleared to logic zero, the addressable area in the RAM is restricted to specific locations.

The EMB flag works in connection with the select memory bank instruction, SMBn. You will recall that the SMBn instruction is used to select RAM bank 0, 1, 2 or 15. The SMB setting is always contained in the upper four bits of a 12-bit RAM address. For this reason, both addressing modes (EMB = "0" and EMB = "1") apply specifically to the memory bank indicated by the SMB instruction, and any restrictions to the addressable area within banks 0, 1, 2 or 15. Direct and indirect 1-bit, 4-bit, and 8-bit addressing methods can be used. Several RAM locations are addressable at all times, regardless of the current EMB flag setting.

Here are a few guidelines to keep in mind regarding data memory addressing:

- When you address peripheral hardware locations in bank 15, the mnemonic for the memory-mapped hardware component can be used as the operand in place of the actual address location.
- Always use an even-numbered RAM address as the operand in 8-bit direct and indirect addressing.
- With direct addressing, use the RAM address as the instruction operand; with indirect addressing, the instruction specifies a register which contains the operand's address.

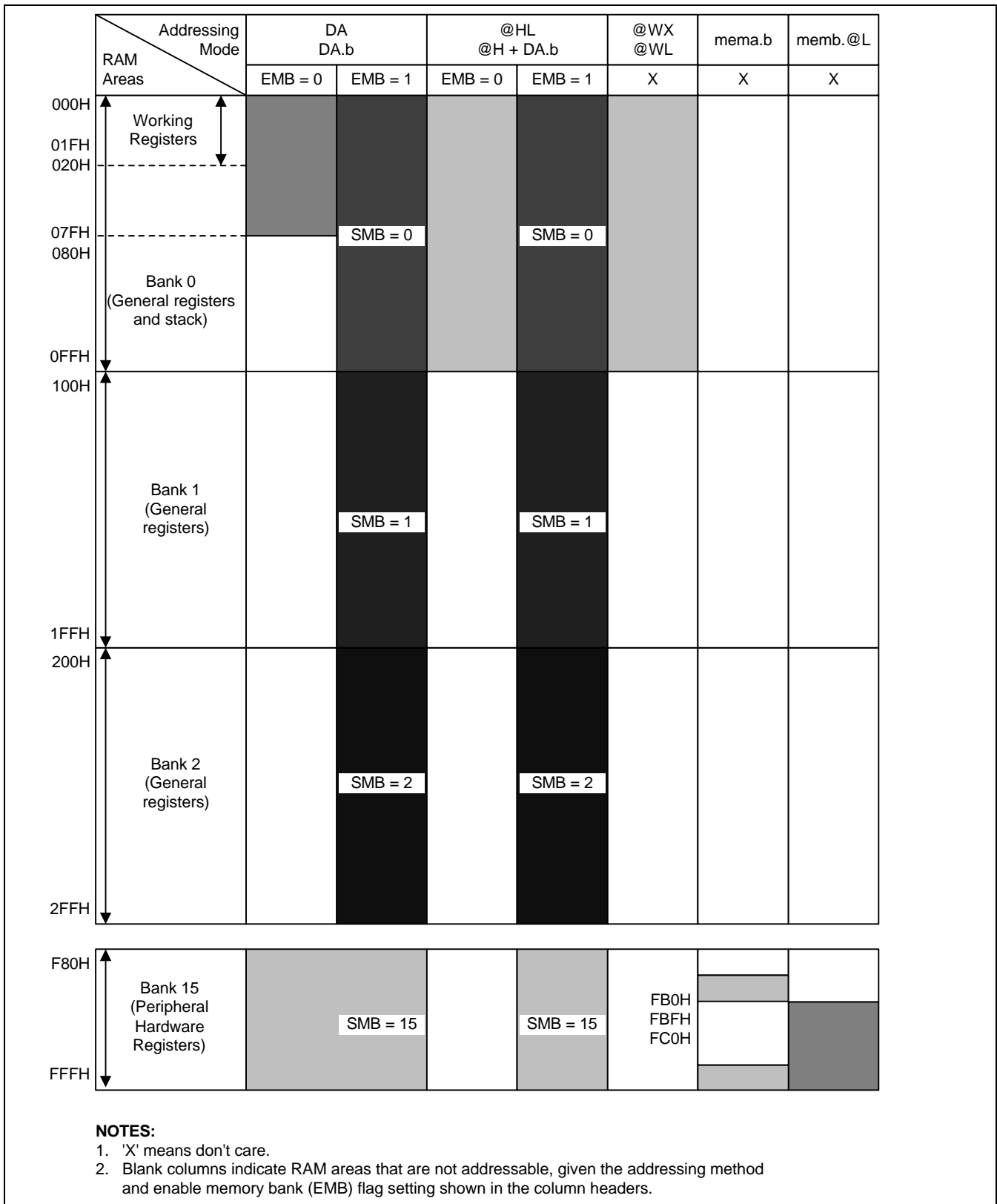


Figure 3-1. RAM Address Structure

### EMB and ERB Initialization Values

The EMB and ERB flag bits are set automatically by the values of the RESET vector address and the interrupt vector address. When a RESET is generated internally, bit 7 of program memory address 0000H is written to the EMB flag, initializing it automatically. When a vectored interrupt is generated, bit 7 of the respective vector address table is written to the EMB. This automatically sets the EMB flag status for the interrupt service routine. When the interrupt is serviced, the EMB value is automatically saved to stack and then restored when the interrupt routine has completed.

At the beginning of a program, the initial EMB and ERB flag values for each vectored interrupt must be set by using VENT instruction. The EMB and ERB can be set or reset by bit manipulation instructions (BITS, BITR) despite the current SMB setting.

#### PROGRAMMING TIP — Initializing the EMB and ERB Flags

The following assembly instructions show how to initialize the EMB and ERB flag settings:

```

ORG      0000H          ; ROM address assignment
VENT0    1,0,RESET     ; EMB ← 1, ERB ← 0, branch RESET
VENT1    0,1,INTB      ; EMB ← 0, ERB ← 1, branch INTB
VENT2    0,1,INT0      ; EMB ← 0, ERB ← 1, branch INT0
VENT3    0,1,INT1      ; EMB ← 0, ERB ← 1, branch INT1
NOP
NOP
VENT5    0,1,INTT0     ; EMB ← 0, ERB ← 1, branch INTT0
VENT6    0,1,INTT1     ; EMB ← 0, ERB ← 1, branch INTT1
•
•
•
RESET   BITR          EMB

```

**ENABLE MEMORY BANK SETTINGS****EMB = "1"**

When the enable memory bank flag EMB is set to logic one, you can address the data memory bank specified by the select memory bank (SMB) value (0, 1, 2 or 15) using 1-, 4-, or 8-bit instructions. You can use both direct and indirect addressing modes. The addressable RAM areas when EMB = "1" are as follows:

- If SMB = 0,           000H–0FFH
- If SMB = 1,           100H–1FFH
- If SMB = 2,           200H–2FFH
- If SMB = 15,         F80H–FFFH

**EMB = "0"**

When the enable memory bank flag EMB is set to logic zero, the addressable area is defined independently of the SMB value, and is restricted to specific locations depending on whether a direct or indirect address mode is used.

If EMB = "0", the addressable area is restricted to locations 000H–07FH in bank 0 and to locations F80H–FFFH in bank 15 for direct addressing. For indirect addressing, only locations 000H–0FFH in bank 0 are addressable, regardless of SMB value.

To address the peripheral hardware register (bank 15) using indirect addressing, the EMB flag must first be set to "1" and the SMB value to "15". When a RESET occurs, the EMB flag is set to the value contained in bit 7 of ROM address 0000H.

**EMB-Independent Addressing**

At any time, several areas of the data memory can be addressed independently of the current status of the EMB flag. These exceptions are described in Table 3-1.

**Table 3-1. RAM Addressing Not Affected by the EMB Value**

Address	Addressing Method	Affected Hardware	Program Examples
000H–0FFH	4-bit indirect addressing using WX and WL register pairs; 8-bit indirect addressing using SP	Not applicable	LD A,@WX  PUSH POP
FB0H–FBFH FF0H–FFFH	1-bit direct addressing	PSW, IEx, IRQx, I/O	BITS EMB BITR IE4
FC0H–FFFH	1-bit indirect addressing using the L register	I/O	BAND C,P3.@L

## SELECT BANK REGISTER (SB)

The select bank register (SB) is used to assign the memory bank and register bank. The 8-bit SB register consists of the 4-bit select register bank register (SRB) and the 4-bit select memory bank register (SMB), as shown in Figure 3-2.

During interrupts and subroutine calls, SB register contents can be saved to stack in 8-bit units by the PUSH SB instruction. You later restore the value to the SB using the POP SB instruction.

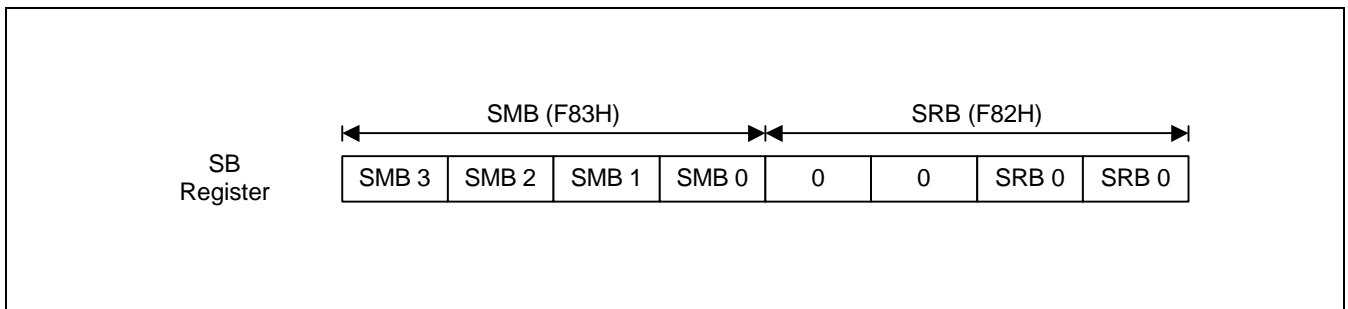


Figure 3-2. SMB and SRB Values in the SB Register

### Select Register Bank (SRB) Instruction

The select register bank (SRB) value specifies which register bank is to be used as a working register bank. The SRB value is set by the 'SRB n' instruction, where n = 0, 1, 2, 3.

One of the four register banks is selected by the combination of ERB flag status and the SRB value that is set using the 'SRB n' instruction. The current SRB value is retained until another register is requested by program software. PUSH SB and POP SB instructions are used to save and restore the contents of SRB during interrupts and subroutine calls. RESET clears the 4-bit SRB value to logic zero.

### Select Memory Bank (SMB) Instruction

To select one of the four available data memory banks, you must execute an SMB n instruction specifying the number of the memory bank you want (0, 1, 2 or 15). For example, the instruction 'SMB 1' selects bank 1 and 'SMB 15' selects bank 15. (And remember to enable the selected memory bank by making the appropriate EMB flag setting.)

The upper four bits of the 12-bit data memory address are stored in the SMB register. If the SMB value is not specified by software (or if a RESET does not occur) the current value is retained. RESET clears the 4-bit SMB value to logic zero.

The PUSH SB and POP SB instructions save and restore the contents of the SMB register to and from the stack area during interrupts and subroutine calls.



## DIRECT AND INDIRECT ADDRESSING

1-bit, 4-bit, and 8-bit data stored in data memory locations can be addressed directly using a specific register or bit address as the instruction operand.

Indirect addressing specifies a memory location that contains the required direct address. The KS57 instruction set supports 1-bit, 4-bit, and 8-bit indirect addressing. For 8-bit indirect addressing, an even-numbered RAM address must always be used as the instruction operand.

### 1-Bit Addressing

**Table 3-2. 1-Bit Direct and Indirect RAM Addressing**

Operand Notation	Addressing Mode Description	EMB Flag Setting	Addressable Area	Memory Bank	Hardware I/O Mapping
DA.b	Direct: bit is indicated by the RAM address (DA), memory bank selection, and specified bit number (b).	0	000H–07FH	Bank 0	–
			F80H–FFFH	Bank 15	All 1-bit addressable peripherals (SMB = 15)
		1	000H–FFFH	SMB = 0, 1, 2, 15	
mema.b	Direct: bit is indicated by addressable area (mema) and bit number (b).	x	FB0H–FBFH FF0H–FFFH	Bank 15	IS0, IS1, EMB, ERB, IEx, IRQx, Pn.n
memb.@L	Indirect: lower two bits of register L as indicated by the upper 10 bits of RAM area (memb) and the upper two bits of register L.	x	FC0H–FFFH	Bank 15	Pn.n
@H + DA.b	Indirect: bit indicated by the lower four bits of the address (DA), memory bank selection, and the H register identifier.	0	000H–0FFH	Bank 0	All 1-bit addressable peripherals (SMB = 15)
		1	000H–FFFH	SMB = 0, 1, 2, 15	

**NOTE:** x = not applicable.

## PROGRAMMING TIP — 1-Bit Addressing Modes

### 1-Bit Direct Addressing

1. If EMB = "0":

AFLAG	EQU	34H.3	
BFLAG	EQU	85H.3	
CFLAG	EQU	0BAH.0	
	SMB	0	
	BITS	AFLAG	; 34H.3 ← 1
	BITS	BFLAG	; F85H.3 (BMOD.3) ← 1
	BTST	CFLAG	; If FBAH.0 (IRQW) = 1, skip
	BITS	BFLAG	; Else if, FBAH.0 (IRQW) = 0, F85H.3 (BMOD.3) ← 1
	BITS	P3.0	; FF3H.0 (P3.0) ← 1

2. If EMB = "1":

AFLAG	EQU	34H.3	
BFLAG	EQU	85H.3	
CFLAG	EQU	0BAH.0	
	SMB	0	
	BITS	AFLAG	; 34H.3 ← 1
	BITS	BFLAG	; 85H.3 ← 1
	BTST	CFLAG	; If 0BAH.0 = 1, skip
	BITS	BFLAG	; Else if 0BAH.0 = 0, 085H.3 ← 1
	BITS	P3.0	; FF3H.0 (P3.0) ← 1

### 1-Bit Indirect Addressing

1. If EMB = "0":

AFLAG	EQU	34H.3	
BFLAG	EQU	85H.3	
CFLAG	EQU	0BAH.0	
	SMB	0	
	LD	H,#0BH	; H ← #0BH
	BTSTZ	@H+CFLAG	; If 0BAH.0 = 1, 0BAH.0 ← 0 and skip
	BITS	CFLAG	; Else if 0BAH.0 = 0, FBAH.0 (IRQW) ← 1

2. If EMB = "1":

AFLAG	EQU	34H.3	
BFLAG	EQU	85H.3	
CFLAG	EQU	0BAH.0	
	SMB	0	
	LD	H,#0BH	; H ← #0BH
	BTSTZ	@H+CFLAG	; If 0BAH.0 = 1, 0BAH.0 ← 0 and skip
	BITS	CFLAG	; Else if 0BAH.0 = 0, 0BAH.0 ← 1

## 4-Bit Addressing

Table 3-3. 4-Bit Direct and Indirect RAM Addressing

Operand Notation	Addressing Mode Description	EMB Flag Setting	Addressable Area	Memory Bank	Hardware I/O Mapping
DA	Direct: 4-bit address indicated by the RAM address (DA) and the memory bank selection	0	000H–07FH	Bank 0	–
			F80H–FFFH	Bank 15	All 4-bit addressable peripherals
		1	000H–FFFH	SMB = 0, 1, 2 15	(SMB = 15)
@HL	Indirect: 4-bit address indicated by the memory bank selection and register HL	0	000H–0FFH	Bank 0	–
		1	000H–FFFH	SMB = 0, 1, 2 15	All 4-bit addressable peripherals (SMB = 15)
@WX	Indirect: 4-bit address indicated by register WX	x	000H–0FFH	Bank 0	–
@WL	Indirect: 4-bit address indicated by register WL	x	000H–0FFH	Bank 0	

**NOTE:** x = not applicable.

 **PROGRAMMING TIP — 4-Bit Addressing Modes**
**4-Bit Direct Addressing**

1. If EMB = "0":

```

ADATA    EQU    46H
BDATA    EQU    8EH
SMB      15          ; Non-essential instruction, since EMB = "0"
LD       A,P3      ; A ← (P3)
SMB      0          ; Non-essential instruction, since EMB = "0"
LD       ADATA,A   ; (046H) ← A
LD       BDATA,A   ; (F8EH) ← A

```

2. If EMB = "1":

```

ADATA    EQU    46H
BDATA    EQU    8EH
SMB      15
LD       A,P3      ; A ← (P3)
SMB      0
LD       ADATA,A   ; (046H) ← A
LD       BDATA,A   ; (08EH) ← A

```

 PROGRAMMING TIP — 4-Bit Addressing Modes (Continued)

**4-Bit Indirect Addressing (Example 1)**

1. If EMB = "0", compare bank 0 locations 040H–046H with bank 0 locations 060H–066H:

ADATA	EQU	46H	
BDATA	EQU	66H	
	SMB	1	; Non-essential instruction, since EMB = "0"
	LD	HL,#BDATA	
	LD	WX,#ADATA	
COMP	LD	A,@WL	; A ← bank 0 (040H–046H)
	CPSE	A,@HL	; If bank 0 (060H–066H) = A, skip
	SRET		
	DECS	L	
	JR	COMP	
	RET		

2. If EMB = "1", compare bank 0 locations 040H–046H to bank 1 locations 160H–166H:

ADATA	EQU	46H	
BDATA	EQU	66H	
	SMB	1	
	LD	HL,#BDATA	
	LD	WX,#ADATA	
COMP	LD	A,@WL	; A ← bank 0 (040H–046H)
	CPSE	A,@HL	; If bank 1 (160H–166H) = A, skip
	SRET		
	DECS	L	
	JR	COMP	
	RET		

 PROGRAMMING TIP — 4-Bit Addressing Modes (Continued)

#### 4-Bit Indirect Addressing (Example 2)

1. If EMB = "0", exchange bank 0 locations 040H–046H with bank 0 locations 060H–066H:

```

ADATA    EQU    46H
BDATA    EQU    66H
          SMB    1                ; Non-essential instruction, since EMB = "0"
          LD     HL,#BDATA
          LD     WX,#ADATA
TRANS    LD     A,@WL            ; A ← bank 0 (040H–046MH)
          XCHD  A,@HL            ; Bank 0 (060H–066H) ← A
          JR     TRANS

```

2. If EMB = "1", exchange bank 0 locations 040H–046H to bank 1 locations 160H–166H:

```


ADATA    EQU    46H
BDATA    EQU    66H
          SMB    1
          LD     HL,#BDATA
          LD     WX,#ADATA
TRANS    LD     A,@WL            ; A ← bank 0 (040H–046H)
          XCHD  A,@HL            ; Bank 1 (160H–166H) ← A
          JR     TRANS

```

## 8-Bit Addressing

Table 3-4. 8-Bit Direct and Indirect RAM Addressing

Instruction Notation	Addressing Mode Description	EMB Flag Setting	Addressable Area	Memory Bank	Hardware I/O Mapping
DA	Direct: 8-bit address indicated by the RAM address ( <i>DA = even number</i> ) and memory bank selection	0	000H–07FH	Bank 0	–
			F80H–FFFH	Bank 15	All 8-bit addressable peripherals
		1	000H–FFFH	SMB = 0, 1, 2, 15	(SMB = 15)
@HL	Indirect: the 8-bit address indicated by the memory bank selection and register HL; (the 4-bit L register value must be an even number)	0	000H–0FFH	Bank 0	–
		1	000H–FFFH	SMB = 0, 1, 2, 15	All 8-bit addressable peripherals (SMB = 15)

 PROGRAMMING TIP — 8-Bit Addressing Modes

**8-Bit Direct Addressing**

1. If EMB = "0":

ADATA	EQU	46H	
BDATA	EQU	8EH	
	SMB	15	; Non-essential instruction, since EMB = "0"
	LD	EA,P4	; E ← (P5), A ← (P4)
	SMB	0	
	LD	ADATA,EA	; (046H) ← A, (047H) ← E
	LD	BDATA,EA	; (F8EH) ← A, (F8FH) ← E

2. If EMB = "1":

ADATA	EQU	46H	
BDATA	EQU	8EH	
	SMB	15	
	LD	EA,P4	; E ← (P5), A ← (P4)
	SMB	0	
	LD	ADATA,EA	; (046H) ← A, (047H) ← E
	LD	BDATA,EA	; (08EH) ← A, (08FH) ← E

**8-Bit Indirect Addressing**

1. If EMB = "0":

ADATA	EQU	146H	
	SMB	1	; Non-essential instruction, since EMB = "0"
	LD	HL,#ADATA	
	LD	EA,@HL	; A ← (046H), E ← (047H)

2. If EMB = "1":

ADATA	EQU	146H	
	SMB	1	
	LD	HL,#ADATA	
	LD	EA,@HL	; A ← (146H), E ← (147H)

# 4 MEMORY MAP

## OVERVIEW

To support program control of peripheral hardware, I/O addresses for peripherals are memory-mapped to bank 15 of the RAM. Memory mapping lets you use a mnemonic as the operand of an instruction in place of the specific memory location.

Access to bank 15 is controlled by the select memory bank (SMB) instruction and by the enable memory bank flag (EMB) setting. If the EMB flag is "0", bank 15 can be addressed using direct addressing, regardless of the current SMB value. 1-bit direct and indirect addressing can be used for specific locations in bank 15, regardless of the current EMB value.

### I/O Map for Hardware Registers

Table 4-1 contains detailed information about I/O mapping for peripheral hardware in bank 15 (register locations F80H–FFFH). Use the I/O map as a quick-reference source when writing application programs. The I/O map gives you the following information:

- Register address
- Register name (mnemonic for program addressing)
- Bit values (both addressable and non-manipulable)
- Read-only, write-only, or read and write addressability
- 1-bit, 4-bit, or 8-bit data manipulation characteristics



Table 4-1. I/O Map for Memory Bank 15

Memory Bank 15							Addressing Mode		
Address	Register	Bit 3	Bit 2	Bit 1	Bit 0	R/W	1-Bit	4-Bit	8-Bit
F80H	SP	.3	.2	.1	"0"	R/W	No	No	Yes
F81H		.7	.6	.5	.4				
Locations F82H–F84H are not mapped.									
F85H	BMOD	.3	.2	.1	.0	W	.3	Yes	No
F86H	BCNT					R	No	No	Yes
F87H									
F88H	WMOD	"0"	.2	.1	"0" (1)	W	No	No	Yes
F89H		.7	"0"	.5	.4				
Locations F8AH–F8FH are not mapped.									
F90H	TMOD0	.3	.2	"0"	"0"	W	.3	No	Yes
F91H		"0"	.6	.5	.4				
F92H		TOE1	TOE0	BOE	"0"	R/W	Yes	Yes	No
F93H		"0"	TOL1	TOL0	"0"				
F94H	TCNT0					R	No	No	Yes
F95H									
F96H	TREF0					W	No	No	Yes
F97H									
F98H	WDMOD	.3	.2	.1	.0	W	No	No	Yes
F99H		.7	.6	.5	.4				
F9AH	WDFLAG	WDTCF	"0"	"0"	"0"	W	Yes	Yes	No
Locations F9BH–F9FH are not mapped.									
FA0H	TMOD1	.3	.2	"0"	"0"	W	.3	No	Yes
FA1H		"0"	.6	.5	.4				
Locations FA2H–FA3H are not mapped.									
FA4H	TCNT1					R	No	No	Yes
FA5H									
Locations FA6H–FA7H are not mapped.									
FA8H	TREF1					W	No	No	Yes
FA9H									
Locations FAAH–FAFH are not mapped.									
FB0H	PSW	IS1	IS0	EMB	ERB	R/W	Yes	Yes	Yes
FB1H		C (2)	SC2	SC1	SC0	R	No	No	
FB2H	IPR	IME	.2	.1	.0	W	IME	Yes	No

Table 4-1. I/O Map for Memory Bank 15 (Continued)

Memory Bank 15							Addressing Mode		
Address	Register	Bit 3	Bit 2	Bit 1	Bit 0	R/W	1-Bit	4-Bit	8-Bit
FB3H	PCON	.3	.2	.1	.0	W	.3, .2	Yes	No
FB4H	IMOD0	"0"	"0"	.1	.0	W	No	Yes	No
FB5H	IMOD1	"0"	"0"	"0"	.0				
FB6H	IMOD2	"0"	"0"	.1	.0				
Locations FB7H is not mapped.									
FB8H		IE4	IRQ4	IEB	IRQB	R/W	Yes	Yes	No
Locations FB9H is not mapped.									
FBAH		"0"	"0"	IEW	IRQW	R/W	Yes	Yes	No
FBBH		"0"	"0"	IET1	IRQT1				
FBCH		"0"	"0"	IET0	IRQT0				
Locations FBDH is not mapped.									
FBEH		IE1	IRQ1	IE0	IRQ0	R/W	Yes	No	No
FBFH		"0"	"0"	IE2	IRQ2				
FC0H	BSC0					R/W	Yes	No	Yes
FC1H	BSC1								
FC2H	BSC2								
FC3H	BSC3								
Locations FC4H–FCFH are not mapped.									
FD0H	CLMOD	.3	"0"	.1	.0	W	No	Yes	No
Locations FD1H is not mapped.									
FD2H	DTMR	"0"	.2	.1	.0	W	No	No	Yes
FD3H		.7	.6	.5	.4				
FD4H	DTGR	.3	.2	.1	.0	W	No	No	Yes
FD5H		"0"	"0"	"0"	.4				
Locations FD6H–FD9H are not mapped.									
FDAH	PNE1	PNE4.3	PNE4.2	PNE4.1	PNE4.0	W	No	No	Yes
FDBH		PNE5.3	PNE5.2	PNE5.1	PNE5.0				
FDCH	PUMOD1	PUR1.3	PUR1.2	PUR1.1	PUR1.0	W	No	No	Yes
FDDH		PUR5	PUR4	PUR3	PUR2				
FDEH	PUMOD2	PUR9	PUR8	PUR7	PUR6	W	No	Yes	No
Locations FDFH–FE7H are not mapped.									
FE8H	PMG1	PM2.3	PM2.2	PM2.1	PM2.0	W	No	No	Yes
FE9H		PM3.3	PM3.2	PM3.1	PM3.0				

Table 4-1. I/O Map for Memory Bank 15 (Continued)

Memory Bank 15							Addressing Mode		
Address	Register	Bit 3	Bit 2	Bit 1	Bit 0	R/W	1-Bit	4-Bit	8-Bit
FEAH	PMG2	PM4.3	PM4.2	PM4.1	PM4.0	W	No	No	Yes
FEBH		PM5.3	PM5.2	PM5.1	PM5.0				
FECH	PMG3	PM6.3	PM6.2	PM6.1	PM6.0				
FEDH		PM7.3	PM7.2	PM7.1	PM7.0				
FEEH	PMG4	PM8.3	PM8.2	PM8.1	PM8.0	W	No	No	Yes
FEFH		"0"	PM9.2	PM9.1	PM9.0				
Locations FF0H is not mapped.									
FF1H	Port 1	.3	.2	.1	.0	R	Yes	Yes	No
FF2H	Port 2	.3	.2	.1	.0	R/W	Yes	Yes	Yes
FF3H	Port 3	.3 / .7	.2 / .6	.1 / .5	.0 / .4				
FF4H	Port 4	.3	.2	.1	.0	R/W	Yes	Yes	Yes
FF5H	Port 5	.3 / .7	.2 / .6	.1 / .5	.0 / .4				
FF6H	Port 6	.3	.2	.1	.0	R/W	Yes	Yes	Yes
FF7H	Port 7	.3 / .7	.2 / .6	.1 / .5	.0 / .4				
FF8H	Port 8	.3	.2	.1	.0	R/W	Yes	Yes	Yes
FF9H	Port 9	"0"	.2 / .6	.1 / .5	.0 / .4				

**NOTES:**

1. Bit 0 in the WMOD register must be set to logic "0"
2. The carry flag can be read or written by specific bit manipulation instructions only.

**REGISTER DESCRIPTIONS**

In this section, register descriptions are presented in a consistent format to familiarize you with the memory-mapped I/O locations in bank 15 of the RAM. Figure 4-1 describes features of the register description format. Register descriptions are arranged in alphabetical order. Programmers can use this section as a quick-reference source when writing application programs.

Counter registers, buffer registers, and reference registers, as well as the stack pointer and port I/O latches, are not included in these descriptions. More detailed information about how these registers are used is included in Part II of this manual, "Hardware Descriptions," in the context of the corresponding peripheral hardware module descriptions.

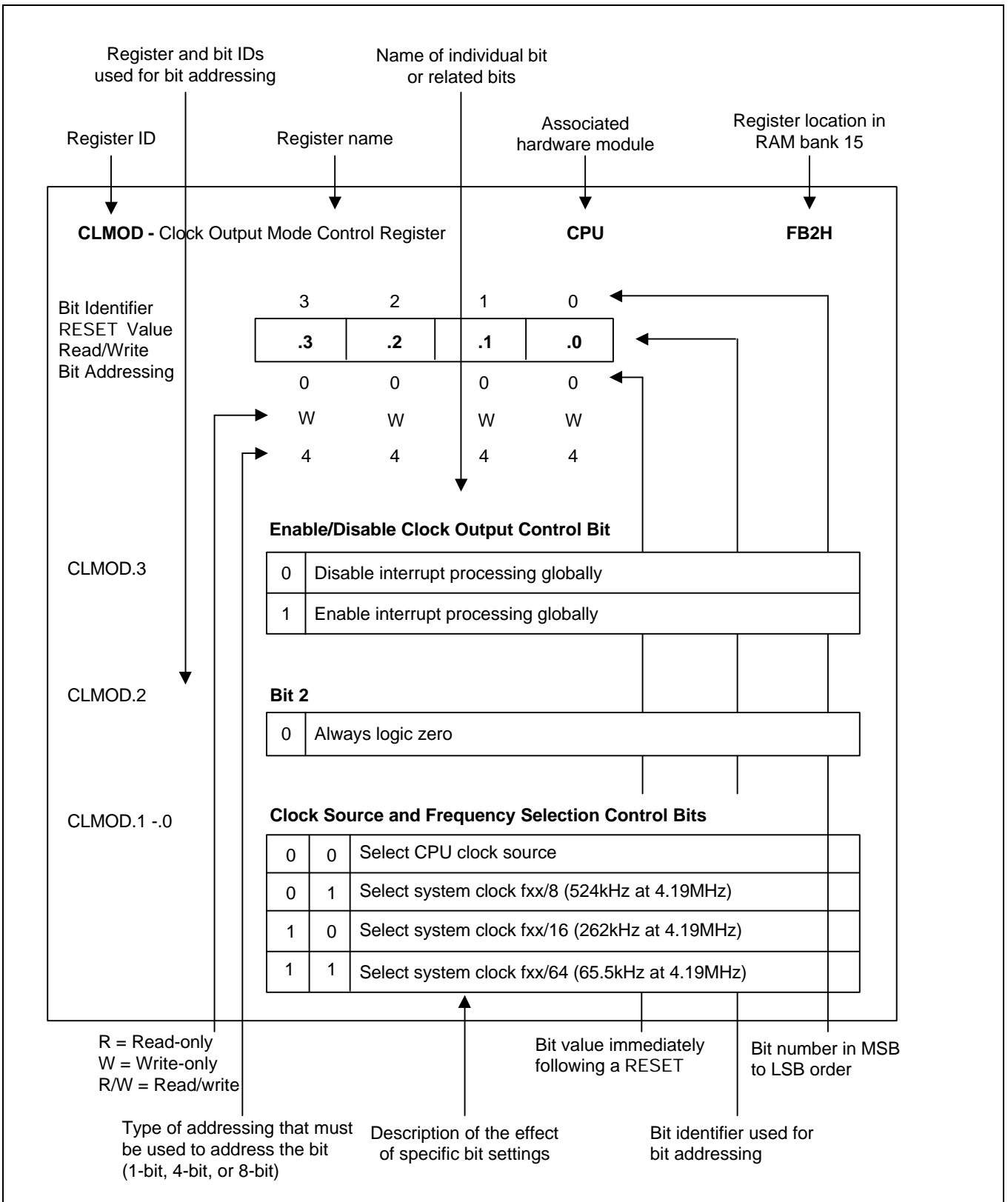


Figure 4-1. Register Description Format

**BMOD — Basic Timer Mode Register**

BT

F85H

Bit	3	2	1	0
Identifier	.3	.2	.1	.0
RESET Value	0	0	0	0
Read/Write	W	W	W	W
Bit Addressing	1/4	4	4	4

**BMOD.3****Basic Timer Restart Bit**

1	Restart basic timer, then clear IRQB flag, BCNT and BMOD.3 to logic zero
---	--

**BMOD.2 – .0****Input Clock Frequency and Signal Stabilization Interval Control Bits**

0	0	0	Input clock frequency: Signal stabilization interval:	$f_x / 2^{12}$ (1.02kHz) $2^{20} / f_x$ (250ms)
0	1	1	Input clock frequency: Signal stabilization interval:	$f_x / 2^9$ (8.18kHz) $2^{17} / f_x$ (31.3ms)
1	0	1	Input clock frequency: Signal stabilization interval:	$f_x / 2^7$ (32.7kHz) $2^{15} / f_x$ (7.82ms)
1	1	1	Input clock frequency: Signal stabilization interval:	$f_x / 2^5$ (131kHz) $2^{13} / f_x$ (1.95ms)

**NOTES:**

- Signal stabilization interval is the time required to stabilize clock signal oscillation after stop mode is terminated by an interrupt. The stabilization interval can also be interpreted as "Interrupt Interval Time".
- When a RESET occurs, the oscillation stabilization time is 31.3 ms ( $2^{17}/f_x$ ) at 4.19MHz.
- 'fx' is the system clock rate, given a clock frequency of 4.19MHz.

**CLMOD** — Clock Output Mode Register

CPU

FD0H

Bit	3	2	1	0
Identifier	.3	"0"	.1	.0
RESET Value	0	0	0	0
Read/Write	W	W	W	W
Bit Addressing	4	4	4	4

**CLMOD.3****Enable/Disable Clock Output Control Bit**

0	Disable clock output
1	Enable clock output

**CLMOD.2****Bit 2**

0	Always logic zero
---	-------------------

**CLMOD.1 – .0****Clock Source and Frequency Selection Control Bits**

0	0	Select CPU clock source $fx/4$ , $fx/8$ or $fx/64$ (1.05MHz, 524kHz or 65.5kHz)
0	1	Select system clock $fx/8$ (524kHz)
1	0	Select system clock $fx/16$ (262kHz)
1	1	Select system clock $fx/64$ (65.5kHz)

**NOTE:** 'fx' is the system clock, given a clock frequency of 4.19MHz.

**DTMR — DTMF Mode Register**

**DTMF**

**FD3H, FD2H**

<b>Bit</b>	3	2	1	0	3	2	1	0
<b>Identifier</b>	.7	.6	.5	.4	"0"	.2	.1	.0
<b>RESET Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	W	W	W	W	W	W	W	W
<b>Bit Addressing</b>	8	8	8	8	8	8	8	8

**DTMR.7 – .4**

**DTMR Bit Values For Keyboard Inputs**

0	0	0	0	Function key D
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	0
1	0	1	1	*
1	1	0	0	#
1	1	0	1	Function key A
1	1	1	0	Function key B
1	1	1	1	Function key C

**DTMR.3**

**Bit 3**

0	Always logic zero
---	-------------------

**DTMR.2 – .1**

**Tone Selection Bits**

0	0	Dual-tone enable
1	0	Dual-tone enable (alternate setting)
0	1	Single-column tone enable
1	1	Single-low tone enable

**DTMR.0**

**DTMF Operation Enable/Disable Bit**

0	Disable DTMF operation
1	Enable DTMF operation

**DTGR — DTMF Gain Register**

FD5H, FD4H

<b>Bit</b>	3	2	1	0	3	2	1	0
<b>Identifier</b>	"0"	"0"	"0"	.4	.3	.2	.1	.0
<b>RESET Value</b>	0	0	0	1	0	0	0	0
<b>Read/Write</b>	W	W	W	W	W	W	W	W
<b>Bit Addressing</b>	8	8	8	8	8	8	8	8

**DTGR.4 – .0****DTMF Signal Gain Inputs**

1	0	0	0	0	DTMF signal is amplified by 1 (Default)
0	?	?	?	?	Gain = $.3 * 0.5 + .2 * 0.25 + .1 * 0.125 + .0 * 0.0625$
1	?	?	?	1	Gain = $1.0625 + .3 * 0.5 + .2 * 0.25 + .1 * 0.125$



**IMOD0** — External Interrupt 0 (INT0) Mode Register <sup>(NOTE)</sup>

CPU

B4H

Bit	3	2	1	0
Identifier	“0”	“0”	.1	.0
RESET Value	0	0	0	0
Read/Write	W	W	W	W
Bit Addressing	4	4	4	4

**IMOD0.3 – .2****Bits 3–2**

0	Always logic zero
---	-------------------

**IMOD0.1 – .0****External Interrupt Mode Control Bits**

0	0	Interrupt requests are triggered by a rising signal edge
0	1	Interrupt requests are triggered by a falling signal edge
1	0	Interrupt requests are triggered by both rising and falling signal edges
1	1	Interrupt request flag (IRQx) cannot be set to logic one

**NOTE:** Interrupt0 is dedicated for caller id interrupt.

**IMOD1** — External Interrupt 1 (INT1) Mode Register

CPU

FB5H

Bit	3	2	1	0
Identifier	“0”	“0”	“0”	.0
RESET Value	0	0	0	0
Read/Write	W	W	W	W
Bit Addressing	4	4	4	4

IMOD1.3 – .1

**Bits 3–1**

0	Always logic zero
---	-------------------

IMOD1.0

**External Interrupt 1 Edge Detection Control Bit**

0	Rising edge detection
1	Falling edge detection

**IMOD2 — External Interrupt 2 (INT2) Mode Register**

CPU

FB6H

<b>Bit</b>	3	2	1	0
<b>Identifier</b>	“0”	“0”	.1	.0
<b>RESET Value</b>	0	0	0	0
<b>Read/Write</b>	W	W	W	W
<b>Bit Addressing</b>	4	4	4	4

**IMOD2.3 – .2****Bits 3–2**

0	Always logic zero
---	-------------------

**IMOD2.1 – .0****External Interrupt 2 Edge Detection Selection Bit**

0	0	Interrupt request at INT2 pin triggered by rising edge
0	1	Interrupt request at KS4–KS7 triggered by falling edge
1	0	Interrupt request at KS2–KS7 triggered by falling edge
1	1	Interrupt request at KS0–KS7 triggered by falling edge

**IE0, 1, IRQ0, 1 — INT0, 1 Interrupt Enable/Request Flags**      CPU      FBEH

Bit	3	2	1	0
Identifier	IE1	IRQ1	IE0	IRQ0
RESET Value	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W
Bit Addressing	1/4	1/4	1/4	1/4

**IE1****INT1 Interrupt Enable Flag**

0	Disable interrupt requests at the INT1 pin
1	Enable interrupt requests at the INT1 pin

**IRQ1****INT1 Interrupt Request Flag**

–	Generate INT1 interrupt (This bit is set and cleared by hardware when rising or falling edge detected at INT1 pin.)
---	---

**IE0****INT0 Interrupt Enable Flag**

0	Disable interrupt requests at the INT0 pin
1	Enable interrupt requests at the INT0 pin

**IRQ0****INT0 Interrupt Request Flag**

–	Generate INT0 interrupt (This bit is set and cleared automatically by hardware when rising or falling edge detected at INT0 pin.)
---	---

**NOTE:** Interrupt0 is dedicated for caller id interrupt.

**IE2, IRQ2 — INT2 Interrupt Enable/Request Flags**

CPU

FBFH

Bit	3	2	1	0
Identifier	"0"	"0"	IE2	IRQ2
RESET Value	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W
Bit Addressing	1/4	1/4	1/4	1/4

.3 – .2

**Bits 3–2**

0	Always logic zero
---	-------------------

**IE2****INT2 Interrupt Enable Flag**

0	Disable INT2 interrupt requests at the INT2 pin (note) or KS0–KS7 pins
1	Enable INT2 interrupt requests at the INT2 pin (note) or KS0–KS7 pins

**IRQ2****INT2 Interrupt Request Flag**

–	Generate INT2 quasi-interrupt (This bit is set and is not cleared automatically by hardware when a rising edge is detected at INT2 pin (note) or when a falling edge is detected at one of the KS0–KS7 pins. Since INT2 is a quasi-interrupt, IRQ2 flag must be cleared by software.)
---	---

**IE4, IRQ4 — INT4 Interrupt Enable/Request Flags** CPU FB8H

**IEB, IRQB — INTB Interrupt Enable/Request Flags** CPU FB8H

Bit	3	2	1	0
Identifier	IE4	IRQ4	IEB	IRQB
RESET Value	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W
Bit Addressing	1/4	1/4	1/4	1/4

**IE4** **INT4 Interrupt Enable Flag**

0	Disable interrupt requests at the INT4 pin
1	Enable interrupt requests at the INT4 pin

**IRQ4** **INT4 Interrupt Request Flag**

–	Generate INT4 interrupt (This bit is set and cleared automatically by hardware when rising and falling signal edge detected at INT4 pin.)
---	---

**IEB** **INTB Interrupt Enable Flag**

0	Disable INTB interrupt requests
1	Enable INTB interrupt requests

**IRQB** **INTB Interrupt Request Flag**

–	Generate INTB interrupt (This bit is set and cleared automatically by hardware when reference interval signal received from basic timer.)
---	---

**IETO, IRQT0 — INTT0 Interrupt Enable/Request Flags**

CPU

FBCH

Bit	3	2	1	0
Identifier	"0"	"0"	IETO	IRQT0
RESET Value	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W
Bit Addressing	1/4	1/4	1/4	1/4

.3 – .2

**Bits 3–2**

0	Always logic zero
---	-------------------

**IETO****INTT0 Interrupt Enable Flag**

0	Disable INTT0 interrupt requests
1	Enable INTT0 interrupt requests

**IRQT0****INTT0 Interrupt Request Flag**

–	Generate INTT0 interrupt (This bit is set and cleared automatically by hardware when contents of TCNT0 and TREF0 registers match.)
---	--

**IET1, IRQT1 — INTT1 Interrupt Enable/Request Flags**

CPU

FBBH

Bit	3	2	1	0
Identifier	"0"	"0"	IET1	IRQT1
RESET Value	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W
Bit Addressing	1/4	1/4	1/4	1/4

.2 – .3

**Bits 2–3**

0	Always logic 0
---	----------------

IET1

**INTT1 Interrupt Enable Flag**

0	Disable INTT1 interrupt requests
1	Enable INTT1 interrupt requests

IRQT1

**INTT1 Interrupt Request Flag**

–	Generate INTT1 interrupt (This bit is set and cleared automatically by hardware when contents of TCNT1 and TREF1 registers match.)
---	--



**IEW, IRQW — INTW Interrupt Enable/Request Flags**

CPU

FBAH

Bit	3	2	1	0
Identifier	"0"	"0"	IEW	IRQW
RESET Value	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W
Bit Addressing	1/4	1/4	1/4	1/4

.3 – .2

**Bits 3–2**

0	Always logic zero
---	-------------------

**IEW****INTW Interrupt Enable Flag**

0	Disable INTW interrupt requests
1	Enable INTW interrupt requests

**IRQW****INTW Interrupt Request Flag**

–	Generate INTW interrupt (This bit is set when the timer interval is set to 0.5 seconds or 3.19 milliseconds at the watch timer frequency of 32.768kHz.)
---	---

**NOTE:** Since INTW is a quasi-interrupt, the IRQW flag must be cleared by software.

**IPR — Interrupt Priority Register**

CPU

FB2H

<b>Bit</b>	3	2	1	0
<b>Identifier</b>	IME	.2	.1	.0
<b>RESET Value</b>	0	0	0	0
<b>Read/Write</b>	W	W	W	W
<b>Bit Addressing</b>	1/4	4	4	4

**IME****Interrupt Master Enable Bit (MSB)**

0	Disable all interrupt processing
1	Enable processing of all interrupt service requests

**IPR.2 – .0****Interrupt Priority Assignment Bits**

0	0	0	Normal interrupt processing according to default priority settings
0	0	1	Process INTB and INT4 <sup>(NOTE)</sup> interrupts at highest priority
0	1	0	Process INT0 interrupts at highest priority
0	1	1	Process INT1 interrupts at highest priority
1	0	1	Process INTT0 interrupts at highest priority
1	1	0	Process INTT1 interrupts at highest priority

**NOTE:** During normal interrupt processing, interrupts are processed in the order in which they occur. If two or more interrupts occur simultaneously, the processing order is determined by the default interrupt priority settings shown below. Using the IPR settings, you can select specific interrupts for high-priority processing in the event of contention. When the high-priority (IPR) interrupt has been processed, waiting interrupts are handled according to their default priorities. The default priorities are as follows ('1' is highest priority; '5' is lowest priority):

INTB, INT4	1
INT0	2
INT1	3
INTT0	4
INTT1	5

**PCON** — Power Control Register

CPU

FB3H

Bit	3	2	1	0
Identifier	.3	.2	.1	.0
RESET Value	0	0	0	0
Read/Write	W	W	W	W
Bit Addressing	1/4	1/4	4	4

**PCON.3 – .2****CPU Operating Mode Control Bits**

0	0	Enable normal CPU operating mode
0	1	Initiate idle power-down mode
1	0	Initiate stop power-down mode

**PCON.1 – .0****CPU Clock Frequency Selection Bits**

0	0	Select fx/64
1	0	Select fx/8
1	1	Select fx/4

**NOTE:** 'fx' is the system clock.

**PSW — Program Status Word****CPU****FB1H, FB0H**

Bit	7	6	5	4	3	2	1	0
Identifier	C	SC2	SC1	SC0	IS1	IS0	EMB	ERB
RESET Value	(1)	0	0	0	0	0	0	0
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W
Bit Addressing	(2)	8	8	8	1/4	1/4	1	1

**C****Carry Flag**

0	No overflow or borrow condition exists
1	An overflow or borrow condition does exist

**SC2 – SC0****Skip Condition Flags**

0	No skip condition exists; no direct manipulation of these bits is allowed
1	A skip condition exists; no direct manipulation of these bits is allowed

**IS1, IS0****Interrupt Status Flags**

0	0	Service all interrupt requests
0	1	Service only the high-priority interrupt(s) as determined in the interrupt priority register (IPR)
1	0	Do not service any more interrupt requests
1	1	Undefined

**EMB****Enable Data Memory Bank Flag**

0	Restrict program access to data memory to bank 15 (F80H–FFFH) and to the locations 000H–07FH in the bank 0 only
1	Enable full access to data memory banks 0, 1, and 15

**ERB****Enable Register Bank Flag**

0	Select register bank 0 as working register area
1	Select register banks 0, 1, 2 or 3 as working register area in accordance with the select register bank (SRB) instruction operand

**NOTES:**

1. The value of the carry flag after a RESET occurs during normal operation is undefined. If a RESET occurs during power-down mode (IDLE or STOP), the current value of the carry flag is retained.
2. The carry flag can only be addressed by a specific set of 1-bit manipulation instructions. See Section 2 for detailed information.

**PMG1 — Port I/O Mode Flags (GROUP 1: PORTS 2, 3) I/O FE9H, FE8H**

Bit	7	6	5	4	3	2	1	0
Identifier	PM3.3	PM3.2 (NOTE)	PM3.1	PM3.0	PM2.3	PM2.2 (NOTE)	PM2.1 (NOTE)	PM2.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	8	8	8	8

**PM3.3****P3.3 I/O Mode Selection Flag**

0	Set P3.3 to input mode
1	Set P3.3 to output mode

**PM3.2 (NOTE)****P3.2 I/O Mode Selection Flag**

0	Set P3.2 to input mode
1	Set P3.2 to input mode

**PM3.1****P3.1 I/O Mode Selection Flag**

0	Set P3.1 to input mode
1	Set P3.1 to output mode

**PM3.0****P3.0 I/O Mode Selection Flag**

0	Set P3.0 to input mode
1	Set P3.0 to output mode

**PM2.3****P2.3 I/O Mode Selection Flag**

0	Set P2.3 to input mode
1	Set P2.3 to output mode

**PM2.2 (NOTE)****P2.2 I/O Mode Selection Flag**

0	Set P2.2 to input mode
1	Set P2.2 to output mode

**PM2.1 (NOTE)****P0.1 I/O Mode Selection Flag**

0	Set P2.1 to input mode
1	Set P2.1 to output mode

**PM2.0****P2.0 I/O Mode Selection Flag**

0	Set P2.0 to input mode
1	Set P2.0 to output mode

**NOTE:** P3.2, P2.2, P2.1 is dedicated for caller id interfacing. (Refer to Chapter 7)

## PMG2 — Port I/O Mode Flags (GROUP 2: PORTS 4, 5) I/O FEBH, FEAH

Bit	7	6	5	4	3	2	1	0
Identifier	PM5.3	PM5.2	PM5.1	PM5.0	PM4.3	PM4.2	PM4.1	PM4.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	8	8	8	8

### PM5.3 P5.3 I/O Mode Selection Flag

0	Set P5.3 to input mode
1	Set P5.3 to output mode

### PM5.2 P5.2 I/O Mode Selection Flag

0	Set P5.2 to input mode
1	Set P5.2 to output mode

### PM5.1 P5.1 I/O Mode Selection Flag

0	Set P5.1 to input mode
1	Set P5.1 to output mode

### PM5.0 P5.0 I/O Mode Selection Flag

0	Set P5.0 to input mode
1	Set P5.0 to output mode

### PM4.3 P4.3 I/O Mode Selection Flag

0	Set P4.3 to input mode
1	Set P4.3 to output mode

### PM4.2 P4.2 I/O Mode Selection Flag

0	Set P4.2 to input mode
1	Set P4.2 to output mode

### PM4.1 P4.1 I/O Mode Selection Flag

0	Set P4.1 to input mode
1	Set P4.1 to output mode

### PM4.0 P4.0 I/O Mode Selection Flag

0	Set P4.0 to input mode
1	Set P4.0 to output mode

## PMG3 — Port I/O Mode Flags (GROUP 3: PORTS 6, 7) I/O FEDH, FECH

Bit	7	6	5	4	3	2	1	0
Identifier	PM7.3	PM7.2	PM7.1	PM7.0	PM6.3	PM6.2	PM6.1	PM6.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	8	8	8	8

### PM7.3 P7.3 I/O Mode Selection Flag

0	Set P7.3 to input mode
1	Set P7.3 to output mode

### PM7.2 P7.2 I/O Mode Selection Flag

0	Set P7.2 to input mode
1	Set P7.2 to output mode

### PM7.1 P7.1 I/O Mode Selection Flag

0	Set P7.1 to input mode
1	Set P7.1 to output mode

### PM7.0 P7.0 I/O Mode Selection Flag

0	Set P7.0 to input mode
1	Set P7.0 to output mode

### PM6.3 P6.3 I/O Mode Selection Flag

0	Set P6.3 to input mode
1	Set P6.3 to output mode

### PM6.2 P6.2 I/O Mode Selection Flag

0	Set P6.2 to input mode
1	Set P6.2 to output mode

### PM6.1 P6.1 I/O Mode Selection Flag

0	Set P6.1 to input mode
1	Set P6.1 to output mode

### PM6.0 P6.0 I/O Mode Selection Flag

0	Set P6.0 to input mode
1	Set P6.0 to output mode

**PMG4 — PORT I/O MODE FLAGS (GROUP 3: PORTS 8, 9) I/O FEFH, FEEH**

Bit	7	6	5	4	3	2	1	0
Identifier	"0"	PM9.2	PM9.1	PM9.0	PM8.3 (NOTE)	PM8.2 (NOTE)	PM8.1 (NOTE)	PM8.0 (NOTE)
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	8	8	8	8

.7

**Bit 7**

0	Always logic zero
---	-------------------

**PM9.2****P9.2 I/O Mode Selection Flag**

0	Set P9.2 to input mode
1	Set P9.2 to output mode

**PM9.1****P9.1 I/O Mode Selection Flag**

0	Set P9.1 to input mode
1	Set P9.1 to output mode

**PM9.0****P9.0 I/O Mode Selection Flag**

0	Set P 9.0 to input mode
1	Set P 9.0 to output mode

**PM8.3 (NOTE)****P8.3 I/O Mode Selection Flag**

0	Set P8.3 to input mode
1	Set P8.3 to output mode

**PM8.2 (NOTE)****P8.2 I/O Mode Selection Flag**

0	Set P8.2 to input mode
1	Set P8.2 to output mode

**PM8.1 (NOTE)****P8.1 I/O Mode Selection Flag**

0	Set P8.1 to input mode
1	Set P8.1 to output mode

**PM8.0 (NOTE)****P8.0 I/O Mode Selection Flag**

0	Set P8.0 to input mode
1	Set P8.0 to output mode

**NOTE:** P8 is dedicated for caller id interfacing. (Refer to Chapter 7)



**PNE 1 — Port Open-Drain Enable Register**

FDBH, FDAH

Bit	7	6	5	4	3	2	1	0
Identifier	PNE5.3	PNE5.2	PNE5.1	PNE5.0	PNE4.3	PNE4.2	PNE4.1	PNE4.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	8	8	8	8

**PNE5.3 P5.3 Output mode Control Bit**

0	Push-pull output
1	N-channel open-drain output

**PNE5.2 P5.2 Output mode Control Bit**

0	Push-pull output
1	N-channel open-drain output

**PNE5.1 P5.1 Output mode Control Bit**

0	Push-pull output
1	N-channel open-drain output

**PNE5.0 P5.0 Output mode Control Bit**

0	Push-pull output
1	N-channel open-drain output

**PNE4.3 P4.3 Output mode Control Bit**

0	Push-pull output
1	N-channel open-drain output

**PNE4.2 P4.2 Output mode Control Bit**

0	Push-pull output
1	N-channel open-drain output

**PNE4.1 P4.1 Output mode Control Bit**

0	Push-pull output
1	N-channel open-drain output

**PNE4.0 P4.0 Output mode Control Bit**

0	Push-pull output
1	N-channel open-drain output

**PUMOD1 — Pull-Up Resistor Mode Register 1**

I/O

FDDH, FDC

Bit	7	6	5	4	3	2	1	0
Identifier	PUR5	PUR4	PUR3	PUR2	PUR1.3	PUR1.2	PUR1.1	"0"
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	8	8	8	8

**PUR5****Connect/Disconnect Port 5 Pull-Up Resistor Control Bit**

0	Disconnect port 5 pull-up resistor
1	Connect port 5 pull-up resistor

**PUR4****Connect/Disconnect Port 4 Pull-Up Resistor Control Bit**

0	Disconnect port 4 pull-up resistor
1	Connect port 4 pull-up resistor

**PUR3****Connect/Disconnect Port 3 Pull-Up Resistor Control Bit**

0	Disconnect port 3 pull-up resistor
1	Connect port 3 pull-up resistor

**PUR2****Connect/Disconnect Port 2 Pull-Up Resistor Control Bit**

0	Disconnect port 2 pull-up resistor
1	Connect port 2 pull-up resistor

**PUR1.3****Connect/Disconnect P1.3 Pull-Up Resistor Control Bit**

0	Disconnect P1.3 pull-up resistor
1	Connect P1.3 pull-up resistor

**PUR1.2****Connect/Disconnect P1.2 Pull-Up Resistor Control Bit**

0	Disconnect P1.2 pull-up resistor
1	Connect P1.2 pull-up resistor

**PUR1.1****Connect/Disconnect P1.1 Pull-Up Resistor Control Bit**

0	Disconnect P1.1 pull-up resistor
1	Connect P1.1 pull-up resistor

**.0****Bit 0**

0	Always logic zero
---	-------------------

**PUMOD2 — Pull-Up Resistor Mode Register 2**

I/O

FDEH

Bit	3	2	1	0
Identifier	PUR9	PUR8 NOTE)	PUR7	PUR6
RESET Value	0	0	0	0
Read/Write	W	W	W	W
Bit Addressing	4	4	4	4

**PUR9****Connect/Disconnect Port 9 Pull-Up Resistor Control Bit**

0	Disconnect port 9 pull-up resistor
1	Connect port 9 pull-up resistor

**PUR8 (note)****Connect/Disconnect Port 8 Pull-Down Resistor Control Bit**

0	Disconnect port 8 pull-down resistor
1	Connect port 8 pull-down resistor

**PUR7****Connect/Disconnect Port 7 Pull-Up Resistor Control Bit**

0	Disconnect port 7 pull-up resistor
1	Connect port 7 pull-up resistor

**PUR6****Connect/Disconnect Port 6 Pull-Up Resistor Control Bit**

0	Disconnect port 6 pull-up resistor
1	Connect port 6 pull-up resistor

**NOTE:** P8 is dedicated for caller id interfacing. (Refer to Chapter 7)

**TMOD0 — Timer/Counter 0 Mode Register****T/C0****F91H, F90H**

Bit	3	2	1	0	3	2	1	0
Identifier	"0"	.6	.5	.4	.3	.2	"0"	"0"
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	1/8	8	8	8

**TMOD0.7****Bit 7**

0	Always logic zero
---	-------------------

**TMOD0.6 – .4****Timer/Counter 0 Input Clock Selection Bits**

0	0	0	External clock input at TCLK pin on rising edge
0	0	1	External clock input at TCLK pin on falling edge
1	0	0	Internal system clock $f_x/2^{10}$ (4.09kHz)
1	0	1	Select clock: $f_x/2^6$ (65.5kHz)
1	1	0	Select clock: $f_x/2^4$ (262kHz)
1	1	1	Select clock: $f_x$ (4.19MHz)

**TMOD0.3****Clear Counter and Resume Counting Control Bit**

1	Clears TCNT0 and IRQT0. TOL0 is retained and resume counting immediately (This bit is cleared automatically when counting starts.)
---	--

**TMOD0.2****Enable/Disable Timer/Counter 0 Bit**

0	Disable timer/counter 0; retain TCNT0 contents
1	Enable timer/counter 0

**TMOD0.1****Bit 1**

0	Always logic zero
---	-------------------

**TMOD0.0****Bit 0**

0	Always logic zero
---	-------------------

**TMOD1 — Timer/Counter 1 Mode Register**

T/C1

FA1H, FA0H

Bit	3	2	1	0	3	2	1	0
Identifier	"0"	.6	.5	.4	.3	.2	"0"	"0"
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	1/8	8	8	8

**TMOD1.7****Bit 7**

0	Always logic zero
---	-------------------

**TMOD1.6 – .4****Timer/Counter 0 Input Clock Selection Bits**

0	0	0	External clock input at TCL1 pin on rising edge
0	0	1	External clock input at TCL1 pin on falling edge
1	0	0	Internal system clock $fx/2^{12}$ (1.02kHz)
1	0	1	Select clock: $fx/2^{10}$ (4.09kHz)
1	1	0	Select clock: $fx/2^8$ (16.4kHz)
1	1	1	Select clock: $fx/2^6$ (65.5kHz)

**TMOD1.3****Clear Counter and Resume Counting Control Bit**

1	Clears TCNT1 and IRQT1. TOL1 is remained and resume counting immediately (This bit is cleared automatically when counting starts.)
---	--

**TMOD1.2****Enable/Disable Timer/Counter 0 Bit**

0	Disable timer/counter 1; retain TCNT1 contents
1	Enable timer/counter 1

**TMOD1.1****Bit 1**

0	Always logic zero
---	-------------------

**TMOD1.0****Bit 0**

0	Always logic zero
---	-------------------

**TOE** — Timer Output Enable Flag Register

T/C

F92H

<b>Bit</b>	3	2	1	0
<b>Identifier</b>	TOE1	TOE0	BOE	"0"
<b>RESET Value</b>	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	W
<b>Bit Addressing</b>	1/4	1/4	1/4	1/4

**TOE1**      **Timer/Counter 1 Output Enable Flag**

0	Disable timer/counter 1 output to the TCLO1 pin
1	Enable timer/counter 1 output to the TCLO1 pin

**TOE0**      **Timer/Counter 0 Output Enable Flag**

0	Disable timer/counter 0 output at the TCLO0 pin
1	Enable timer/counter 0 output at the TCLO0 pin

**BOE**      **Basic Timer Output Enable Flag**

0	Disable basic timer output at the BTCO pin
1	Enable basic timer output at the BTCO pin

**.0**      **Bit 0**

0	Always logic zero
---	-------------------

**WDMOD — Watchdog Timer Mode Register****F99H, F98H**

Bit	7	6	5	4	3	2	1	0
Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	0	1	0	0	1	0	1
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	8	8	8	8

**WDMOD****Watchdog Timer Enable/Disable Control**

5AH	Disable watchdog timer function
Any other value	Enable watchdog timer function

**WDFLAG** — Watchdog Timer Counter Clear Flag Register**F9AH**

Bit	3	2	1	0
Identifier	WDTCF	"0"	"0"	"0"
RESET Value	0	0	0	0
Read/Write	W	W	W	W
Bit Addressing	1/4	1/4	1/4	1/4

**WDTCF****Watchdog Timer Counter Clear Flag**

1	Clears the watchdog timer counter
---	-----------------------------------

**.2-0****Bits 2-0**

0	Always logic zero
---	-------------------

**NOTE:** After watchdog timer is cleared by writing "1", this bit is cleared to "0" automatically.



**WMOD — Watch Timer Mode Register**

WT

F89H, F88H

Bit	3	2	1	0	3	2	1	0
Identifier	.7	"0"	.5	.4	"0"	.2	.1	"0"
RESET Value	0	0	0	0	0	0	0	0
Read/Write	W	W	W	W	W	W	W	W
Bit Addressing	8	8	8	8	8	8	8	8

**WMOD.7****Enable/Disable Buzzer Output Bit**

0	Disable buzzer (BUZ) signal output
1	Enable buzzer (BUZ) signal output

**WMOD.6****Bit 6**

0	Always logic zero
---	-------------------

**WMOD.5 – .4****Output Buzzer Frequency Selection Bits**

0	0	fw/16 buzzer (BUZ) signal output (2kHz)
0	1	fw/8 buzzer (BUZ) signal output (4kHz)
1	0	fw/4 buzzer (BUZ) signal output (8kHz)
1	1	fw/2 buzzer (BUZ) signal output (16kHz)

**WMOD.3****Bit 3**

0	Always logic zero
---	-------------------

**WMOD.2****Enable/Disable Watch Timer Bit**

0	Disable watch timer and clear frequency dividing circuits
1	Enable watch timer

**WMOD.1****Watch Timer Speed Control Bit**

0	Normal speed; set IRQW to 0.5 seconds
1	High-speed operation; set IRQW to 3.91ms

**WMOD.0****Bit 0**

0	Always logic zero
---	-------------------

**NOTE:** System clock of 4.19MHz is assumed.

# 5 INSTRUCTION SET

## OVERVIEW

The instruction set includes 1-bit, 4-bit, and 8-bit instructions for data manipulation, logical and arithmetic operations, program control, and CPU control. I/O instructions for peripheral hardware devices are flexible and easy to use. Symbolic hardware names can be substituted as the instruction operand in place of the actual address. Other important features of the instruction set include:

- 1-byte referencing of long instructions (REF instruction)
- Redundant instruction reduction (string effect)
- Skip feature for ADC and SBC instructions

Instruction operands conform to the operand format defined for each instruction. Several instructions have multiple operand formats.

Predefined values or labels can be used as instruction operands when addressing immediate data. Many of the symbols for specific registers and flags may also be substituted as labels for operations such DA, mema, memb, b, and so on. Using instruction labels can greatly simplify program writing and debugging tasks.

## INSTRUCTION SET FEATURES

In this section, the following instruction set features are described in detail:

- Instruction reference area
- Instruction redundancy reduction
- Flexible bit manipulation
- ADC and SBC instruction skip condition

### Instruction Reference Area

Using the 1-byte REF (REFerence) instruction, you can reference instructions stored in addresses 0020H–007FH of program memory (the REF instruction look-up table). The location referenced by REF may contain either two 1-byte instructions or a single 2-byte instruction. The starting address of the instruction being referenced must always be an even number.

3-byte instructions such as JP or CALL may also be referenced using REF. To reference these 3-byte instructions, the 2-byte pseudo commands TJP and TCALL must be written in the reference.

The PC is not incremented when a REF instruction is executed. After it executes, the program's instruction execution sequence resumes at the address immediately following the REF instruction. By using REF instructions to execute instructions larger than one byte, as well as branches and subroutines, you can reduce the total number of program steps. To summarize, the REF instruction can be used in three ways:

- Using the 1-byte REF instruction to execute one 2-byte or two 1-byte instructions;
- Branching to any location by referencing a branch address that is stored in the look-up table;
- Calling subroutines at any location by referencing a call address that is stored in the look-up table.

If necessary, a REF instruction can be circumvented by means of a skip operation prior to the REF in the execution sequence. In addition, the instruction immediately following a REF can also be skipped by using an appropriate reference instruction or instructions.

Two-byte instructions which can be referenced using a REF instruction are limited to instructions with an execution time of two machine cycles. (An exception to this rule is XCH A,DA. ) In addition, when you use REF to reference two 1-byte instructions stored in the reference area, specific combinations must be used for the first and second 1-byte instruction. These combinations are described in Table 5-1.

**Table 5-1. Valid 1-Byte Instruction Combinations for REF Look-Ups**

First 1-Byte Instruction		Second 1-Byte Instruction	
Instruction	Operand	Instruction	Operand
LD	A,@HL	INCS	L
LD	@HL,A	DECS	L
		DECS	H
		INCS	HL
LD	A,@WX	INCS	X
		INCS	W
		DECS	W
		INCS	WX
LD	A,@WL	INCS	L
		INCS	W
		DECS	W

**NOTE:** If the MSB value of the first one-byte instruction is "0", the instruction cannot be referenced by a REF instruction.

### Reducing Instruction Redundancy

When redundant instructions such as LD A,#im and LD EA,#imm are used consecutively in a program sequence, only the first instruction is executed. The redundant instructions which follow are ignored, that is, they are handled like a NOP instruction. When LD HL,#imm instructions are used consecutively, redundant instructions are also ignored.

In the following example, only the 'LD A, #im' instruction will be executed. The 8-bit load instruction which follows it is interpreted as redundant and is ignored:

```
LD      A,#im           ; Load 4-bit immediate data (#im) to accumulator
LD      EA,#imm        ; Load 8-bit immediate data (#imm) to extended
                        ; accumulator
```

In this example, the statements 'LD A,#2H' and 'LD A,#3H' are ignored:

```
BITR    EMB
LD      A,#1H          ; Execute instruction
LD      A,#2H          ; Ignore, redundant instruction
LD      A,#3H          ; Ignore, redundant instruction
LD      23H,A         ; Execute instruction, 023H ← #1H
```

If consecutive LD HL, #imm instructions (load 8-bit immediate data to the 8-bit memory pointer pair, HL) are detected, only the first LD is executed and the LDs which immediately follow are ignored. For example,

```
LD      HL,#10H        ; HL ← 10H
LD      HL,#20H        ; Ignore, redundant instruction
LD      A,#3H          ; A ← 3H
LD      EA,#35H        ; Ignore, redundant instruction
LD      @HL,A         ; (10H) ← 3H
```

If an instruction reference with a REF instruction has a redundancy effect, the following conditions apply:

- If the instruction *preceding* the REF has a redundancy effect, this effect is cancelled and the referenced instruction is not skipped.
- If the instruction *following* the REF has a redundancy effect, the instruction following the REF is skipped.

 **PROGRAMMING TIP — Example of the Instruction Redundancy Effect**

```
ABC      ORG      0020H
          LD      EA,#30H      ; Stored in REF instruction reference area
          ORG      0080H
          .
          .
          .
          LD      EA,#40H      ; Redundancy effect is encountered
          REF      ABC        ; No skip (EA ← #30H)
          .
          .
          .
          REF      ABC        ; EA ← #30H
          LD      EA,#50H     ; Skip
```

### Flexible Bit Manipulation

In addition to normal bit manipulation instructions like set and clear, the instruction set can also perform bit tests, bit transfers, and bit Boolean operations. Bits can also be addressed and manipulated by special bit addressing modes. Three types of bit addressing are supported:

- mema.b
- memb.@L
- @H+DA.b

The parameters of these bit addressing modes are described in more detail in Table 5-2.

**Table 5-2. Bit Addressing Modes and Parameters**

Addressing Mode	Addressable Peripherals	Address Range
mema.b	ERB, EMB, IS1, IS0, IEx, IRQx	FB0H–FBFH
	Ports 1–9	FF1H–FF9H
memb.@L	Ports 1–9, and BSC	FC0H–FF9H
@H+DA.b	All bit-manipulable peripheral hardware	All bits of the memory bank specified by EMB and SMB that are bit-manipulable

### Instructions Which Have Skip Conditions

The following instructions have a skip function when an overflow or borrow occurs:

XCHI	INCS
XCHD	DECS
LDI	ADS
LDD	SBS

If there is an overflow or borrow from the result of an increment or decrement, a skip signal is generated and a skip is executed. However, the carry flag value is unaffected.

The instructions BTST, BTSF, and CPSE also generate a skip signal and execute a skip when they meet a skip condition, and the carry flag value is also unaffected.

### Instructions Which Affect the Carry Flag

The only instructions which do not generate a skip signal, but which do affect the carry flag are as follows:

ADC	LDB	C,(operand)
SBC	BAND	C,(operand)
SCF	BOR	C,(operand)
RCF	BXOR	C,(operand)
CCF	IRET	
RRC		

### ADC and SBC Instruction Skip Conditions

The instructions 'ADC A,@HL' and 'SBC A,@HL' can generate a skip signal, and set or clear the carry flag, when they are executed in combination with the instruction 'ADS A,#im'.

If an 'ADS A,#im' instruction immediately follows an 'ADC A,@HL' or 'SBC A,@HL' instruction in a program sequence, the ADS instruction does not skip the instruction following ADS, even if it has a skip function. If, however, an 'ADC A,@HL' or 'SBC A,@HL' instruction is immediately followed by an 'ADS A,#im' instruction, the ADC (or SBC) skips on overflow (or if there is no borrow) to the instruction immediately following the ADS, and program execution continues. Table 5-3 contains additional information and examples of the 'ADC A,@HL' and 'SBC A,@HL' skip feature.

**Table 5-3. Skip Conditions for ADC and SBC Instructions**

Sample Instruction Sequences		If the Result of Instruction 1 is:	Then, the Execution Sequence is:	Reason
ADC A,@HL	1	Overflow	1, 3, 4	ADS cannot skip instruction 3, even if it has a skip function.
ADS A,#im	2	No overflow	1, 2, 3, 4	
xxx	3			
xxx	4			
SBC A,@HL	1	Borrow	1, 2, 3, 4	ADS cannot skip instruction 3, even if it has a skip function.
ADS A,#im	2	No borrow	1, 3, 4	
xxx	3			
xxx	4			

## Symbols and Conventions

Table 5-4. Data Type Symbols

Symbol	Data Type
d	Immediate data
a	Address data
b	Bit data
r	Register data
f	Flag data
i	Indirect addressing data
t	memc × 0.5 immediate data

Table 5-5. Register Identifiers

Full Register Name	ID
4-bit accumulator	A
4-bit working registers	E, L, H, X, W, Z, Y
8-bit extended accumulator	EA
8-bit memory pointer	HL
8-bit working registers	WX, YZ, WL
Select register bank 'n'	SRB n
Select memory bank 'n'	SMB n
Carry flag	C
Program status word	PSW
Port 'n'	Pn
'm'-th bit of port 'n'	Pn.m
Interrupt priority register	IPR
Enable memory bank flag	EMB
Enable register bank flag	ERB

Table 5-6. Instruction Operand Notation

Symbol	Definition
DA	Direct address
@	Indirect address prefix
src	Source operand
dst	Destination operand
(R)	Contents of register R
.b	Bit location
im	4-bit immediate data (number)
imm	8-bit immediate data (number)
#	Immediate data prefix
ADR	000H–1FFFH immediate address
ADRn	'n' bit address
R	A, E, L, H, X, W, Z, Y
Ra	E, L, H, X, W, Z, Y
RR	EA, HL, WX, YZ
RRa	HL, WX, WL
RRb	HL, WX, YZ
RRc	WX, WL
mema	FB0H–FBFH, FF1H–FF9H
memb	FC0H–FF9H
memc	Code direct addressing: 0020H–007FH
SB	Select bank register (8 bits)
XOR	Logical exclusive-OR
OR	Logical OR
AND	Logical AND
[(RR)]	Contents addressed by RR



## Opcode Definitions

Table 5-7. Opcode Definitions (Direct)

Register	r2	r1	r0
A	0	0	0
E	0	0	1
L	0	1	0
H	0	1	1
X	1	0	0
W	1	0	1
Z	1	1	0
Y	1	1	1
EA	0	0	0
HL	0	1	0
WX	1	0	0
YZ	1	1	0

r = Immediate data for register

Table 5-8. Opcode Definitions (Indirect)

Register	i2	i1	i0
@HL	1	0	1
@WX	1	1	0
@WL	1	1	1

i = Immediate data for indirect addressing

## Calculating Additional Machine Cycles for Skips

A machine cycle is defined as one cycle of the selected CPU clock. Three different clock rates can be selected using the PCON register.

In this document, the letter 'S' is used in tables when describing the number of additional machine cycles required for an instruction to execute, given that the instruction has a skip function ('S' = skip). The addition number of machine cycles that will be required to perform the skip usually depends on the size of the instruction being skipped — whether it is a 1-byte, 2-byte, or 3-byte instruction. A skip is also executed for SMB and SRB instructions.

The values in additional machine cycles for 'S' for the three cases in which skip conditions occur are as follows:

Case 1: No skip S = 0 cycles

Case 2: Skip is 1-byte or 2-byte instruction S = 1 cycle

Case 3: Skip is 3-byte instruction S = 2 cycles

**NOTE**

REF instructions are skipped in one machine cycle.

## HIGH-LEVEL SUMMARY

This section contains a high-level summary of the instruction set in table format. The tables are designed to familiarize you with the range of instructions that are available in each instruction category.

These tables are a useful quick-reference resource when writing application programs.

If you are reading this user's manual for the first time, however, you may want to scan this detailed information briefly, and then return to it later on. The following information is provided for each instruction:

- Instruction name
- Operand(s)
- Brief operation description
- Number of bytes of the instruction and operand(s)
- Number of machine cycles required to execute the instruction

The tables in this section are arranged according to the following instruction categories:

- CPU control instructions
- Program control instructions
- Data transfer instructions
- Logic instructions
- Arithmetic instructions
- Bit manipulation instructions

Table 5-9. CPU Control Instructions — High-Level Summary

Name	Operand	Operation Description	Bytes	Cycles
SCF		Set carry flag to logic one	1	1
RCF		Reset carry flag to logic zero	1	1
CCF		Complement carry flag	1	1
EI		Enable all interrupts	2	2
DI		Disable all interrupts	2	2
IDLE		Engage CPU idle mode	2	2
STOP		Engage CPU stop mode	2	2
NOP		No operation	1	1
SMB	n	Select memory bank	2	2
SRB	n	Select register bank	2	2
REF	memc	Reference code	1	3
VENTn	EMB (0,1) ERB (0,1) ADR	Load enable memory bank flag (EMB) and the enable register bank flag (ERB) and program counter to vector address, then branch to the corresponding location	2	2

Table 5-10. Program Control Instructions — High-Level Summary

Name	Operand	Operation Description	Bytes	Cycles
CPSE	R,#im	Compare and skip if register equals #im	2	2 + S
	@HL,#im	Compare and skip if indirect data memory equals #im	2	2 + S
	A,R	Compare and skip if A equals R	2	2 + S
	A,@HL	Compare and skip if A equals indirect data memory	1	1 + S
	EA,@HL	Compare and skip if EA equals indirect data memory	2	2 + S
	EA,RR	Compare and skip if EA equals RR	2	2 + S
JP	ADR14	Jump to direct address (14 bits)	3	3
JPS	ADR12	Jump direct in page (12 bits)	2	2
JR	#im	Jump to immediate address	1	2
	@WX	Branch relative to WX register	2	3
	@EA	Branch relative to EA	2	3
CALL	ADR14	Call direct in page (14 bits)	3	4
CALLS	ADR11	Call direct in page (11 bits)	2	3
RET	–	Return from subroutine	1	3
IRET	–	Return from interrupt	1	3
SRET	–	Return from subroutine and skip	1	3 + S

Table 5-11. Data Transfer Instructions — High-Level Summary

Name	Operand	Operation Description	Bytes	Cycles
XCH	A,DA	Exchange A and direct data memory contents	2	2
	A,Ra	Exchange A and register (Ra) contents	1	1
	A,@RRa	Exchange A and indirect data memory	1	1
	EA,DA	Exchange EA and direct data memory contents	2	2
	EA,RRb	Exchange EA and register pair (RRb) contents	2	2
	EA,@HL	Exchange EA and indirect data memory contents	2	2
XCHI	A,@HL	Exchange A and indirect data memory contents; increment contents of register L and skip on carry	1	2 + S
XCHD	A,@HL	Exchange A and indirect data memory contents; decrement contents of register L and skip on carry	1	2 + S
LD	A,#im	Load 4-bit immediate data to A	1	1
	A,@RRa	Load indirect data memory contents to A	1	1
	A,DA	Load direct data memory contents to A	2	2
	A,Ra	Load register contents to A	2	2
	Ra,#im	Load 4-bit immediate data to register	2	2
	RR,#imm	Load 8-bit immediate data to register	2	2
	DA,A	Load contents of A to direct data memory	2	2
	Ra,A	Load contents of A to register	2	2
	EA,@HL	Load indirect data memory contents to EA	2	2
	EA,DA	Load direct data memory contents to EA	2	2
	EA,RRb	Load register contents to EA	2	2
	@HL,A	Load contents of A to indirect data memory	1	1
	DA,EA	Load contents of EA to data memory	2	2
	RRb,EA	Load contents of EA to register	2	2
@HL,EA	Load contents of EA to indirect data memory	2	2	
LDI	A,@HL	Load indirect data memory to A; increment register L contents and skip on carry	1	2 + S
LDD	A,@HL	Load indirect data memory contents to A; decrement register L contents and skip on carry	1	2 + S
LDC	EA,@WX	Load code byte from WX to EA	1	3
	EA,@EA	Load code byte from EA to EA	1	3
RRC	A	Rotate right through carry bit	1	1
PUSH	RR	Push register pair onto stack	1	1
	SB	Push SMB and SRB values onto stack	2	2
POP	RR	Pop to register pair from stack	1	1
	SB	Pop SMB and SRB values from stack	2	2

Table 5-12. Logic Instructions — High-Level Summary

Name	Operand	Operation Description	Bytes	Cycles
AND	A,#im	Logical-AND A immediate data to A	2	2
	A,@HL	Logical-AND A indirect data memory to A	1	1
	EA,RR	Logical-AND register pair (RR) to EA	2	2
	RRb,EA	Logical-AND EA to register pair (RRb)	2	2
OR	A, #im	Logical-OR immediate data to A	2	2
	A, @HL	Logical-OR indirect data memory contents to A	1	1
	EA,RR	Logical-OR double register to EA	2	2
	RRb,EA	Logical-OR EA to double register	2	2
XOR	A,#im	Exclusive-OR immediate data to A	2	2
	A,@HL	Exclusive-OR indirect data memory to A	1	1
	EA,RR	Exclusive-OR register pair (RR) to EA	2	2
	RRb,EA	Exclusive-OR register pair (RRb) to EA	2	2
COM	A	Complement accumulator (A)	2	2

Table 5-13. Arithmetic Instructions — High-Level Summary

Name	Operand	Operation Description	Bytes	Cycles
ADC	A,@HL	Add indirect data memory to A with carry	t	1
	EA,RR	Add register pair (RR) to EA with carry	2	2
	RRb,EA	Add EA to register pair (RRb) with carry	2	2
ADS	A, #im	Add 4-bit immediate data to A and skip on carry	1	1 + S
	EA,#imm	Add 8-bit immediate data to EA and skip on carry	2	2 + S
	A,@HL	Add indirect data memory to A and skip on carry	1	1 + S
	EA,RR	Add register pair (RR) contents to EA and skip on carry	2	2 + S
	RRb,EA	Add EA to register pair (RRb) and skip on carry	2	2 + S
SBC	A,@HL	Subtract indirect data memory from A with carry	1	1
	EA,RR	Subtract register pair (RR) from EA with carry	2	2
	RRb,EA	Subtract EA from register pair (RRb) with carry	2	2
SBS	A,@HL	Subtract indirect data memory from A; skip on borrow	1	1 + S
	EA,RR	Subtract register pair (RR) from EA; skip on borrow	2	2 + S
	RRb,EA	Subtract EA from register pair (RRb); skip on borrow	2	2 + S
DECS	R	Decrement register (R); skip on borrow	1	1 + S
	RR	Decrement register pair (RR); skip on borrow	2	2 + S
INCS	R	Increment register (R); skip on carry	1	1 + S
	DA	Increment direct data memory; skip on carry	2	2 + S
	@HL	Increment indirect data memory; skip on carry	2	2 + S
	RRb	Increment register pair (RRb); skip on carry	1	1 + S

Table 5-14. Bit Manipulation Instructions — High-Level Summary

Name	Operand	Operation Description	Bytes	Cycles
BTST	C	Test specified bit and skip if carry flag is set	1	1 + S
	DA.b	Test specified bit and skip if memory bit is set	2	2 + S
	mema.b			
	memb.@L			
@H+DA.b				
BTSF	DA.b	Test specified memory bit and skip if bit equals "0"		
	mema.b			
	memb.@L			
	@H+DA.b			
BTSTZ	mema.b	Test specified bit; skip and clear if memory bit is set		
	memb.@L			
	@H+DA.b			
BITS	DA.b	Set specified memory bit	2	2
	mema.b			
	memb.@L			
	@H+DA.b			
BITR	DA.b	Clear specified memory bit to logic zero		
	mema.b			
	memb.@L			
	@H+DA.b			
BAND	C,mema.b	Logical-AND carry flag with specified memory bit		
	C,memb.@L			
	C,@H+DA.b			
BOR	C,mema.b	Logical-OR carry with specified memory bit		
	C,memb.@L			
	C,@H+DA.b			
BXOR	C,mema.b	Exclusive-OR carry with specified memory bit		
	C,memb.@L			
	C,@H+DA.b			
LDB	mema.b,C	Load carry bit to a specified memory bit		
	memb.@L,C	Load carry bit to a specified indirect memory bit		
	@H+DA.b,C			
	C,mema.b	Load specified memory bit to carry bit		
	C,memb.@L	Load specified indirect memory bit to carry bit		
	C,@H+DA.b			

**BINARY CODE SUMMARY**

This section contains binary code values and operation notation for each instruction in the instruction set in an easy-to-read, tabular format. It is intended to be used as a quick-reference source for programmers who are experienced with the instruction set. The same binary values and notation are also included in the detailed descriptions of individual instructions later in Section 5.

If you are reading this user's manual for the first time, please just scan this very detailed information briefly. Most of the general information you will need to write application programs can be found in the high-level summary tables in the previous section. The following information is provided for each instruction:

- Instruction name
- Operand(s)
- Binary values
- Operation notation

The tables in this section are arranged according to the following instruction categories:

- CPU control instructions
- Program control instructions
- Data transfer instructions
- Logic instructions
- Arithmetic instructions
- Bit manipulation instructions

Table 5-15. CPU Control Instructions — Binary Code Summary

Name	Operand	Binary Code								Operation Notation
SCF		1	1	1	0	0	1	1	1	$C \leftarrow 1$
RCF		1	1	1	0	0	1	1	0	$C \leftarrow 0$
CCF		1	1	0	1	0	1	1	0	$C \leftarrow C$
EI		1	1	1	1	1	1	1	1	$IME \leftarrow 1$
		1	0	1	1	0	0	1	0	
DI		1	1	1	1	1	1	1	0	$IME \leftarrow 0$
		1	0	1	1	0	0	1	0	
IDLE		1	1	1	1	1	1	1	1	$PCON.2 \leftarrow 1$
		1	0	1	0	0	0	1	1	
STOP		1	1	1	1	1	1	1	1	$PCON.3 \leftarrow 1$
		1	0	1	1	0	0	1	1	
NOP		1	0	1	0	0	0	0	0	No operation
SMB	n	1	1	0	1	1	1	0	1	$SMB \leftarrow n$ ( $n = 0, 1, 15$ )
		0	1	0	0	d3	d2	d1	d0	
SRB	n	1	1	0	1	1	1	0	1	$SRB \leftarrow n$ ( $n = 0, 1, 2, 3$ )
		0	1	0	1	0	0	d1	d0	
REF	memc	t7	t6	t5	t4	t3	t2	t1	t0	$PC12-0 = memc7-4, memc3-0 < 1$
VENTn	EMB (0,1) ERB (0,1) ADR	E	E	0	a12	a11	a10	a9	a8	ROM (2 x n) 7-6 $\leftarrow$ EMB, ERB ROM (2 x n) 5-4 $\leftarrow$ 0, PC12 ROM (2 x n) 3-0 $\leftarrow$ PC12-8 ROM (2 x n + 1) 7-0 $\leftarrow$ PC7-0 ( $n = 0, 1, 2, 3, 4, 5, 6, 7$ )
		a7	a6	a5	a4	a3	a2	a1	a0	



Table 5-16. Program Control Instructions — Binary Code Summary

Name	Operand	Binary Code								Operation Notation
CPSE	R,#im	1	1	0	1	1	0	0	1	Skip if R = im
		d3	d2	d1	d0	0	r2	r1	r0	
	@HL,#im	1	1	0	1	1	1	0	1	Skip if (HL) = im
		0	1	1	1	d3	d2	d1	d0	
	A,R	1	1	0	1	1	1	0	1	Skip if A = R
		0	1	1	0	1	r2	r1	r0	
	A,@HL	0	0	1	1	1	0	0	0	Skip if A = (HL)
	EA,@HL	1	1	0	1	1	1	0	0	Skip if A = (HL), E = (HL+1)
0		0	0	0	1	0	0	1		
EA,RR	1	1	0	1	1	1	0	0	Skip if EA = RR	
	1	1	1	0	1	r2	r1	0		
JP	ADR14	1	1	0	1	1	0	1	1	PC12-0 ← ADR14
		0	0	0	a12	a11	a10	a9	a8	
		a7	a6	a5	a4	a3	a2	a1	a0	
JPS	ADR12	1	0	0	1	a11	a10	a9	a8	PC12-0 ← PC12 + ADR11-0
		a7	a6	a5	a4	a3	a2	a1	a0	
JR	#im *									PC12-0 ← ADR (PC-15 to PC+16)
	@WX	1	1	0	1	1	1	0	1	PC12-0 ← PC12-8 + (WX)
		0	1	1	0	0	1	0	0	
	@EA	1	1	0	1	1	1	0	1	PC12-0 ← PC12-8 + (EA)
0		1	1	0	0	0	0	0		
CALL	ADR14	1	1	0	1	1	0	1	1	[(SP-1) (SP-2)] ← EMB, ERB
		0	1	0	a12	a11	a10	a9	a8	[(SP-3) (SP-4)] ← PC7-0
		a7	a6	a5	a4	a3	a2	a1	a0	[(SP-5) (SP-6)] ← PC12-8
CALLS	ADR11	1	1	1	0	1	a10	a9	a8	[(SP-1) (SP-2)] ← EMB, ERB
		a7	a6	a5	a4	a3	a2	a1	a0	[(SP-3) (SP-4)] ← PC7-0 [(SP-5) (SP-6)] ← PC10-8

	First Byte								Condition
* JR #im	0	0	0	1	a3	a2	a1	a0	PC ← PC+2 to PC+16
	0	0	0	0	a3	a2	a1	a0	PC ← PC-1 to PC-15

Table 5-16. Program Control Instructions — Binary Code Summary (Continued)

Name	Operand	Binary Code								Operation Notation
RET	–	1	1	0	0	0	1	0	1	PC12–8 $\leftarrow$ (SP + 1) (SP) PC7–0 $\leftarrow$ (SP + 2) (SP + 3) EMB,ERB $\leftarrow$ (SP + 5) (SP + 4) SP $\leftarrow$ SP + 6
IRET	–	1	1	0	1	0	1	0	1	PC12–8 $\leftarrow$ (SP + 1) (SP) PC7–0 $\leftarrow$ (SP + 2) (SP + 3) PSW $\leftarrow$ (SP + 4) (SP + 5) SP $\leftarrow$ SP + 6
SRET	–	1	1	1	0	0	1	0	1	PC12–8 $\leftarrow$ (SP + 1) (SP) PC7–0 $\leftarrow$ (SP + 3) (SP + 2) EMB,ERB $\leftarrow$ (SP + 5) (SP + 4) SP $\leftarrow$ SP + 6

Table 5-17. Data Transfer Instructions — Binary Code Summary

Name	Operand	Binary Code								Operation Notation
XCH	A,DA	0	1	1	1	1	0	0	1	A $\leftrightarrow$ DA
		a7	a6	a5	a4	a3	a2	a1	a0	
	A,Ra	0	1	1	0	1	r2	r1	r0	A $\leftrightarrow$ Ra
	A,@RRa	0	1	1	1	1	i2	i1	i0	A $\leftrightarrow$ (RRa)
	EA,DA	1	1	0	0	1	1	1	1	A $\leftrightarrow$ DA, E $\leftrightarrow$ DA + 1
		a7	a6	a5	a4	a3	a2	a1	a0	
	EA,RRb	1	1	0	1	1	1	0	0	EA $\leftrightarrow$ RRb
		1	1	1	0	0	r2	r1	0	
EA,@HL	1	1	0	1	1	1	0	0	A $\leftrightarrow$ (HL), E $\leftrightarrow$ (HL + 1)	
	0	0	0	0	0	0	0	1		
XCHI	A,@HL	0	1	1	1	1	0	1	0	A $\leftrightarrow$ (HL), then L $\leftarrow$ L+1; skip if L = 0H
XCHD	A,@HL	0	1	1	1	1	0	1	1	A $\leftrightarrow$ (HL), then L $\leftarrow$ L-1; skip if L = 0FH
LD	A,#im	1	0	1	1	d3	d2	d1	d0	A $\leftarrow$ im
	A,@RRa	1	0	0	0	1	i2	i1	i0	A $\leftarrow$ (RRa)
	A,DA	1	0	0	0	1	1	0	0	A $\leftarrow$ DA
		a7	a6	a5	a4	a3	a2	a1	a0	
	A,Ra	1	1	0	1	1	1	0	1	A $\leftarrow$ Ra
		0	0	0	0	1	r2	r1	r0	

Table 5-17. Data Transfer Instructions — Binary Code Summary (Continued)

Name	Operand	Binary Code								Operation Notation
LD	Ra,#im	1	1	0	1	1	0	0	1	Ra ← im
		d3	d2	d1	d0	1	r2	r1	r0	
	RR,#imm	1	0	0	0	0	r2	r1	1	RR ← imm
		d7	d6	d5	d4	d3	d2	d1	d0	
	DA,A	1	0	0	0	1	0	0	1	DA ← A
		a7	a6	a5	a4	a3	a2	a1	a0	
	Ra,A	1	1	0	1	1	1	0	1	Ra ← A
		0	0	0	0	0	r2	r1	r0	
	EA,@HL	1	1	0	1	1	1	0	0	A ← (HL), E ← (HL + 1)
		0	0	0	0	1	0	0	0	
	EA,DA	1	1	0	0	1	1	1	0	A ← DA, E ← DA + 1
		a7	a6	a5	a4	a3	a2	a1	a0	
	EA,RRb	1	1	0	1	1	1	0	0	EA ← RRb
		1	1	1	1	1	r2	r1	0	
@HL,A	1	1	0	0	0	1	0	0	(HL) ← A	
DA,EA	1	1	0	0	1	1	0	1	DA ← A, DA + 1 ← E	
	a7	a6	a5	a4	a3	a2	a1	a0		
RRb,EA	1	1	0	1	1	1	0	0	RRb ← EA	
	1	1	1	1	0	r2	r1	0		
@HL,EA	1	1	0	1	1	1	0	0	(HL) ← A, (HL + 1) ← E	
	0	0	0	0	0	0	0	0		
LDI	A,@HL	1	0	0	0	1	0	1	0	A ← (HL), then L ← L+1; skip if L = 0H
LDD	A,@HL	1	0	0	0	1	0	1	1	A ← (HL), then L ← L-1; skip if L = 0FH
LDC	EA,@WX	1	1	0	0	1	1	0	0	EA ← [PC12-8 + (WX)]
	EA,@EA	1	1	0	0	1	0	0	0	EA ← [PC12-8 + (EA)]
RRC	A	1	0	0	0	1	0	0	0	C ← A.0, A3 ← C A.n-1 ← A.n (n = 1, 2, 3)
PUSH	RR	0	0	1	0	1	r2	r1	1	((SP-1)) ((SP-2)) ← (RR), (SP) ← (SP)-2
	SB	1	1	0	1	1	1	0	1	((SP-1)) ← (SMB), ((SP-2)) ← (SRB), (SP) ← (SP)-2
		0	1	1	0	0	1	1	1	

Table 5-17. Data Transfer Instructions — Binary Code Summary (Concluded)

Name	Operand	Binary Code								Operation Notation
POP	RR	0	0	1	0	1	r2	r1	0	$RR_L \leftarrow (SP), RR_H \leftarrow (SP + 1)$ $SP \leftarrow SP + 2$
	SB	1	1	0	1	1	1	0	1	$(SRB) \leftarrow (SP), SMB \leftarrow (SP + 1),$ $SP \leftarrow SP + 2$
		0	1	1	0	0	1	1	0	

Table 5-18. Logic Instructions — Binary Code Summary

Name	Operand	Binary Code								Operation Notation
AND	A,#im	1	1	0	1	1	1	0	1	$A \leftarrow A \text{ AND } im$
		0	0	0	1	d3	d2	d1	d0	
	A,@HL	0	0	1	1	1	0	0	1	$A \leftarrow A \text{ AND } (HL)$
		EA,RR	1	1	0	1	1	1	0	
	RRb,EA	0	0	0	1	1	r2	r1	0	$RRb \leftarrow RRb \text{ AND } EA$
		1	1	0	1	1	1	0	0	
OR	A, #im	1	1	0	1	1	1	0	1	$A \leftarrow A \text{ OR } im$
		0	0	1	0	d3	d2	d1	d0	
	A, @HL	0	0	1	1	1	0	1	0	$A \leftarrow A \text{ OR } (HL)$
		EA,RR	1	1	0	1	1	1	0	
	RRb,EA	0	0	1	0	1	r2	r1	0	$RRb \leftarrow RRb \text{ OR } EA$
		1	1	0	1	1	1	0	0	
XOR	A,#im	1	1	0	1	1	1	0	1	$A \leftarrow A \text{ XOR } im$
		0	0	1	1	d3	d2	d1	d0	
	A,@HL	0	0	1	1	1	0	1	1	$A \leftarrow A \text{ XOR } (HL)$
		EA,RR	1	1	0	1	1	1	0	
	RRb,EA	0	0	1	1	0	r2	r1	0	$RRb \leftarrow RRb \text{ XOR } EA$
		1	1	0	1	1	1	0	0	
COM	A	1	1	0	1	1	1	0	1	$A \leftarrow A$
		0	0	1	1	1	1	1	1	

Table 5-19. Arithmetic Instructions — Binary Code Summary

Name	Operand	Binary Code								Operation Notation
ADC	A,@HL	0	0	1	1	1	1	1	0	$C, A \leftarrow A + (HL) + C$
	EA,RR	1	1	0	1	1	1	0	0	$C, EA \leftarrow EA + RR + C$
		1	0	1	0	1	r2	r1	0	
	RRb,EA	1	1	0	1	1	1	0	0	$C, RRb \leftarrow RRb + EA + C$
1		0	1	0	0	r2	r1	0		
ADS	A, #im	1	0	1	0	d3	d2	d1	d0	$A \leftarrow A + im$ ; skip on carry
	EA,#imm	1	1	0	0	1	0	0	1	$EA \leftarrow EA + imm$ ; skip on carry
		d7	d6	d5	d4	d3	d2	d1	d0	
	A,@HL	0	0	1	1	1	1	1	1	$A \leftarrow A + (HL)$ ; skip on carry
	EA,RR	1	1	0	1	1	1	0	0	$EA \leftarrow EA + RR$ ; skip on carry
		1	0	0	1	1	r2	r1	0	
RRb,EA	1	1	0	1	1	1	0	0	$RRb \leftarrow RRb + EA$ ; skip on carry	
	1	0	0	1	0	r2	r1	0		
SBC	A,@HL	0	0	1	1	1	1	0	0	$C, A \leftarrow A - (HL) - C$
	EA,RR	1	1	0	1	1	1	0	0	$C, EA \leftarrow EA - RR - C$
		1	1	0	0	1	r2	r1	0	
	RRb,EA	1	1	0	1	1	1	0	0	$C, RRb \leftarrow RRb - EA - C$
1		1	0	0	0	r2	r1	0		
SBS	A,@HL	0	0	1	1	1	1	0	1	$A \leftarrow A - (HL)$ ; skip on borrow
	EA,RR	1	1	0	1	1	1	0	0	$EA \leftarrow EA - RR$ ; skip on borrow
		1	0	1	1	1	r2	r1	0	
	RRb,EA	1	1	0	1	1	1	0	0	$RRb \leftarrow RRb - EA$ ; skip on borrow
1		0	1	1	0	r2	r1	0		
DECS	R	0	1	0	0	1	r2	r1	r0	$R \leftarrow R - 1$ ; skip on borrow
	RR	1	1	0	1	1	1	0	0	$RR \leftarrow RR - 1$ ; skip on borrow
		1	1	0	1	1	r2	r1	0	
INCS	R	0	1	0	1	1	r2	r1	r0	$R \leftarrow R + 1$ ; skip on carry
	DA	1	1	0	0	1	0	1	0	$DA \leftarrow DA + 1$ ; skip on carry
		a7	a6	a5	a4	a3	a2	a1	a0	
	@HL	1	1	0	1	1	1	0	1	$(HL) \leftarrow (HL) + 1$ ; skip on carry
		0	1	1	0	0	0	1	0	
RRb	1	0	0	0	0	r2	r1	0	$RRb \leftarrow RRb + 1$ ; skip on carry	

Table 5-20. Bit Manipulation Instructions — Binary Code Summary

Name	Operand	Binary Code								Operation Notation
BTST	C	1	1	0	1	0	1	1	1	Skip if C = 1
	DA.b	1	1	b1	b0	0	0	1	1	Skip if DA.b = 1
		a7	a6	a5	a4	a3	a2	a1	a0	
	mema.b *	1	1	1	1	1	0	0	1	Skip if mema.b = 1
	memb.@L	1	1	1	1	1	0	0	1	Skip if [memb.7–2 + L.3–2]. [L.1–0] = 1
0		1	0	0	a5	a4	a3	a2		
@H+DA.b	1	1	1	1	1	0	0	1	Skip if [H + DA.3–0].b = 1	
	0	0	b1	b0	a3	a2	a1	a0		
BTSF	DA.b	1	1	b1	b0	0	0	1	0	Skip if DA.b = 0
		a7	a6	a5	a4	a3	a2	a1	a0	
	mema.b *	1	1	1	1	1	0	0	0	Skip if mema.b = 0
	memb.@L	1	1	1	1	1	0	0	0	Skip if [memb.7–2 + L.3–2]. [L.1–0] = 0
		0	1	0	0	a5	a4	a3	a2	
@H DA.b	1	1	1	1	1	0	0	0	Skip if [H + DA.3–0].b = 0	
	0	0	b1	b0	a3	a2	a1	a0		
BTSTZ	mema.b *	1	1	1	1	1	1	0	1	Skip if mema.b = 1 and clear
	memb.@L	1	1	1	1	1	1	0	1	Skip if [memb.7–2 + L.3–2]. [L.1–0] = 1 and clear
		0	1	0	0	a5	a4	a3	a2	
@H+DA.b	1	1	1	1	1	1	0	1	Skip if [H + DA.3–0].b = 1 and clear	
	0	0	b1	b0	a3	a2	a1	a0		
BITS	DA.b	1	1	b1	b0	0	0	0	1	DA.b ← 1
		a7	a6	a5	a4	a3	a2	a1	a0	
	mema.b *	1	1	1	1	1	1	1	1	mema.b ← 1
	memb.@L	1	1	1	1	1	1	1	1	[memb.7–2 + L.3–2].b [L.1–0] ← 1
		0	1	0	0	a5	a4	a3	a2	
@H+DA.b	1	1	1	1	1	1	1	1	[H + DA.3–0].b ← 1	
	0	0	b1	b0	a3	a2	a1	a0		

Table 5-20. Bit Manipulation Instructions — Binary Code Summary (Continued)

Name	Operand	Binary Code								Operation Notation	
BITR	DA.b	1	1	b1	b0	0	0	0	0	DA.b ← 0	
		a7	a6	a5	a4	a3	a2	a1	a0		
	mema.b *	1	1	1	1	1	1	1	0	mema.b ← 0	
	memb.@L		1	1	1	1	1	1	1	0	[memb.7–2 + L3–2].[L.1–0] ← 0
			0	1	0	0	a5	a4	a3	a2	
@H+DA.b		1	1	1	1	1	1	1	0	[H + DA.3–0].b ← 0	
		0	0	b1	b0	a3	a2	a1	a0		
BAND	C,mema.b *	1	1	1	1	0	1	0	1	C ← C AND mema.b	
	C,memb.@L		1	1	1	1	0	1	0	1	C ← C AND [memb.7–2 + L.3–2]. [L.1–0]
			0	1	0	0	a5	a4	a3	a2	
	C,@H+DA.b		1	1	1	1	0	1	0	1	C ← C AND [H + DA.3–0].b
			0	0	b1	b0	a3	a2	a1	a0	
BOR	C,mema.b *	1	1	1	1	0	1	1	0	C ← C OR mema.b	
	C,memb.@L		1	1	1	1	0	1	1	0	C ← C OR [memb.7–2 + L.3–2]. [L.1–0]
			0	1	0	0	a5	a4	a3	a2	
	C,@H+DA.b		1	1	1	1	0	1	1	0	C ← C OR [H + DA.3–0].b
			0	0	b1	b0	a3	a2	a1	a0	
BXOR	C,mema.b *	1	1	1	1	0	1	1	1	C ← C XOR mema.b	
	C,memb.@L		1	1	1	1	0	1	1	1	C ← C XOR [memb.7–2 + L.3–2]. [L.1–0]
			0	1	0	0	a5	a4	a3	a2	
	C,@H+DA.b		1	1	1	1	0	1	1	1	C ← C XOR [H + DA.3–0].b
			0	0	b1	b0	a3	a2	a1	a0	

* mema.b	Second Byte								Bit Addresses
	1	0	b1	b0	a3	a2	a1	a0	FB0H–FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H–FF9H

Table 5-20. Bit Manipulation Instructions — Binary Code Summary (Concluded)

Name	Operand	Binary Code								Operation Notation
LDB	mema.b,C *	1	1	1	1	1	1	0	0	mema.b ← C
	memb.@L,C	1	1	1	1	1	1	0	0	memb.7-2 + [L.3-2]. [L.1-0] ← C
		0	1	0	0	a5	a4	a3	a2	
	@H+DA.b,C	1	1	1	1	1	1	0	0	H + [DA.3-0].b ← (C)
		0	b2	b1	b0	a3	a2	a1	a0	
	C,mema.b *	1	1	1	1	0	1	0	0	C ← mema.b
	C,memb.@L	1	1	1	1	0	1	0	0	C ← memb.7-2 + [L.3-2] . [L.1-0]
		0	1	0	0	a5	a4	a3	a2	
	C,@H+DA.b	1	1	1	1	0	1	0	0	C ← [H + DA.3-0].b
		0	b2	b1	b0	a3	a2	a1	a0	

* mema.b	Second Byte								Bit Addresses
	1	0	b1	b0	a3	a2	a1	a0	FB0H-FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H-FF9H



## INSTRUCTION DESCRIPTIONS

This section contains detailed information and programming examples for each instruction of the instruction set. Information is arranged in a consistent format to improve readability and for use as a quick-reference resource for application programmers.

If you are reading this user's manual for the first time, please just scan this very detailed information briefly in order to acquaint yourself with the basic features of the instruction set. The information elements of the instruction description format are as follows:

- Instruction name (mnemonic)
- Full instruction name
- Source/destination format of the instruction operand
- Operation overview (from the "High-Level Summary" table)
- Textual description of the instruction's effect
- Binary code overview (from the "Binary Code Summary" table)
- Programming example(s) to show how the instruction is used

## ADC — Add With Carry

ADC           dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,@HL	Add indirect data memory to A with carry	1	1
	EA,RR	Add register pair (RR) to EA with carry	2	2
	RRb,EA	Add EA to register pair (RRb) with carry	2	2

**Description:** 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. If there is an overflow from the most significant bit of the 1s is no overflow, the ADS instruction is executed normally. (This condition is valid only for 'ADC A,@HL' instructions. If an overflow occurs following an 'ADS A,#im' instruction, the next instruction will not be skipped.)

Operand	Binary Code								Operation Notation
	0	0	1	1	1	1	1	0	
A,@HL	0	0	1	1	1	1	1	0	C, A ← A + (HL) + C
EA,RR	1	1	0	1	1	1	0	0	C, EA ← EA + RR + C
	1	0	1	0	1	r2	r1	0	
RRb,EA	1	1	0	1	1	1	0	0	C, RRb ← RRb + EA + C
	1	0	1	0	0	r2	r1	0	

**Examples:** 1. The extended accumulator contains the value 0C3H, register pair HL the value 0AAH, and the carry flag is set to "1":

```
SCF                ; C ← "1"
ADC    EA,HL        ; EA ← 0C3H + 0AAH + 1H = 6EH, C ← "1"
JPS    XXX          ; Jump to XXX; no skip after ADC
```

2. If the extended accumulator contains the value 0C3H, register pair HL the value 0AAH, and the carry flag is cleared to "0":

```
RCF                ; C ← "0"
ADC    EA,HL        ; EA ← 0C3H + 0AAH + 0H = 6EH, C ← "1"
JPS    XXX          ; Jump to XXX; no skip after ADC
```

## ADC — Add With Carry

ADC (Continued)

**Examples:** 3. If ADC A,@HL is followed by an ADS A,#im, the ADC skips on carry to the instruction immediately after the ADS. An ADS instruction immediately after the ADC does not skip even if an overflow occurs. This function is useful for decimal adjustment operations.

a. 8 + 9 decimal addition (the contents of the address specified by the HL register is 9H):

```
RCF                ; C ← "0"
LD      A,#8H      ; A ← 8H
ADS     A,#6H      ; A ← 8H + 6H = 0EH
ADC     A,@HL      ; A ← 7H, C ← "1"
ADS     A,#0AH     ; Skip this instruction because C = "1" after ADC result
JPS     XXX
```

b. 3 + 4 decimal addition (the contents of the address specified by the HL register is 4H):

```
RCF                ; C ← "0"
LD      A,#3H      ; A ← 3H
ADS     A,#6H      ; A ← 3H + 6H = 9H
ADC     A,@HL      ; A ← 9H + 4H + C(0) = 0DH
ADS     A,#0AH     ; No skip. A ← 0DH + 0AH = 7H
                ; (The skip function for 'ADS A,#im' is inhibited after an
                ; 'ADC A,@HL' instruction even if an overflow occurs.)
JPS     XXX
```

## ADS — Add and Skip on Overflow

ADS           dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A, #im	Add 4-bit immediate data to A and skip on overflow	1	1 + S
	EA,#imm	Add 8-bit immediate data to EA and skip on overflow	2	2 + S
	A,@HL	Add indirect data memory to A and skip on overflow	1	1 + S
	EA,RR	Add register pair (RR) contents to EA and skip on overflow	2	2 + S
	RRb,EA	Add EA to register pair (RRb) and skip on overflow	2	2 + S

**Description:** The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. If there is an overflow from the most significant bit of the result, the skip signal is generated and a skip is executed, but the carry flag value is unaffected.

If 'ADS A,#im' follows an 'ADC A,@HL' instruction in a program, ADC skips the ADS instruction if an overflow occurs. If there is no overflow, the ADS instruction is executed normally. This skip condition is valid only for 'ADC A,@HL' instructions, however. If an overflow occurs following an ADS instruction, the next instruction is not skipped.

Operand	Binary Code								Operation Notation
A, #im	1	0	1	0	d3	d2	d1	d0	$A \leftarrow A + im$ ; skip on overflow
EA,#imm	1	1	0	0	1	0	0	1	$EA \leftarrow EA + imm$ ; skip on overflow
	d7	d6	d5	d4	d3	d2	d1	d0	
A,@HL	0	0	1	1	1	1	1	1	$A \leftarrow A + (HL)$ ; skip on overflow
EA,RR	1	1	0	1	1	1	0	0	$EA \leftarrow EA + RR$ ; skip on overflow
	1	0	0	1	1	r2	r1	0	
RRb,EA	1	1	0	1	1	1	0	0	$RRb \leftarrow RRb + EA$ ; skip on overflow
	1	0	0	1	0	r2	r1	0	

**Examples:** 1. The extended accumulator contains the value 0C3H, register pair HL the value 0AAH, and the carry flag = "0":

```

ADS    EA,HL                ; EA ← 0C3H + 0AAH = 6DH, C ← "0"
                          ; ADS skips on overflow, but carry flag value is not
                          ; affected.
JPS    XXX                  ; This instruction is skipped since ADS had an overflow.
JPS    YYY                  ; Jump to YYY.

```

## ADS — Add and Skip on Overflow

ADS (Continued)

**Examples:** 2. If the extended accumulator contains the value 0C3H, register pair HL the value 12H, and the carry flag = "0":

```
ADS    EA,HL           ; EA ← 0C3H + 12H = 0D5H, C ← "0"
JPS    XXX             ; Jump to XXX; no skip after ADS.
```

3. If 'ADC A,@HL' is followed by an 'ADS A,#im', the ADC skips on overflow to the instruction immediately after the ADS. An 'ADS A,#im' instruction immediately after the 'ADC A,@HL' does not skip even if overflow occurs. This function is useful for decimal adjustment operations.

a. 8 + 9 decimal addition (the contents of the address specified by the HL register is 9H):

```
RCF                    ; C ← "0"
LD    A,#8H           ; A ← 8H
ADS   A,#6H           ; A ← 8H + 6H = 0EH
ADC   A,@HL           ; A ← 7H, C ← "1"
ADS   A,#0AH          ; Skip this instruction because C = "1" after ADC result.
JPS   XXX
```

b. 3 + 4 decimal addition (the contents of the address specified by the HL register is 4H):

```
RCF                    ; C ← "0"
LD    A,#3H           ; A ← 3H
ADS   A,#6H           ; A ← 3H + 6H = 9H
ADC   A,@HL           ; A ← 9H + 4H + C(0) = 0DH
ADS   A,#0AH          ; No skip. A ← 0DH + 0AH = 7H
                        ; (The skip function for 'ADS A,#im' is inhibited after an
                        ; 'ADC A,@HL' instruction even if an overflow occurs.)
JPS   XXX
```

## AND — Logical And

AND           dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,#im	Logical-AND A immediate data to A	2	2
	A,@HL	Logical-AND A indirect data memory to A	1	1
	EA,RR	Logical-AND register pair (RR) to EA	2	2
	RRb,EA	Logical-AND EA to register pair (RRb)	2	2

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

Operand	Binary Code								Operation Notation
A,#im	1	1	0	1	1	1	0	1	A ← A AND im
	0	0	0	1	d3	d2	d1	d0	
A,@HL	0	0	1	1	1	0	0	1	A ← A AND (HL)
EA,RR	1	1	0	1	1	1	0	0	EA ← EA AND RR
	0	0	0	1	1	r2	r1	0	
RRb,EA	1	1	0	1	1	1	0	0	RRb ← RRb AND EA
	0	0	0	1	0	r2	r1	0	

**Example:** If the extended accumulator contains the value 0C3H (11000011B) and register pair HL the value 55H (01010101B), the instruction

```
AND     EA,HL
```

leaves the value 41H (01000001B) in the extended accumulator EA .

## BAND — Bit Logical And

**BAND** C,src.b

Operation:	Operand	Operation Summary	Bytes	Cycles
	C,mema.b	Logical-AND carry flag with memory bit	2	2
	C,memb.@L		2	2
	C,@H+DA.b		2	2

**Description:** The specified bit of the source is logically ANDed with the carry flag bit value. If the Boolean value of the source bit is a logic zero, the carry flag is cleared to "0"; otherwise, the current carry flag setting is left unaltered. The bit value of the source operand is not affected.

Operand	Binary Code								Operation Notation
C,mema.b *	1	1	1	1	0	1	0	1	C ← C AND mema.b
C,memb.@L	1	1	1	1	0	1	0	1	C ← C AND [memb.7–2 + L.3–2]. [L.1–0]
	0	1	0	0	a5	a4	a3	a2	
C,@H+DA.b	1	1	1	1	0	1	0	1	C ← C AND [H + DA.3–0].b
	0	0	b1	b0	a3	a2	a1	a0	

	Second Byte								Bit Addresses
* mema.b	1	0	b1	b0	a3	a2	a1	a0	FB0H–FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H–FF9H

**Examples:** 1. The following instructions set the carry flag if P1.0 (port 1.0) is equal to "1" (and assuming the carry flag is already set to "1"):

```
SMB    15                ; C ← "1"
BAND   C,P1.0           ; If P1.0 = "1", C ← "1"
                          ; If P1.0 = "0", C ← "0"
```

2. Assume the P1 address is FF1H and the value for register L is 9H (1001B). The address (memb.7–2) is 111100B; (L.3–2) is 10B. The resulting address is 11110010B or FF2H, specifying P2. The bit value for the BAND instruction, (L.1–0) is 01B which specifies

bit 1.

Therefore, P1.@L = P2.1:

```
LD     L,#9H
BAND   C,P1.@L          ; P1.@L is specified as P2.1
                          ; C AND P2.1
```

## BAND — Bit Logical And

**BAND** (Continued)

**Examples:** 3. Register H contains the value 2H and FLAG = 20H.3. The address of H is 0010B and FLAG(3–0) is 0000B. The resulting address is 00100000B or 20H. The bit value for the BAND instruction is 3. Therefore, @H+FLAG = 20H.3:

FLAG	EQU	20H.3
LD	H,#2H	
BAND	C,@H+FLAG	; C AND FLAG (20H.3)



## BITR — Bit Reset

BITR           dst.b

Operation:	Operand	Operation Summary	Bytes	Cycles
	DA.b	Clear specified memory bit to logic zero	2	2
	mema.b		2	2
	memb.@L		2	2
	@H+DA.b		2	2

**Description:** A BITR instruction clears to logic zero (resets) the specified bit within the destination operand. No other bits in the destination are affected.

Operand	Binary Code								Operation Notation
DA.b	1	1	b1	b0	0	0	0	0	DA.b ← 0
	a7	a6	a5	a4	a3	a2	a1	a0	
mema.b *	1	1	1	1	1	1	1	0	mema.b ← 0
memb.@L	1	1	1	1	1	1	1	0	[memb.7–2 + L3–2].[L.1–0] ← 0
	0	1	0	0	a5	a4	a3	a2	
@H+DA.b	1	1	1	1	1	1	1	0	[H + DA.3–0].b ← 0
	0	0	b1	b0	a3	a2	a1	a0	

	Second Byte								Bit Addresses
* mema.b	1	0	b1	b0	a3	a2	a1	a0	FB0H–FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H–FF9H

**Examples:**

1. Bit location 30H.2 in the RAM has a current value of logic one. The following instruction clears the third bit in RAM location 30H (bit 2) to logic zero:

```
BITR    30H.2           ; 30H.2 ← "0"
```

2. You can use BITR in the same way to manipulate a port address bit:

```
BITR    P2.0           ; P2.0 ← "0"
```

## BITR — Bit Reset

BITR (Continued)

**Examples:** 3. Assuming that P2.2, P2.3, and P3.0–P3.3 are cleared to "0":

```

BP2  LD    L,#0AH
      BITR  P1.@L           ; First, P1.@0AH = P2.2
                                ; (1111100B) + 10B.10B = 0F2H.2

      INCS  L
      JR    BP2
  
```

4. If bank 0, location 0A0H.0 is cleared (and regardless of whether the EMB value is logic zero), BITR has the following effect:

```

FLAG EQU    0A0H.0
      •
      •
      •
      BITR  EMB
      •
      •
      •
      LD    H,#0AH
      BITR  @H+FLAG           ; Bank 0 (AH + 0H).0 = 0A0H.0 ← "0"
  
```

**NOTE:** Since the BITR instruction is used for output functions, the pin names used in the examples above may change for different devices in the product family.



## BITS — Bit Set

**BITS** (Continued)

**Examples:** 3. Given that P2.2, P2.3, and P3.0–P3.3 are set to "1":

```

BP2    LD      L,#0AH
        BITS   P1.@L           ; First, P1.@0AH = P2.2
                                   ; (1111100B) + 10B.10B = 0F2H.2
        INCS   L
        JR     BP2
  
```

4. If bank 0, location 0A0H.0, is set to "1" and the EMB = "0", BITS has the following effect:

```

FLAG  EQU     0A0H.0
      •
      •
      •
      BITR    EMB
      •
      •
      •
      LD      H,#0AH
      BITS   @H+FLAG           ; Bank 0 (AH + 0H).0 = 0A0H.0 ← "1"
  
```

**NOTE:** Since the BITS instruction is used for output functions, pin names used in the examples above may change for different devices in the product family.

## BOR — Bit Logical OR

BOR C,src.b

Operation:	Operand	Operation Summary	Bytes	Cycles
	C,mema.b	Logical-OR carry with specified memory bit	2	2
	C,memb.@L		2	2
	C,@H+DA.b		2	2

**Description:** The specified bit of the source is logically ORed with the carry flag bit value. The value of the source is unaffected.

Operand	Binary Code								Operation Notation
C,mema.b *	1	1	1	1	0	1	1	0	C ← C OR mema.b
C,memb.@L	1	1	1	1	0	1	1	0	C ← C OR [memb.7–2 + L.3–2]. [L.1–0]
	0	1	0	0	a5	a4	a3	a2	
C,@H+DA.b	1	1	1	1	0	1	1	0	C ← C OR [H + DA.3–0].b
	0	0	b1	b0	a3	a2	a1	a0	

	Second Byte								Bit Addresses
* mema.b	1	0	b1	b0	a3	a2	a1	a0	FB0H–FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H–FF9H

**Examples:** 1. The carry flag is logically ORed with the P1.0 value:

```
RCF                                ; C ← "0"
BOR    C,P1.0                       ; If P1.0 = "1", then C ← "1"; if P1.0 = "0", then C ← "0"
```

2. The P1 address is FF1H and register L contains the value 9H (1001B). The address (memb.7–2) is 111100B and (L.3–2) = 10B. The resulting address is 11110010B or FF2H, specifying P2. The bit value for the BOR instruction, (L.1–0) is 01B which specifies bit 1. Therefore, P1.@L = P2.1:

```
LD    L,#9H
BOR    C,P1.@L                       ; P1.@L is specified as P2.1; C OR P2.1
```

## BOR — Bit Logical OR

**BOR** (Continued)

**Examples:** 3. Register H contains the value 2H and FLAG = 20H.3. The address of H is 0010B and FLAG(3–0) is 0000B. The resulting address is 00100000B or 20H. The bit value for the BOR instruction is 3. Therefore, @H+FLAG = 20H.3:

```
FLAG EQU 20H.3
LD H,#2H
BOR C,@H+FLAG ; C OR FLAG (20H.3)
```

## BTSF — Bit Test and Skip on False

BTSF dst.b

Operation:	Operand	Operation Summary	Bytes	Cycles
	DA.b	Test specified memory bit and skip if bit equals "0"	2	2 + S
	mema.b		2	2 + S
	memb.@L		2	2 + S
	@H+DA.b		2	2 + S

**Description:** The specified bit within the destination operand is tested. If it is a "0", the BTSF instruction skips the instruction which immediately follows it; otherwise the instruction following the BTSF is executed. The destination bit value is not affected.

Operand	Binary Code								Operation Notation
DA.b	1	1	b1	b0	0	0	1	0	Skip if DA.b = 0
	a7	a6	a5	a4	a3	a2	a1	a0	
mema.b *	1	1	1	1	1	0	0	0	Skip if mema.b = 0
memb.@L	1	1	1	1	1	0	0	0	Skip if [memb.7-2 + L.3-2]. [L.1-0] = 0
	0	1	0	0	a5	a4	a3	a2	
@H + DA.b	1	1	1	1	1	0	0	0	Skip if [H + DA.3-0].b = 0
	0	0	b1	b0	a3	a2	a1	a0	

		Second Byte							Bit Addresses	
*	mema.b	1	0	b1	b0	a3	a2	a1	a0	FB0H-FBFH
		1	1	b1	b0	a3	a2	a1	a0	FF1H-FF9H

**Examples:**

1. If RAM bit location 30H.2 is set to logic zero, the following instruction sequence will cause the program to continue execution from the instruction identified as LABEL2:

```

BTSF    30H.2           ; If 30H.2 = "0", then skip
RET     ; If 30H.2 = "1", return
JP      LABEL2

```

2. You can use BTSF in the same way to manipulate a port pin address bit:

```

BTSF    P2.0           ; If P2.0 = "0", then skip
RET     ; If P2.0 = "1", then return
JP      LABEL3

```

## BTSF — Bit Test and Skip on False

**BTSF** (Continued)

**Examples:** 3. P2.2, P2.3 and P3.0–P3.3 are tested:

```

LD      L,#0AH
BP2    BTSF  P1.@L           ; First, P1.@0AH = P2.2
                                   ; (1111100B) + 10B.10B = 0F2H.2

      RET
      INCS  L
      JR    BP2

```

4. Bank 0, location 0A0H.0, is tested and (regardless of the current EMB value) BTSF has the following effect:

```

FLAG  EQU    0A0H.0
      •
      •
      •
      BITR   EMB
      •
      •
      •
      LD     H,#0AH
      BTSF  @H+FLAG       ; If bank 0 (AH + 0H).0 = 0A0H.0 = "0", then skip
      RET
      •
      •
      •

```



## BTST — Bit Test and Skip on True

BTST dst.b

Operation:	Operand	Operation Summary	Bytes	Cycles
	C	Test carry bit and skip if set (= "1")	1	1 + S
	DA.b	Test specified bit and skip if memory bit is set	2	2 + S
	mema.b		2	2 + S
	memb.@L		2	2 + S
	@H+DA.b		2	2 + S

**Description:** The specified bit within the destination operand is tested. If it is "1", the instruction that immediately follows the BTST instruction is skipped; otherwise the instruction following the BTST instruction is executed. The destination bit value is not affected.

Operand	Binary Code								Operation Notation
C	1	1	0	1	0	1	1	1	Skip if C = 1
DA.b	1	1	b1	b0	0	0	1	1	Skip if DA.b = 1
	a7	a6	a5	a4	a3	a2	a1	a0	
mema.b *	1	1	1	1	1	0	0	1	Skip if mema.b = 1
memb.@L	1	1	1	1	1	0	0	1	Skip if [memb.7–2 + L.3–2]. [L.1–0] = 1
	0	1	0	0	a5	a4	a3	a2	
@H+DA.b	1	1	1	1	1	0	0	1	Skip if [H + DA.3–0].b = 1
	0	0	b1	b0	a3	a2	a1	a0	

	Second Byte								Bit Addresses
* mema.b	1	0	b1	b0	a3	a2	a1	a0	FB0H–FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H–FF9H

**Examples:** 1. If RAM bit location 30H.2 is set to logic zero, the following instruction sequence will execute the RET instruction:

```
BTST    30H.2           ; If 30H.2 = "1", then skip
RET     LABEL2         ; If 30H.2 = "0", return
JP      LABEL2
```

## BTST — Bit Test and Skip on True

**BTST** (Continued)

**Examples:** 2. You can use BTST in the same way to manipulate a port pin address bit:

```

BTST    P2.0           ; If P2.0 = "1", then skip
RET     ; If P2.0 = "0", then return
JP      LABEL3

```

3. Assume that P2.2, P2.3 and P3.0–P3.3 are cleared to "0":

```

BP2    LD    L,#0AH
        BTST P1.@L           ; First, P1.@0AH = P2.2
                                (1111100B) + 10B.10B = 0F2H.2
        RET
        INCS L
        JR   BP2

```

4. Bank 0, location 0A0H.0, is tested and (regardless of the current EMB value) BTST has the following effect:

```

FLAG EQU    0A0H.0
.
.
.
BITR  EMB
.
.
.
LD    H,#0AH
BTST  @H+FLAG           ; If bank 0 (AH + 0H).0 = 0A0H.0 = "1", then skip
RET
.
.
.

```

## BTSTZ — Bit Test and Skip on True; Clear Bit

BTSTZ      dst.b

Operation:	Operand	Operation Summary	Bytes	Cycles
	mema.b	Test specified bit; skip and clear if memory bit is set	2	2 + S
	memb.@L		2	2 + S
	@H+DA.b		2	2 + S

**Description:** The specified bit within the destination operand is tested. If it is a "1", the instruction immediately following the BTSTZ instruction is skipped; otherwise the instruction following the BTSTZ is executed. The destination bit value is cleared.

Operand	Binary Code								Operation Notation
mema.b *	1	1	1	1	1	1	0	1	Skip if mema.b = 1 and clear
memb.@L	1	1	1	1	1	1	0	1	Skip if [memb.7–2 + L.3–2]. [L.1–0] = 1 and clear
	0	1	0	0	a5	a4	a3	a2	
@H+DA.b	1	1	1	1	1	1	0	1	Skip if [H + DA.3–0].b = 1 and clear
	0	0	b1	b0	a3	a2	a1	a0	

	Second Byte								Bit Addresses
* mema.b	1	0	b1	b0	a3	a2	a1	a0	FB0H–FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H–FF9H

**Examples:** 1. Port pin P2.0 is toggled by checking the P2.0 value (level):

```
BTSTZ  P2.0           ; If P2.0 = "1", then P2.0 ← "0" and skip
BITS   P2.0           ; If P2.0 = "0", then P2.0 ← "1"
JP     LABEL3
```

2. Assume that port pins P2.2, P2.3 and P3.0–P3.3 are toggled:

```
LD     L,#0AH
BP2   BTSTZ  P1.@L     ; First, P1.@0AH = P2.2
                               ; (1111100B) + 10B.10B = 0F2H.2

RET
INCS  L
JR    BP2
```

## BTSTZ — Bit Test and Skip on True; Clear Bit

**BTSTZ** (Continued)

**Examples:** 3. Bank 0, location 0A0H.0, is tested and EMB = "0":

```

FLAG    EQU    0A0H.0
        .
        .
        .
        BITR   EMB
        .
        .
        .
LD      H,#0AH
BTSTZ  @H+FLAG    ; If bank 0 (AH + 0H).0 = 0A0H.0 = "1", clear and skip
BITS   @H+FLAG    ; If 0A0H.0 = "0", then 0A0H.0 ← "1"

```

## BXOR — Bit Exclusive OR

**BXOR** C,src.b

Operation:	Operand	Operation Summary	Bytes	Cycles
	C,mema.b	Exclusive-OR carry with memory bit	2	2
	C,memb.@L		2	2
	C,@H+DA.b		2	2

**Description:** The specified bit of the source is logically XORed with the carry bit value. The resultant bit is written to the carry flag. The source value is unaffected.

Operand	Binary Code								Operation Notation
C,mema.b *	1	1	1	1	0	1	1	1	C ← C XOR mema.b
C,memb.@L	1	1	1	1	0	1	1	1	C ← C XOR [memb.7–2 + L.3-2]. [L.1–0]
	0	1	0	0	a5	a4	a3	a2	
C,@H+DA.b	1	1	1	1	0	1	1	1	C ← C XOR [H + DA.3–0].b
	0	0	b1	b0	a3	a2	a1	a0	

Second Byte								Bit Addresses	
* mema.b	1	0	b1	b0	a3	a2	a1	a0	FB0H–FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H–FF9H

**Examples:** 1. The carry flag is logically XORed with the P1.0 value:

```
RCF                                ; C ← "0"
BXOR  C,P1.0                        ; If P1.0 = "1", then C ← "1"; if P1.0 = "0", then C ← "0"
```

2. The P1 address is FF1H and register L contains the value 9H (1001B). The address (memb.7–2) is 111100B and (L.3–2) = 10B. The resulting address is 11110010B or FF2H, specifying P2. The bit value for the BXOR instruction, (L.1–0) is 01B which specifies bit 1. Therefore, P1.@L = P2.1:

```
LD      L,#9H
BXOR   C,P1.@L                      ; P1.@L is specified as P2.1; C XOR P2.1
```

## BXOR — Bit Exclusive OR

**BXOR** (Continued)

**Examples:** 3. Register H contains the value 2H and FLAG = 20H.3. The address of H is 0010B and FLAG(3–0) is 0000B. The resulting address is 00100000B or 20H. The bit value for the BOR instruction is 3. Therefore, @H+FLAG = 20H.3:

```
FLAG EQU                20H.3
LD    H,#2H
BXOR  C,@H+FLAG         ; C XOR FLAG (20H.3)
```

## CALL — Call Procedure

CALL       dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	ADR14	Call direct in page (14 bits)	3	4

**Description:** CALL calls a subroutine located at the destination address. The instruction adds three to the program counter to generate the return address and then pushes the result onto the stack, decreasing the stack pointer by six. The EMB and ERB are also pushed to the stack. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 16 K byte program memory address space.

Operand	Binary Code								Operation Notation
ADR14	1	1	0	1	1	0	1	1	[(SP-1) (SP-2)] ← EMB, ERB
	0	1	0	a12	a11	a10	a9	a8	[(SP-3) (SP-4)] ← PC7-0
	a7	a6	a5	a4	a3	a2	a1	a0	[(SP-5) (SP-6)] ← PC12-8

**Example:** The stack pointer value is 00H and the label 'PLAY' is assigned to program memory location 0E3FH. Executing the instruction

CALL    PLAY

at location 0123H will generate the following values:

```

SP      =      0FAH
0FFH   =      0H
0FEH   =      EMB, ERB
0FDH   =      2H
0FCH   =      6H
0FBH   =      0H
0FAH   =      1H
PC     =      0E3FH

```

Data is written to stack locations 0FFH-0FAH as follows:

0FAH	PC11 – PC8			
0FBH	0	0	0	PC12
0FCH	PC3 – PC0			
0FDH	PC7 – PC4			
0FEH	0	0	EMB	ERB
0FFH	0	0	0	0

## CALLS — Call Procedure (Short)

CALLS      dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	ADR11	Call direct in page (11 bits)	2	3

**Description:** The CALLS instruction unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction. Then, it pushes the result onto the stack, decreasing the stack pointer six times. The higher bits of the PC, with the exception of the lower 11 bits, are cleared. The subroutine call must therefore be located within the 2 K byte block (0000H–07FFH) of program memory.

Operand	Binary Code								Operation Notation
ADR11	1	1	1	0	1	a10	a9	a8	[(SP-1) (SP-2)] ← EMB, ERB
	a7	a6	a5	a4	a3	a2	a1	a0	[(SP-3) (SP-4)] ← PC7-0 [(SP-5) (SP-6)] ← PC10-8

**Example:** The stack pointer value is 00H and the label 'PLAY' is assigned to program memory location 0345H. Executing the instruction

CALLS    PLAY

at location 0123H will generate the following values:

```

SP      =      0FAH
0FFH   =      0H
0FEH   =      EMB, ERB
0FDH   =      2H
0FCH   =      5H
0FBH   =      0H
0FAH   =      1H
PC     =      0345H

```

Data is written to stack locations 0FFH–0FAH as follows:

0FAH	0	PC10 – PC8		
0FBH	0	0	0	0
0FCH	PC3 – PC0			
0FDH	PC7 – PC4			
0FEH	0	0	EMB	ERB
0FFH	0	0	0	0



## CCF — Complement Carry Flag

### CCF

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Complement carry flag	1	1

**Description:** The carry flag is complemented; if C = "1" it is changed to C = "0" and vice-versa.

Operand	Binary Code								Operation Notation
–	1	1	0	1	0	1	1	0	$C \leftarrow \bar{C}$

**Example:** If the carry flag is logic zero, the instruction  
CCF  
changes the value to logic one.

## COM — Complement Accumulator

COM      A

Operation:	Operand	Operation Summary	Bytes	Cycles
	A	Complement accumulator (A)	2	2

**Description:** The accumulator value is complemented; if the bit value of A is "1", it is changed to "0" and vice versa.

Operand	Binary Code								Operation Notation
A	1	1	0	1	1	1	0	1	A ← A
	0	0	1	1	1	1	1	1	

**Example:** If the accumulator contains the value 4H (0100B), the instruction

COM      A

leaves the value 0BH (1011B) in the accumulator.

## CPSE — Compare and Skip If Equal

CPSE      dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	R,#im	Compare and skip if register equals #im	2	2 + S
	@HL,#im	Compare and skip if indirect data memory equals #im	2	2 + S
	A,R	Compare and skip if A equals R	2	2 + S
	A,@HL	Compare and skip if A equals indirect data memory	1	1 + S
	EA,@HL	Compare and skip if EA equals indirect data memory	2	2 + S
	EA,RR	Compare and skip if EA equals RR	2	2 + S

**Description:** CPSE compares the source operand (subtracts it from) the destination operand, and skips the next instruction if the values are equal. Neither operand is affected by the comparison.

Operand	Binary Code								Operation Notation
R,#im	1	1	0	1	1	0	0	1	Skip if R = im
	d3	d2	d1	d0	0	r2	r1	r0	
@HL,#im	1	1	0	1	1	1	0	1	Skip if (HL) = im
	0	1	1	1	d3	d2	d1	d0	
A,R	1	1	0	1	1	1	0	1	Skip if A = R
	0	1	1	0	1	r2	r1	r0	
A,@HL	0	0	1	1	1	0	0	0	Skip if A = (HL)
EA,@HL	1	1	0	1	1	1	0	0	Skip if A = (HL), E = (HL+1)
	0	0	0	0	1	0	0	1	
EA,RR	1	1	0	1	1	1	0	0	Skip if EA = RR
	1	1	1	0	1	r2	r1	0	

**Example:** The extended accumulator contains the value 34H and register pair HL contains 56H. The second instruction (RET) in the instruction sequence

```
CPSE   EA,HL
RET
```

is not skipped. That is, the subroutine returns since the result of the comparison is 'not equal.'

## DECS — Decrement and Skip On Borrow

DECS      dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	R	Decrement register (R); skip on borrow	1	1 + S
	RR	Decrement register pair (RR); skip on borrow	2	2 + S

**Description:** The destination is decremented by one. An original value of 00H will underflow to 0FFH. If a borrow occurs, a skip is executed. The carry flag value is unaffected.

Operand	Binary Code								Operation Notation
R	0	1	0	0	1	r2	r1	r0	R ← R-1; skip on borrow
RR	1	1	0	1	1	1	0	0	RR ← RR-1; skip on borrow
	1	1	0	1	1	r2	r1	0	

**Examples:** 1. Register pair HL contains the value 7FH (01111111B). The following instruction leaves the value 7EH in register pair HL:

```
DECS    HL
```

2. Register A contains the value 0H. The following instruction sequence leaves the value 0FFH in register A. Since a "borrow" occurs, the 'CALL PLAY1' instruction is skipped and the 'CALL PLAY2' instruction is executed:

```
DECS    A           ; "Borrow" occurs
CALL    PLAY1       ; Skipped
CALL    PLAY2       ; Executed
```

## DI — Disable Interrupts

DI

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Disable all interrupts	2	2

**Description:** Bit 3 of the interrupt priority register IPR, IME, is cleared to logic zero, disabling all interrupts. Interrupts can still set their respective interrupt status latches, but the CPU will not directly service them.

Operand	Binary Code								Operation Notation
–	1	1	1	1	1	1	1	0	IME ← 0
	1	0	1	1	0	0	1	0	

**Example:** If the IME bit (bit 3 of the IPR) is logic one (e.g., all instructions are enabled), the instruction DI sets the IME bit to logic zero, disabling all interrupts.

## EI — Enable Interrupts

EI

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Enable all interrupts	2	2

**Description:** Bit 3 of the interrupt priority register IPR (IME) is set to logic one. This allows all interrupts to be serviced when they occur, assuming they are enabled. If an interrupt's status latch was previously enabled by an interrupt, this interrupt can also be serviced.

Operand	Binary Code								Operation Notation
–	1	1	1	1	1	1	1	1	IM← 1
	1	0	1	1	0	0	1	0	

**Example:** If the IME bit (bit 3 of the IPR) is logic zero (e.g., all instructions are disabled), the instruction EI sets the IME bit to logic one, enabling all interrupts.

## IDLE — Idle Operation

### IDLE

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Engage CPU idle mode	2	2

**Description:** IDLE causes the CPU clock to stop while the system clock continues oscillating by setting bit 2 of the power control register (PCON). After an IDLE instruction has been executed, peripheral hardware remains operative.

In application programs, an IDLE instruction should 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.

Operand	Binary Code								Operation Notation
–	1	1	1	1	1	1	1	1	PCON.2 ← 1
	1	0	1	0	0	0	1	1	

**Example:** The instruction sequence

```
IDLE
NOP
NOP
NOP
```

sets bit 2 of the PCON register to logic one, stopping the CPU clock. The three NOP instructions provide the necessary timing delay for clock stabilization before the next instruction in the program sequence is executed.

## INCS — Increment and Skip on Carry

INCS           dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	R	Increment register (R); skip on carry	1	1 + S
	DA	Increment direct data memory; skip on carry	2	2 + S
	@HL	Increment indirect data memory; skip on carry	2	2 + S
	RRb	Increment register pair (RRb); skip on carry	1	1 + S

**Description:** The instruction INCS increments the value of the destination operand by one. An original value of 0FH will, for example, overflow to 00H. If a carry occurs, the next instruction is skipped. The carry flag value is unaffected.

Operand	Binary Code								Operation Notation
R	0	1	0	1	1	r2	r1	r0	$R \leftarrow R + 1$ ; skip on carry
DA	1	1	0	0	1	0	1	0	$DA \leftarrow DA + 1$ ; skip on carry
	a7	a6	a5	a4	a3	a2	a1	a0	
@HL	1	1	0	1	1	1	0	1	$(HL) \leftarrow (HL) + 1$ ; skip on carry
	0	1	1	0	0	0	1	0	
RRb	1	0	0	0	0	r2	r1	0	$RRb \leftarrow RRb + 1$ ; skip on carry

**Example:** Register pair HL contains the value 7EH (01111110B). RAM location 7EH contains 0FH. The instruction sequence

```
INCS    @HL                ; 7EH ← "0"
INCS    HL                  ; Skip
INCS    @HL                ; 7EH ← "1"
```

leaves the register pair HL with the value 7EH and RAM location 7EH with the value 1H. Since a carry occurred, the second instruction is skipped. The carry flag value remains unchanged.



## IRET — Return From Interrupt

### IRET

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Return from interrupt	1	3

**Description:** IRET is used at the end of an interrupt service routine. It pops the PC values successively from the stack and restores them to the program counter. The stack pointer is incremented by six and the PSW, enable memory bank (EMB) bit, and enable register bank (ERB) bit are also automatically restored to their pre-interrupt values. Program execution continues from the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower-level or same-level interrupt was pending when the IRET was executed, IRET will be executed before the pending interrupt is processed.

Since the 'a14' bit of an interrupt return address is not stored in the stack, this bit location is always interpreted as a logic zero. The start address in the ROM must for this reason be 3FFFH.

Operand	Binary Code							Operation Notation	
–	1	1	0	1	0	1	0	1	$PC_{12-8} \leftarrow (SP + 1)$ (SP) $PC_{7-0} \leftarrow SP + 2$ (SP + 3) $PSW \leftarrow (SP + 4)$ (SP + 5) $SP \leftarrow SP + 6$

**Example:** The stack pointer contains the value 0FAH. An interrupt is detected in the instruction at location 0122H. RAM locations 0FDH, 0FCH, and 0FAH contain the values 2H, 3H, and 1H, respectively. The instruction

IRET

leaves the stack pointer with the value 00H and the program returns to continue execution at location 123H.

During a return from interrupt, data is popped from the stack to the program counter. The data in stack locations 0FFH–0FAH is organized as follows:

0FAH	PC11 – PC8			
0FBH	0	0	0	PC12
0FCH	PC3 – PC0			
0FDH	PC7 – PC4			
0FEH	IS1	IS0	EMB	ERB
0FFH	C	SC2	SC1	SC0

## JP — Jump

JP           dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	ADR14	Jump to direct address (14 bits)	3	3

**Description:** JP causes an unconditional branch to the indicated address by replacing the contents of the program counter with the address specified in the destination operand. The destination can be anywhere in the 16 K byte program memory address space.

Operand	Binary Code								Operation Notation
ADR14	1	1	0	1	1	0	1	1	PC12-0 ← ADR14
	0	0	0	a12	a11	a10	a9	a8	
	a7	a6	a5	a4	a3	a2	a1	a0	

**Example:** The label 'SYSICON' is assigned to the instruction at program location 07FFH. The instruction

JP           SYSICON

at location 0123H will load the program counter with the value 07FFH.

## JPS — Jump (Short)

JPS           dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	ADR12	Jump direct in page (12 bits)	2	2

**Description:** JPS causes an unconditional branch to the indicated address with the 4 K byte program memory address space. Bits 0–11 of the program counter are replaced with the directly specified address. The destination address for this jump is specified to the assembler by a label or by an actual address in program memory.

Operand	Binary Code								Operation Notation
ADR12	1	0	0	1	a11	a10	a9	a8	PC12–0 ← PC12+ ADR11–0
	a7	a6	a5	a4	a3	a2	a1	a0	

**Example:** The label 'SUB' is assigned to the instruction at program memory location 00FFH. The instruction

```
JPS       SUB
```

at location 0EABH will load the program counter with the value 00FFH. Normally, the JPS instruction jumps to the address in the block in which the instruction is located. If the first byte of the instruction code is located at address xFFEh or xFFFh, the instruction will jump to the next block. If the instruction 'JPS SUB' were located instead at program memory address 0FFEh or 0FFFh, the instruction 'JPS SUB' would load the PC with the value 10FFh, causing a program malfunction.

## JR — Jump Relative (Very Short)

JR dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	#im	Branch to relative immediate address	1	2
	@WX	Branch relative to contents of WX register	2	3
	@EA	Branch relative to contents of EA	2	3

**Description:** JR causes the relative address to be added to the program counter and passes control to the instruction whose address is now in the PC. The range of the relative address is current PC – 15 to current PC + 16. The destination address for this jump is specified to the assembler by a label, an actual address, or by immediate data using a plus sign (+) or a minus sign (–).

For immediate addressing, the (+) range is from 2 to 16 and the (–) range is from –1 to –15. If a 0, 1, or any other number that is outside these ranges are used, the assembler interprets it as an error.

For JR @WX and JR @EA branch relative instructions, the valid range for the relative address is 0H–0FFH. The destination address for these jumps can be specified to the assembler by a label that lies anywhere within the current 256-byte block.

Normally, the 'JR @WX' and 'JR @EA' instructions jump to the address in the page in which the instruction is located. However, if the first byte of the instruction code is located at address xxFEH or xxFFH, the instruction will jump to the next page.

Operand	Binary Code								Operation Notation
#im *									PC12–0 ← ADR (PC–15 to PC+16)
@WX	1	1	0	1	1	1	0	1	PC12–0 ← PC12–8 + (WX)
	0	1	1	0	0	1	0	0	
@EA	1	1	0	1	1	1	0	1	PC12–0 ← PC12–8 + (EA)
	0	1	1	0	0	0	0	0	

		First Byte							Condition	
* JR #im		0	0	0	1	a3	a2	a1	a0	PC ← PC+2 to PC+16
		0	0	0	0	a3	a2	a1	a0	PC ← PC–1 to PC–15

## JR — Jump Relative (Very Short)

JR (Continued)

**Examples:** 1. A short form for a relative jump to label 'KK' is the instruction

```
JR KK
```

where 'KK' must be within the allowed range of current PC-15 to current PC+16. The JR instruction has in this case the effect of an unconditional JP instruction.

2. In the following instruction sequence, if the instruction 'LD WX, #02H' were to be executed in place of 'LD WX,#00H', the program would jump to 1002H and 'JPS BBB' would be executed. If 'LD EA,#04H' were to be executed, the jump would be to 1004H and 'JPS CCC' executed.

would be

```
ORG 1000H
```

```
JPS AAA
JPS BBB
JPS CCC
JPS DDD
```

```
LD WX,#00H ; WX ← 00H
LD EA,WX
ADS WX,EA ; WX ← (WX) + (WX)
JR @WX ; Current PC12-8 (10H) + WX (00H) = 1000H
; Jump to address 1000H and execute JPS AAA
```

3. Here is another example:

```
ORG 1100H
```

```
LD A,#0H
LD A,#1H
LD A,#2H
LD A,#3H
LD 30H,A ; Address 30H ← A
JPS YYY
```

```
XXX LD EA,#00H ; EA ← 00H
JR @EA ; Jump to address 1100H
; Address 30H ← 00H
```

If 'LD EA,#01H' were to be executed in place of 'LD EA,#00H', the program would jump to 1001H and address 30H would contain the value 1H. If 'LD EA,#02H' were to be executed, the jump would be to 1002H and address 30H would contain the value 2H.

## LD — Load

LD dst,src

Operation:

Operand	Operation Summary	Bytes	Cycles
A,#im	Load 4-bit immediate data to A	1	1
A,@RRa	Load indirect data memory contents to A	1	1
A,DA	Load direct data memory contents to A	2	2
A,Ra	Load register contents to A	2	2
Ra,#im	Load 4-bit immediate data to register	2	2
RR,#imm	Load 8-bit immediate data to register	2	2
DA,A	Load contents of A to direct data memory	2	2
Ra,A	Load contents of A to register	2	2
EA,@HL	Load indirect data memory contents to EA	2	2
EA,DA	Load direct data memory contents to EA	2	2
EA,RRb	Load register contents to EA	2	2
@HL,A	Load contents of A to indirect data memory	1	1
DA,EA	Load contents of EA to data memory	2	2
RRb,EA	Load contents of EA to register	2	2
@HL,EA	Load contents of EA to indirect data memory	2	2

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

If an instruction such as 'LD A,#im' (LD EA,#imm) or 'LD HL,#imm' is written more than two times in succession, only the first LD will be executed; the other similar instructions that immediately follow the first LD will be treated like a NOP. This is called the 'redundancy effect' (see examples below).

Operand	Binary Code								Operation Notation
A,#im	1	0	1	1	d3	d2	d1	d0	$A \leftarrow im$
A,@RRa	1	0	0	0	1	i2	i1	i0	$A \leftarrow (RRa)$
A,DA	1	0	0	0	1	1	0	0	$A \leftarrow DA$
	a7	a6	a5	a4	a3	a2	a1	a0	
A,Ra	1	1	0	1	1	1	0	1	$A \leftarrow Ra$
	0	0	0	0	1	r2	r1	r0	
Ra,#im	1	1	0	1	1	0	0	1	$Ra \leftarrow im$
	d3	d2	d1	d0	1	r2	r1	r0	

**LD — Load**

LD (Continued)

Description:	Operand		Binary Code						Operation Notation	
RR,#imm	1	0	0	0	0	r2	r1	1	RR ← imm	
	d7	d6	d5	d4	d3	d2	d1	d0		
DA,A	1	0	0	0	1	0	0	1	DA ← A	
	a7	a6	a5	a4	a3	a2	a1	a0		
Ra,A	1	1	0	1	1	1	0	1	Ra ← A	
	0	0	0	0	0	r2	r1	r0		
EA,@HL	1	1	0	1	1	1	0	0	A ← (HL), E ← (HL + 1)	
	0	0	0	0	1	0	0	0		
EA,DA	1	1	0	0	1	1	1	0	A ← DA, E ← DA + 1	
	a7	a6	a5	a4	a3	a2	a1	a0		
EA,RRb	1	1	0	1	1	1	0	0	EA ← RRb	
	1	1	1	1	1	r2	r1	0		
@HL,A	1	1	0	0	0	1	0	0	(HL) ← A	
DA,EA	1	1	0	0	1	1	0	1	DA ← A, DA + 1 ← E	
	a7	a6	a5	a4	a3	a2	a1	a0		
RRb,EA	1	1	0	1	1	1	0	0	RRb ← EA	
	1	1	1	1	0	r2	r1	0		
@HL,EA	1	1	0	1	1	1	0	0	(HL) ← A, (HL + 1) ← E	
	0	0	0	0	0	0	0	0		

**Examples:** 1. RAM location 30H contains the value 4H. The RAM location values are 40H, 41H and 0AH, 3H respectively. The following instruction sequence leaves the value 40H in point pair HL, 0AH in the accumulator and in RAM location 40H, and 3H in register E.

```
LD    HL,#30H           ; HL ← 30H
LD    A,@HL            ; A ← 4H
LD    HL,#40H          ; HL ← 40H
LD    EA,@HL           ; A ← 0AH, E ← 3H
LD    @HL,A           ; RAM (40H) ← 0AH
```

## LD — Load

LD (Continued)

**Examples:** 2. If an instruction such as LD A,#im (LD EA,#imm) or LD HL,#imm is written more than two times in succession, only the first LD is executed; the next instructions are treated as NOPs. Here are two examples of this 'redundancy effect':

```
LD    A,#1H           ; A ← 1H
LD    EA,#2H         ; NOP
LD    A,#3H          ; NOP
LD    23H,A          ; (23H) ← 1H

LD    HL,#10H        ; HL ← 10H
LD    HL,#20H        ; NOP
LD    A,#3H          ; A ← 3H
LD    EA,#35         ; NOP
LD    @HL,A          ; (10H) ← 3H
```

The following table contains descriptions of special characteristics of the LD instruction when used in different addressing modes:

Instruction	Operation Description and Guidelines
LD A,#im	Since the 'redundancy effect' occurs with instructions like LD EA,#imm, if this instruction is used consecutively, the second and additional instructions of the same type will be treated like NOPs.
LD A,@RRa	Load the data memory contents pointed to by 8-bit RRa register pairs (HL, WX, WL) to the A register.
LD A,DA	Load direct data memory contents to the A register.
LD A,Ra	Load 4-bit register Ra (E, L, H, X, W, Z, Y) to the A register.
LD Ra,#im	Load 4-bit immediate data into the Ra register (E, L, H, X, W, Y, Z).
LD RR,#imm	Load 8-bit immediate data into the Ra register (EA, HL, WX, YZ). There is a redundancy effect if the operation addresses the HL or EA registers.
LD DA,A	Load contents of register A to direct data memory address.
LD Ra,A	Load contents of register A to 4-bit Ra register (E, L, H, X, W, Z, Y).



**LD — Load**

LD (Concluded)

**Examples:**

Instruction	Operation Description and Guidelines
LD EA,@HL	Load data memory contents pointed to by 8-bit register HL to the A register, and the contents of HL+1 to the E register. The contents of register L must be an even number. If the number is odd, the LSB of register L is recognized as a logic zero (an even number), and it is not replaced with the true value. For example, 'LD HL,#36H' loads immediate 36H to HL and the next instruction 'LD EA,@HL' loads the contents of 36H to register A and the contents of 37H to register E.
LD EA,DA	Load direct data memory contents of DA to the A register, and the next direct data memory contents of DA + 1 to the E register. The DA value must be an even number. If it is an odd number, the LSB of DA is recognized as a logic zero (an even number), and it is not replaced with the true value. For example, 'LD EA,37H' loads the contents of 36H to the A register and the contents of 37H to the E register.
LD EA,RRb	Load 8-bit RRb register (HL, WX, YZ) to the EA register. H, W, and Y register values are loaded into the E register, and the L, X, and Z values into the A register.
LD @HL,A	Load A register contents to data memory location pointed to by the 8-bit HL register value.
LD DA,EA	Load the A register contents to direct data memory and the E register contents to the next direct data memory location. The DA value must be an even number. If it is an odd number, the LSB of the DA value is recognized as logic zero (an even number), and is not replaced with the true value.
LD RRb,EA	Load contents of EA to the 8-bit RRb register (HL, WX, YZ). The E register is loaded into the H, W, and Y register and the A register into the L, X, and Z register.
LD @HL,EA	Load the A register to data memory location pointed to by the 8-bit HL register, and the E register contents to the next location, HL + 1. The contents of the L register must be an even number. If the number is odd, the LSB of the L register is recognized as logic zero (an even number), and is not replaced with the true value. For example, 'LD HL,#36H' loads immediate 36H to register HL; the instruction 'LD @HL,EA' loads the contents of A into address 36H and the contents of E into address 37H.

## LDB — Load Bit

LDB dst,src.b  
LDB dst.b,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	mema.b,C	Load carry bit to a specified memory bit	2	2
	memb.@L,C	Load carry bit to a specified indirect memory bit	2	2
	@H+DA.b,C		2	2
	C,mema.b	Load memory bit to a specified carry bit	2	2
	C,memb.@L	Load indirect memory bit to a specified carry bit	2	2
	C,@H+DA.b		2	2

**Description:** The Boolean variable indicated by the first or second operand is copied into the location specified by the second or first operand. One of the operands must be the carry flag; the other may be any directly or indirectly addressable bit. The source is unaffected.

Operand	Binary Code								Operation Notation
mema.b,C *	1	1	1	1	1	1	0	0	mema.b ← C
memb.@L,C	1	1	1	1	1	1	0	0	memb.7–2 + [L.3–2]. [L.1–0] ← C
	0	1	0	0	a5	a4	a3	a2	
@H+DA.b,C	1	1	1	1	1	1	0	0	H + [DA.3–0].b ← (C)
	0	b2	b1	b0	a3	a2	a1	a0	
C,mema.b*	1	1	1	1	0	1	0	0	C ← mema.b
C,memb.@L	1	1	1	1	0	1	0	0	C ← memb.7–2 + [L.3–2]. [L.1–0]
	0	1	0	0	a5	a4	a3	a2	
C,@H+DA.b	1	1	1	1	0	1	0	0	C ← [H + DA.3–0].b
	0	b2	b1	b0	a3	a2	a1	a0	

	Second Byte								Bit Addresses
* mema.b	1	0	b1	b0	a3	a2	a1	a0	FB0H–FBFH
	1	1	b1	b0	a3	a2	a1	a0	FF1H–FF9H

## LDB — Load Bit

**LDB** (Continued)

**Examples:** 1. The carry flag is set and the data value at input pin P1.0 is logic zero. The following instruction clears the carry flag to logic zero.

```
LDB    C,P1.0
```

2. The P1 address is FF1H and the L register contains the value 9H (1001B). The address (memb.7–2) is 111100B and (L.3–2) is 10B. The resulting address is 11110010B or FF2H and P2 is addressed. The bit value (L.1–0) is specified as 01B (bit 1).

```
LD     L,#9H
LDB    C,P1.@L          ; P1.@L specifies P2.1 and C ← P2.1
```

3. The H register contains the value 2H and FLAG = 20H.3. The address for H is 0010B and for FLAG(3–0) the address is 0000B. The resulting address is 00100000B or 20H. The bit value is 3. Therefore, @H+FLAG = 20H.3.

```
FLAG   EQU  20H.3
LD     H,#2H
LDB    C,@H+FLAG       ; C ← FLAG (20H.3)
```

4. The following instruction sequence sets the carry flag and the loads the "1" data value to the output pin P2.0, setting it to output mode:

```
SCF                    ; C ← "1"
LDB    P2.0,C          ; P2.0 ← "1"
```

5. The P1 address is FF1H and L = 9H (1001B). The address (memb.7–2) is 111100B and (L.3–2) is 10B. The resulting address, 11110010B specifies P2. The bit value (L.1–0) is specified as 01B (bit 1). Therefore, P1.@L = P2.1.

```
SCF                    ; C ← "1"
LD     L,#9H
LDB    P1.@L,C         ; P1.@L specifies P2.1
                          ; P2.1 ← "1"
```

6. In this example, H = 2H and FLAG = 20H.3 and the address 20H is specified. Since the bit value is 3, @H+FLAG = 20H.3:

```
FLAG   EQU  20H.3
RCF                    ; C ← "0"
LD     H,#2H
LDB    @H+FLAG,C      ; FLAG(20H.3) ← "0"
```

**NOTE:** Port pin names used in examples 4 and 5 may vary with different devices.

## LDC — Load Code Byte

LDC dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	EA,@WX	Load code byte from WX to EA	1	3
	EA,@EA	Load code byte from EA to EA	1	3

**Description:** This instruction is used to load a byte from program memory into an extended accumulator. The address of the byte fetched is the five highest bit values in the program counter and the contents of an 8-bit working register (either WX or EA). The contents of the source are unaffected.

Operand	Binary Code								Operation Notation
EA,@WX	1	1	0	0	1	1	0	0	$EA \leftarrow [PC12-8 + (WX)]$
EA,@EA	1	1	0	0	1	0	0	0	$EA \leftarrow [PC12-8 + (EA)]$

**Examples:** 1. The following instructions will load one of four values defined by the define byte (DB) directive to the extended accumulator:

```

LD      EA,#00H
CALL   DISPLAY
JPS    MAIN

ORG    0500H

DB     66H
DB     77H
DB     88H
DB     99H
•
•
•
DISPLAY LDC   EA,@EA      ; EA ← address 0500H = 66H
RET

```

If the instruction 'LD EA,#01H' is executed in place of 'LD EA,#00H', The content of 0501H (77H) is loaded to the EA register. If 'LD EA,#02H' is executed, the content of address 0502H (88H) is loaded to EA.

## LDC — Load Code Byte

LDC (Continued)

**Examples:** 2. The following instructions will load one of four values defined by the define byte (DB) directive to the extended accumulator:

```

        ORG      0500

        DB      66H
        DB      77H
        DB      88H
        DB      99H
        .
        .
        .
DISPLAY LD      WX,#00H
        LDC     EA,@WX          ; EA ← address 0500H = 66H
        RET

```

If the instruction 'LD WX,#01H' is executed in place of 'LD WX,#00H', then  
EA ← address 0501H = 77H.

If the instruction 'LD WX,#02H' is executed in place of 'LD WX,#00H', then  
EA ← address 0502H = 88H.

3. Normally, the LDC EA, @EA and the LDC EA, @WX instructions reference the table data on the page on which the instruction is located. If, however, the instruction is located at address xxFFH, it will reference table data on the next page. In this example, the upper 4 bits of the address at location 0200H is loaded into register E and the lower 4 bits into register A:

```

        ORG      01FDH

01FDH  LD      WX,#00H
01FFH  LDC     EA,@WX          ; E ← upper 4 bits of 0200H address
                                   ; A ← lower 4 bits of 0200H address

```

4. Here is another example of page referencing with the LDC instruction:

```

        ORG      0100

        DB      67H
        SMB     0
        LD      HL,#30H      ; Even number
        LD      WX,#00H
        LDC     EA,@WX      ; E ← upper 4 bits of 0100H address
                                   ; A ← lower 4 bits of 0100H address
        LD      @HL,EA      ; RAM (30H) ← 7, RAM (31H) ← 6

```

## LDD — Load Data Memory and Decrement

LDD           dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,@HL	Load indirect data memory contents to A; decrement register L contents and skip on borrow	1	2 + S

**Description:** The contents of a data memory location are loaded into the accumulator, and the contents of the register L are decreased by one. If a "borrow" occurs (e.g., if the resulting value in register L is 0FH), the next instruction is skipped. The contents of data memory and the carry flag value are not affected.

Operand	Binary Code								Operation Notation
A,@HL	1	0	0	0	1	0	1	1	A ← (HL), then L ← L-1; skip if L = 0FH

**Example:** In this example, assume that register pair HL contains 20H and internal RAM location 20H contains the value 0FH:

```
LD      HL,#20H
LDD     A,@HL      ; A ← (HL) and L ← L-1
JPS     XXX        ; Skip
JPS     YYY        ; H ← 2H and L ← 0FH
```

The instruction 'JPS XXX' is skipped since a "borrow" occurred after the 'LDD A,@HL' and instruction 'JPS YYY' is executed.

## LDI — Load Data Memory and Increment

LDI dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,@HL	Load indirect data memory to A; increment register L contents and skip on overflow	1	2 + S

**Description:** The contents of a data memory location are loaded into the accumulator, and the contents of the register L are incremented by one. If an overflow occurs (e.g., if the resulting value in register L is 0H), the next instruction is skipped. The contents of data memory and the carry flag value are not affected.

Operand	Binary Code								Operation Notation
A,@HL	1	0	0	0	1	0	1	0	$A \leftarrow (HL)$ , then $L \leftarrow L+1$ ; skip if $L = 0H$

**Example:** Assume that register pair HL contains the address 2FH and internal RAM location 2FH contains the value 0FH:

```
LD    HL,#2FH
LDI   A,@HL           ; A ← (HL) and L ← L+1
JPS   XXX             ; Skip
JPS   YYY             ; H ← 2H and L ← 0H
```

The instruction 'JPS XXX' is skipped since an overflow occurred after the 'LDI A,@HL' and the instruction 'JPS YYY' is executed.

## NOP — No Operation

### NOP

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	No operation	1	1

**Description:** No operation is performed by a NOP instruction. It is typically used for timing delays.

One NOP causes a 1-cycle delay: with a 1  $\mu$ s cycle time, five NOPs would therefore cause a 5  $\mu$ s delay. Program execution continues with the instruction immediately following the NOP. Only the PC is affected. At least three NOP instructions should follow a STOP or IDLE instruction.

Operand	Binary Code								Operation Notation
–	1	0	1	0	0	0	0	0	No operation

**Example:** Three NOP instructions follow the STOP instruction to provide a short interval for clock stabilization before power-down mode is initiated:

```
STOP
NOP
NOP
NOP
```



## OR — Logical OR

OR            dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A, #im	Logical-OR immediate data to A	2	2
	A, @HL	Logical-OR indirect data memory contents to A	1	1
	EA,RR	Logical-OR double register to EA	2	2
	RRb,EA	Logical-OR EA to double register	2	2

**Description:** The source operand is logically ORed with the destination operand. The result is stored in the destination. The contents of the source are unaffected.

Operand	Binary Code								Operation Notation
A, #im	1	1	0	1	1	1	0	1	A ← A OR im
	0	0	1	0	d3	d2	d1	d0	
A, @HL	0	0	1	1	1	0	1	0	A ← A OR (HL)
EA,RR	1	1	0	1	1	1	0	0	EA ← EA OR RR
	0	0	1	0	1	r2	r1	0	
RRb,EA	1	1	0	1	1	1	0	0	RRb ← RRb OR EA
	0	0	1	0	0	r2	r1	0	

**Example:** If the accumulator contains the value 0C3H (11000011B) and register pair HL the value 55H (01010101B), the instruction

OR        EA,@HL

leaves the value 0D7H (11010111B) in the accumulator .

## POP — Pop From Stack

POP           dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	RR	Pop to register pair from stack	1	1
	SB	Pop SMB and SRB values from stack	2	2

**Description:** The contents of the RAM location addressed by the stack pointer is read, and the SP is incremented by two. The value read is then transferred to the variable indicated by the destination operand.

Operand	Binary Code								Operation Notation
RR	0	0	1	0	1	r2	r1	0	$RR_L \leftarrow (SP), RR_H \leftarrow (SP+1)$ $SP \leftarrow SP+2$
SB	1	1	0	1	1	1	0	1	$(SRB) \leftarrow (SP), SMB \leftarrow (SP+1),$ $SP \leftarrow SP+2$
	0	1	1	0	0	1	1	0	

**Example:** The SP value is equal to 0EDH, and RAM locations 0EFH through 0EDH contain the values 2H, 3H, and 4H, respectively. The instruction

```
POP     HL
```

leaves the stack pointer set to 0EFH and the data pointer pair HL set to 34H.

## PUSH — Push Onto Stack

PUSH            src

Operation:	Operand	Operation Summary	Bytes	Cycles
	RR	Push register pair onto stack	1	1
	SB	Push SMB and SRB values onto stack	2	2

**Description:** The SP is then decreased by two and the contents of the source operand are copied into the RAM location addressed by the stack pointer, thereby adding a new element to the top of the stack.

Operand	Binary Code								Operation Notation
RR	0	0	1	0	1	r2	r1	1	$(SP-1) \leftarrow RR_H, (SP-2) \leftarrow RR_L$ $SP \leftarrow SP-2$
SB	1	1	0	1	1	1	0	1	$(SP-1) \leftarrow SMB, (SP-2) \leftarrow SRB;$ $(SP) \leftarrow SP-2$
	0	1	1	0	0	1	1	1	

**Example:** As an interrupt service routine begins, the stack pointer contains the value 0FAH and the data pointer register pair HL contains the value 20H. The instruction

```
PUSH    HL
```

leaves the stack pointer set to 0F8H and stores the values 2H and 0H in RAM locations 0F9H and 0F8H, respectively.

## RCF — Reset Carry Flag

### RCF

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Reset carry flag to logic zero	1	1

**Description:** The carry flag is cleared to logic zero, regardless of its previous value.

Operand	Binary Code								Operation Notation
–	1	1	1	0	0	1	1	0	$C \leftarrow 0$

**Example:** Assuming the carry flag is set to logic one, the instruction  
RCF  
resets (clears) the carry flag to logic zero.

## REF — Reference Instruction

REF           dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	memc	Reference code	1	3 *

\*           The REF instruction for a 16K CALL instruction is 4 cycles.

**Description:** The REF instruction is used to rewrite into 1-byte form, arbitrary 2-byte or 3-byte instructions (or two 1-byte instructions) stored in the REF instruction reference area in program memory. REF reduces the number of program memory accesses for a program.

Operand	Binary Code								Operation Notation
memc	t7	t6	t5	t4	t3	t2	t1	t0	PC12-0 = memc7-4, memc3-0 <1

TJP and TCALL are 2-byte pseudo-instructions that are used only to specify the reference area:

1. When the reference area is specified by the TJP instruction,  
 $\text{memc.7-6} = 00$   
 $\text{PC11-0} \leftarrow \text{memc.3-0} + (\text{memc} + 1)$
2. When the reference area is specified by the TCALL instruction,  
 $\text{memc.7-6} = 01$   
 $(\text{SP-4}) (\text{SP-1}) (\text{SP-2}) \leftarrow \text{PC11-0}$   
 $\text{SP-3} \leftarrow \text{EMB, ERB, 0, 0}$   
 $\text{PC11-0} \leftarrow \text{memc.3-0} + (\text{memc} + 1)$   
 $\text{SP} \leftarrow \text{SP-4}$

When the reference area is specified by any other instruction, the 'memc' and 'memc + 1' instructions are executed.

Instructions referenced by REF occupy 2 bytes of memory space (for two 1-byte instructions or one 2-byte instruction) and must be written as an even number from 0020H to 007FH in ROM. In addition, the destination address of the TJP and TCALL instructions must be located with the 3FFFH address. TJP and TCALL are reference instructions for JP/JPS and CALL/CALLS.

If the instruction following a REF is subject to the 'redundancy effect', the redundant instruction is skipped. If, however, the REF follows a redundant instruction, it is executed.

On the other hand, the binary code of a REF instruction is 1 byte. The upper 4 bits become the higher address bits of the referenced instruction, and the lower 4 bits of the referenced instruction ( x 1/2) becomes the lower address, producing a total of 8 bits or 1 byte (see Example 3 below).

## REF — Reference Instruction

REF (Continued)

**Examples:** 1. Instructions can be executed efficiently using REF, as shown in the following example:

```

        ORG    0020H
AAA     LD     HL,#00H
BBB     LD     EA,#FFH
CCC     TCALL  SUB1
DDD     TJP   SUB2
        .
        .
        .
ORG 0080H
REF     AAA           ; LD     HL,#00H
REF     BBB           ; LD     EA,#FFH
REF     CCC           ; CALL  SUB1
REF     DDD           ; JP    SUB2

```

2. The following example shows how the REF instruction is executed in relation to LD instructions that have a 'redundancy effect':

```

        ORG    0020H
AAA     LD     EA,#40H
        .
        .
        .
        ORG    0100H
LD      EA,#30H
REF     AAA           ; Not skipped
        .
        .
        .
REF     AAA
LD      EA,#50H       ; Skipped
SRB    2

```

## REF — Reference Instruction

REF (Concluded)

**Examples:** 3. In this example the binary code of 'REF A1' at locations 20H–21H is 20H, for 'REF A2' at locations 22H–23H, it is 21H, and for 'REF A3' at 24H–25H, the binary code is 22H :

Opcode	Symbol	Instruction	
		ORG	0020H
8300	A1	LD	HL,#00H
8303	A2	LD	HL,#03H
8305	A3	LD	HL,#05H
8310	A4	LD	HL,#10H
8326	A5	LD	HL,#26H
8308	A6	LD	HL,#08H
830F	A7	LD	HL,#0FH
83F0	A8	LD	HL,#0F0H
8367	A9	LD	HL,#067H
410B	A10	TCALL	SUB1
010D	A11	TJP	SUB2
		•	
		•	
		•	
		ORG	0100H
20	REF	A1	; LD HL,#00H
21	REF	A2	; LD HL,#03H
22	REF	A3	; LD HL,#05H
23	REF	A4	; LD HL,#10H
24	REF	A5	; LD HL,#26H
25	REF	A6	; LD HL,#08H
26	REF	A7	; LD HL,#0FH
27	REF	A8	; LD HL,#0F0H
30	REF	A9	; LD HL,#067H
31	REF	A10	; CALL SUB1
32	REF	A11	; JP SUB2

## RET — Return From Subroutine

### RET

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Return from subroutine	1	3

**Description:** RET pops the PC values successively from the stack, incrementing the stack pointer by six. Program execution continues from the resulting address, generally the instruction immediately following a CALL or CALLS.

Operand	Binary Code								Operation Notation
–	1	1	0	0	0	1	0	1	PC12–8 ← (SP+1) (SP) PC7–0 ← (SP+2) (SP+3) PSW ← EMB,ERB SP ← SP+6

**Example:** The stack pointer contains the value 0FAH. RAM locations 0FAH, 0FBH, 0FCH, and and 0FDH contain 1H, 0H, 5H, and 2H, respectively. The instruction

RET

leaves the stack pointer with the new value of 00H and program execution continues from location 0125H.

During a return from subroutine, PC values are popped from stack locations as follows:

SP →	PC11 – PC8			
SP + 1	0	0	0	PC12
SP + 2	PC3 – PC0			
SP + 3	PC7 – PC4			
SP + 4	0	0	EMB	ERB
SP + 5	0	0	0	0
SP + 6				



## RRC — Rotate Accumulator Right Through Carry

RRC      A

Operation:	Operand	Operation Summary	Bytes	Cycles
	A	Rotate right through carry bit	1	1

**Description:** The four bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag and the original carry value moves into the bit 3 accumulator position.

Operand	Binary Code								Operation Notation
A	1	0	0	0	1	0	0	0	$C \leftarrow A.0, A3 \leftarrow C$ $A.n-1 \leftarrow A.n \quad (n = 1, 2, 3)$

**Example:** The accumulator contains the value 5H (0101B) and the carry flag is cleared to logic zero. The instruction

RRC      A

leaves the accumulator with the value 2H (0010B) and the carry flag set to logic one.

## SBC — Subtract With Carry

**SBC** dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,@HL	Subtract indirect data memory from A with carry	1	1
	EA,RR	Subtract register pair (RR) from EA with carry	2	2
	RRb,EA	Subtract EA from register pair (RRb) with carry	2	2

**Description:** SBC subtracts the source and carry flag value from the destination operand, leaving the result in the destination. SBC sets the carry flag if a borrow is needed for the most significant bit; otherwise it clears the carry flag. The contents of the source are unaffected.

If the carry flag was set before the SBC instruction was executed, a borrow was needed for the previous step in multiple precision subtraction. In this case, the carry bit is subtracted from the destination along with the source operand.

Operand	Binary Code								Operation Notation
A,@HL	0	0	1	1	1	1	0	0	$C, A \leftarrow A - (HL) - C$
EA,RR	1	1	0	1	1	1	0	0	$C, EA \leftarrow EA - RR - C$
	1	1	0	0	1	r2	r1	0	
RRb,EA	1	1	0	1	1	1	0	0	$C, RRb \leftarrow RRb - EA - C$
	1	1	0	0	0	r2	r1	0	

**Examples:**

1. The extended accumulator contains the value 0C3H, register pair HL the value 0AAH, and the carry flag is set to "1":

```
SCF                ; C ← "1"
SBC    EA,HL       ; EA ← 0C3H - 0AAH - 1H, C ← "0"
JPS    XXX         ; Jump to XXX; no skip after SBC
```

2. If the extended accumulator contains the value 0C3H, register pair HL the value 0AAH, and the carry flag is cleared to "0":

```
RCF                ; C ← "0"
SBC    EA,HL       ; EA ← 0C3H - 0AAH - 0H = 19H, C ← "0"
JPS    XXX         ; Jump to XXX; no skip after SBC
```

## SBC — Subtract With Carry

**SBC** (Continued)

**Examples:** 3. If SBC A,@HL is followed by an ADS A,#im, the SBC skips on 'no borrow' to the instruction immediately after the ADS. An 'ADS A,#im' instruction immediately after the 'SBC A,@HL' instruction does not skip even if an overflow occurs. This function is useful for decimal adjustment operations.

a. 8 – 6 decimal addition (the contents of the address specified by the HL register is 6H):

```
RCF                ; C ← "0"
LD      A,#8H      ; A ← 8H
SBC     A,@HL      ; A ← 8H – 6H – C(0) = 2H, C ← "0"
ADS     A,#0AH     ; Skip this instruction because no borrow after SBC result
JPS     XXX
```

b. 3 – 4 decimal addition (the contents of the address specified by the HL register is 4H):

```
RCF                ; C ← "0"
LD      A,#3H      ; A ← 3H
SBC     A,@HL      ; A ← 3H – 4H – C(0) = 0FH, C ← "1"
ADS     A,#0AH     ; No skip. A ← 0FH + 0AH = 9H
                ; (The skip function of 'ADS A,#im' is inhibited after a
                ; 'SBC A,@HL' instruction even if an overflow occurs.)
JPS     XXX
```

## SBS — Subtract

SBS dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,@HL	Subtract indirect data memory from A; skip on borrow	1	1 + S
	EA,RR	Subtract register pair (RR) from EA; skip on borrow	2	2 + S
	RRb,EA	Subtract EA from register pair (RRb); skip on borrow	2	2 + S

**Description:** The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. A skip is executed if a borrow occurs. The value of the carry flag is not affected.

Operand	Binary Code								Operation Notation
A,@HL	0	0	1	1	1	1	0	1	$A \leftarrow A - (HL)$ ; skip on borrow
EA,RR	1	1	0	1	1	1	0	0	$EA \leftarrow EA - RR$ ; skip on borrow
	1	0	1	1	1	r2	r1	0	
RRb,EA	1	1	0	1	1	1	0	0	$RRb \leftarrow RRb - EA$ ; skip on borrow
	1	0	1	1	0	r2	r1	0	

**Examples:** 1. The accumulator contains the value 0C3H, register pair HL contains the value 0C7H, and the carry flag is cleared to logic zero:

```

RCF                                ; C ← "0"
SBS    EA,HL                        ; EA ← 0C3H – 0C7H, C ← "0"
                                ; SBS instruction skips on borrow,
                                ; but carry flag value is not affected
JPS    XXX                          ; Skip because a borrow occurred
JPS    YYY                          ; Jump to YYY is executed

```

2. The accumulator contains the value 0AFH, register pair HL contains the value 0AAH, and the carry flag is set to logic one:

```

SCF                                ; C ← "1"
SBS    EA,HL                        ; EA ← 0AFH – 0AAH, C ← "1"
JPS    XXX                          ; Jump to XXX
                                ; JPS was not skipped since no "borrow" occurred
                                ; after SBS

```

## SCF — Set Carry Flag

### SCF

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Set carry flag to logic one	1	1

**Description:** The SCF instruction sets the carry flag to logic one, regardless of its previous value.

Operand	Binary Code								Operation Notation
–	1	1	1	0	0	1	1	1	$C \leftarrow 1$

**Example:** If the carry flag is cleared to logic zero, the instruction  
SCF  
sets the carry flag to logic one.

## SMB — Select Memory Bank

SMB            n

Operation:	Operand	Operation Summary	Bytes	Cycles
	n	Select memory bank	2	2

**Description:** The SMB instruction sets the upper four bits of a 12-bit data memory address to select a specific memory bank. The constants 0, 1, and 15 are usually used as the SMB operand to select the corresponding memory bank. All references to data memory addresses fall within the following address ranges:

Please note that since data memory spaces differ for various devices in the SAM4 product family, the 'n' value of the SMB instruction will also vary.

Addresses	Register Areas	Bank	SMB
000H–01FH	Working registers	0	0
020H–0FFH	Stack and general-purpose registers		
100H–1DFH	General-purpose registers	1	1
1E0H–1FFH	Display registers		
F80H–FFFH	I/O-mapped hardware registers	15	15

The enable memory bank (EMB) flag must always be set to "1" in order for the SMB instruction to execute successfully for memory banks 0, 1, and 15.

Format	Binary Code								Operation Notation
n	1	1	0	1	1	1	0	1	SMB ← n (n = 0, 1, 15)
	0	1	0	0	d3	d2	d1	d0	

**Example:** If the EMB flag is set, the instruction

SMB        0

selects the data memory address range for bank 0 (000H–0FFH) as the working memory bank.

## SRB — Select Register Bank

SRB            n

Operation:	Operand	Operation Summary	Bytes	Cycles
	n	Select register bank	2	2

**Description:** The SRB instruction selects one of four register banks in the working register memory area. The constant value used with SRB is 0, 1, 2, or 3. The following table shows the effect of SRB settings:

ERB Setting	SRB Settings				Selected Register Bank
	3	2	1	0	
0	0	0	x	x	Always set to bank 0
1	0	0	0	0	Bank 0
			0	1	Bank 1
			1	0	Bank 2
			1	1	Bank 3

**NOTE:** 'x' = not applicable.

The enable register bank flag (ERB) must always be set for the SRB instruction to execute successfully for register banks 0, 1, 2, and 3. In addition, if the ERB value is logic zero, register bank 0 is always selected, regardless of the SRB value.

Operand	Binary Code								Operation Notation
n	1	1	0	1	1	1	0	1	SRB ← n (n = 0, 1, 2, 3)
	0	1	0	1	0	0	d1	d0	

**Example:** If the ERB flag is set, the instruction

SRB        3

selects register bank 3 (018H–01FH) as the working memory register bank.

## SRET — Return from Subroutine and Skip

### SRET

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Return from subroutine and skip	1	3 + S

**Description:** SRET is normally used to return to the previously executing procedure at the end of a subroutine that was initiated by a CALL or CALLS instruction. SRET skips the resulting address, which is generally the instruction immediately after the point at which the subroutine was called. Then, program execution continues from the resulting address and the contents of the location addressed by the stack pointer are popped into the program counter.

Operand	Binary Code								Operation Notation
–	1	1	1	0	0	1	0	1	$PC_{12-8} \leftarrow (SP + 1) (SP)$ $PC_{7-0} \leftarrow (SP + 3) (SP + 2)$ $EMB, ERB \leftarrow (SP + 5) (SP + 4)$ $SP \leftarrow SP + 6$

**Example:** If the stack pointer contains the value 0FAH and RAM locations 0FAH, 0FBH, 0FCH, and 0FDH contain the values 1H, 0H, 5H, and 2H, respectively, the instruction

SRET

leaves the stack pointer with the value 00H and the program returns to continue execution at location 0125H.

During a return from subroutine, data is popped from the stack to the PC as follows:

SP →	PC11 – PC8			
SP + 1	0	0	0	PC12
SP + 2	PC3 – PC0			
SP + 3	PC7 – PC4			
SP + 4	0	0	EMB	ERB
SP + 5	0	0	0	0
SP + 6				



## STOP — Stop Operation

### STOP

Operation:	Operand	Operation Summary	Bytes	Cycles
	–	Engage CPU stop mode	2	2

**Description:** The STOP instruction stops the system clock by setting bit 3 of the power control register (PCON) to logic one. When STOP executes, all system operations are halted with the exception of some peripheral hardware with special power-down mode operating conditions.

In application programs, a STOP instruction should 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.

Operand	Binary Code								Operation Notation
–	1	1	1	1	1	1	1	1	PCON.3 ← 1
	1	0	1	1	0	0	1	1	

**Example:** Given that bit 3 of the PCON register is cleared to logic zero, and all systems are operational, the instruction sequence

```
STOP
NOP
NOP
NOP
```

sets bit 3 of the PCON register to logic one, stopping all controller operations (with the exception of some peripheral hardware). The three NOP instructions provide the necessary timing delay for clock stabilization before the next instruction in the program sequence is executed.

## VENT — Load EMB, ERB, and Vector Address

VENTn dst

Operation:	Operand	Operation Summary	Bytes	Cycles
	EMB (0,1) ERB (0,1) ADR	Load enable memory bank flag (EMB) and the enable register bank flag (ERB) and program counter to vector address, then branch to the corresponding location.	2	2

**Description:** The VENT instruction loads the contents of the enable memory bank flag (EMB) and enable register bank flag (ERB) into the respective vector addresses. It then points the interrupt service routine to the corresponding branching locations. The program counter is loaded automatically with the respective vector addresses which indicate the starting address of the respective vector interrupt service routines.

The EMB and ERB flags should be modified using VENT before the vector interrupts are acknowledged. Then, when an interrupt is generated, the EMB and ERB values of the previous routine are automatically pushed onto the stack and then popped back when the routine is completed.

After the return from interrupt (IRET) you do not need to set the EMB and ERB values again. Instead, use BTR and BITS to clear these values in your program routine.

The starting addresses for vector interrupts and reset operations are pointed to by the VENTn instruction. These addresses must be stored in ROM locations 0000H–3FFFH. Generally, the VENTn instructions are coded starting at location 0000H.

The format for VENT instructions is as follows:

VENTn d1,d2,ADDR

EMB ← d1 ("0" or "1")

ERB ← d2 ("0" or "1")

PC ← ADDR (address to branch)

n = device-specific module address code (n = 0–n)

Operand	Binary Code								Operation Notation
EMB (0,1) ERB (0,1) ADR	E M B	E R B	0	a12	a11	a10	a9	a8	ROM (2 x n) 7–6 ← EMB, ERB ROM (2 x n) 5–4 ← 0, PC12 ROM (2 x n) 3–0 ← PC12–8 ROM (2 x n + 1) 7–0 ← PC7–0 (n = 0, 1, 2, 3, 4, 5, 6, 7)
	a7	a6	a5	a4	a3	a2	a1	a0	

## VENT — Load EMB, ERB, and Vector Address

VENTn (Continued)

**Example:** The instruction sequence

```
ORG    0000H
VENT0  1,0,RESET
VENT1  0,1,INTB
VENT2  0,1,INT0
VENT3  0,1,INT1
NOP
NOP
VENT5  0,1,INTT0
VENT6  0,1,INTT1
```

causes the program sequence to branch to the RESET routine labeled 'RESET,' setting EMB to "1" and ERB to "0" when RESET is activated. When a basic timer interrupt is generated, VENT1 causes the program to branch to the basic timer's interrupt service routine, INTB, and to set the EMB value to "0" and the ERB value to "1". VENT2 then branches to INT0, VENT3 to INT1, and so on, setting the appropriate EMB and ERB values.

## XCH — Exchange a or EA With Nibble or Byte

XCH           dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,DA	Exchange A and data memory contents	2	2
	A,Ra	Exchange A and register (Ra) contents	1	1
	A,@RRa	Exchange A and indirect data memory	1	1
	EA,DA	Exchange EA and direct data memory contents	2	2
	EA,RRb	Exchange EA and register pair (RRb) contents	2	2
	EA,@HL	Exchange EA and indirect data memory contents	2	2

**Description:** The instruction XCH loads the accumulator with the contents of the indicated destination variable and writes the original contents of the accumulator to the source.

Operand	Binary Code								Operation Notation
A,DA	0	1	1	1	1	0	0	1	A ↔ DA
	a7	a6	a5	a4	a3	a2	a1	a0	
A,Ra	0	1	1	0	1	r2	r1	r0	A ↔ Ra
A,@RRa	0	1	1	1	1	i2	i1	i0	A ↔ (RRa)
EA,DA	1	1	0	0	1	1	1	1	A ↔ DA, E ↔ DA + 1
	a7	a6	a5	a4	a3	a2	a1	a0	
EA,RRb	1	1	0	1	1	1	0	0	EA ↔ RRb
	1	1	1	0	0	r2	r1	0	
EA,@HL	1	1	0	1	1	1	0	0	A ↔ (HL), E ↔ (HL + 1)
	0	0	0	0	0	0	0	1	

**Example:** Double register HL contains the address 20H. The accumulator contains the value 3FH (0011111B) and internal RAM location 20H the value 75H (01110101B). The instruction

XCH       EA,@HL

leaves RAM location 20H with the value 3FH (0011111B) and the extended accumulator with the value 75H (01110101B).

## XCHD — Exchange and Decrement

XCHD dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,@HL	Exchange A and data memory contents; decrement contents of register L and skip on borrow	1	2 + S

**Description:** The instruction XCHD exchanges the contents of the accumulator with the RAM location addressed by register pair HL and then decrements the contents of register L. If the content of register L is 0FH, the next instruction is skipped. The value of the carry flag is not affected.

Operand	Binary Code								Operation Notation
A,@HL	0	1	1	1	1	0	1	1	A ↔ (HL), then L ← L-1; skip if L = 0FH

**Example:** Register pair HL contains the address 20H and internal RAM location 20H contains the value 0FH:

```
LD      HL,#20H
LD      A,#0H
XCHD   A,@HL      ; A ← 0FH and L ← L - 1, (HL) ← "0"
JPS    XXX        ; Skipped since a borrow occurred
JPS    YYY        ; H ← 2H, L ← 0FH

YYY   XCHD   A,@HL      ; (2FH) ← 0FH, A ← (2FH), L ← L - 1 = 0EH
      .
      .
      .
```

The 'JPS YYY' instruction is executed since a skip occurs after the XCHD instruction.

## XCHI — Exchange and Increment

XCHI dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,@HL	Exchange A and data memory contents; increment contents of register L and skip on overflow	1	2 + S

**Description:** The instruction XCHI exchanges the contents of the accumulator with the RAM location addressed by register pair HL and then increments the contents of register L. If the content of register L is 0H, a skip is executed. The value of the carry flag is not affected.

Operand	Binary Code								Operation Notation
A,@HL	0	1	1	1	1	0	1	0	A ↔ (HL), then L ← L+1; skip if L = 0H

**Example:** Register pair HL contains the address 2FH and internal RAM location 2FH contains 0FH:

```
LD      HL,#2FH
LD      A,#0H
XCHI   A,@HL      ; A ← 0FH and L ← L + 1 = 0, (HL) ← "0"
JPS    XXX        ; Skipped since an overflow occurred
JPS    YYY        ; H ← 2H, L ← 0H

YYY   XCHI   A,@HL      ; (20H) ← 0FH, A ← (20H), L ← L + 1 = 1H
      •
      •
      •
```

The 'JPS YYY' instruction is executed since a skip occurs after the XCHI instruction.

## XOR — LOGICAL EXCLUSIVE OR

XOR           dst,src

Operation:	Operand	Operation Summary	Bytes	Cycles
	A,#im	Exclusive-OR immediate data to A	2	2
	A,@HL	Exclusive-OR indirect data memory to A	1	1
	EA,RR	Exclusive-OR register pair (RR) to EA	2	2
	RRb,EA	Exclusive-OR register pair (RRb) to EA	2	2

**Description:** XOR performs a bitwise logical XOR operation between the source and destination variables and stores the result in the destination. The source contents are unaffected.

Operand	Binary Code								Operation Notation
A,#im	1	1	0	1	1	1	0	1	A ← A XOR im
	0	0	1	1	d3	d2	d1	d0	
A,@HL	0	0	1	1	1	0	1	1	A ← A XOR (HL)
EA,RR	1	1	0	1	1	1	0	0	EA ← EA XOR (RR)
	0	0	1	1	0	r2	r1	0	
RRb,EA	1	1	0	1	1	1	0	0	RRb ← RRb XOR EA
	0	0	1	1	0	r2	r1	0	

**Example:** If the extended accumulator contains 0C3H (11000011B) and register pair HL contains 55H (01010101B), the instruction

```
XOR     EA,HL
```

leaves the value 96H (10010110B) in the extended accumulator.

# 6

## OSCILLATOR CIRCUITS

### OVERVIEW

The S3P7588X microcontrollers have one oscillator circuit, the system clock circuit, ( $f_x$ ). The CPU and peripheral hardware operates on the system clock frequency supplied through this circuit. Specifically, a clock pulse is required by the following peripheral modules:

- Basic timer
- Timer/counters 0
- Watch timer
- Clock output circuit

### CLOCK CONTROL REGISTERS

The power control register, PCON, is used to select normal CPU operating mode or one of two power-down modes — stop or idle. Bits 3 and 2 of the PCON register can be manipulated by a STOP or IDLE instruction to engage stop or idle power-down mode.

The system clock frequencies can be divided by 4, 8, or 64. By manipulating PCON bits 1 and 0, you select one of the following frequencies as the selected system clock.

$$\frac{f_x}{4} , \frac{f_x}{8} , \frac{f_x}{64}$$



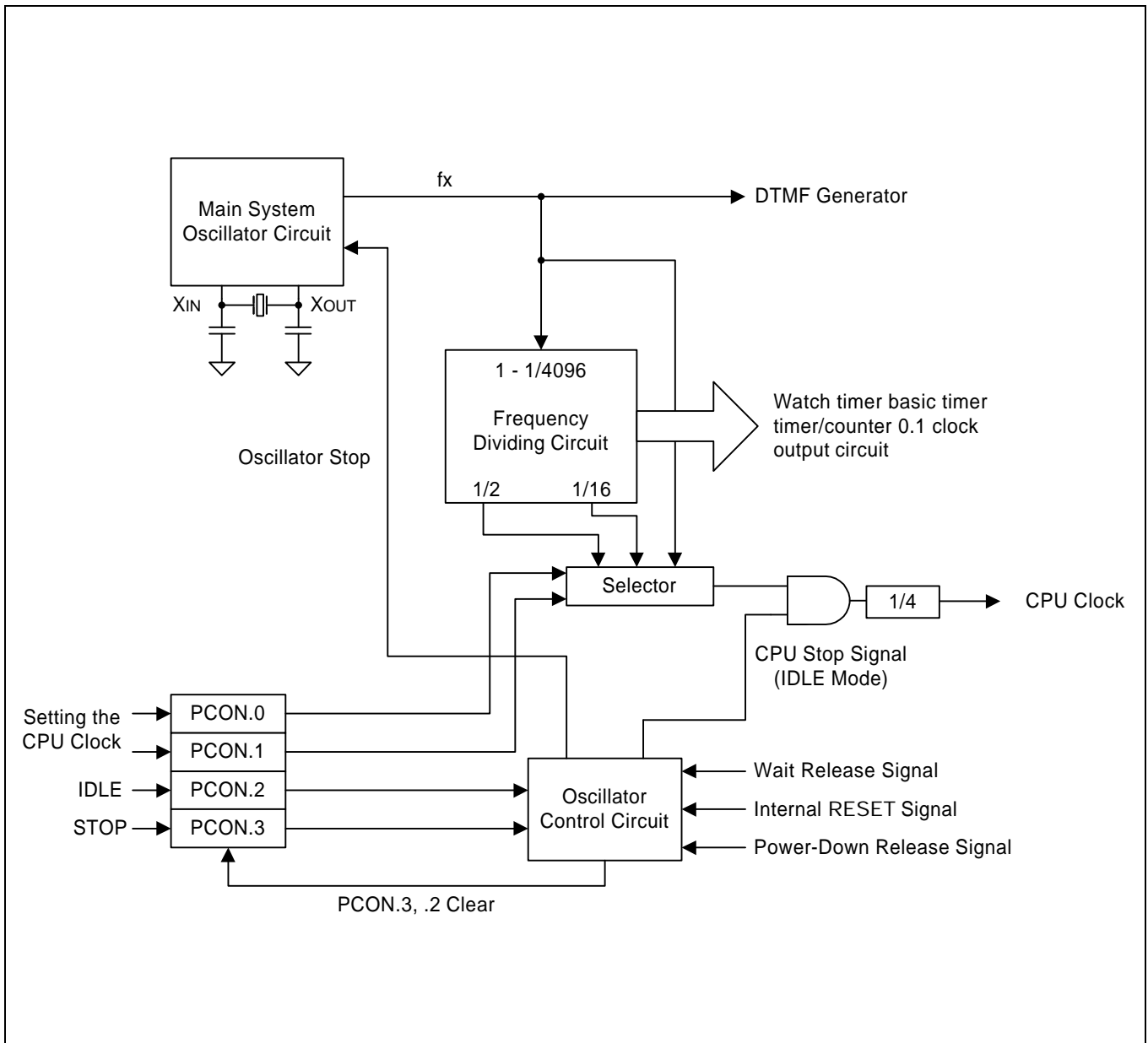


Figure 6-1. Clock Circuit Diagram

System Oscillator Circuits

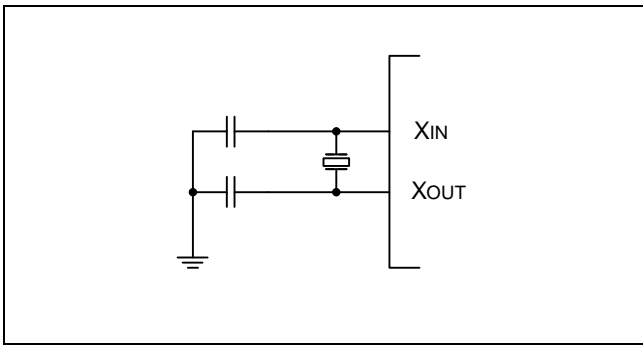


Figure 6-2. Crystal/Ceramic Oscillator

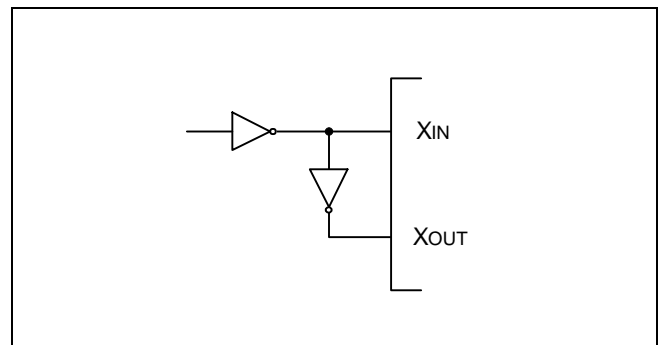
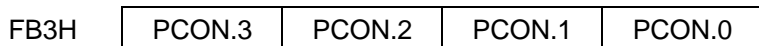


Figure 6-3. External Oscillator

**POWER CONTROL REGISTER (PCON)**

The power control register, PCON, is a 4-bit register that is used to select the CPU clock frequency and to control CPU operating and power-down modes. PCON can be addressed directly by 4-bit write instructions or indirectly by the instructions IDLE and STOP.



PCON bits 3 and 2 are addressed by the STOP and IDLE instructions, respectively, to engage the IDLE and STOP power-down modes. IDLE and STOP modes can be initiated by these instruction despite the current value of the enable memory bank flag (EMB). By manipulating bits 1 and 0 of the PCON register, the system clock frequency can be divided by 4, 8, or 64.

RESET sets PCON register values to logic zero: PCON.1 and PCON.0 divide the fx frequency by 64, and PCON.3 and PCON.2 enable normal CPU operating mode.

**Table 6-1. Power Control Register (PCON) Organization**

PCON Bit Settings		Resulting CPU Operating Mode
PCON.3	PCON.2	
0	0	Normal CPU operating mode
0	1	IDLE power-down mode
1	0	Stop power-down mode

PCON Bit Settings		Resulting CPU Clock Frequency
PCON.1	PCON.0	
0	0	fx/64
1	0	fx/8
1	1	fx/4

**PROGRAMMING TIP — Setting the CPU Clock**

To set the CPU clock to 1.05MHz at 4.19MHz:

```

BITS      EMB
SMB      15
LD        A,#3H
LD        PCON,A
    
```

### Instruction Cycle Times

The unit of time that equals one machine cycle varies depending on how the oscillator clock signal is divided (by 4, 8, or 64) by the system clock. Table 6-2 shows corresponding cycle times in microseconds.

**Table 6-2. Instruction Cycle Times for CPU Clock Rates**

Selected CPU Clock	Resulting Frequency	Oscillation Source	Cycle Time ( $\mu$ sec)
fx/64	65.5kHz	fx = 4.19MHz	15.3
fx/8	524.0kHz		1.91
fx/4	1.05MHz		0.95

### CLOCK OUTPUT MODE REGISTER (CLMOD)

The clock output mode register, CLMOD, is a 4-bit register that is used to enable or disable clock output to the CLO pin and to select the CPU clock source and frequency. CLMOD is addressable by 4-bit write instruction only.

FD0H	CLMOD.3	"0"	CLMOD.1	CLMOD.0
------	---------	-----	---------	---------

RESET clears CLMOD to logic zero, which automatically selects the CPU clock as the clock source (without initiating clock oscillation), and disable clock output.

CLMOD.3 is the enable/disable clock output control bit; CLMOD.1 and CLMOD.0 are used to select one of four possible clock sources and frequencies: normal CPU clock, fx/8, fx/16, or fx/64.

**Table 6-3. Clock Output Mode Register (CLMOD) Organization**

CLMOD Bit Setting		Resulting Clock Output	
CLMOD.1	CLMOD.0	Clock Source	Frequency
0	0	CPU Clock (fx/4, fx/8, fx/64)	1.05MHz, 524kHz, 65.5kHz
0	1	fx/8	524kHz
1	0	fx/16	262kHz
1	1	fx/64	65.5kHz

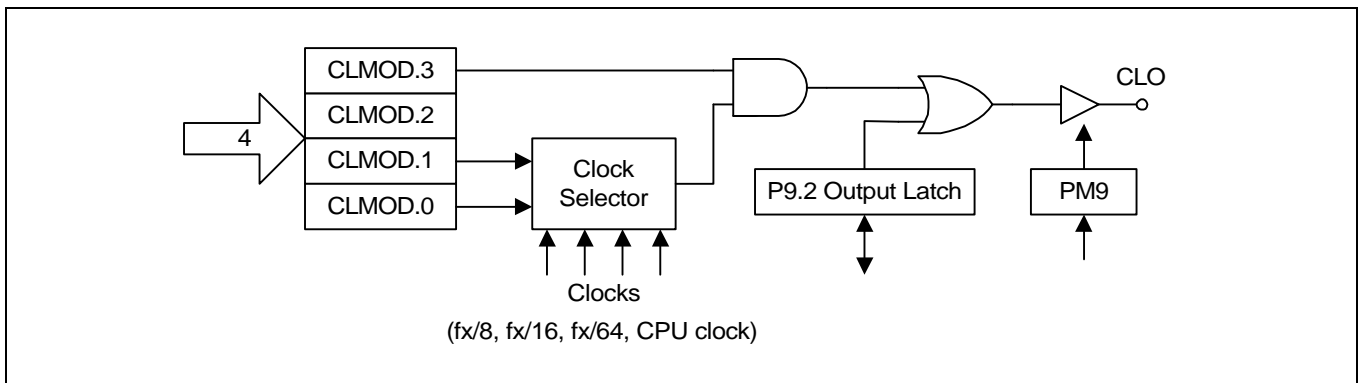
CLMOD.3	Result of CLMOD.3 setting
0	Clock output is disable
1	Clock output is enable

**NOTE:** Frequencies assume that fx = 4.19MHz.

**CLOCK OUTPUT CIRCUIT**

The clock output circuit, used to output clock pulses to the CLO pin, has the following components:

- 4-bit clock output mode register (CLMOD)
- Clock selector
- output latch
- Port mode flag
- CLO output pin (P2.2)



**Figure 6-4. CLO Output Pin Circuit Diagram**

**Clock Output Procedure**

The procedure for outputting clock pulses to the CLO pin may be summarized as follows:

- Disable clock output by clearing CLMOD.3 to logic zero.
- Set the clock output frequency (CLMOD.1, CLMOD.0).
- Load a “0” to the output latch of the CLO pin (P9.2).
- Set the P9.2 mode flag (PM 9) to output mode.
- Enable clock output by setting CLMOD.3 to logic one.

**PROGRAMMING TIP — CPU Clock Output to the CLO Pin**

To output the CPU clock to the CLO pin

```

BITS      EMB
SMB      15
LD        EA,#040H
LD        PMG4,EA      ; P 9.2 ← Output mode
BITR     P9.2          ; Clear P9.2 output latch
LD        A,#8H
LD        CLMOD,A
    
```

# 7

## CALLER ID

### OVERVIEW

The S3P7588X has a caller id receiver unit in which it has the following features.

- 1200 baud FSK (Frequency Shift Keying) demodulator with sensitivity  $-38\text{dBm}$  ( $600\Omega$ ) conforms to Bell 202 and CCITT V.23 standards
- CAS receiver with receive sensitivity of  $-32\text{dBm}$  (in  $600\Omega$ )
- Stutter Dial Tone (SDT) detector with sensitivity of  $-36\text{dBm}$
- Ring or Line Reversal detector
- On-hook and off-hook applications according to Bellcore TR-NWT-000030 and SR-TSV-002476 specifications
- Compatible with ETSI standards ETS 300 659-1 and ETS 3000 659-2

## APPLICATION

### Board Configuration for Developing Caller id Application

When the test board is designed, you must acknowledge the S3P7588X's operating modes. There are three modes for S3P7588X.

#### 1. Normal Operating Mode

This mode is a normal operating mode as it is. In this mode internal MCU and caller id are cooperate with 4 signals. These signals are viewed as ports in programmers view. That is, programmers can control the caller id as if it is connected to the external I/O ports.

These I/O ports and it's functions are as follows.

**Table 7-1. Interconnections Between Internal MCU and Caller ID**

Programmers View	Caller ID Signal	Function
P1.0	INT	Caller id interrupt request
P2.1	SCK	SCK signal for I <sup>2</sup> C interfacing with Caller ID
P2.2	SDT	SDT signal for I <sup>2</sup> C interfacing with Caller ID
P3.1 (or P8.0)	CID_RESETB	Reset signal dedicated to Caller id block Because KS57C5308 doesn't have P8 ports, P3.1 is used instead as default reset signal. But if you develop application with KS57C5208 SMDS, you can use P8.0 by setting P8.1 to logic-1 ahead. (NOTE)

**NOTE:** The caller id receiver remains in reset state during this dedicated signal is not released. that is, the caller id receiver can be released from reset state only by toggling the dedicated signal. (high → low → high)

#### 2. Caller ID Mode

In this mode, internal MCU is disabled and S3P7588X is operating as if it is a caller id chip. This mode is useful when you develop some application system with S3P7588X device. S3P7588X's functions are compatible to KS57C5208/KS57C5308 device so you can utilize the SMDS system of KS57C5208/S57C5308 without modification with S3P7588X configured as this mode.

When Caller id Mode is enabled, 7 pins are assigned as follows for caller id interfacing.

Table 7-2. Pin Assignment in Caller ID Mode

Pin Name	Caller ID Signal	Function
P9.0	INT	Caller ID interrupt request
P9.1	SCK	SCK signal for I <sup>2</sup> C interfacing with Caller ID
P9.2	SDT	SDT signal for I <sup>2</sup> C interfacing with Caller ID
P1.1	CID_RESETB	Reset signal dedicated to caller ID block
P1.2, P3.0, P3.1	–	Must be tied to ground.

**NOTE:** Other ports must be floated from the development board. That is SMDS board must feed these ports to the development board. (see Figure 7-1)

So, when you develop some application system, you first develop a S/W for your system with this mode, then download it to S3P7588X's OTP ROM and test if it runs correctly in normal operating mode.

### 3. OTP Programming Mode

In this mode, you can download your own program to internal EPROM. It is useful in that it can diminish the risk of MASK-ROM version, and is helpful for S/W development.

Refer to chapter 15 for detailed OTP programming method.

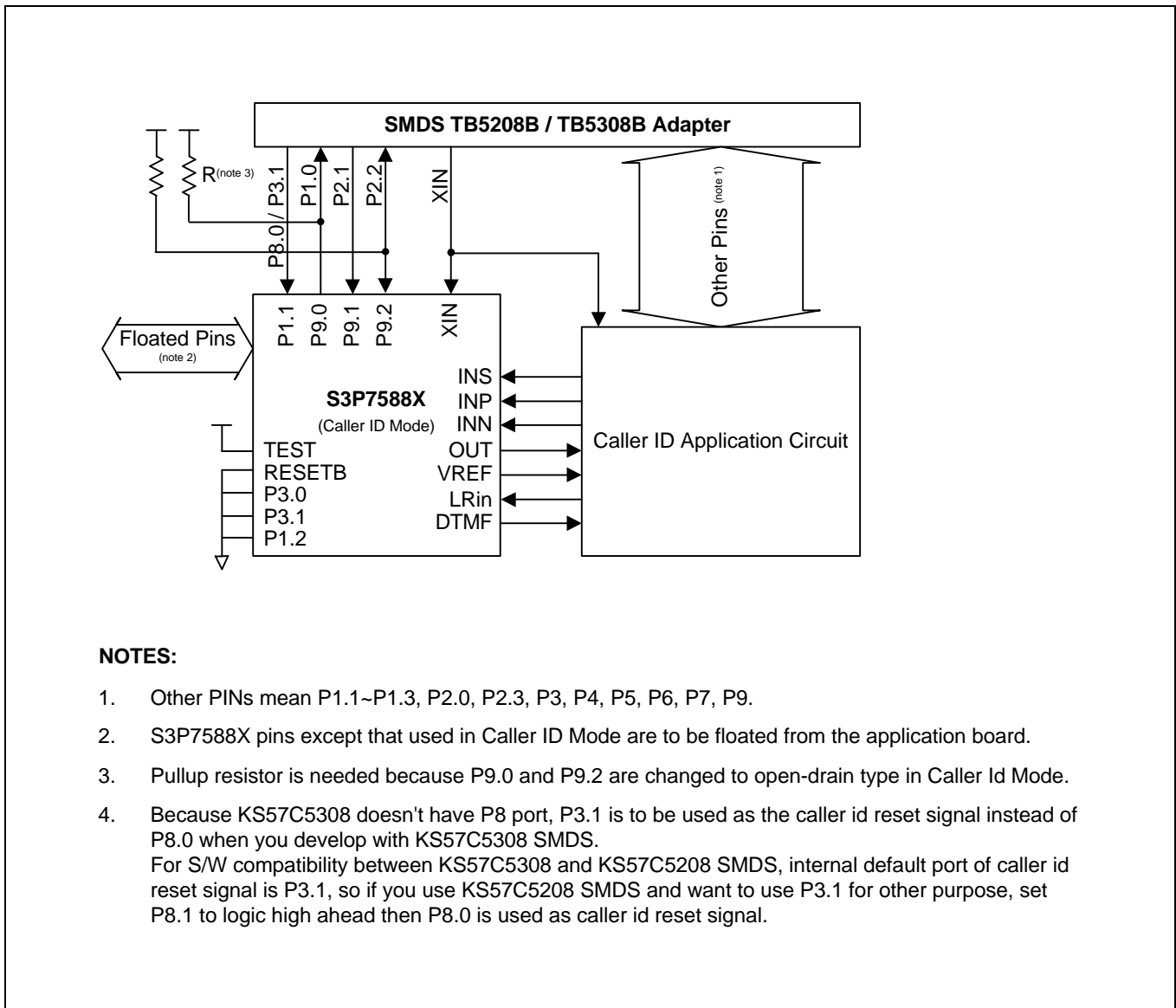
These three modes can be selected by configuring external pins. Table 7-3 represents this configurations.

Table 7-3. Pin Configurations for Selecting Operation Modes

TEST	RESETB	Operation Mode
0	1	Normal operation mode
1	0	Caller ID mode
V <sub>PP</sub> (12.5V)	0	OTP programming mode

Figure 7-1 represents overall interconnection between the development board and SMDS of KS57C5208/ KS57C5308.

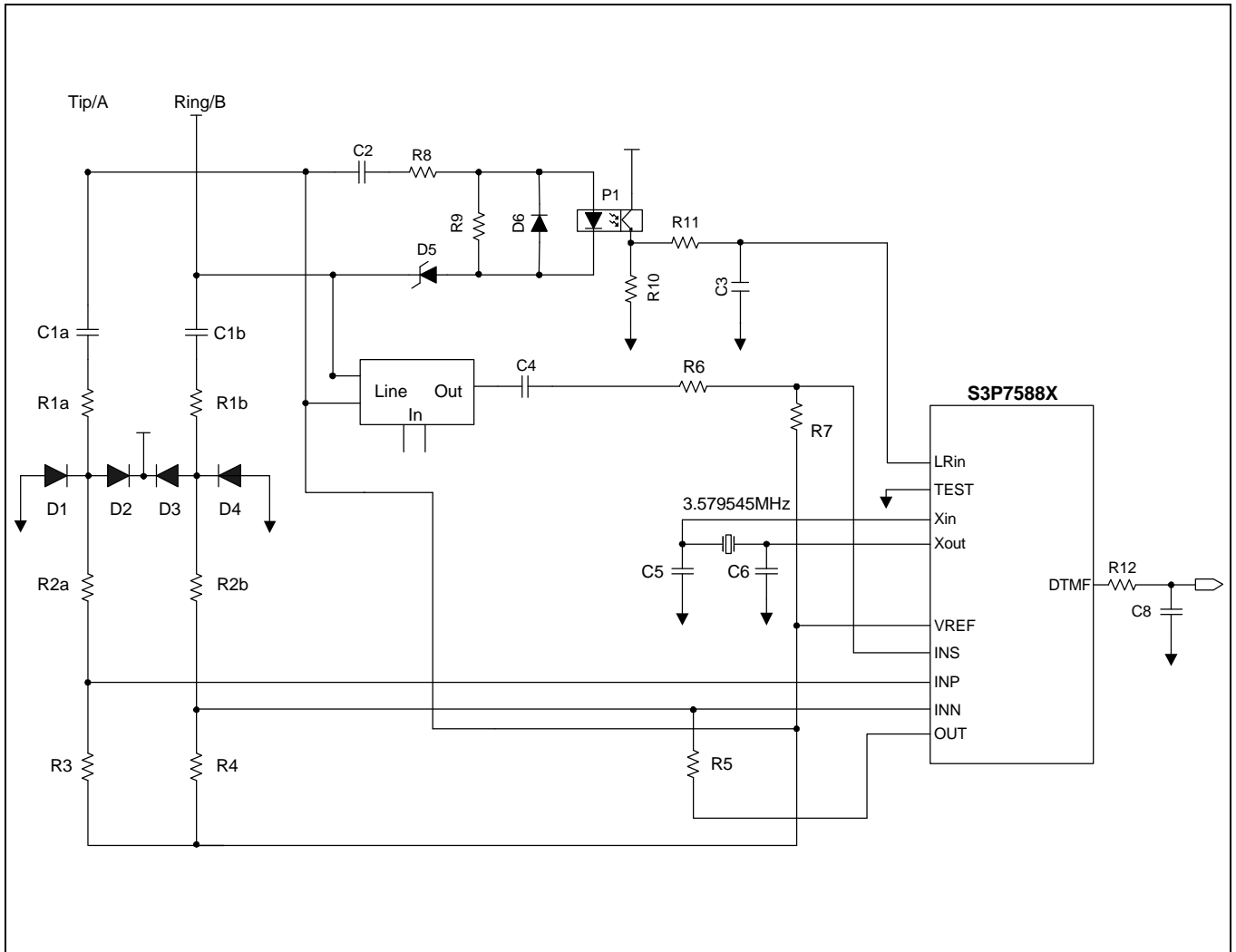




**Figure 7-1. Application Diagram for S3P7588X Development System with KS57C5208 SMDS**

**Analog Application Diagram**

All analog parts in S3P7588X are related to interfacing between caller id and telephone line. Figure 7-2 and Table 7-4 represents the recommended diagram and its component values for typical application. Note that the components specified are for a typical application. For conformance to standards in certain applications, other component values and/or ratings may be necessary.



**Figure 7-2. Recommended Diagram for Typical Application**

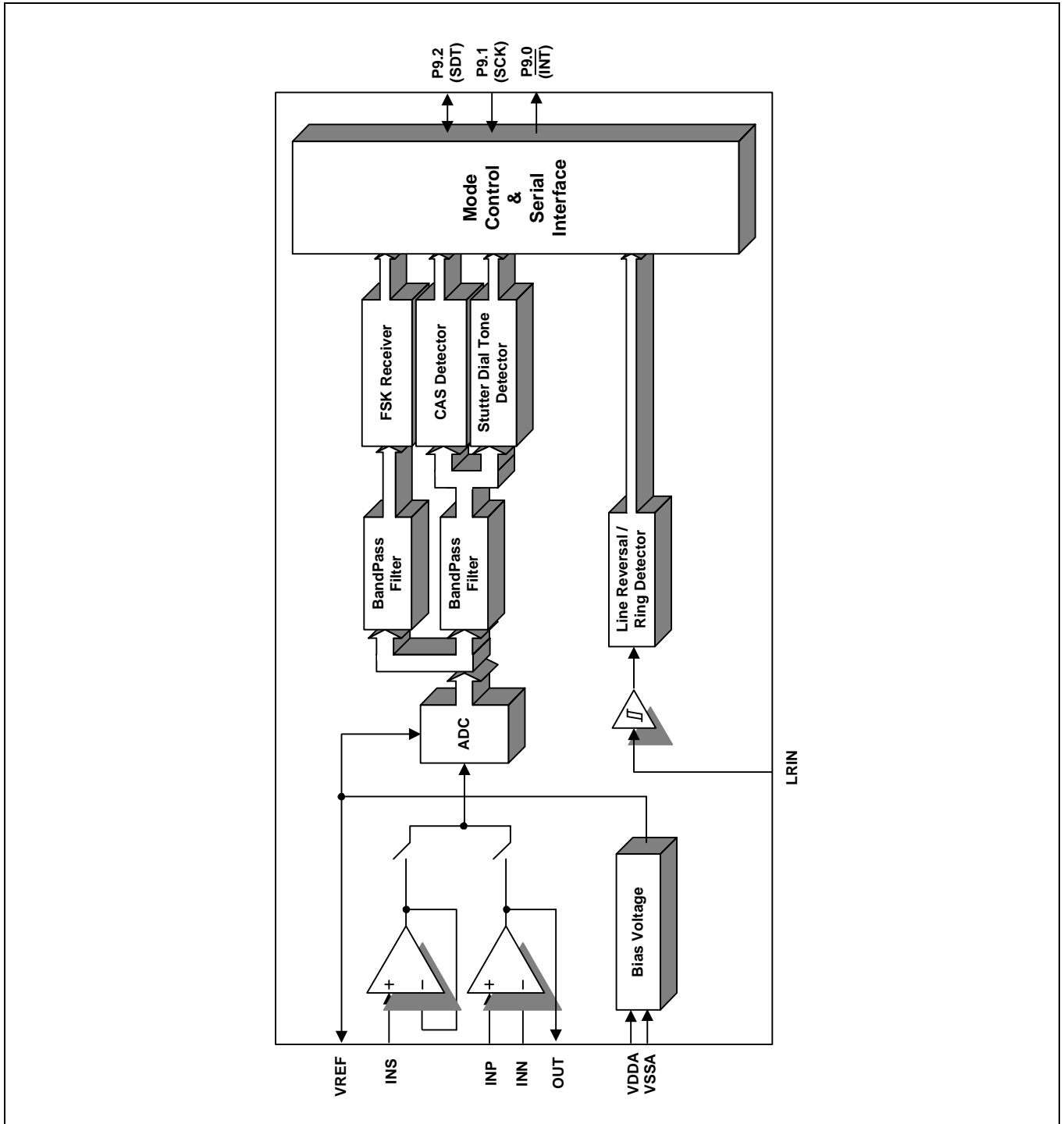
Table 7-4. Recommended External Component Values for Typical Application

Differential Input Stage		Single Ended Input Stage	
C1a, C1b	2.2nF (1KV)	C4	100nF
R1a, R1b	390k $\Omega$ (0.5W)	R6 (NOTE)	100k $\Omega$
R2a, R2b	47k $\Omega$	R7 (NOTE)	100k $\Omega$
R3	68k $\Omega$	<b>Crystal Oscillator</b>	
R4	220k $\Omega$	C5,C6	20pF
R5	100k $\Omega$	X1	3.579545MHz $\pm$ 0.1%
D1, D2, D3, D4	1N4007	O1	LM358
<b>Ring or Line Reversal Detector</b>			
C2	0.22 $\mu$ F (250V)	R10,R11	20k $\Omega$
C3	10nF	D5	24V
R8	36k $\Omega$	D6	1N4148
R9	3.9k $\Omega$	P1	PC817/LTV817
<b>DC Input Stage</b>			
<b>Tone Generator</b>			
C8	10nF	R12	2.0k $\Omega$

**NOTE:** Values for R6 and R7 are based on a hybrid that has a loss free path from LINE to OUT

**FUNCTIONAL DESCRIPTIONS OF CALLER ID BLOCK**

**FUNCTIONAL BLOCK DIAGRAM**



**Figure 7-3. Block Diagram of CID Module**

## ANALOG INPUT AND PREPROCESSOR

The preprocessor for the FSK receiver and the CAS, the SDT detectors, comprises two input signal buffers, an 14bit Analog-to-Digital Converter (ADC) and digital bandpass filters. Bandpass filters are used to attenuate out band noise and interfering signals, which might otherwise reach the FSK receiver and CAS, SDT detectors. The CAS and SDT detectors share a single digital filter while the FSK receiver has its own separate filter.

The CID block can be forced into a power-down state by switching off the 3.579545MHz system clock and ADC and op-amps.

### Differential Input Buffer

The differential input buffer is used to convert the balanced telephone line signal to the input signal of ADC in the CID block.

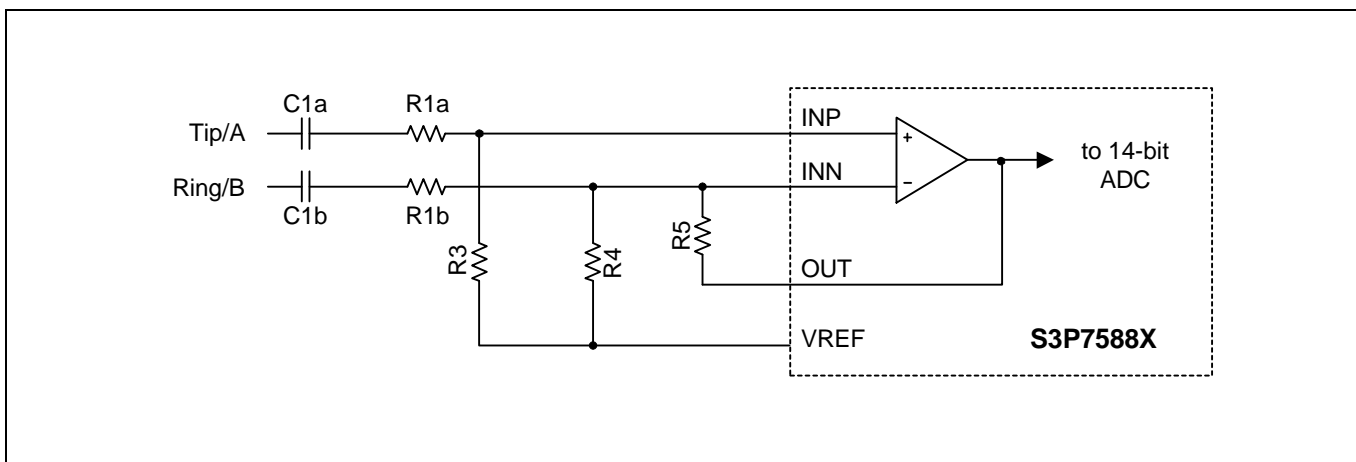


Figure 7-4. Differential Input Buffer of S3P7588X

Design equations for this buffer are

$$\begin{aligned} \text{The differential voltage gain} &= R5/R1b. \\ R1a &= R1b \\ C1a &= C1b \\ R3 &= R4 * R5 / (R4 + R5) \end{aligned}$$

The target differential voltage gain should be adjusted to obtain the expected signal level at the 'OUT' pin.

### Single Ended Input Buffer

The single ended input buffer may also be used with the telephone line signal connected to the hybrid as shown in Figure 7-5. The voltage gain is  $R7 / (R6 + R7)$ .

The target voltage gain should be adjusted to obtain the expected signal level at the INS input.

The BFS (Buffer selection) bit in the Function register chooses between the output of the single-ended input buffer and the output of the differential input buffer, sending the selected output to the ADC. The differential input buffer is selected when BFS is '0' and the single ended input buffer is selected when BFS is '1'. The default value of BFS is '0'.

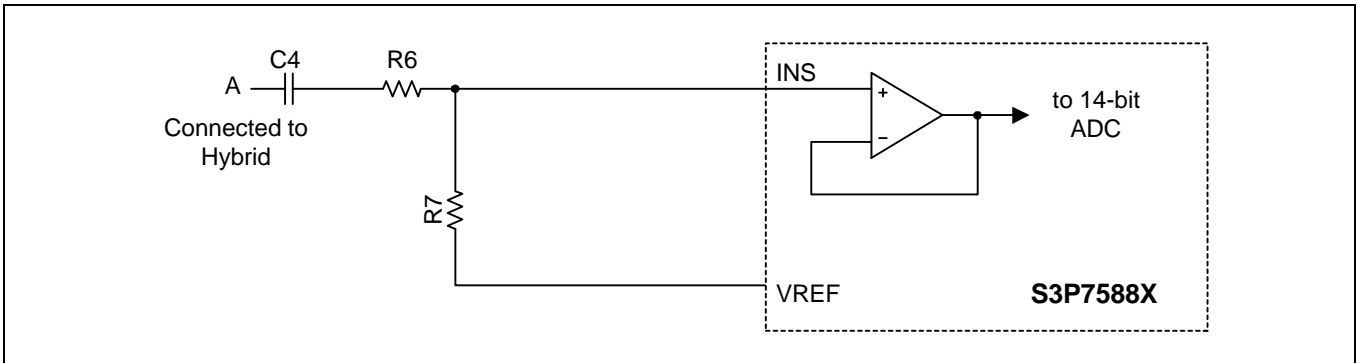


Figure 7-5. Single Ended Buffer of S3P7588X

**CAS TONE DETECTION**

The CAS detection block is capable of detecting the CAS signals during speech with high talk-down and talk-off performance without the use of a hybrid, and 100% Bellcore compliant performance with the use of a hybrid. If the CAS detection is enabled the Caller id block will generate an interrupt (Interrupt register, bit 1 is set) when a correct dual tone (2130 and 2750Hz) is detected. CAS detection is enabled when the CASenable bit in the Function register is set and the FSK and SDT enable bits in the Function register are cleared. The parameters of the CAS Detector are shown in Table 7-5.

**Table 7-5. CAS Detector Parameters**

Parameter	Value
Low tone frequency	2130Hz ± 0.5%
High tone frequency	2750Hz ± 0.5%
Accepted signal level	-5.2dBm to -32dBm
Twist	-6dB to +6dB

When a valid CAS signal is detected, the CASdetect status bit of the Status register and the CASint bit of the interrupt register are set and an interrupt is generated. When the signal level is below the accepted signal level the status bit of the status register is cleared and the CASint interrupt bit is set, generating another interrupt. The CASint interrupt bit is reset when the interrupt register is read (see Figure 7-6).

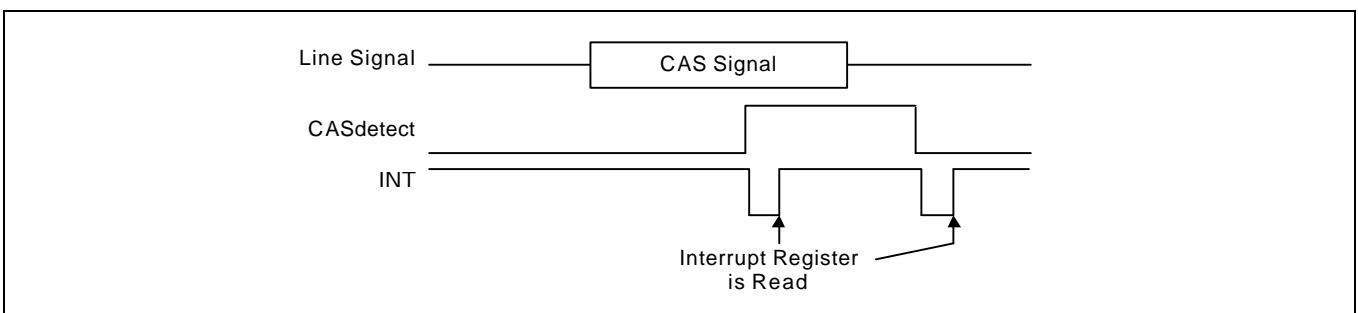


Figure 7-6. CASdetect, CASint and INT Related to the CAS Tone

In order to accurately detect the end of a CAS tone, it is recommended to mute the near end speech immediately after the CAS tone has been detected.

## FSK RECEPTION

### FSK Data Reception Sequence

The on-chip FSK Receiver satisfies all target specifications of Bellcore. The FSK receiver function can be enabled by setting the FSKenable bit (Function register, bit2) and clearing the CASenable (Function register, bit1) and the SDTenable (Function register, bit5) bits.

When the FSK Receiver is enabled, the CID block continuously checks for a signal in the FSK band (~1200 - ~2200Hz) above the minimum signal level threshold. An FSK data word consists of one start bit (space) followed by eight data bits and one stop bit (mark). After the FSK receiver has detected a start bit it starts receiving the data bits (LSB first). After the 8<sup>th</sup> data bit the FSKint interrupt bit (Interrupt register, bit2) is set and an interrupt is generated.

The FSKint interrupt bit is cleared when the Interrupt register is read. The interrupt register and the FSKDT register should be read every time an interrupt occurs.

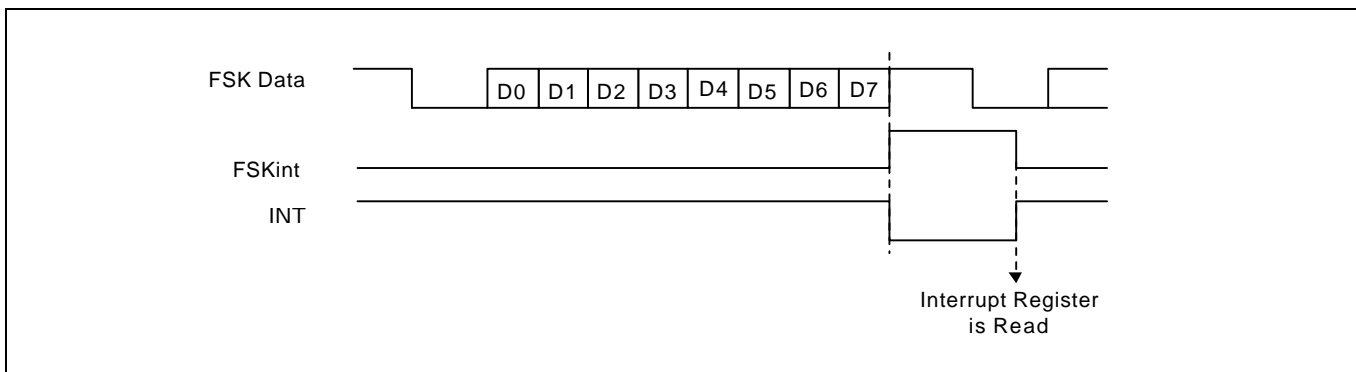


Figure 7-7. Sequence to Receive an FSK Data Byte

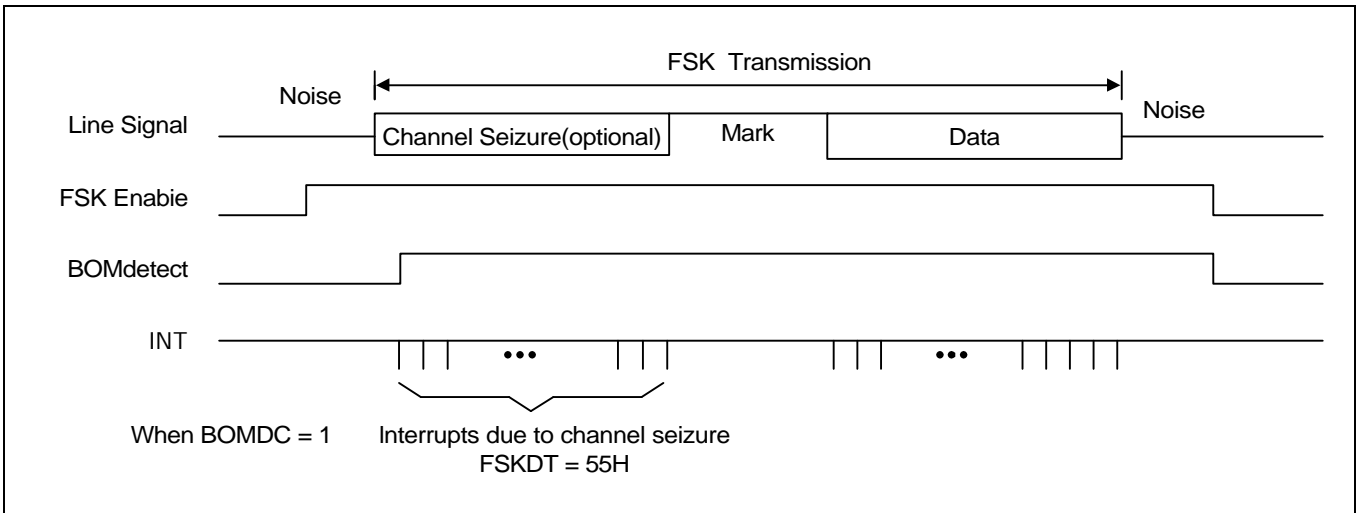
Table 7-6. FSK Receiver Parameters

Parameter	Bellcore	CCITT / V23
Mark frequency (logic 1)	1200Hz $\pm$ 1%	1300Hz $\pm$ 1.5%
Space frequency (logic 0)	2200Hz $\pm$ 1%	2100Hz $\pm$ 1.5%
Maximum allowed signal level	0dBm	-8dBV
Minimum signal level threshold	< -38dBm	< -40dBV
Twist	-10dB to +10dB	-6dB to +6dB
Accepted S/N (0Hz – 200Hz)	< -20dB	< -20dB
Accepted S/N (200Hz – 3200Hz)	< 6dB	< 6dB
Accepted S/N (3200Hz – 15000Hz)	< -20dB	< -20dB
Transmission rate	1200 bits per second $\pm$ 1%	1200 bits per second $\pm$ 1%

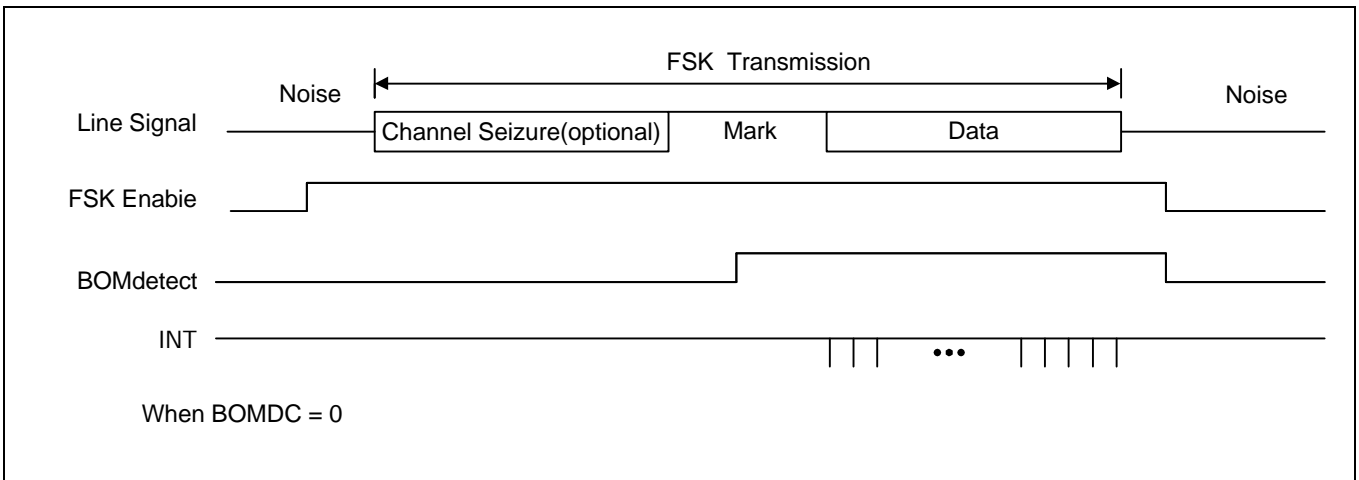
**Begin of Mark (BOM) Detection**

BOMDC bit of MODE register (MODE register, bit 6) is utilized for detecting begin of mark or channel seizure. If BOMDC is set to '0', the BOMdetect signal (INTR register, bit 6) will be set after the begin of mark has been detected, and if BOMDC is '1', BOMdetect will be set after the channel seizure detected.

When BOMDC is '1' and BOMdetect is set, the interrupts occur due to channel seizure and the value of FSKDT will be 55H as shown in Figure 7-8. If BOMDC is '0', interrupt will therefore not be generated during the channel seizure and during the block of marks as shown in Figure 7-9. The FSK interrupts of data bytes will be generated after a mark period of at least 16 sequential 1's has been detected. Behavior of BOMdetect (STAT register, bit 4) is shown in Figure 7-8 and 7-9. This bit will be cleared when the FSK receiver is disabled or a signal drop out occurs for more than 18.3ms. In the latter case the FSK receiver will behave as if it has just been disabled.



**Figure 7-8. Interrupt Behavior of the FSK Receiver with BOMDC = 1**



**Figure 7-9. Interrupt Behavior of the FSK Receiver with BOMDC = 0**

During FSK data reception, no new interrupts will occur after a signal dropout when BOMDC = '0'. If it is necessary to receive as much data as possible (even with a part missing) then the BOMDC can be set to '1' when reception of data starts.



### STUTTER DIAL TONE (SDT) DETECTOR

This block is enabled when the S3P7588X is set to SDT enable mode (Function register, bit5) and all the other functions in the Function register are disabled.

The detector measures the total signal level for every 31.5ms. When the total signal level is above -36dBm in the 350Hz to 440Hz dial tone band, the SDTdetect bit in the Status register is set. When the total signal level is below -36dBm the SDTdetect bit is cleared (see Table 7-7). Each time SDTdetect changes the SDTint bit is set and an interrupt is generated. The SDTint bit is cleared when the Interrupt register is read. This behavior is shown in Figure 7-10.

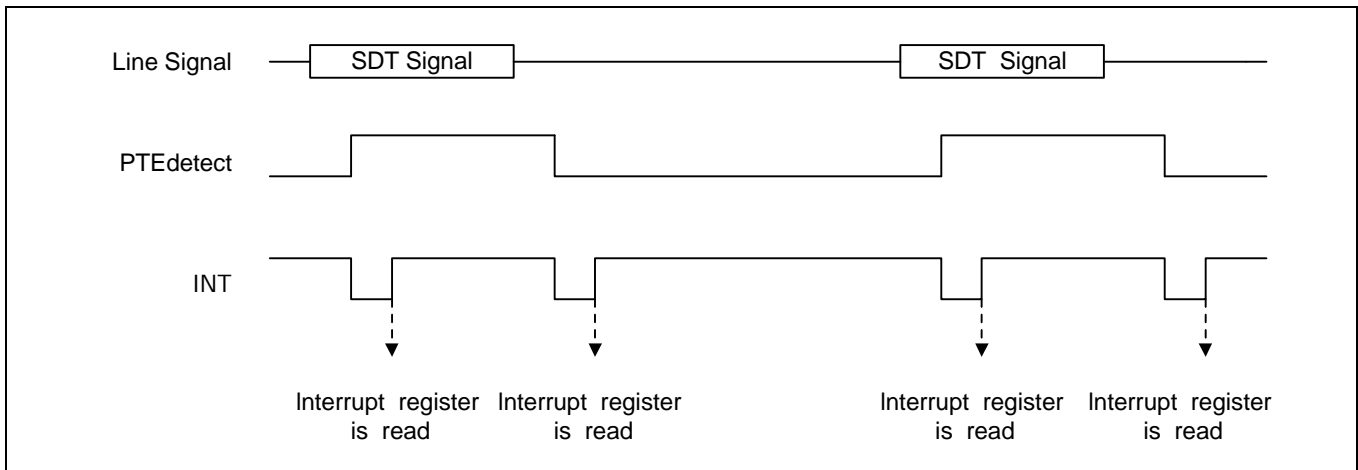


Figure 7-10. SDT Detector Operation

Table 7-7. Stutter Dial Tone Parameters

Parameters	Values
Frequencies	330Hz to 440Hz
Signal amplitude power	-10dBm to -36dBm
Duration	80 to 160ms on/off, with a duty cycle from 40% to 60%

### Ring or Line Reversal Detector

For ring or line reversal detection, some external components are needed to generate a pulse each time a ring or line reversal occurs, as shown in Figure 7-11. Interrupt generation of the ring or line reversal detector is controlled by the LRenable bit in the Function register. When LRenable is set to '1', the LRint bit of the interrupt register will be set and interrupts will be generated at every transition of the LRstatus bit. When LRenable is '0', interrupts will not be generated.

The LRstatus bit (reset value is high) in the Status register is cleared to '0' when LRin is high. If no positive edges of LRin are detected in Tguard time the LRstatus bit is set to '1'. The LRint bit is cleared when the Interrupt register is read.

If an LRint interrupt has been generated in power-down mode, it is recommended to disable power-down mode to be able to count the guard time counter using the main clock (XIN). The guard time counter is reset at the positive edge of LRin. The guard time (Tguard) can be programmed by writing the GTIME register as follows.

$$T_{guard} = 183\mu s * ( GTIME[6:0] * 4 + 3 )$$

(Ex.  $T_{guard} = 44,469ms = 0.153ms * (0111100B * 4 + 3) )$ )

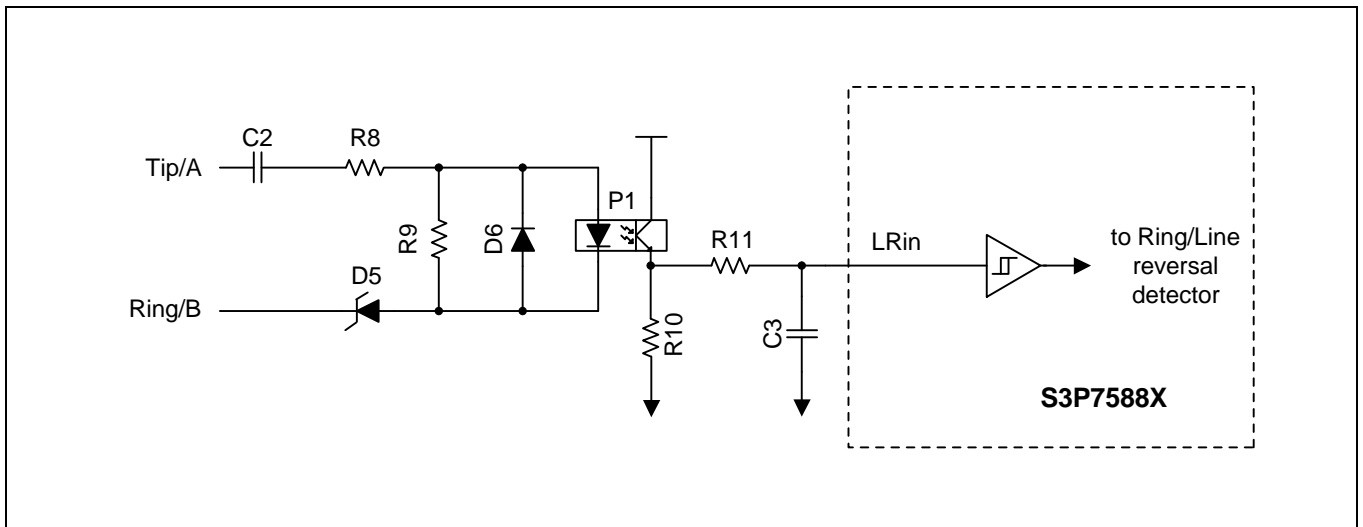


Figure 7-11. External Component to Generate LRIin

The following Figures are shown for the behavior of line reversal and ring detection respectively.

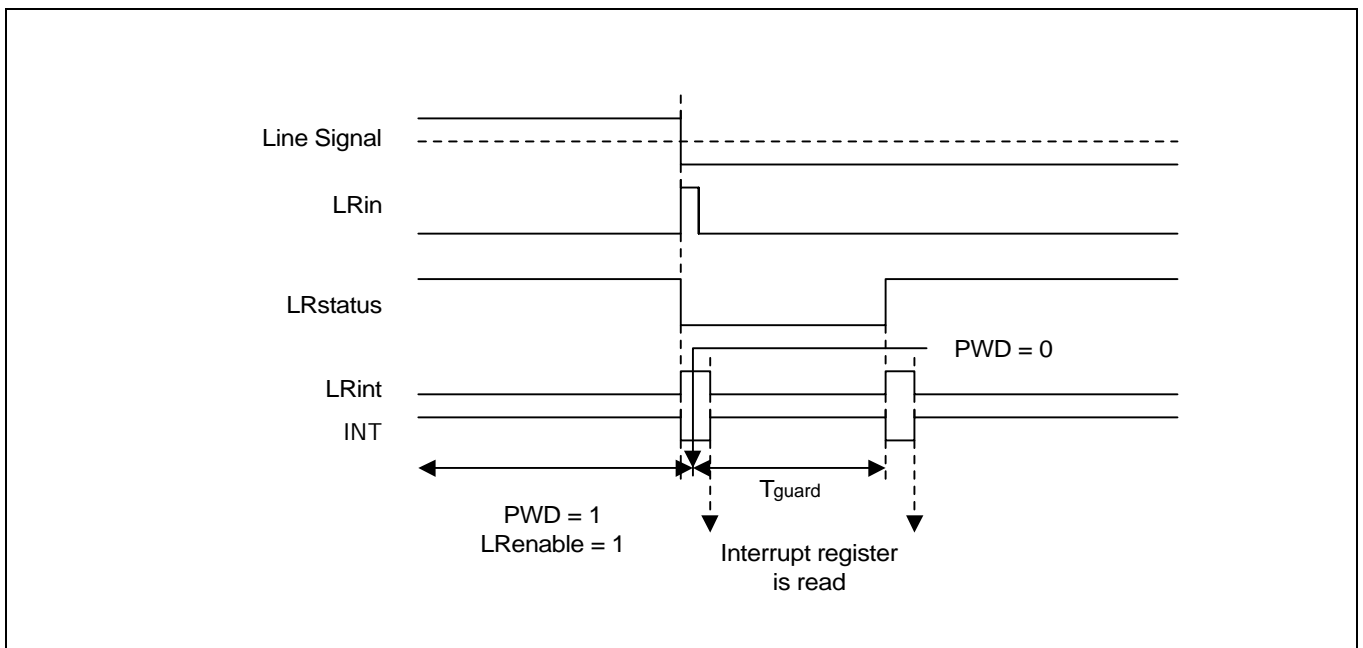


Figure 7-12. Behavior of Signals on a Line Reversal

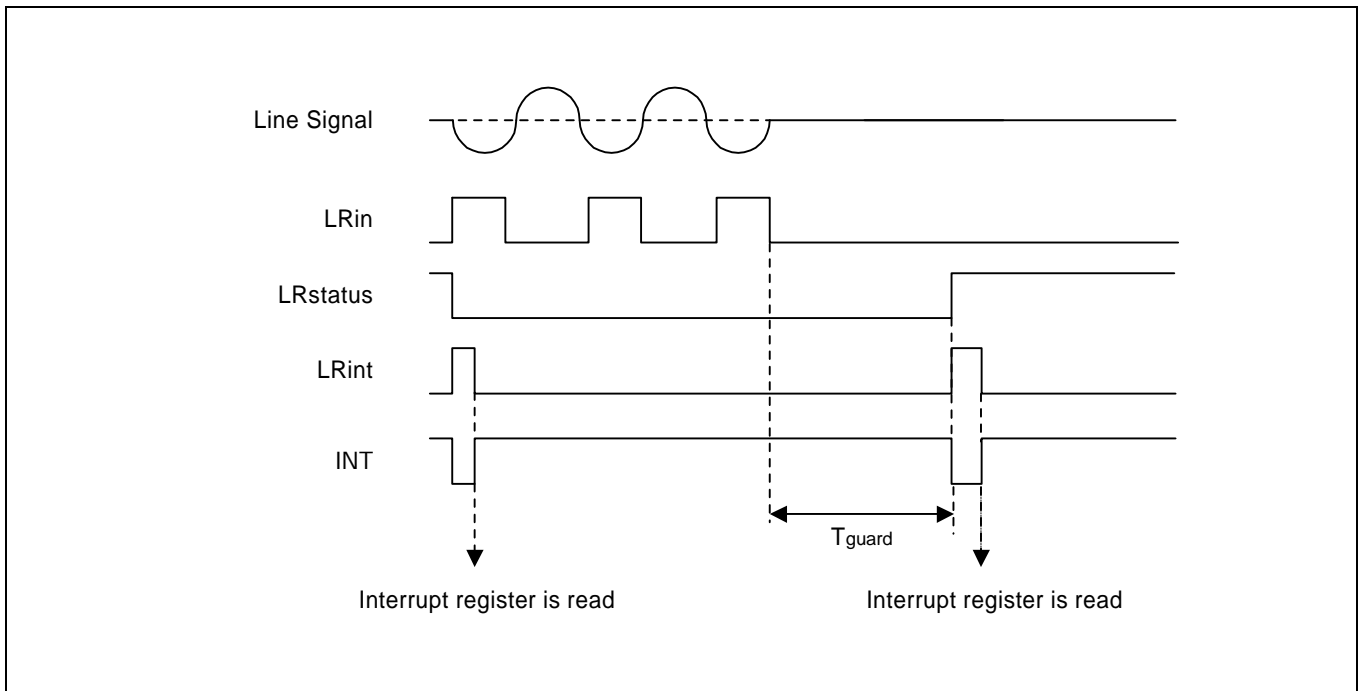


Figure 7-13. Behavior of Signals During Ring

**DTMF Generator**

The DTMF generator is able to generate 16 standard dual tones (see Table 7-8). These tones can be programmed by writing the DTMF register via the serial interface or directly writing DTMR(FD2H), DTGR(FD4H) register with 'ld' instruction. That is, there are two method to control DTMF generator, one is to use caller id register and the other is to use memory mapped register of DTMF generator. There are only caller id registers described in this chapter. Please refer to chapter 13 to know about the memory mapped registers.

It is recommended to use caller id's register because you'd better to use S3P7588X's DTMF output rather than KS57C5208/KS57C5308 SMDS's DTMF output. When you develop your own application system by setting S3P7588X as Caller id Mode, all registers except that of caller id are unable to be used, so if you use memory mapped register of DTMF generator, SMDS's DTMF generator is activated instead of S3P7588X's one.

The DTMF generator is enabled when the DTMFenable bit (Function register, bit3) is set to '1'. When the DTMFT.7 (On/Off) bit is programmed to '0', no tone will be generated; when it is programmed to '1', the tone specified in DTMFT.3 to DTMFT.0 will be generated. The code for each dual tone is shown in Table 7-8.

The DTMFG register can control the output gain of DTMF signal. The default power of the DTMF signal is -7.5 dBm for high tone and -9.5dBm for low tone. The DTMFG register contains the gain factor that is multiplied to the default signal power to obtain the DTMF signal power. The gain factor is an unsigned number. The most significant bit (M) of the DTMFG register is the mantissa and the remaining bits (E6 to E0) denote the exponent. The output power of the DTMF signal can be obtained by the following equation.

$$\text{DTMF signal power} = \text{Default signal power} * \text{DTMFG}$$

<b>Symbol</b>	7	6	5	4	3	2	1	0
<b>DTMFG</b>	M	E6	E5	E4	E3	E2	E1	E0

The DTMFG register can be programmed within the range from 0.0000001B (0.0078 in decimal) to 1.1000111B (1.5546 in decimal). For example, if DTMFG is set to 80H (1.0000000 in binary or 1.0 in decimal) the DTMF signal power will be the same as the default power. If the DTMFG register is 0.1100110H (0.7969 in decimal) the DTMF signal power will be 1.97dB lower than the default power as follows.

$$20\log(\text{default power} \times 0.7969 - \text{default power}) = 20\log 0.7969 = -1.97\text{dB}$$

The high tone power = -9.47dB  
The low Tone power = -11.47dB

When you want to use memory mapped register of DTMF generator (DTMR, DTGR), you should write zero to DTMFT register and 80H to DTMFG register ahead.

**Table 7-8. DTMF Frequencies Code Table**

D3	D2	D1	D0	Character	Low Frequency	High Frequency
0	0	0	1	1	697.0Hz	1209Hz
0	0	1	0	2	697.0Hz	1336Hz
0	0	1	1	3	697.0Hz	1477Hz
0	1	0	0	4	770.0Hz	1209Hz
0	1	0	1	5	770.0Hz	1336Hz
0	1	1	0	6	770.0Hz	1477Hz
0	1	1	1	7	852.0Hz	1209Hz
1	0	0	0	8	852.0Hz	1336Hz
1	0	0	1	9	852.0Hz	1477Hz
1	0	1	0	0	941.0Hz	1336Hz
1	0	1	1	*	941.0Hz	1209Hz
1	1	0	0	#	941.0Hz	1477Hz
1	1	0	1	A	697.0Hz	1633Hz
1	1	1	0	B	770.0Hz	1633Hz
1	1	1	1	C	852.0Hz	1633Hz
0	0	0	0	D	941.0Hz	1633Hz

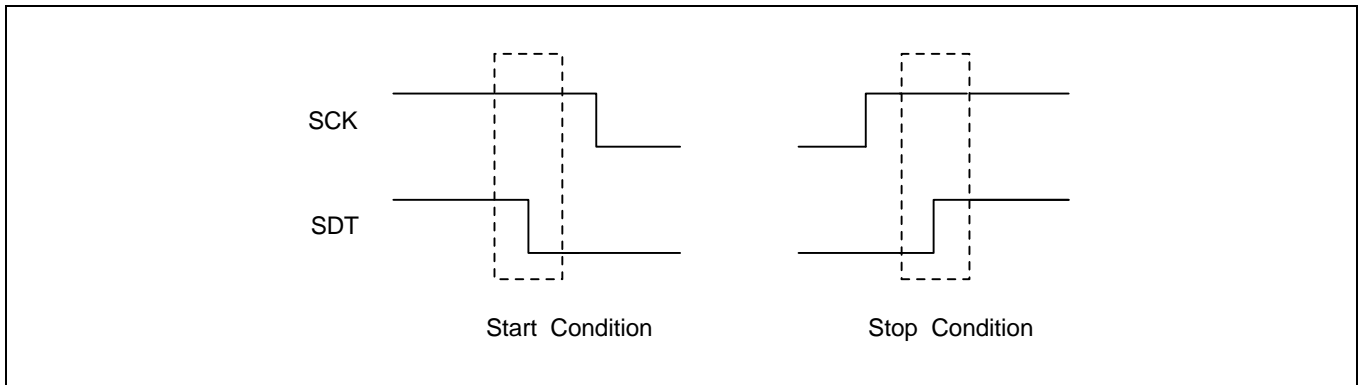
### Serial Interface

The data interface between Caller id and MCU block is a serial interface. This interface is processed through the internal P2.1 (SCK) and P2.2 (SDT) signal. The SCK is a transmission clock and the SDT transmits bi-directional data. The MCU always initiates a transmission and generates the transmission clock on the SCK line.

**Start and Stop Conditions**

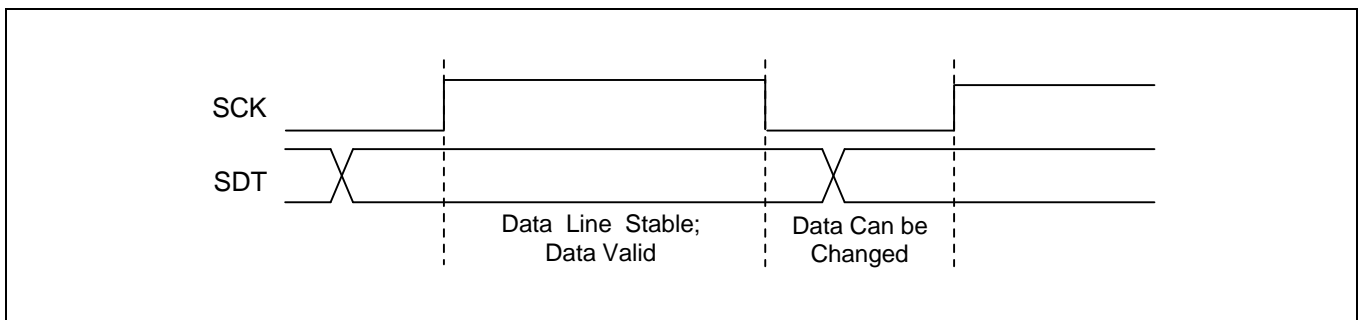
The SDT and SCK lines remain high when the bus is not busy. A high-to-low transition of the SDT line while the SCK is high is defined as the start condition. A low-to-high transition of the SDT while the SCK is high is defined as a stop condition.

When a start condition occurs between a normal start condition and a stop condition, this is called a repeated start condition.



**Figure 7-14. Start and Stop Conditions**

**BIT TRANSFER**



**Figure 7-15. Bit Transfer Timing**

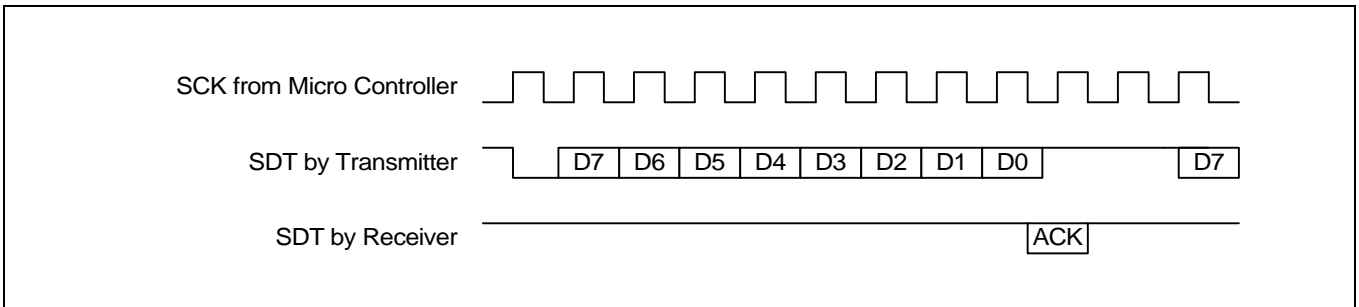
### Byte Transfer and Acknowledge

The number of data bytes transferred between the start and the stop conditions from the transmitter to the receiver is unlimited. Each byte of eight bits is followed by an acknowledge bit. The acknowledge bit is a high level signal put on the bus by the transmitter during which time the micro-controller generates an extra acknowledge-related clock pulse.

The Caller id block must generate an acknowledge bit after the reception of address field data or a register start address. Also the micro-controller must generate an acknowledge bit after the reception of each byte that has been clocked out of the Caller id block.

The device that acknowledges must pull down the SDT line during the acknowledge clock period immediately after the 8<sup>th</sup> SCK pulse, so that the SDT line is stable low during the high period of the acknowledge-related SCK pulse.

The micro-controller must signal an end-of-data to the Caller id block by not generating an acknowledge on the last byte that has been clocked out of the Caller id block. In this event the Caller id block must leave the SDT line high to enable the micro-controller to generate a stop condition.



**Figure 7-16. Byte Transmission and Acknowledge**

### Address Field

Before any data is transmitted on the SDT line, the Caller id block, which should respond, is addressed first. The addressing is always carried out with the first byte (Address field) transmitted after the start procedure. The interface address is reserved for the Caller id block, 30H for write and 31H for read.

When the address matches the address of the Caller id block, the acknowledge is given; when it does not match, no acknowledge is given.

The address field is built of two parts as follows

- Interface Address (A6 to A0)
- Read/Write control (R/W)

**Table 7-9. Bit Specification of the Address Field**

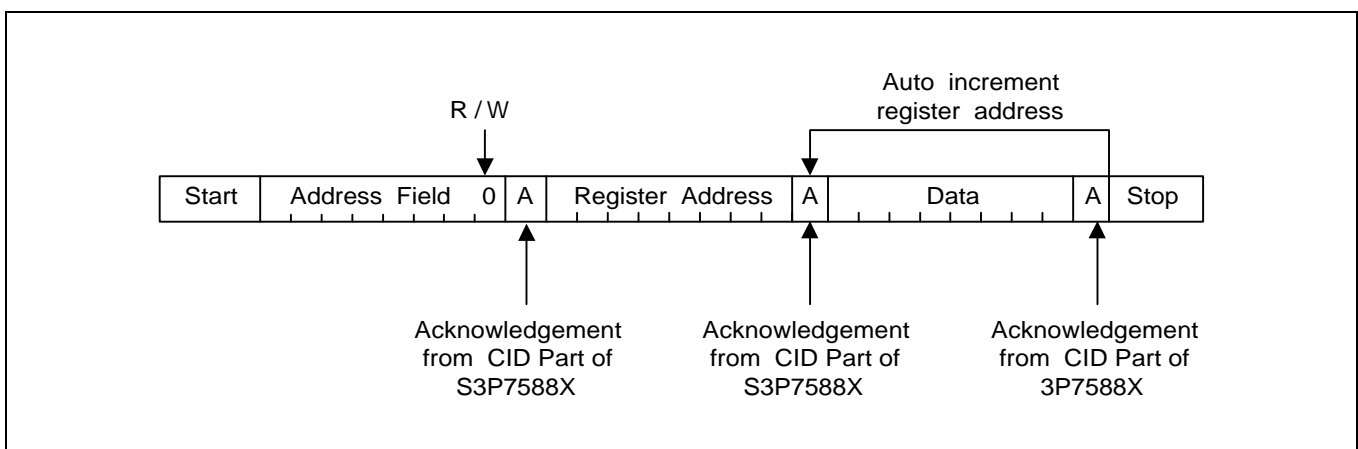
A6	A5	A4	A3	A2	A1	A0	R/W
0	0	1	1	0	0	0	1/0

### Register Address

The register address is the second byte transmitted by the micro-controller. This address is stored in the CID block and used for the following read and write actions. When multiple bytes are accessed, the first byte is written to the specified register address and the register address of the CID block is auto-incremented on each acknowledge.

### Serial Communication Protocol

The serial communication protocol is shown in Figure 7-17 and Figure 7-18. The micro-controller can initiate two kinds of sequence, the write sequence and the read sequence. Both sequences are initiated with a start condition that is followed by the Caller id block address with the read/write control bit cleared. The first byte after the Caller id block address is interpreted as the address of a Caller id block register. During the write sequence the register address of the Caller id block is increased automatically on each acknowledge. The write sequence is ended with the stop condition from the micro-controller.



**Figure 7-17. Write Sequence of the Serial Interface**

For the read sequence, after a register address of the Caller id block, a repeated start condition is generated by the micro-controller which is followed by the Caller id block address with the read/write control bit set. The data is read from the previously set register address. When the micro-controller responds with an acknowledge the address of the register is auto incremented and the Caller id block will put the data from the next register on the SDT line. When the micro-controller stops giving an acknowledge the Caller id block will stop transmitting data and the micro-controller will generate a stop condition.

When the read sequence is initiated with a start condition that is followed by the Caller id block address with the read/write control bit set, the data is read from the last set register address. (See Figure 7-18)

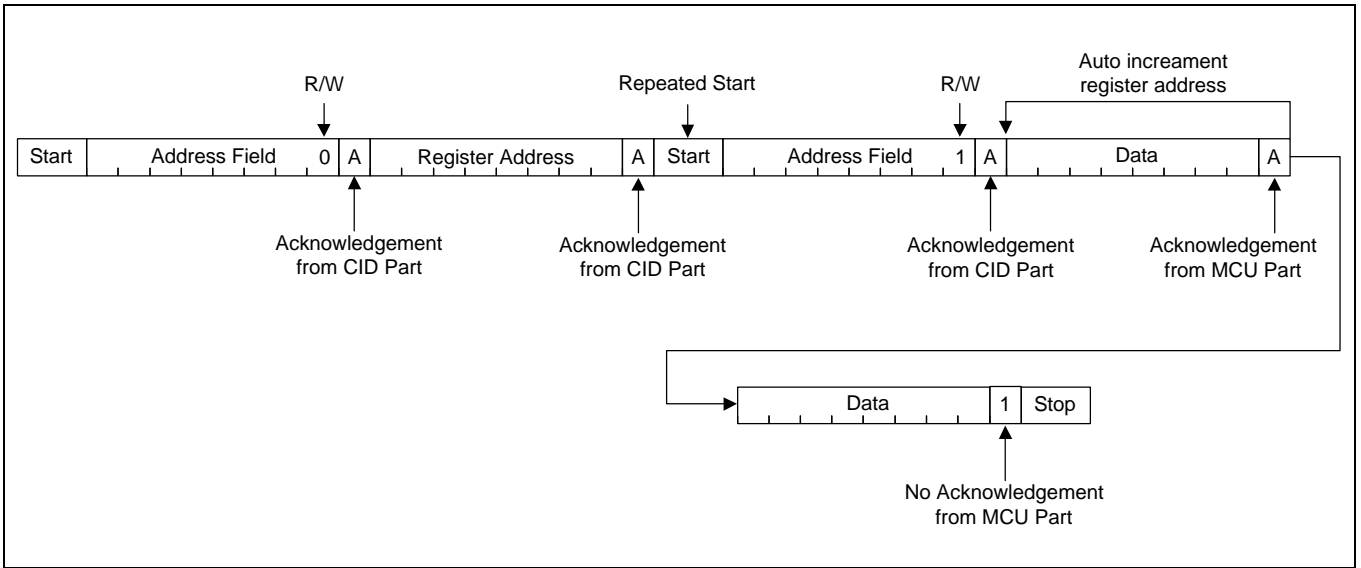


Figure 7-18. (a) Read Sequence of the Serial Interface when new Register Start Address is Programmed

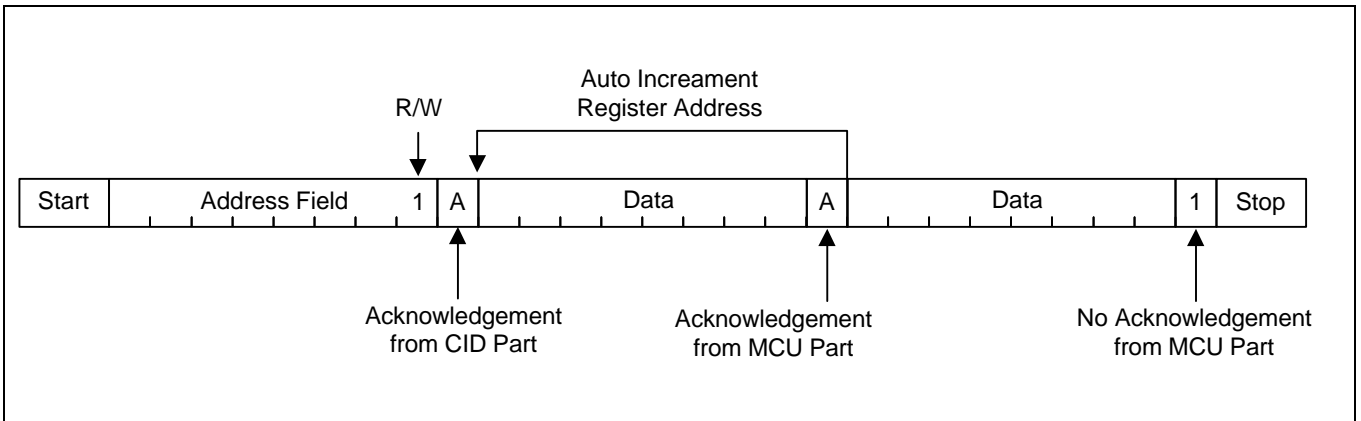


Figure 7-18. (b) Read Sequence of the Serial Interface when no Register Start Address is Programmed

**Power-Down Mode**

The Caller id block can be put in power-down mode by programming the PDW bit in the Mode register to '1'. In this mode the input signal buffers, ADC, the reference bias generator and the internal clock are switched off. However the Ring/Line Reversal detection can be active by programming the LRenable bit in the function register to be set. The serial interface can always be accessed, even in power-down mode. In power-down mode, if ring or line reversal occurs when LRenable bit is '1', the LRint bit is set and an interrupt is generated. When the Caller id block is put in power-down mode, all interrupt bits in the interrupt register cannot be set except for the LRint bit.



**Interrupt**

The interrupt signal of caller id is active low. So it must be programmed that INT0 interrupt is falling edge detection mode. The flag in the interrupt register of caller id indicates the interrupt cause. Interrupt flags are set by hardware but must be reset by software. All flags of the interrupt register are reset when the register is read via the serial interface.

The Table 7-10 shows interrupt sources of the CID block.

**Table 7-10. Interrupt Sources of the CID Block**

<b>Source Block</b>	<b>Generation</b>
Ring / line reversal detector	When LRstatus changes
FSK receiver	Reception of a new FSK data byte
CAS detector	When CASdetect changes
SDT detector	When SDTdetect changes

## REGISTER MAPS OF CALLER ID BLOCK

The registers that are available in the caller id block are shown in the following tables.

**Table 7-11. Register Overview**

Register Name	Address	Function	Default Value	Read / Write
MODE	00H	Mode register	0000 0000	Read / Write
FUNC	01H	Function register	0000 0000	Read / Write
DTMFT	02H	DTMF tone select register	0000 0000	Read / Write
GTIME	0AH	Guard time register	0000 0000	Read / Write
INTR	80H	Interrupt register	0000 0000	Read Only
STAT	81H	Status register	0000 0100	Read Only
FSKDT	82H	FSK data register	0000 0000	Read Only
DTMFG	F0H	DTMF output gain control register	0000 0000	Read / Write
CONT1	F1H	Special control register 1	0000 0000	Read / Write
CONT2	F5H	Special control register 2	0000 0000	Read / Write

**MODE REGISTER (MODE)**

Address 00H; read / write.

<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
PDW	BOMDC	-	-	-	-	-	-

Description of MODE bits

<b>Bit</b>	<b>Symbol</b>	<b>Description</b>
MODE.7	PWD	1: Puts the CID part of CID block in power-down mode 0: Puts the CID part of CID block in active mode
MODE.6	BOMDC	0: Forbids FSK interrupts until BOMDC is '1' 1: Allows FSK interrupts before BOMDC is '0'

**FUNCTION REGISTER (FUNC)**

Address 01H; read / write.

<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
BFS	-	SDTenable	-	DTMFenable	FSKenable	CASenable	LRenable

Description of FUNC bits

<b>Bit</b>	<b>Symbol</b>	<b>Description</b>
FUNC.7	BFS	1: Selects the single-ended input buffer 0: Selects the differential input buffer
FUNC.5	SDTenable	1: Enables the SDT detector 0: Disables the SDT detector
FUNC.3	DTMFenable	1: Enables the DTMF generator 0: Disables the DTMF generator
FUNC.2	FSKenable	1: Enables FSK receiver 0: Disables FSK receiver
FUNC.1	CASenable	1: Enables CAS detector 0: Disables CAS detector
FUNC.0	LRenable	1: Enables LR interrupts 0: Disables LR interrupts

**DTMF TONE SELECT REGISTER (DTMFT)**

Address 02H; read / write.

<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
ON-OFF	-	-	-	T3	T2	T1	T0

Description of DTFMT bits

Bit	Symbol	Description
DTMFT.7	ON-OFF	1: Enables DTMF tone output 0: Disables DTMF tone output
DTMFT.3 to DTMFT.0	T3 to T0	DTMF code to be generated (See Table 7)

**GUARD TIME REGISTER (GTIME)**

Address 0AH; read / write.

<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
-	G6	G5	G4	G3	G2	G1	G0

Description of GTIME bits

Bit	Symbol	Description
GTIME.6 to GTIME.0	D6 to D0	Guard time to indicate the end of a line reversal or ring

**INTERRUPT REGISTER (INTR)**

Address 80H; read only.

<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
-	BOMdetect	SDTint	-	-	FSKint	CASint	LRint

Description of INTR bits

Bit	Symbol	Description
INTR.6	BOMdetect	1: Indicates that the begin of the mark period during FSK reception has been detected
INTR.5	SDTint	1: Indicates that SDTdetect has been changed
INTR.2	FSKint	1: Indicates that a new FSK frame has been received
INTR.1	CASint	1: Indicates that CASdetect has been detected
INTR.0	LRint	1: Indicates that LRstatus has been changed

**STATUS REGISTER (STAT)**

Address 81H; read only.

<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
-	-	SDTdetect	-	-	-	CASdetect	LRstatus

Description of STAT bits

Bit	Symbol	Description
STAT.5	SDTdetect	1: Indicates that the SDT detector detects the signal that satisfies the specified frequency and energy level; 0: No more Progress Tone is detected
STAT.1	CASdetect	1: Indicates that a CAS tone has been detected 0: No more CAS Tone is detected
STAT.0	LRstatus	1: LRint has not occurred until expiring GTIME (reset value) 0: LRint has occurred before expiring GTIME

**FSK DATA REGISTER (FSKDT)**

Address 82H; read only.

<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
D7	D6	D5	D4	D3	D2	D1	D0

Description of FSKDT bits

Bit	Symbol	Description
FSKDT.7 to FSKDT.0	D7 to D0	Last received FSK data byte

**DTMF OUTPUT GAIN CONTROL REGISTER (DTMFG)**

Address 0H; read / write

<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
D7	D6	D5	D4	D3	D2	D1	D0

Description of DTMFG bits

Bit	Symbol	Description
DTMFG.7 to DTMFG.0	D7 to D0	This byte multiplied to control the output gain of DTMF generator

**SPECIAL CONTROL REGISTER (CONT1)**

Address F1H; read / write

7	6	5	4	3	2	1	0
-	-	0	0	0	1	1	1

This register should be written with 'xx00 0111b'.

**SPECIAL CONTROL REGISTER (CONT2)**

Address F5H; read / write

7	6	5	4	3	2	1	0
-	-	0	0	0	0	0	0

This register should be written with 'xx00 0000b'.

NOTES

# 8 INTERRUPTS

## OVERVIEW

The S3P7588X interrupt control circuit has five functional components:

- Interrupt enable flags (IEx)
- Interrupt request flags (IRQx)
- Interrupt mask enable register (IME)
- Interrupt priority register (IPR)
- Power-down release signal circuit

Three kinds of interrupts are supported:

- Internal interrupts generated by on-chip processes
- External interrupts generated by external peripheral devices
- Quasi-interrupts used for edge detection and as clock sources

**Table 8-1. Interrupt Types and Corresponding Port Pin(s)**

Interrupt Type	Interrupt Name	Corresponding Port Pin
External interrupts	INT1, INT4	P1.1, P1.3
Internal interrupts	INT0 (note), INTB, INTT0, INTT1	Not applicable
Quasi-interrupts	INT2	P1.2, KS0–KS7
	INTW	Not applicable

**NOTE:** INT0 is dedicated to caller id interrupt.



## VECTORED INTERRUPTS

Interrupt requests may be processed as vectored interrupts in hardware, or they can be generated by program software. A vectored interrupt is generated when the following flags and register settings, corresponding to the specific interrupt (INTn) are set to logic one:

- Interrupt enable flag (IEx)
- Interrupt master enable flag (IME)
- Interrupt request flag (IRQx)
- Interrupt status flags (IS0, IS1)
- Interrupt priority register (IPR)

If all conditions are satisfied for the execution of a requested service routine, the start address of the interrupt is loaded into the program counter and the program starts executing the service routine from this address.

EMB and ERB flags for RAM memory banks and registers are stored in the vector address area of the ROM during interrupt service routines. The flags are stored at the beginning of the program with the VENT instruction. The initial flag values determine the vectors for resets and interrupts. Enable flag values are saved during the main routine, as well as during service routines. Any changes that are made to enable flag values during a service routine are not stored in the vector address.

When an interrupt occurs, the enable flag values before the interrupt is initiated are saved along with the program status word (PSW), and the enable flag values for the interrupt is fetched from the respective vector address. Then, if necessary, you can modify the enable flags during the interrupt service routine. When the interrupt service routine is returned to the main routine by the IRET instruction, the original values saved in the stack are restored and the main program continues program execution with these values.

### Software-Generated Interrupts

To generate an interrupt request from software, the program manipulates the appropriate IRQx flag. When the interrupt request flag value is set, it is retained until all other conditions for the vectored interrupt have been met, and the service routine can be initiated.

### Multiple Interrupts

By manipulating the two interrupt status flags (IS0 and IS1), you can control service routine initialization and thereby process multiple interrupts simultaneously.

If more than four interrupts are being processed at one time, you can avoid possible loss of working register data by using the PUSH RR instruction to save register contents to the stack before the service routines are executed in the same register bank. When the routines have executed successfully, you can restore the register contents from the stack to working memory using the POP instruction.

### Power-Down Mode Release

An interrupt can be used to release power-down mode (stop or idle). Interrupts for power-down mode release are initiated by setting the corresponding interrupt enable flag. Even if the IME flag is cleared to zero, power-down mode will be released by an interrupt request signal when the interrupt enable flag has been set. In such cases, the interrupt routine will not be executed since IME = "0".

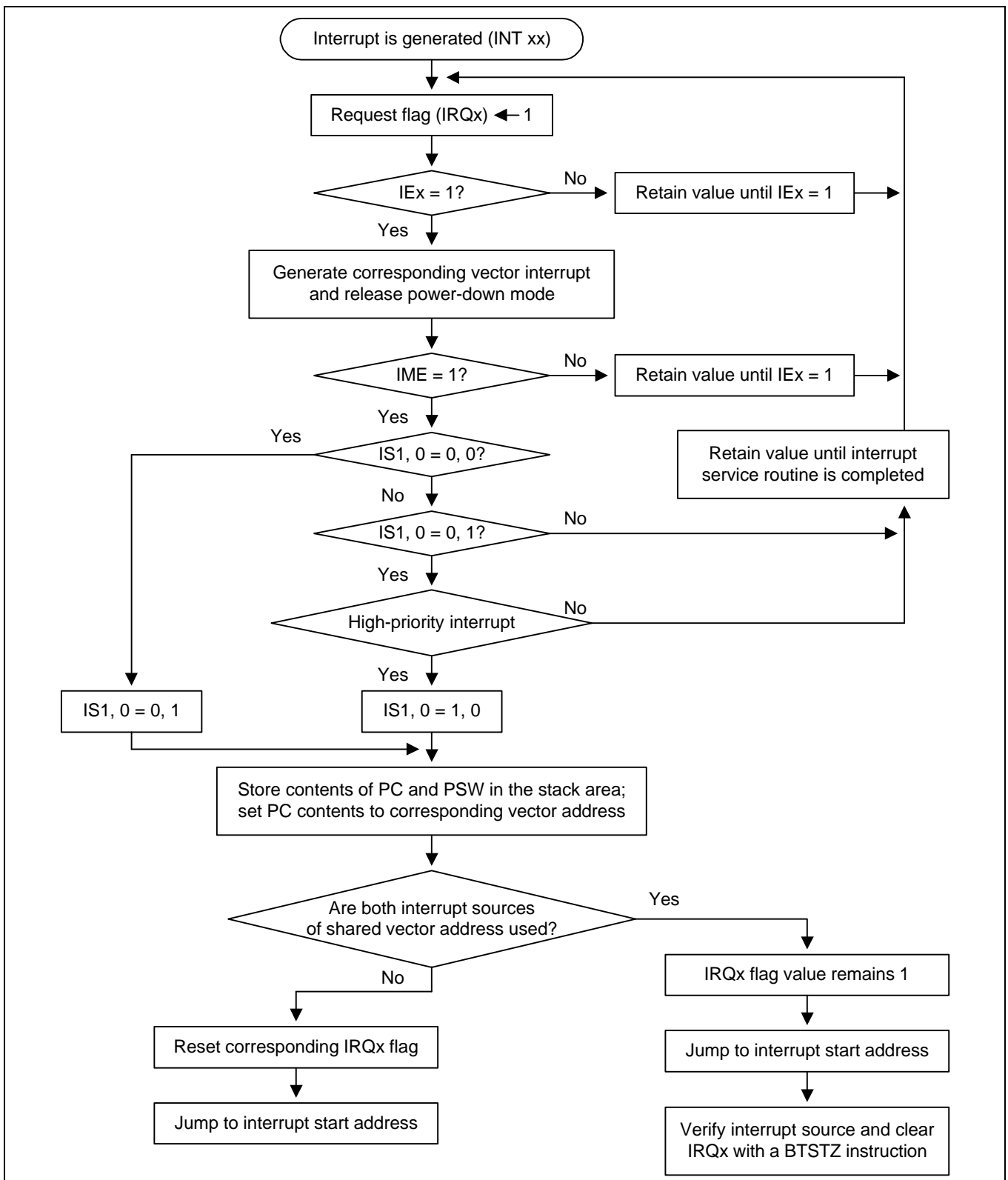


Figure 8-1. Interrupt Execution Flowchart

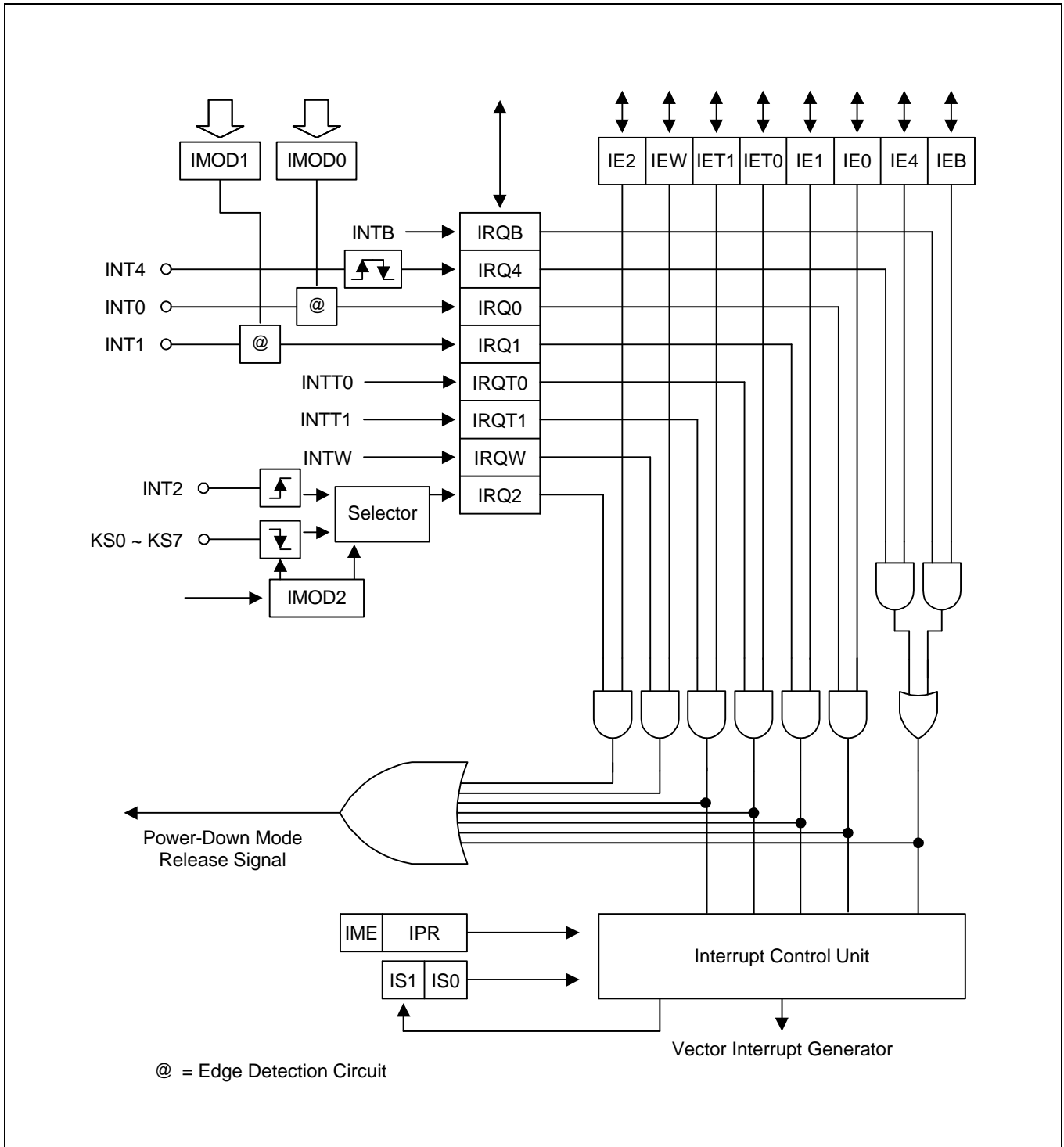


Figure 8-2. Interrupt Control Circuit Diagram

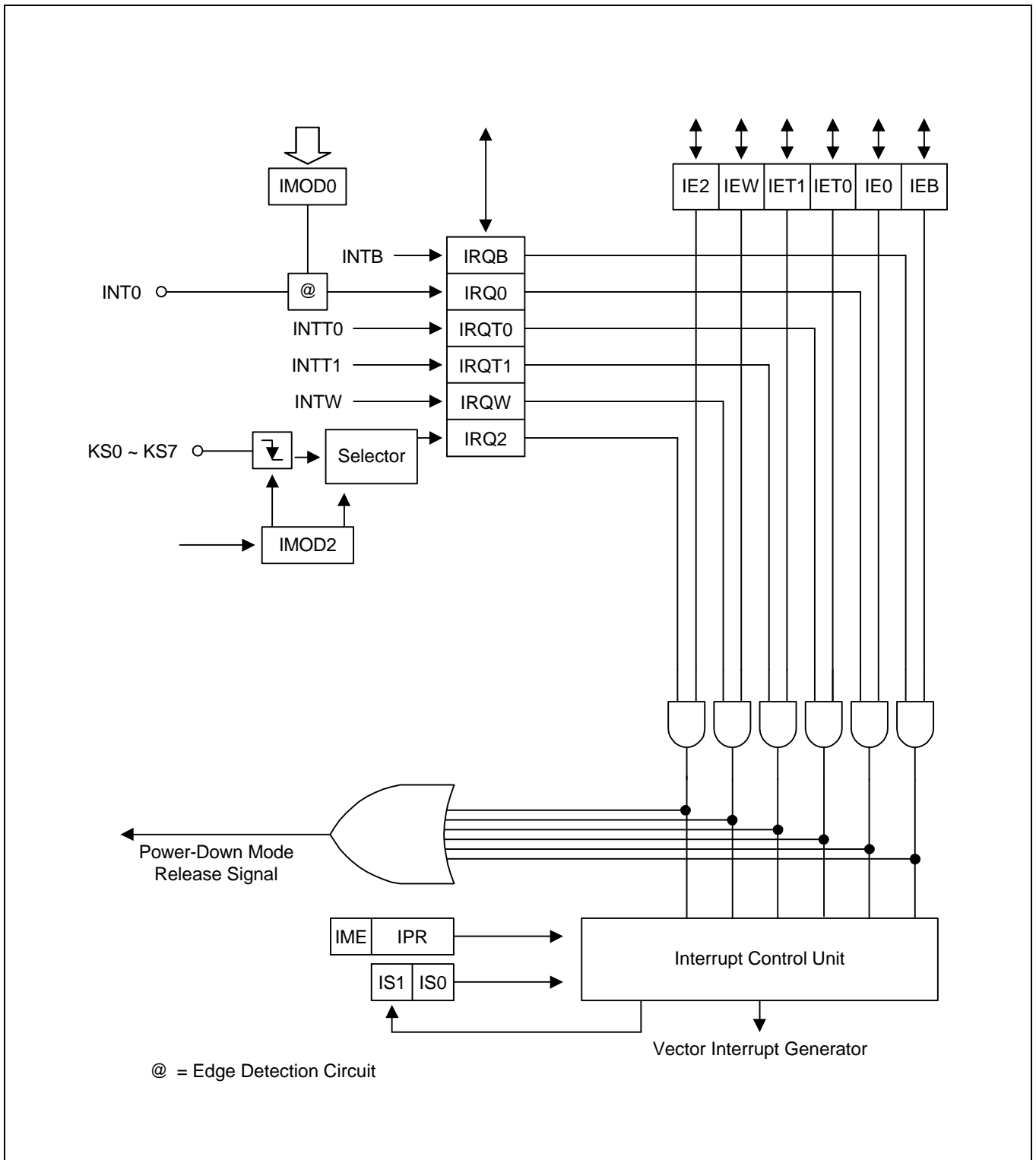


Figure 8-3. Interrupt Control Circuit Diagram

**Multiple Interrupts**

The interrupt controller can service multiple interrupts in two ways: as two-level interrupts, where either all interrupt requests or only those of highest priority are serviced, or as multi-level interrupts, when the interrupt service routine for a lower-priority request is accepted during the execution of a higher priority routine.

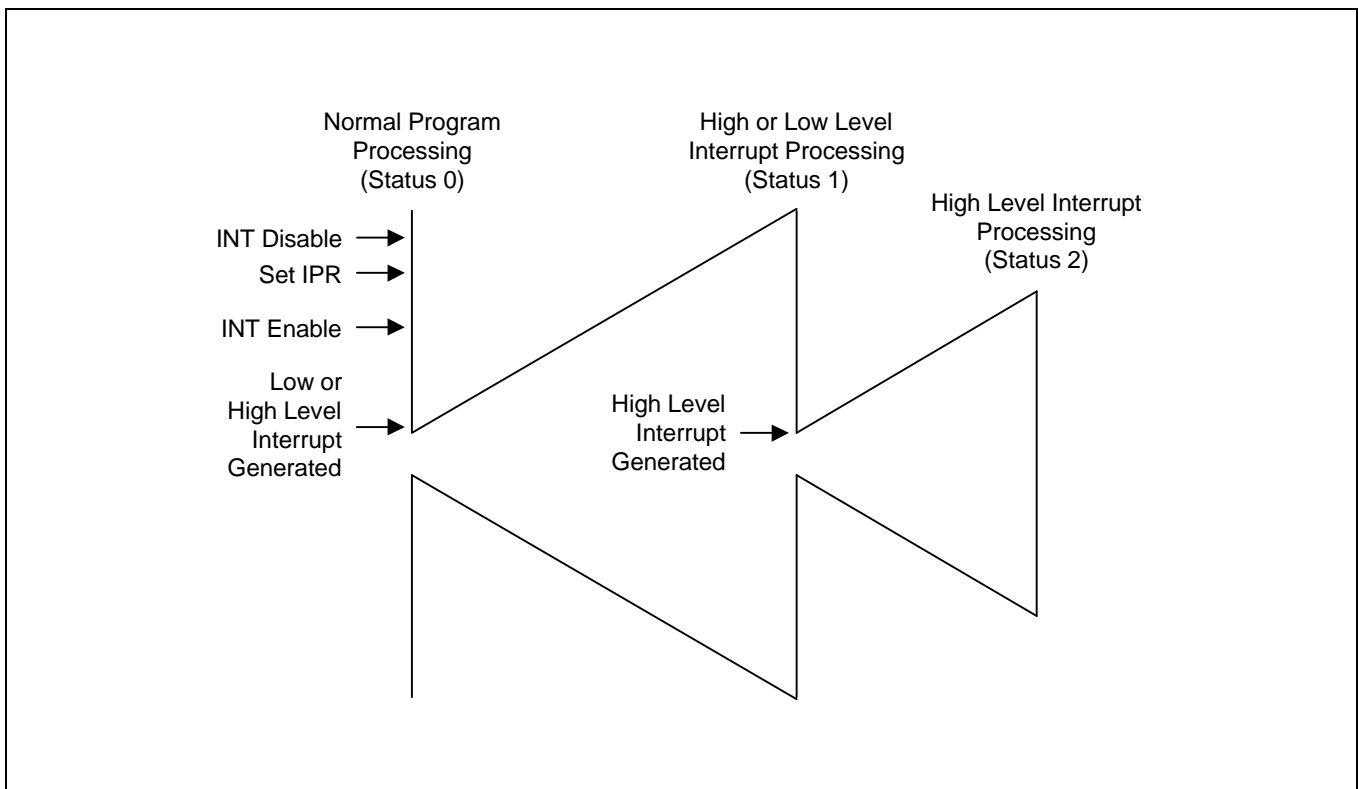
**Two-Level Interrupt Handling**

Two-level interrupt handling is the standard method for processing multiple interrupts. When the IS1 and IS0 bits of the PSW (FB0H.3 and FB0H.2, respectively) are both logic zero, program execution mode is normal and all interrupt requests are serviced (see Figure 8-3).

Whenever an interrupt request is accepted, IS1 and IS0 are incremented by one ("0" → "1" or "1" → "0"), and the values are stored in the stack along with the other PSW bits. After the interrupt routine has been serviced, the modified IS1 and IS0 values are automatically restored from the stack by an IRET instruction.

IS0 and IS1 can be manipulated directly by 1-bit write instructions, regardless of the current value of the enable memory bank flag (EMB). Before you can modify an interrupt status flag, however, you must first disable interrupt processing with a DI instruction.

When IS1 = "0" and IS0 = "1", all interrupt service routines are inhibited except for the highest priority interrupt currently defined by the interrupt priority register (IPR).



**Figure 8-4. Two-Level Interrupt Handling**

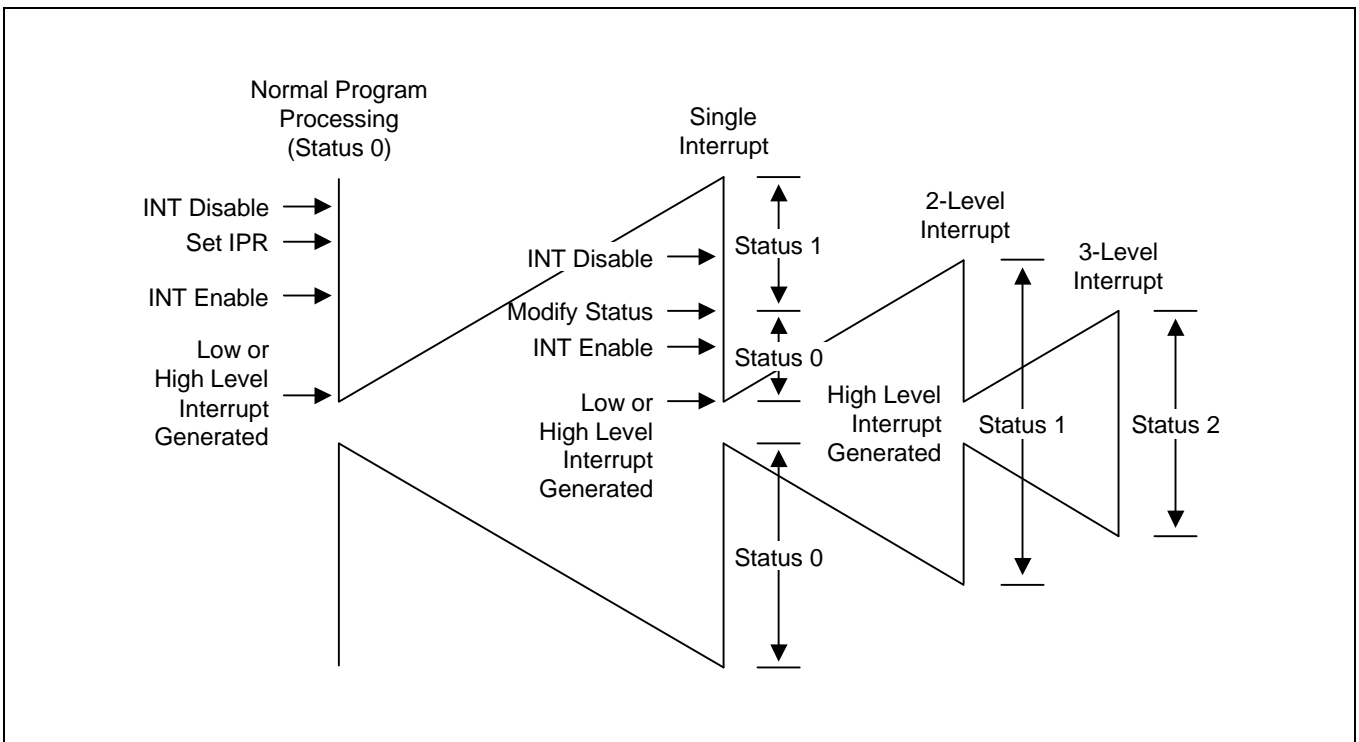
**Multi-Level Interrupt Handling**

With multi-level interrupt handling, a lower-priority interrupt request can be executed while a high-priority interrupt is being serviced. This is done by manipulating the interrupt status flags, IS0 and IS1 (see Table 8-2).

When an interrupt is requested during normal program execution, interrupt status flags IS0 and IS1 are set to "1" and "0", respectively. This setting allows only highest-priority interrupts to be serviced. When a high-priority request is accepted, both interrupt status flags are then cleared to "0" by software so that a request of any priority level can be serviced. In this way, the high- and low-priority requests can be serviced in parallel (see Figure 8-4).

**Table 8-2. IS1 and IS0 Bit Manipulation for Multi-Level Interrupt Handling**

Process Status	Before INT		Effect of ISx Bit Setting	After INT ACK	
	IS1	IS0		IS1	IS0
0	0	0	All interrupt requests are serviced.	0	1
1	0	1	Only high-priority interrupts as determined by the current settings in the IPR register are serviced.	1	0
2	1	0	No additional interrupt requests will be serviced.	–	–
–	1	1	Value undefined	–	–



**Figure 8-5. Multi-Level Interrupt Handling**

### INTERRUPT PRIORITY REGISTER (IPR)

The 4-bit interrupt priority register (IPR) is used to control multi-level interrupt handling. Its reset value is logic zero. Before the IPR can be modified by 4-bit write instructions, all interrupts must first be disabled by a DI instruction.

FB2H	IME	IPR.2	IPR.1	IPR.0
------	-----	-------	-------	-------

By manipulating the IPR settings, you can choose to process all interrupt requests with the same priority level, or you can select one type of interrupt for high-priority processing. A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt cannot be interrupted by any other interrupt source.

**Table 8-3. Standard Interrupt Priorities**

Interrupt	Default Priority
INTB, INT4	1
INT0 (note)	2
INT1	3
INTT0	4
INTT1	5

The MSB of the IPR, the interrupt master enable flag (IME), enables and disables all interrupt processing. Even if an interrupt request flag and its corresponding enable flag are set, a service routine cannot be executed until the IME flag is set to logic one. The IME flag can be directly manipulated by EI and DI instructions, regardless of the current enable memory bank (EMB) value.

**Table 8-4. Interrupt Priority Register Settings**

IPR.2	IPR.1	IPR.0	Result of IPR Bit Setting
0	0	0	Normal interrupt handling according to default priority settings
0	0	1	Process INTB and INT4 interrupts at highest priority
0	1	0	Process INT0 (NOTE) interrupts at highest priority
0	1	1	Process INT1 interrupts at highest priority
1	0	1	Process INTT0 interrupts at highest priority
1	1	0	Process INTT1 interrupts at highest priority

#### NOTES:

- During normal interrupt processing, interrupts are processed in the order in which they occur. If two or more interrupts occur simultaneously, the processing order is determined by the default interrupt priority settings shown in Table 8-3. Using the IPR settings, you can select specific interrupts for high-priority processing in the event of contention. When the high-priority (IPR) interrupt has been processed, waiting interrupts are handled according to their default priorities.
- INT0 is dedicated to caller id interrupt.

### PROGRAMMING TIP — Setting the INT Interrupt Priority

The following instruction sequence sets the INT1 interrupt to high priority:

```

BITS      EMB
SMB      15
DI                          ; IPR.3 (IME) ← 0
LD        A,#3H
LD        IPR,A
EI                          ; IPR.3 (IME) ← 1

```

#### External Interrupt 0 AND 1 Mode Registers (IMOD0, IMOD1)

The following components are used to process external interrupts at the INT0 (note) and INT1 pin:

- Edge detection circuit
- Two mode registers, IMOD0 and IMOD1

The mode registers are used to control the triggering edge of the input signal. IMOD0 and IMOD1 settings let you choose either the rising or falling edge of the incoming signal as the interrupt request trigger. The INT4 interrupt is an exception since its input signal generates an interrupt request on both rising and falling edges.

FB4H	"0"	"0"	IMOD0.1	IMOD0.0
FB5H	"0"	"0"	"0"	IMOD1.0

IMOD0 and IMOD1 bits are addressable by 4-bit write instructions. RESET clears all IMOD values to logic zero, selecting rising edges as the trigger for incoming interrupt requests.

**Table 8-5. IMOD0 and IMOD1 Register Organization**

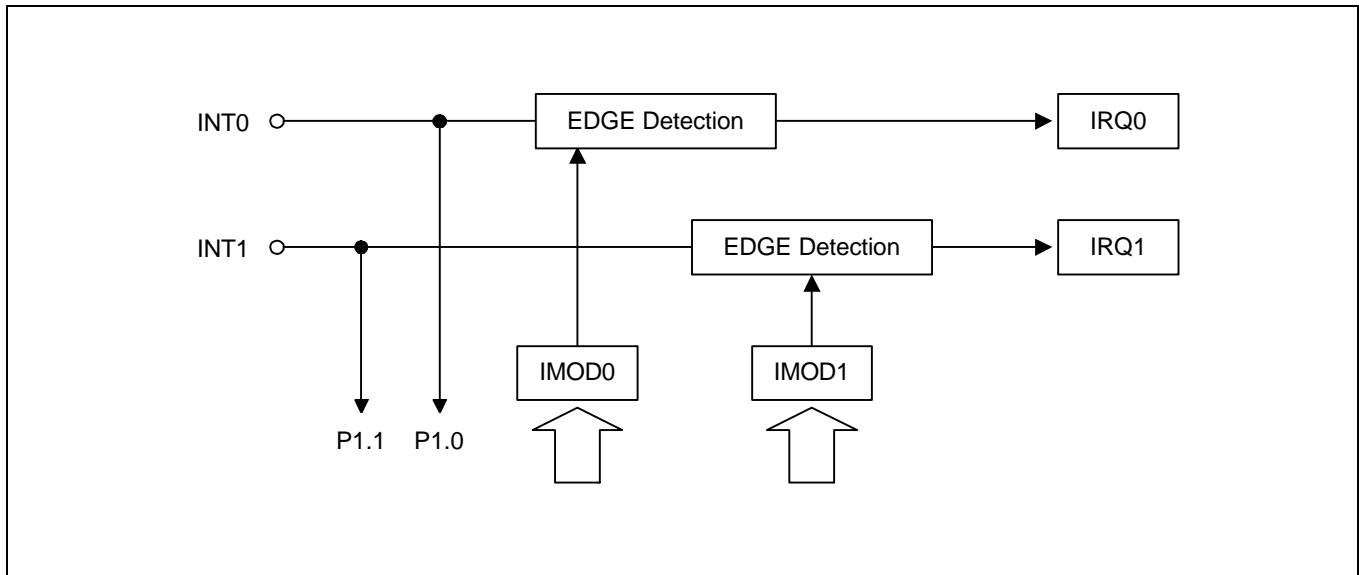
IMOD0	0	0	IMOD0.1	IMOD0.0	Effect of IMOD0 Settings
			0	0	Rising edge detection
			0	1	Falling edge detection
			1	0	Both rising and falling edge detection
			1	1	IRQ0 flag cannot be set to "1"
IMOD1	0	0	0	IMOD1.0	Effect of IMOD1 Settings
				0	Rising edge detection
				1	Falling edge detection

**NOTE:** INT0 is dedicated to caller id interrupt, so it is unable to receive external event from INT0 pin.



**EXTERNAL INTERRUPT 0 AND 1 MODE REGISTERS (CONTINUED)**

When a sampling clock rate of  $f_x/64$  is used for INT0, an interrupt request flag must be cleared before 16 machine cycles have elapsed.



**Figure 8-6. Circuit Diagram for INT0 and INT1 Pins**

When modifying the IMOD0 and IMOD1 registers, it is possible to accidentally set an interrupt request flag. To avoid unwanted interrupts, take these precautions when writing your programs:

1. Disable all interrupts with a DI instruction.
2. Modify the IMOD0 or IMOD1 register.
3. Clear all relevant interrupt request flags.
4. Enable the interrupt by setting the appropriate IEx flag.
5. Enable all interrupts with an EI instructions.

**EXTERNAL INTERRUPT 2 MODE REGISTER (IMOD2)**

The mode register for external interrupts at the KS0–KKS7 pins, IMOD2, is addressable only by 4-bit write instructions. RESET clears all IMOD2 bits to logic zero.

FB6H	"0"	"0"	IMOD2.1	IMOD2.0
------	-----	-----	---------	---------

When IMOD2 is cleared to logic zero, INT2 uses the rising edge of an incoming signal as the interrupt request trigger. If a rising edge is detected at the INT2 pin, or when a falling edge is detected at any one of the pins KS0–KS7, the IRQ2 flag is set to logic one and a release signal for power-down mode is generated.

**Table 8-6. IMOD2 Register Bit Settings**

IMOD2	0	0	IMOD2.1	IMOD2.0	Effect of IMOD2 Settings
				0	0
			0	1	Select falling edge at KS4–KS7
			1	0	Select falling edge at KS2–KS7
			1	1	Select falling edge at KS0–KS7

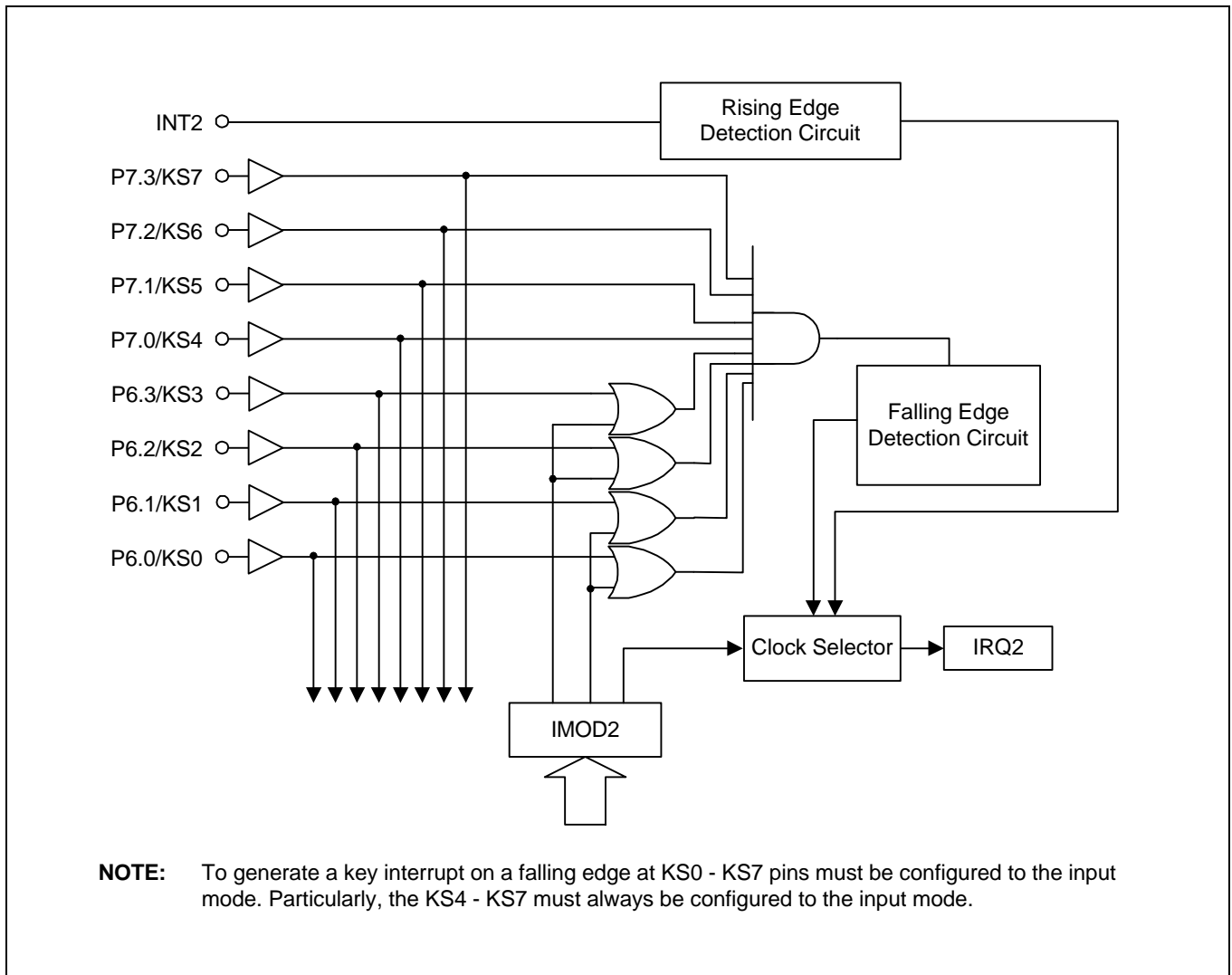


Figure 8-7. Circuit Diagram for INT2 and KS0-KS7 Pins

 **PROGRAMMING TIP — Using INT2 as a Key Input Interrupt**

When the INT2 interrupt is used as a key interrupt, the selected key interrupt source pin must be set to input:

1. When KS0–KS7 are selected (eight pins):

```

BITS      EMB
SMB       15
LD        A, #3H
LD        IMOD2, A          ; (IMOD2) ← #3H, KS0–KS7 falling edge select
LD        EA, #00H
LD        PMG3, EA         ; P6, 7 ← input mode
LD        A, #3H
LD        PUMOD2, A        ; Enable P6 and P7 pull-up resistors

```

2. When KS2–KS7 are selected (six pins):

```

BITS      EMB
SMB       15
LD        A, #2H
LD        IMOD2, A          ; (IMOD2) ← #2H, KS2–KS7 falling edge select
LD        EA, #03H
LD        PMG3, EA         ; P7, P6.2–P6.3 ← input mode
LD        A, #3H
LD        PUMOD2, A        ; Enable P6 and P7 pull-up resistors

```

3. When KS4–KS7 are selected (four pins), P7 must be specified as a key strobe signal input:

```

BITS      EMB
SMB       15
LD        A, #1H
LD        IMOD2, A          ; (IMOD2) ← #1H, KS4–KS7 falling edge select
LD        EA, #0FH
LD        PMG3, EA
LD        A, #2
LD        PUMOD2, A        ; Enable P7 pull-up resistor

```

## INTERRUPT FLAGS

There are three types of interrupt flags: interrupt request and interrupt enable flags that correspond to each interrupt, the interrupt master enable flag, which enables or disables all interrupt processing.

### Interrupt Master Enable Flag (IME)

The interrupt master enable flag, IME, enables or disables all interrupt processing. Therefore, even when an IRQx flag is set and its corresponding IEx flag is enabled, the interrupt service routine is not executed until the IME flag is set to logic one.

The IME flag is located in the IPR register (IPR.3). It can be directly manipulated by EI and DI instructions, regardless of the current value of the enable memory bank flag (EMB).

IME	IPR.2	IPR.1	IPR.0	Effect of Bit Settings
0				Inhibit all interrupts
1				Enable all interrupts

### Interrupt Enable Flags (IEx)

IEx flags, when set to logical one, enable specific interrupt requests to be serviced. When the interrupt request flag is set to logic one, an interrupt will not be serviced until its corresponding IEx flag is also enabled.

Interrupt enable flags can be read, written, or tested directly by 1-bit instructions (BITS and BITR) or 4-bit instructions. IEx flags can be addressed directly at their specific RAM addresses, despite the current value of the enable memory bank (EMB) flag.

**Table 8-7. Interrupt Enable and Interrupt Request Flag Addresses**

Address	Bit 3	Bit 2	Bit 1	Bit 0
FB8H	IE4	IRQ4	IEB	IRQB
FBAH	0	0	IEW	IRQW
FBBH	0	0	IET1	IRQT1
FBCH	0	0	IET0	IRQT0
FBEH	IE1	IRQ1	IE0	IRQ0
FBFH	0	0	IE2	IRQ2

#### NOTES:

1. IEx refers generically to all interrupt enable flags.
2. IRQx refers generically to all interrupt request flags.
3. IEx = 0 is interrupt disable mode.
4. IEx = 1 is interrupt enable mode.

### Interrupt Request Flags (IRQx)

Interrupt request flags, are read/write addressable by 1-bit or 4-bit instructions. IRQx flags can be addressed directly at their specific RAM addresses, regardless of the current value of the enable memory bank (EMB) flag.

When a specific IRQx flag is set to logic one, the corresponding interrupt request is generated. The flag is then automatically cleared to logic zero when the interrupt has been serviced. Exceptions are the watch timer interrupt request flags, IRQW, and the external interrupt 2 flag IRQ2, which must be cleared by software after the interrupt service routine has executed. IRQx flags are also used to execute interrupt requests from software. In summary, follow these guidelines for using IRQx flags:

1. IRQx is set to request an interrupt when an interrupt meets the set condition for interrupt generation.
2. IRQx is set to "1" by hardware and then cleared by hardware when the interrupt has been serviced (with the exception of IRQW and IRQ2).
3. If IRQx is set to "1" by software, an interrupt is also generated.

When two interrupts share the same service routine start address, interrupt processing may occur in one of two ways:

- When only one interrupt is enabled, the IRQx flag is cleared automatically when the interrupt has been serviced.
- When two interrupts are enabled, the request flag is not automatically cleared so that the user has an opportunity to locate the source of the interrupt request. In this case, the IRQx setting must be cleared manually using a BTSTZ instruction.

**Table 8-8. Interrupt Request Flag Conditions and Priorities**

Interrupt Source	Internal / External	Pre-condition for IRQx Flag Setting	Interrupt Priority	IRQ Flag Name
INTB	I	Reference time interval signal from basic timer	1	IRQB
INT4	E	Both rising and falling edges detected at INT4	1	IRQ4
INT0 (NOTE2)	I	Falling edge detected at Caller ID interrupt	2	IRQ0
INT1	E	Rising or falling edge detected at INT1 pin	3	IRQ1
INTT0	I	Signals for TCNT0 and TREF0 registers match	5	IRQT0
INTT1	I	Signals for TCNT1 and TREF1 registers match	6	IRQT1
INT2	E	Rising edge detected at INT2 pin or else a falling edge is detected at any of the KS0–KS7 pins	–	IRQ2
INTW	I	Time interval of 0.5 secs or 3.19 msecs	–	IRQW

**NOTES:**

1. The quasi-interrupt INT2 is only used for testing incoming signals.
2. INT0 is dedicated to caller id interrupt, and must be programmed as falling edge detection mode.

 **PROGRAMMING TIP — Enabling the INTB and INT4 Interrupts**

To simultaneously enable INTB and INT4 interrupts:

```
INTB      DI
          BTSTZ      IRQB      ; IRQB = 1 ?
          JR          INT4      ; If no, INT4 interrupt; if yes, INTB interrupt is processed
          .
          .
          .
          EI
          IRET

INT4      BITR      IRQ4      ; INT4 is processed
          .
          .
          .
          EI
          IRET
```

# 9

## POWER-DOWN

### OVERVIEW

The S3P7588X microcontroller has two power-down modes to reduce power consumption: idle and stop. Idle mode is initiated by the IDLE instruction and stop mode by the instruction STOP. (Several NOP instructions must always follow an IDLE or STOP instruction in a program.) In idle mode, the CPU clock stops while peripherals and the oscillation source continue to operate normally.

When RESET occurs during normal operation or during a power-down mode, a reset operation is initiated and the CPU enters idle mode. When the standard oscillation stabilization time interval (31.3ms at 4.19MHz) has elapsed, normal CPU operation resumes.

In stop mode, system clock oscillation is halted (assuming it is currently operating), and peripheral hardware components are powered-down. The effect of stop mode on specific peripheral hardware components — CPU, basic timer, timer/counters, and watch-timer — and on external interrupt requests, is detailed in Table 9-1.

#### NOTE

Do not use stop mode if you are using an external clock source because  $X_{in}$  input must be restricted internally to  $V_{SS}$  to reduce current leakage.

Idle or stop modes are terminated either by a RESET, or by an interrupt with the exception of INT0, which are enabled by the corresponding interrupt enable flag, IEx. When power-down mode is terminated by RESET input, a normal reset operation is executed. Assuming that both the interrupt enable flag and the interrupt request flag are set to "1", power-down mode is released immediately upon entering power-down mode.

When an interrupt is used to release power-down mode, the operation differs depending on the value of the interrupt master enable flag (IME):

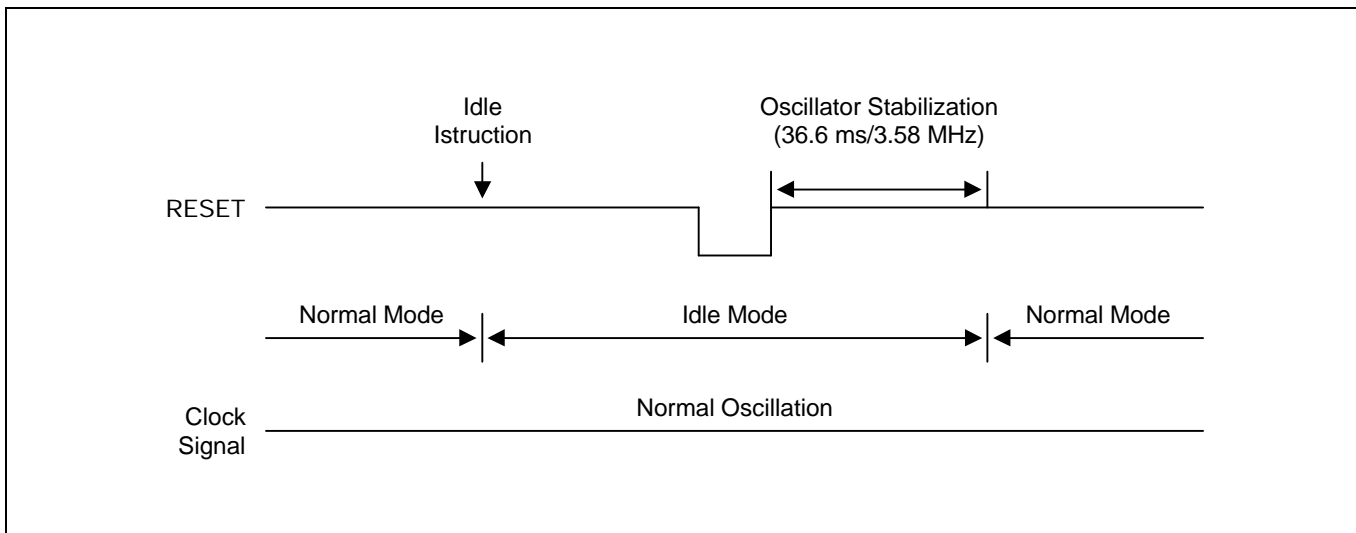
- If the IME flag = "0", program execution is started immediately after the instruction which issues the request to enter power-down mode. The interrupt request flag remains set to logic one.
- If the IME flag = "1", two instructions are executed after the power-down mode release. Then, the vectored interrupt is initiated. However, when the release signal is caused by INT2 or INTW, the operation is identical to the IME = 0 condition. That is, a vector interrupt is not generated.



**Table 9-1. Hardware Operation During Power-Down Modes**

Operation	Stop Mode (STOP)	Idle Mode (IDLE)
Clock oscillator	System clock oscillation stops	CPU clock oscillation stops. (system clock oscillation continues)
Basic timer	Basic timer stops	Basic timer operates. (with IRQB set at each reference interval)
Caller ID	Caller ID stops except LR detector. To save power consumption of 14bit ADC, user must set the additional mode register in caller id module (Refer to Chapter 7)	Caller ID operates. Caller ID can be stopped by setting the mode register in caller id module (Refer to Chapter 7)
Timer/counter 0	Operates only if TCL0 is selected as the counter clock	Timer/counter 0 operates
Timer/counter 1	Operates only if TCL1 is selected as the counter clock	Timer/counter 1 operates
Watch timer	Watch timer operation is stopped	Watch timer operates
External interrupts	INT0, INT1, INT2, and INT4 are acknowledged. (Caller ID's LR interrupt can wake up system clock through INT0 interrupt)	INT0, INT1, INT2, and INT4 are acknowledged. (Any interrupt of caller id can wake up CPU through INT0 interrupt)
CPU	All CPU operations are disabled	All CPU operations are disabled
Power-down mode release signal	Interrupt request signals are enabled by an interrupt enable flag or by RESET input	Interrupt request signals are enabled by an interrupt enable flag or by RESET input

**IDLE MODE TIMING DIAGRAMS**



**Figure 9-1. Timing When Idle Mode is Released by RESET**

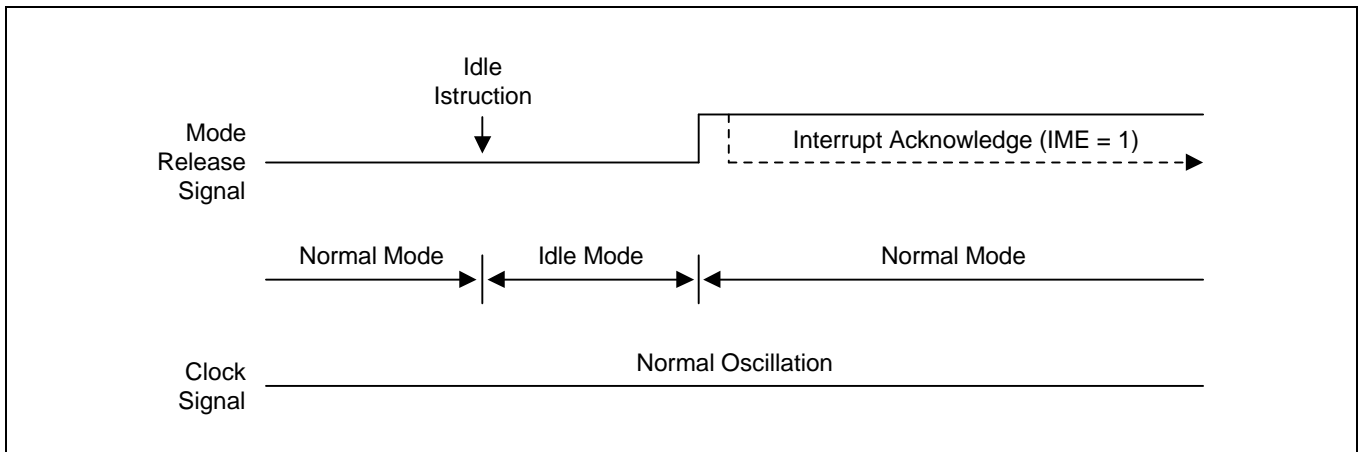


Figure 9-2. Timing When Idle Mode is Released by an Interrupt

**STOP MODE TIMING DIAGRAMS**

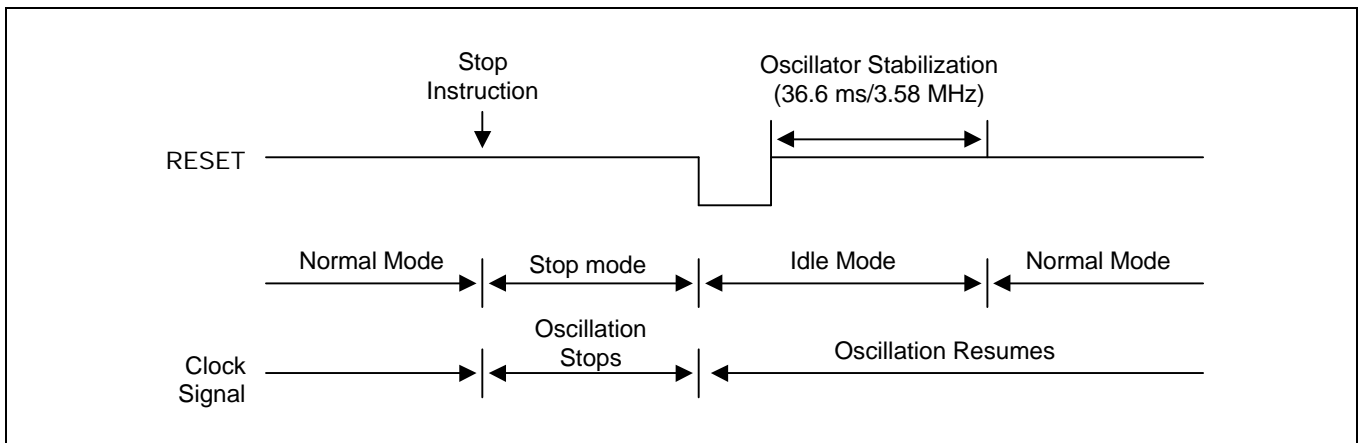


Figure 9-3. Timing When Stop Mode is Released by RESET

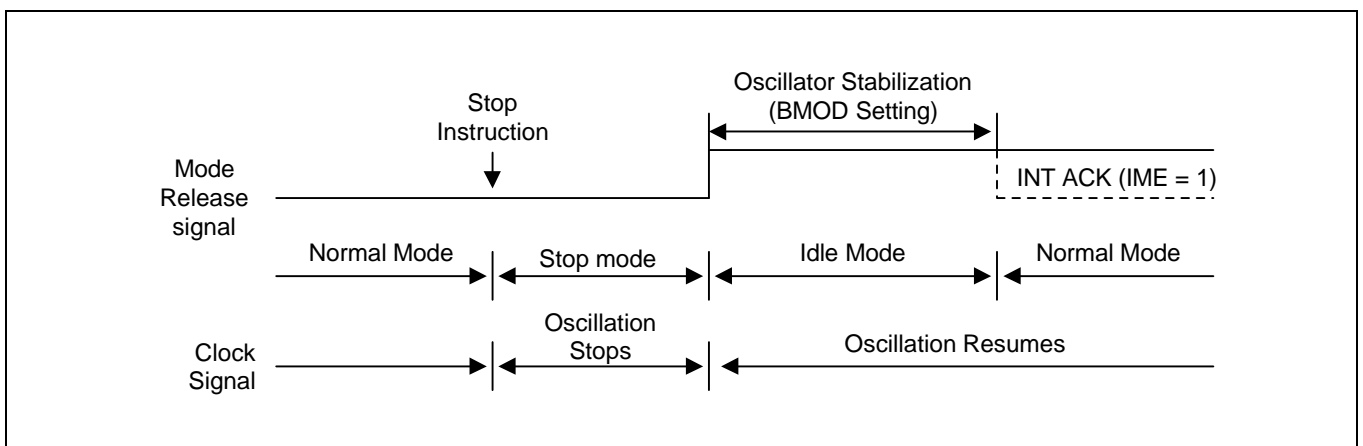


Figure 9-4. Timing When Stop Mode is Release by an Interrupt

## PORT PIN CONFIGURATION FOR POWER-DOWN

The following method describes how to configure I/O port pins to reduce power consumption during power-down modes (STOP, IDLE):

**Condition 1:** If the microcontroller is not configured to an external device:

1. Connect unused port pins according to the information in Table 9-2.
2. Disable all pull-up resistors for output pins by making the appropriate modifications to the pull-up resistor mode register, PUMOD. Reason: If output goes low when the pull-up resistor is enabled, there may be unexpected surges of current through the pull-up.
3. Disable pull-up resistors for input pins configured to  $V_{DD}$  or  $V_{SS}$  levels in order to check the current input option. Reason: If the input level of a port pin is set to  $V_{SS}$  when a pull-up resistor is enabled, it will draw an unnecessarily large current.

**Condition 2:** If the microcontroller is configured to an external device and the external device's  $V_{DD}$  source is turned off in power-down mode.

1. Connect unused port pins according to the information in Table 9-2.
2. Disable the pull-up resistors of output pins by making the appropriate modifications to the pull-up resistor mode register, PUMOD. Reason: If output goes low when the pull-up resistor is enabled, there may be unexpected surges of current through the pull-up.
3. Disable pull-up resistors for input pins configured to  $V_{DD}$  or  $V_{SS}$  levels in order to check the current input option. Reason: If the input level of a port pin is set to  $V_{SS}$  when a pull-up resistor is enabled, it will draw an unnecessarily large current.
4. Disable the pull-up resistors of input pins connected to the external device by making the necessary modifications to the PUMOD register.
5. Configure the output pins that are connected to the external device to low level. Reason: When the external device's  $V_{DD}$  source is turned off, and if the microcontroller's output pins are set to high level,  $V_{DD} - 0.7V$  is supplied to the  $V_{DD}$  of the external device through its input pin. This causes the device to operate at the level  $V_{DD} - 0.7V$ . In this case, total current consumption would not be reduced.
6. Determine the correct output pin state necessary to block current pass in according with the external transistors (PNP, NPN).

## RECOMMENDED CONNECTIONS FOR UNUSED PINS

To reduce overall power consumption, please configure unused pins according to the guidelines described in Table 9-2.

**Table 9-2. Unused Pin Connections for Reduced Power Consumption**

Pin/Share Pin Names	Recommended Connection
P1.1 / INT1 –P 1.2 / INT2	Connect to $V_{DD}$
P1.3 / INT4	
P2.0 / TCLO0 P2.3 / BUZ P3.0 / TCL0 P3.1 / TCL1 P3.2 P3.3 P4.0 / BTCL0 P4.1–P4.3 P5.0–P5.3 P6.0 / KS0–P6.3 / KS3 P7.0 / KS4–P7.3 / KS7 P9.0 P9.1 / TCLO1 P9.2 / CLO	Input mode: Connect to $V_{DD}$ Output mode: No connection
DTMF	No connection
VREF, OUT	No connection
INS, INN, INP, LRin	Connect to $V_{SS}$
NC	Connect to $V_{SS}$

NOTES

# 10

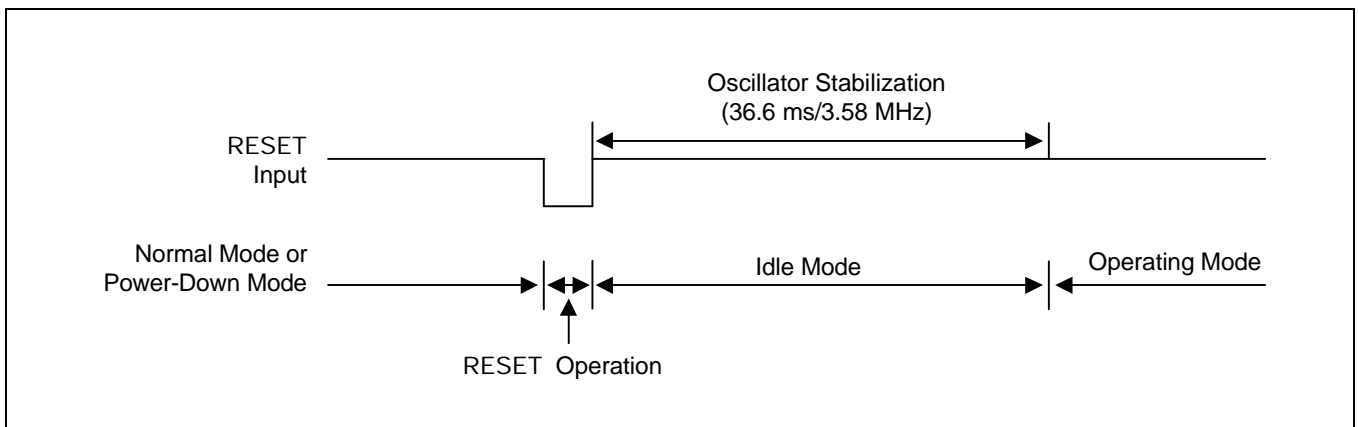
## RESET

### OVERVIEW

When a RESET signal is input during normal operation or power-down mode, a hardware reset operation is initiated and the CPU enters idle mode. Then, when the standard oscillation stabilization interval of 36.6 ms at 3.58 MHz has elapsed, normal system operation resumes.

Regardless of when the RESET occurs — during normal operating mode or during a power-down mode — most hardware register values are set to the reset values described in Table 10-1 below. The current status of several register values is, however, always retained when a RESET occurs during idle or stop mode; If a RESET occurs during normal operating mode, their values are undefined. Current values that are retained in this case are as follows:

- Carry flag
- General-purpose registers E, A, L, H, X, W, Z, and Y



**Figure 10-1. Timing for Oscillation Stabilization After RESET**

### CALLER ID RESET SIGNAL

Caller ID receiver has the dedicated reset signal that is come from the output of P8.0 or P3.1. Whichever port you choose, you should make the active-low reset pulse (high → low → high) at the start of the program, or the caller id receiver remains reset state regardless of the RESET signal.

P8 is a internal redundant port and not used for external interface, so it is recommended to use P8.0 as caller id receiver reset signal. To use P8.0, set P8.1 as 1 ahead. (Refer to Chapter 7)

**HARDWARE RESET VALUES AFTER RESET**

Table 10-1 gives you detailed information about hardware register values after a RESET occurs during power-down mode or during normal operation.

**Table 10-1. Hardware Register Values After RESET**

Hardware Component or Subcomponent	If RESET Occurs During Power-Down Mode	If RESET Occurs During Normal Operation
Program counter (PC)	Lower five bits of address 0000H are transferred to PC12–8, and the contents of 0001H to PC7–0.	Lower five bits of address 0000H are transferred to PC12–8, and the contents of 0001H to PC7–0.
<b>Program Status Word (PSW):</b>		
Carry flag (C)	Values retained	Undefined
Skip flag (SC0–SC2)	0	0
Interrupt status flags (IS0, IS1)	0	0
Bank enable flags (EMB, ERB)	Bit 6 of address 0000H in program memory is transferred to the ERB flag, and bit 7 of the address to the EMB flag.	Bit 6 of address 0000H in program memory is transferred to the ERB flag, and bit 7 of the address to the EMB flag.
Stack pointer (SP)	Undefined	Undefined
<b>Data Memory (RAM):</b>		
General registers E, A, L, H, X, W, Z, Y	Values retained	Undefined
General-purpose registers	Values retained (1)	Undefined
Bank selection registers (SMB, SRB)	0, 0	0, 0
BSC register (BSC0–BSC)	0	0
<b>Clocks:</b>		
Power control register (PCON)	0	0
Clock output mode register (CLMOD)	0	0
<b>Interrupts:</b>		
Interrupt request flags (IRQx)	0	0
Interrupt enable flags (IEx)	0	0
Interrupt priority flag (IPR)	0	0
Interrupt master enable flag (IME)	0	0
INT0 mode register (IMOD0)	0	0
INT1 mode register (IMOD1) (2)	0	0
INT2 mode register (IMOD2)	0	0

**NOTE:** The value of the 0F8H–0FDH are not retained when a RESET signal is input.

Table 10-1. Hardware Register Values After RESET (Continued)

Hardware Component or Subcomponent	If RESET Occurs During Power-Down Mode	If RESET Occurs During Normal Operation
<b>I/O Ports:</b>		
Output buffers	Off	Off
Output latches	0	0
Port mode flags (PMG)	0	0
Pull-up resistor mode reg (PUMOD1/2)	0	0
Port open-drain enable register (PNE1)	0	0
<b>Basic Timer:</b>		
Count register (BCNT)	Undefined	Undefined
Mode register (BMOD)	0	0
Output enable flag (BOE)	0	0
Timer/Counters 0 and 1:		
Count registers (TCNT0/1)	0	0
Reference registers (TREF0/1)	FFH/FFH	FFH/FFH
Mode registers (TMOD0/1)	0	0
T/C output enable flags (TOE0/1)	0	0
T/C output latch (TOL0/1)	0	0
<b>Watch Timer:</b>		
Watch timer mode register(WMOD)	0	0
Watchdog Timer		
WDT mode register (WDMOD)	A5H	A5H
WDT clear flag (WDTCF)	0	0
<b>DTMF Generator:</b>		
DTMF mode register (DTMR)	0	0
Caller ID		
All registers	0	0



## NOTES

# 11 I/O PORTS

## OVERVIEW

The S3P7588X has one input port and seven I/O ports. Pin addresses for all I/O ports are mapped in bank 15 of the RAM. The contents of I/O port pin latches can be read, written, or tested at the corresponding address using bit manipulation instructions.

S3P7588X has three input pins and 25 configurable I/O pins for a maximum number of 28 I/O pins.

### Port Mode Flags

Port mode flags (PM) are used to configure I/O ports 2 and 3 (port mode group 1), ports 4 and 5 (port mode group 2), ports 6 and 7 (port mode group 3), and port 8 and 9 (port mode group 4) to input or output mode by setting or clearing the corresponding I/O buffer. PMG flags are grouped in four 8-bit registers, and are addressable by 8-bit write instructions only.

### PUMOD Control Register

The pull-up mode registers, PUMOD1 and 2 are 8-bit and 4-bit registers, respectively, used to assign internal pull-up resistors by software to specific I/O ports.

When configurable I/O ports 2 through 9 serves as an output pin, its assigned pull-up resistor is automatically disabled, even though the pin's pull-up resistor is enabled by a corresponding bit setting in the pull-up resistor mode register (PUMOD).

PUMOD1 is addressable by 8-bit write instructions only, PUMOD2 is addressable by 4-bit write instructions only. RESET clears PUMOD register values to logic zero, automatically disconnecting all software-assignable port pull-up resistors.

**Table 11-1. I/O Port Overview**

Port	I/O	Pins	Pin Names	Address	Function Description
1	I	4	P1.1–P1.3	FF1H	4-bit input port. 1-bit and 4-bit read and test is possible. 1-bit pull-up resistors are software assignable
2, 3	I/O	8	P2.0, P2.3 (NOTE) P3.0–P3.3	FF2H FF3H	4-bit I/O ports. 1-bit and 4-bit read/write/test is possible. Ports 2 and 3 pins are individually software configurable as input or output. 4-bit Pull-up resistors are software assignable; pull-up resistors are automatically disabled for output pins. Ports 2 and 3 can be paired for 8-bit data transfer.

**NOTE:** P2.1, P2.2 is dedicated to interface with caller id, and unable to be used from outside chip. (Refer Chapter 7)

Table 11-1. I/O Port Overview (Continued)

Port	I/O	Pins	Pin Names	Address	Function Description
4, 5	I/O	8	P4.0–P4.3 P5.0–P5.3	FF4H FF5H	4-bit I/O ports. 1-bit and 4-bit read/write/test is possible. Port 4 and 5 pins are individually software configurable as input or output. 4-bit pull-up resistors are software assignable; pull-up registers are automatically disabled for output pins. N-Ch open drain or push-pull output may be selected by software. Ports 4 and 5 can be paired for 8-bit data transfer.
6, 7	I/O	8	P6.0–P6.3 P7.0–P7.3	FF6H FF7H	4-bit I/O ports. 1-bit and 4-bit read/write/test is possible. Port 6 and 7 pins are individually software configurable as input or output. 4-bit pull-up resistors are software assignable; pull-up registers are automatically disabled for output pins. Ports 6 and 7 can be paired for 8-bit data transfer.
8, 9 (note)	I/O	8	P8.0–P8.3 P9.0–P9.2	FF8H FF9H	4-bit I/O port. 1-bit and 4-bit read/write and test is possible. Ports 8 and 9 pins are individually software configurable as input or output. 4-bit pull-up resistors are software assignable; pull-up registers are automatically disabled for output pins. Ports 8 and 9 can be paired for 8-bit data transfer.

**NOTE:** Port 8 is dedicated to interface with caller id, and unable to be used from outside chip. (Refer Chap. 7)

Table 11-2. Port Pin Status During Instruction Execution

Instruction Type	Example	Input Mode Status	Output Mode Status
1-bit test 1-bit input 4-bit input 8-bit input	BTST P2.3 LDB C,P1.0 LD A,P7 LD EA,P4	Input or test data at each pin	Input or test data at output latch
1-bit output	BITR P2.3	Output latch contents undefined	Output pin status is modified
4-bit output 8-bit output	LD P2,A LD P6,EA	Transfer accumulator data to the output latch	Transfer accumulator data to the output pin

### PORT MODE FLAGS (PM FLAGS)

Port mode flags (PM) are used to configure I/O ports 2–9 to input or output mode by setting or clearing the corresponding I/O buffer.

For convenient program reference, PM flags are organized into four groups — PMG1, PMG2, PMG3, and PMG4 as shown in Table 11-3. PM flags are addressable by 8-bit write instructions only.

When a PM flag is “0”, the port is set to input mode; when it is “1”, the port is enabled for output. RESET clears all port mode flags to logic zero, automatically configuring the corresponding I/O ports to input mode.

**Table 11-3. Port Mode Group Flags**

PM Group ID	Address	Bit 3	Bit 2	Bit 1	Bit 0
PMG1	FE8H	PM2.3	PM2.2	PM2.1	PM2.0
	FE9H	PM3.3	PM3.2	PM3.1	PM3.0
PMG2	FEAH	PM4.3	PM4.2	PM4.1	PM4.0
	FEBH	PM5.3	PM5.2	PM5.1	PM5.0
PMG3	FECH	PM6.3	PM6.2	PM6.1	PM6.0
	FEDH	PM7.3	PM7.2	PM7.1	PM7.0
PMG4	FEEH	PM8.3	PM8.2	PM8.1	PM8.0
	FEFH	“0”	PM9.2	PM9.1	PM9.0

**NOTE:** If bit = “0”, the corresponding I/O pin is set to input mode. If bit = “1”, the pin is set to output mode: PM4 for port 4 and so on. All flags are cleared to “0” following RESET.

### PROGRAMMING TIP — Configuring I/O Ports to Input or Output

Configure P2.3 and P3 as an output port and the other ports as input ports:

```

BITS      EMB
SMB       15
LD        EA,#0F8H
LD        PMG1,EA      ; P2.3, P3 ←          ← Input
           EA,#00H
           PMG2,EA      P4, P5
LD        PMG4,EA      ; P8, P9 ← Input

```

The pull-up resistor mode registers (PUMOD1 and 2) are 8-bit registers used to assign internal pull-up resistors by software to specific I/O ports.

disabled, even though the pin's pull-up is enabled by a corresponding PUMOD bit setting.

PUMOD1 is addressable by 8-bit write instructions only. PUMOD2 is addressable by 4bit write instructions only. clears PUMOD register values to logic zero, automatically disconnecting all software-assignable port pull-

**Table 11-4. Pull-Up Resistor Mode Register (PUMOD) Organization**

	Address	Bit 3		Bit 1	Bit 0
	FDCH	PUR1.3		PUR1.1	PUR1.0
		PUR5	PUR4		PUR2
PUMOD2		PUR9	PUR8		PUR6

**NOTE:**

port 2, and so on.

 **PROGRAMMING TIP — Enabling and Disabling I/O Port Pull-Up Resistors**

P2–P5 enable pull-up resistors, P1 disable pull-up resistors.

```

BITS      EMB
SMB      15
LD      EA,#0F0H
LD      PUMOD1,EA      ; P2–P5 enable

```

**N-CHANNEL OPEN-DRAIN MODE REGISTER (PNE)**

The n-channel, open-drain mode register (PNE) is used to configure ports 4 and 5 to n-channel open-drain or as push-pull outputs. When a bit in the PNE register is set to "1", the corresponding output pin is configured to n-channel open-drain; when set to "0", the output pin is configured to push-pull. The PNE register consists of an 8-bit register; PNE1 can be addressed by 8-bit write instructions only.

FDAH	PNE4.3	PNE4.2	PNE4.1	PNE4.0	PNE1
FDBH	PNE5.3	PNE5.2	PNE5.1	PNE5.0	

PORT 1 CIRCUIT DIAGRAM

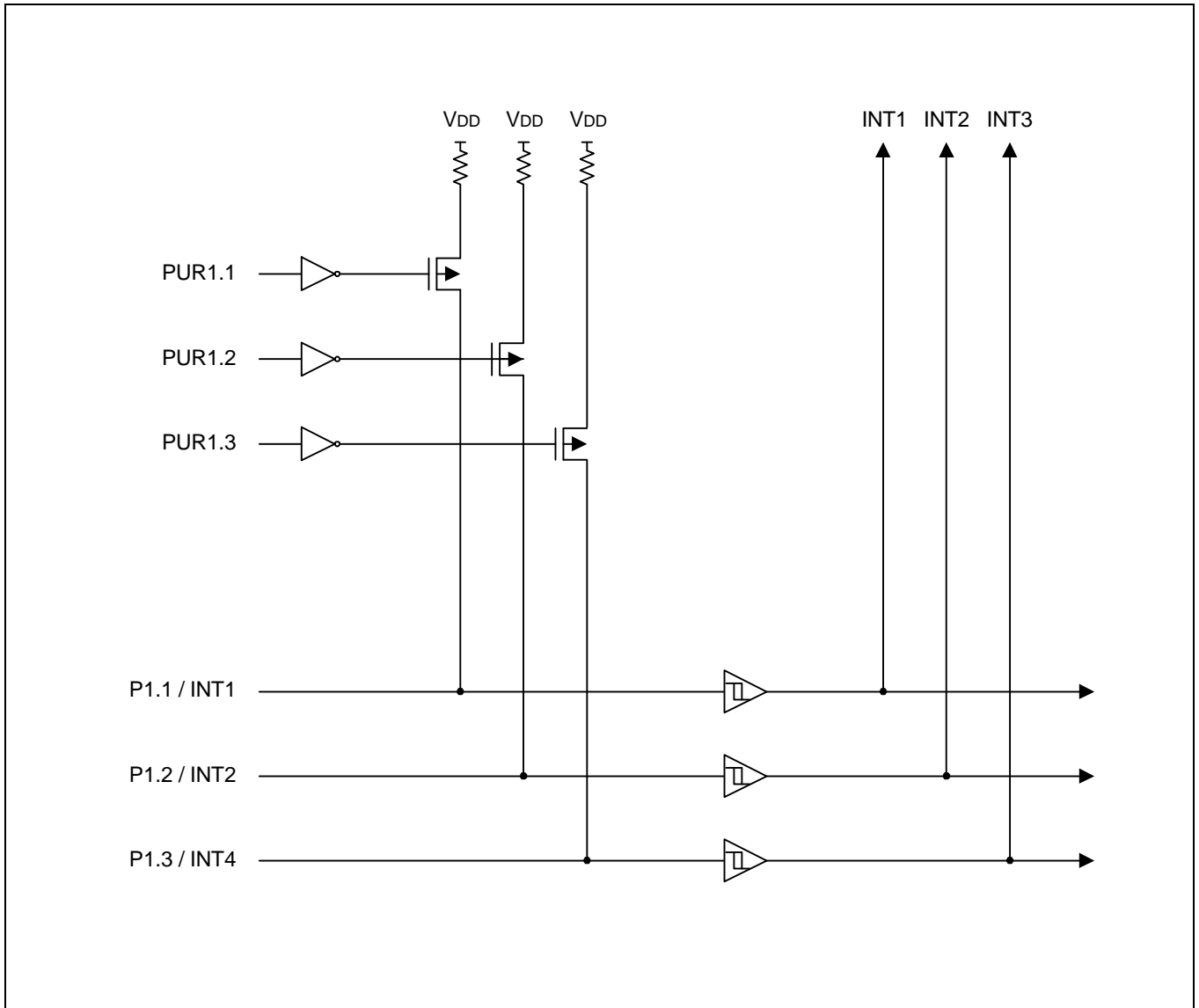


Figure 11-1. Port 1 Circuit Diagram

PORT 2, 3, 6, 7, 8, and 9 CIRCUIT DIAGRAM

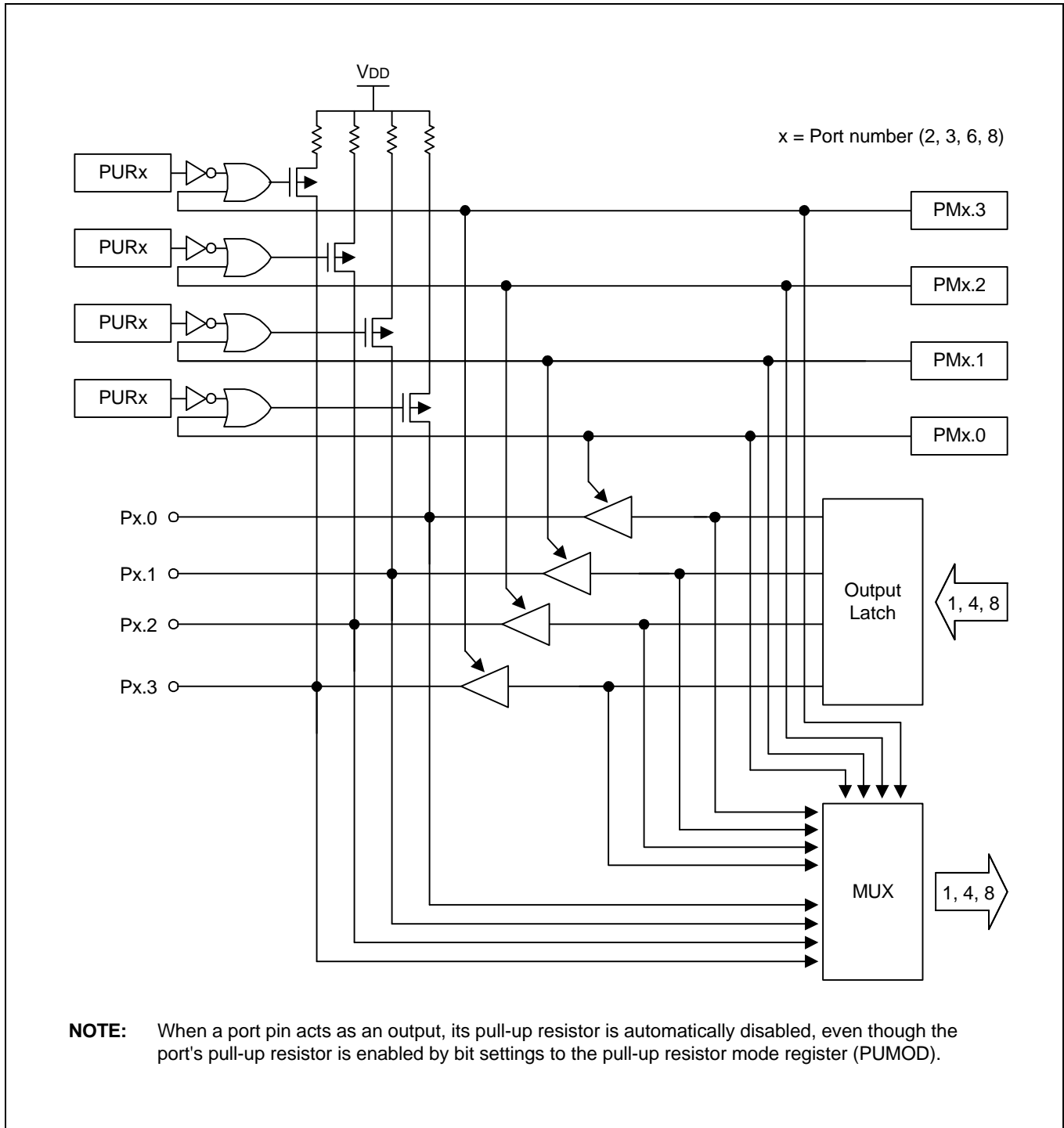


Figure 11-2. Port 2, 3, 6, 7, 8, and 9 Circuit Diagram

PORT 4, 5 CIRCUIT DIAGRAM

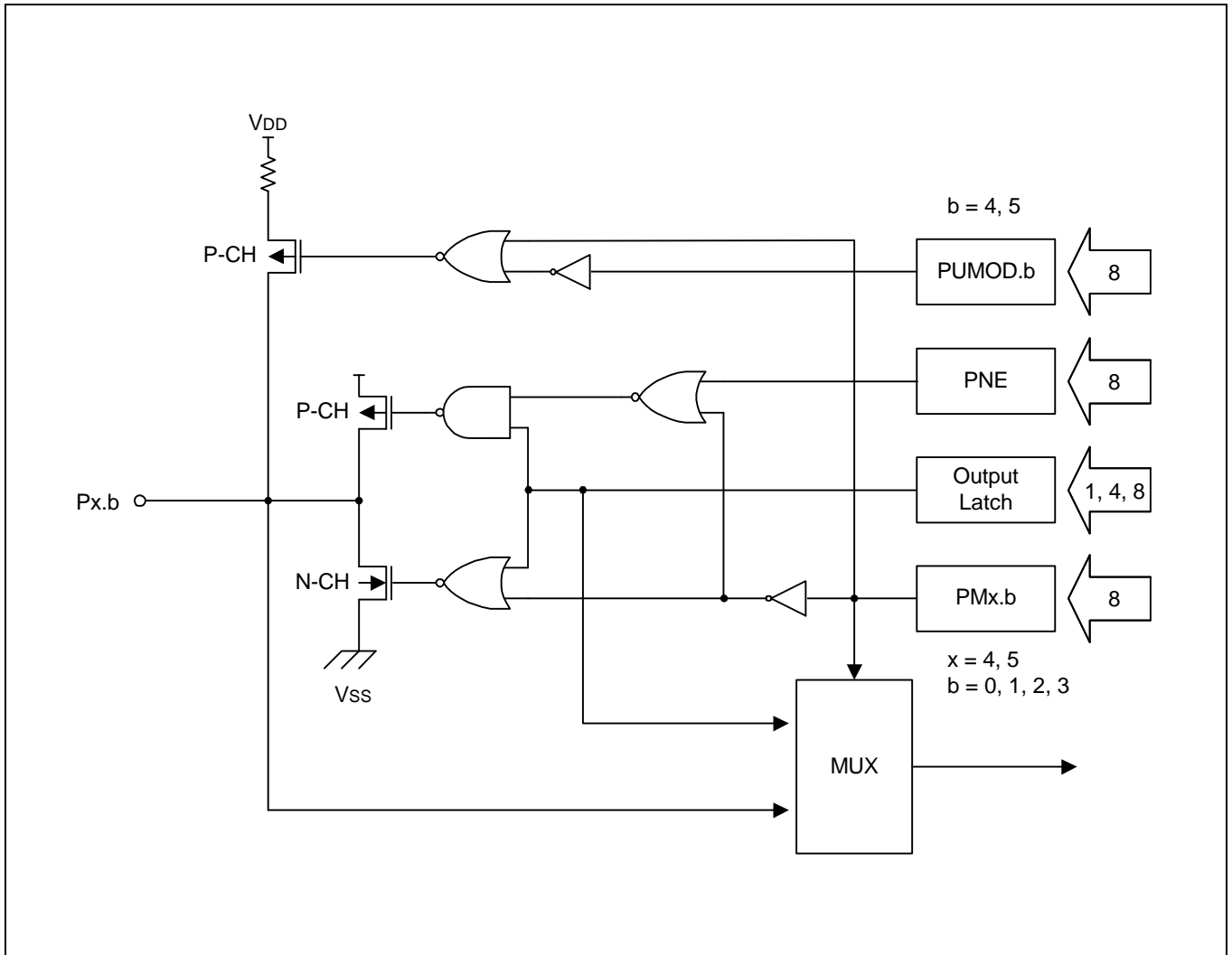


Figure 11-3. Port 4 and 5 Circuit Diagram



NOTES

# 12

## TIMERS and TIMER/COUNTERS

### OVERVIEW

The S3P7588X microcontroller has four timer and timer/counter modules:

- 8-bit basic timer (BT)
- 8-bit timer/counters (TC0, TC1)
- Watch timer (WT)

The 8-bit basic timer (BT) is the microcontroller's main interval timer. It generates an interrupt request at a fixed time interval when the appropriate modification is made to its mode register. When the contents of the basic timer counter register BCNT overflows, a pulse is output to the basic timer output pin, BTCO. The basic timer also functions as a 'watchdog' timer and is used to determine clock oscillation stabilization time when stop mode is released by an interrupt and after a RESET.

The 8-bit timer/counters (TC0, TC1) are programmable timer/counters that are used primarily for event counting and for clock frequency modification and output.

The watch timer (WT) module consists of an 8-bit watch timer mode register, a clock selector, and a frequency divider circuit. Watch timer functions include real-time and watch-time measurement, system clock interval timing, buzzer output generation.

## BASIC TIMER (BT)

### OVERVIEW

The 8-bit basic timer (BT) has six functional components:

- Clock selector logic
- 4-bit mode register (BMOD)
- 8-bit counter register (BCNT)
- Output enable flag (BOE)
- 8-bit watchdog timer mode register (WDMOD)
- Watchdog timer counter clear flag (WDTCF)

The basic timer generates interrupt requests at precise intervals, based on the frequency of the system clock. Timer pulses are output from the basic timer's counter register BCNT to the output pin BTCO when an overflow occurs in the counter register BCNT. You can use the basic timer as a "watchdog" timer for monitoring system events or use BT output to stabilize clock oscillation when stop mode is released by an interrupt and following RESET. Bit settings in the basic timer mode register BMOD turns the BT module on and off, selects the input clock frequency, and controls interrupt or stabilization intervals.

### Interval Timer Function

The basic timer's primary function is to measure elapsed time intervals. The standard time interval is equal to 256 basic timer clock pulses.

To restart the basic timer, one bit setting is required: bit 3 of the mode register BMOD should be set to logic one. The input clock frequency and the interrupt and stabilization interval are selected by loading the appropriate bit values to BMOD.2–BMOD.0.

The 8-bit counter register, BCNT, is incremented each time a clock signal is detected that corresponds to the frequency selected by BMOD. BCNT continues incrementing as it counts BT clocks until an overflow occurs ( $\geq 255$ ). An overflow causes the BT interrupt request flag (IRQB) to be set to logic one to signal that the designated time interval has elapsed. An interrupt request is then generated, BCNT is cleared to logic zero, and counting continues from 00H.

### Watchdog Timer Function

The basic timer can also be used as a "watchdog" timer to signal the occurrence of system or program operation error. For this purpose, instruction that clear the watchdog timer (BITS WDTCF) should be executed at proper points in a program within given period. If an instruction that clears the watchdog timer is not executed within the given period and the watchdog timer overflows, reset signal is generated and the system restarts with reset status. An operation of watchdog timer is as follows:

- Write some values (except #5AH) to watchdog timer mode register, WDMOD
- If WDCNT overflows, system reset is generated.

### Oscillation Stabilization Interval Control

Bits 2–0 of the BMOD register are used to select the input clock frequency for the basic timer. This setting also determines the time interval (also referred to as ‘wait time’) required to stabilize clock signal oscillation when stop mode is released by an interrupt. When a RESET signal is inputted, the standard stabilization interval for system clock oscillation following the RESET is 31.3 ms at 4.19MHz.

**Table 12-1. Basic Timer Register Overview**

Register Name	Type	Description	Size	RAM Address	Addressing Mode	Reset Value
BMOD	Control	Controls the clock frequency (mode) of the basic timer; also, the oscillation stabilization interval after stop mode release or RESET	4-bit	F85H	4-bit write-only; BMOD.3: 1-bit writeable	“0”
BCNT	Counter	Counts clock pulses matching the BMOD frequency setting	8-bit	F86H–F87H	8-bit read-only	U (NOTE)
BOE	Flag	Controls output of basic timer output latch to the BTCO pin	1-bit	F92H.1	1-, 4-bit read/write	“0”
WDMOD	Control	Controls watchdog timer operation.	8-bit	F98H–F99H	8-bit write-only	A5H
WDTCF	Control	Clears the watchdog timer’s counter.	1-bit	F9AH.3	1-, 4-bit write-only	“0”

**NOTE:** 'U' means the value is undetermined after a RESET.

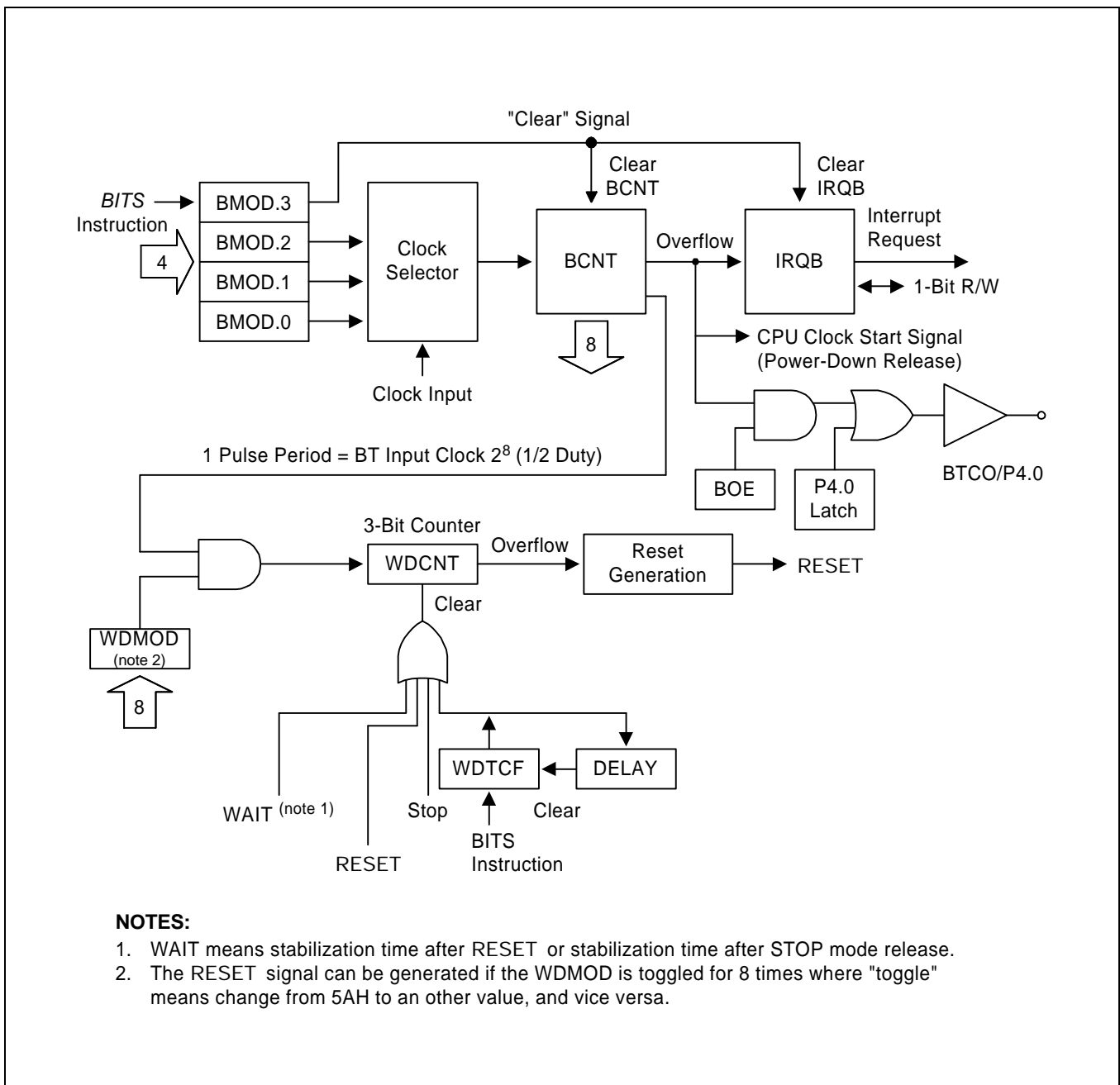


Figure 12-1. Basic Timer Circuit Diagram

**BASIC TIMER MODE REGISTER (BMOD)**

The basic timer mode register, BMOD, is a 4-bit write-only register. Bit 3, the basic timer start control bit, is also 1-bit addressable. All BMOD values are set to logic zero following RESET and interrupt request signal generation is set to the longest interval. (BT counter operation cannot be stopped.) BMOD settings have the following effects:

- Restart the basic timer;
- Control the frequency of clock signal input to the basic timer;
- Determine time interval required for clock oscillation to stabilize following the release of stop mode by an interrupt.

By loading different values into the BMOD register, you can dynamically modify the basic timer clock frequency during program execution. Four BT frequencies, ranging from  $f_x/2^{12}$  to  $f_x/2^5$ , are selectable. Since BMOD's reset value is logic zero, the default clock frequency setting is  $f_x/2^{12}$ .

The most significant bit of the BMOD register, BMOD.3, is used to restart the basic timer. When BMOD.3 is set to logic one by a 1-bit write instruction, the contents of the BT counter register (BCNT) and the BT interrupt request flag (IRQB) are both cleared to logic zero, and timer operation restarts.

The combination of bit settings in the remaining three registers — BMOD.2, BMOD.1, and BMOD.0 — determine the clock input frequency and oscillation stabilization interval.

**Table 12-2. Basic Timer Mode Register (BMOD) Organization**

BMOD.3			Basic Timer Start Control Bit	
1			Start basic timer; clear IRQB, BCNT, and BMOD.3 to "0"	

BMOD.2	BMOD.1	BMOD.0	Basic Timer Input Clock	Oscillation Stabilization
0	0	0	$f_x/2^{12}$ (1.02kHz)	$2^{20}/f_x$ (250ms)
0	1	1	$f_x/2^9$ (8.18kHz)	$2^{17}/f_x$ (31.3ms)
1	0	1	$f_x/2^7$ (32.7kHz)	$2^{15}/f_x$ (7.82ms)
1	1	1	$f_x/2^5$ (131kHz)	$2^{13}/f_x$ (1.95ms)

**NOTES:**

1. Clock frequencies and oscillation stabilization assume a system oscillator clock frequency ( $f_x$ ) of 4.19MHz.
2.  $f_x$  = system clock frequency.
3. Oscillation stabilization time is the time required to stabilize clock signal oscillation after stop mode is released. The data in the table column 'Oscillation Stabilization' can also be interpreted as "Interrupt Interval Time."
4. The standard stabilization time for system clock oscillation following a RESET is 31.3 ms at 4.19MHz.

**BASIC TIMER COUNTER (BCNT)**

BCNT is an 8-bit counter for the basic timer. It can be addressed by 8-bit read instructions. RESET leaves the BCNT counter value undetermined. BCNT is automatically cleared to logic zero whenever the BMOD register control bit (BMOD.3) is set to "1" to restart the basic timer. It is incremented each time a clock pulse of the frequency determined by the current BMOD bit settings is detected.

When BCNT has incremented to hexadecimal 'FFH' ( $\geq 255$  clock pulses), it is cleared to '00H' and an overflow is generated. The overflow causes the interrupt request flag, IRQB, to be set to logic one. When the interrupt request is generated, BCNT immediately resumes counting incoming clock signals.

**NOTE**

Always execute a BCNT read operation twice to eliminate the possibility of reading unstable data while the counter is incrementing. If, after two consecutive reads, the BCNT values match, you can select the latter value as valid data. Until the results of the consecutive reads match, however, the read operation must be repeated until the validation condition is met.

**BASIC TIMER OUTPUT ENABLE FLAG (BOE)**

The basic timer output enable flag (BOE) enables and disables basic timer output to the BTCO pin at I/O port 4 (P4.0). When BOE is logic zero, basic timer output to the BTCO pin is disabled; when it is logic one, BT output to the BTCO pin is enabled. A RESET clears the BOE flag to "0", disabling basic timer output to the BTCO pin. When the BOE flag is set to "1" and the BCNT register overflows, the overflow signal is sent to the BTCO pin. BOE can be addressed by 1-bit read and write instructions.

	Bit 3	Bit 2	Bit 1	Bit 0
F92H	TOE1	TOE0	<b>BOE</b>	0

**BASIC TIMER OPERATION SEQUENCE**

The basic timer's sequence of operations may be summarized as follows:

1. Set BMOD.3 to logic one to restart the basic timer
2. BCNT is then incremented by one after each clock pulse corresponding to BMOD selection
3. BCNT overflows if BCNT = 255 (BCNT = FFH)
4. When an overflow occurs, the IRQB flag is set by hardware to logic one
5. The interrupt request is generated
6. BCNT is then cleared by hardware to logic zero
7. Basic timer resumes counting clock pulses

**PROGRAMMING TIP — Using the Basic Timer**

- To read the basic timer count register (BCNT):

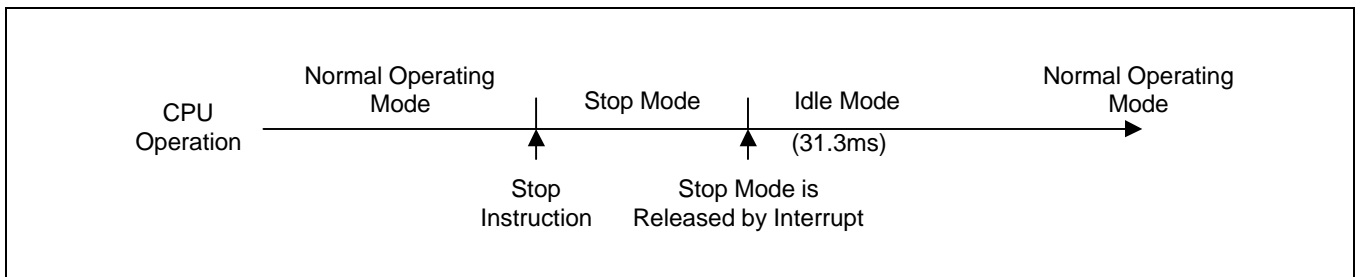
```

BCNTR
BITS      EMB
SMB      15
LD        EA,BCNT
LD        YZ,EA
LD        EA,BCNT
CPSE     EA,YZ
JR        BCNTR
    
```

- When stop mode is released by an interrupt, set the oscillation stabilization interval to 31.3ms:

```

BITS      EMB
SMB      15
LD        A,#0BH
LD        BMOD,A           ; Wait time is 31.3ms
STOP      ; Set stop power-down mode
NOP
NOP
NOP
    
```



- To set the basic timer interrupt interval time to 1.95ms (at 4.19MHz):

```

BITS      EMB
SMB      15
LD        A,#0FH
LD        BMOD,A
EI
BITS      IEB           ; Basic timer interrupt enable flag is set to "1"
    
```

- Clear BCNT and the IRQB flag and restart the basic timer:

```

BITS      EMB
SMB      15
BITS      BMOD.3
    
```



**WATCHDOG TIMER MODE REGISTER (WDMOD)**

The watchdog timer mode register, WDMOD, is a 8-bit write-only register. WDMOD register controls to enable or disable the watchdog function. WDMOD values are set to logic "A5H" following RESET and this value enables the watchdog timer. Watchdog timer is set to the longest interval because BT overflow signal is generated with the longest interval.

WDMOD	Watchdog Timer Enable/Disable Control
5AH	Disable watchdog timer function
Any other value	Enable watchdog timer function

**WATCHDOG TIMER COUNTER (WDCNT)**

The watchdog timer counter, WDCNT, is a 3-bit counter. WDCNT is automatically cleared to logic zero, and restarts whenever the WDTCF register control bit is set to "1". RESET, stop, and wait signal clears the WDCNT to logic zero also.

WDCNT increments each time a clock pulse of the overflow frequency determined by the current BMOD bit setting is generated. When WDCNT has incremented to hexadecimal '07H', it is cleared to '00H' and an overflow is generated. The overflow causes the system RESET. When the interrupt request is generated, BCNT immediately resumes counting incoming clock signals.

**WATCHDOG TIMER COUNTER CLEAR FLAG (WDTCF)**

The watchdog timer counter clear flag, WDTCF, is a 1-bit write instruction. When WDTCF is set to one, it clears the WDCNT to zero and restarts the WDCNT. WDTCF register bits 2–0 are always logic zero.

**Table 12-3. Watchdog Timer Interval Time**

BMOD	BT Input Clock	WDCNT Input Clock	WDT Interval Time	Main Clock
x000b	$212/fx$	$212/fx \times 28$	$212/fx \times 28 \times 23$	2 second
x011b	$29/fx$	$29/fx \times 28$	$29/fx \times 28 \times 23$	250 ms
x101b	$27/fx$	$27/fx \times 28$	$27/fx \times 28 \times 23$	62.5 ms
x111b	$25/fx$	$25/fx \times 28$	$25/fx \times 28 \times 23$	15.6 ms

**NOTES:**

1. Clock frequencies assume a system oscillator clock frequency (fx) of 4.19MHz
2. fx = system clock frequency.

 PROGRAMMING TIP — Using the Watchdog Timer

```

RESET    DI
         LD      EA,#00H
         LD      SP,EA
         .
         .
         .
         LD      A,#0DH           ; WDCNT input clock is 7.82ms
         LD      BMOD,A
         .
         .
         .
MAIN     BITS    WDTCF           ; Main routine operation period must be shorter than
                                   ; watchdog
                                   ; timer's period
         .
         .
         .
         JP      MAIN

```

## 8-BIT TIMER/COUNTERS 0 AND 1 (TC0, TC1)

### OVERVIEW

The S3P7588X TC0 and TC1 are identical except that they have different counter clock sources, which are controlled by the TMODn register. Timer/counters 0 and 1 (TC0, TC1) are used to count system 'events' by identifying the transition (high-to-low or low-to-high) of incoming square wave signals. To indicate that an event has occurred, or that a specified time interval has elapsed, TC generates an interrupt request. By counting signal transitions and comparing the current counter value with the reference register value, TC can be used to measure specific time intervals.

TC has a reloadable counter that consists of two parts: an 8-bit reference register, TREFn (n = 0, 1) into which you write the counter reference value, and an 8-bit counter register, TCNTn (n = 0, 1) whose value is automatically incremented by counter logic.

8-bit mode register, TMODn (n = 0, 1), is used to activate the timer/counter and to select the basic clock frequency to be used for timer/counter operations. To dynamically modify the basic frequency, new values can be loaded into the TMODn register during program execution.

### TC FUNCTION SUMMARY

8-bit programmable timer	Generates interrupts at specific time intervals based on the selected clock frequency.
External event counter	Counts various system "events" based on edge detection of external clock signals at the TC input pin, TCLn (n = 0, 1).
Arbitrary frequency output	Outputs clock frequencies to the TC output pin, TCLOn (n = 0, 1).
External signal divider	Divides the frequency of an incoming external clock signal according to a modifiable reference value (TREFn), and outputs the modified frequency to the TCLOn pin.

**TC COMPONENT SUMMARY**

Mode register (TMODn)	Activates the timer/counter and selects the internal clock frequency or the external clock source at the TCLn pin.
Reference register (TREFn)	Stores the reference value for the desired number of clock pulses between interrupt requests.
Counter register (TCNTn)	Counts internal or external clock pulses based on the bit settings in TMODn and TREFn.
Clock selector circuit	Together with the mode register (TMODn), lets you select one of four internal clock frequencies or an external clock.
8-bit comparator	Determines when to generate an interrupt by comparing the current value of the counter register (TCNTn) with the reference value previously programmed into the reference register (TREFn).
Output latch (TOLn)	When the contents of the TCNTn and TREFn registers coincide, the timer/counter interrupt request flag (IRQn) is set to "1", the status of TOLn is inverted, and an interrupt is generated.
Output enable flag (TOEn)	Must be set to logic one before the contents of the TOLn latch can be output to TCLOn.
Interrupt request flag (IRQn)	Cleared when TC operation starts and the TC interrupt service routine is executed and set to one whenever the counter value and reference value coincide.
Interrupt enable flag (IETn)	Must be set to logic one before the interrupt requests generated by timer/counters can be processed.

**Table 12-4. TC Register Overview**

Register Name	Type	Description	Size	RAM Address	Addressing Mode	Reset Value
TMOD0 TMOD1	Control	Controls TC0 and TC1 enable/disable (bit 2); clears and resumes counting operation (bit 3); sets input clock and clock frequency (bits 6–4)	8-bit	F90H–F91H FA0H–FA1H	8-bit write only; (TMODn.3 is also 1-bit writeable)	"0"
TCNT0 TCNT1	Counter	Counts clock pulses matching the TMODn frequency setting	8-bit	F94H–F95H FA4H–FA5H	8-bit read-only	"0"
TREF0 TREF1	Reference	Stores reference value for the timer/counters interval setting	8-bit	F96H–F97H FA8H–FA9H	8-bit write-only	FFH
TOE0 TOE1	Flag	Controls timer/counters output to the TCLOn pin	1-bit	F92H.2 F92H.3	1-, 4-bit read/write	"0"

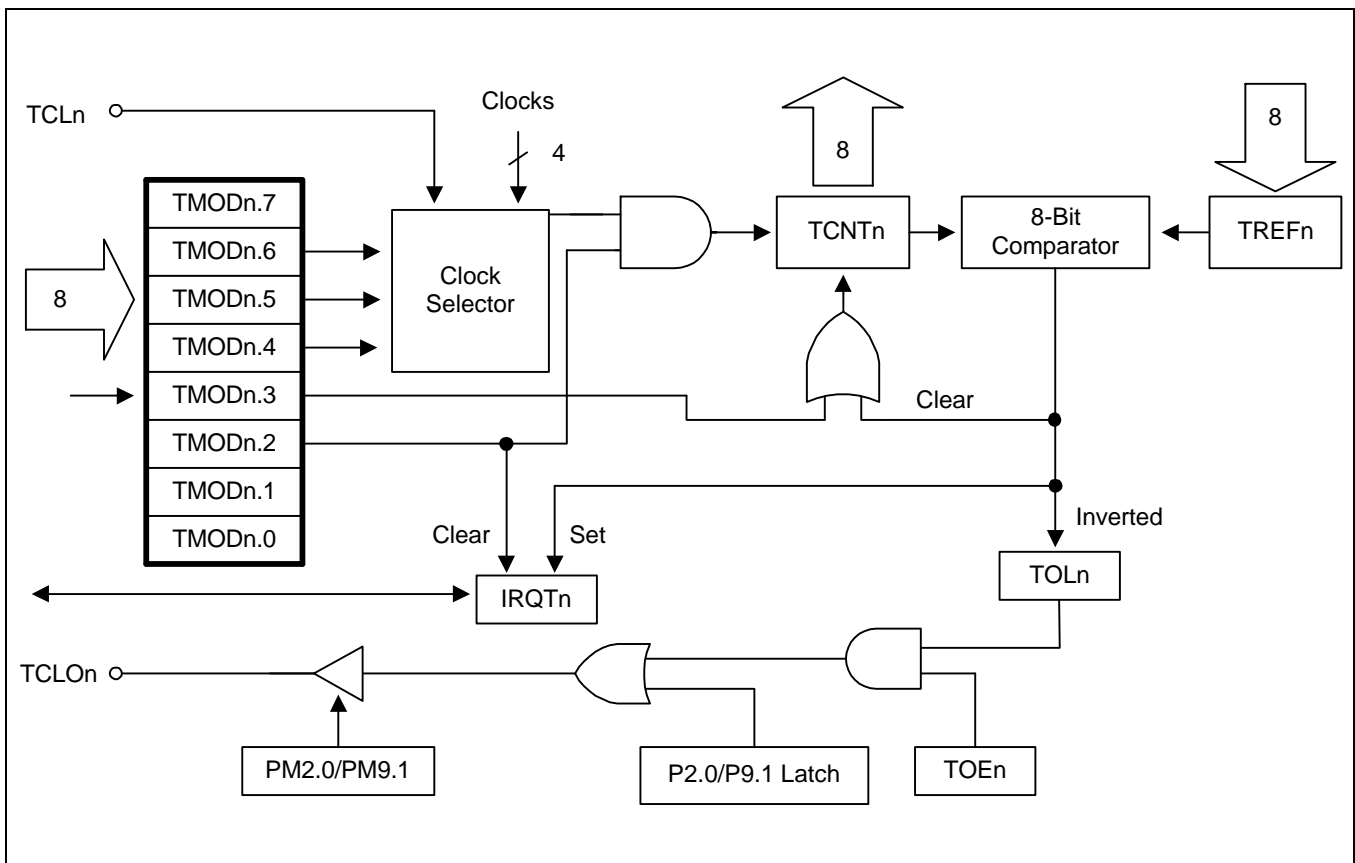


Figure 12-2. TC Circuit Diagram

### TC ENABLE/DISABLE PROCEDURE

#### Enable Timer/Counter

- Set TMODn.2 to logic one
- Set the TC interrupt enable flag IETn to logic one
- Set TMODn.3 to logic one

TCNTn and IRQTn are cleared to logic zero, and timer/counter operation starts.

#### Disable Timer/Counter

- Set TMODn.2 to logic zero

Clock signal input to the counter register TCNTn is halted. The current TCNTn value is retained and can be read if necessary.

## TC PROGRAMMABLE TIMER/COUNTER FUNCTION

Timer/counters can be programmed to generate interrupt requests at various intervals based on the selected system clock frequency. Its 8-bit TC mode register TMODn is used to activate the timer/counter and to select the clock frequency. The reference register TREFn stores the value for the number of clock pulses to be generated between interrupt requests. The counter register, TCNTn, counts the incoming clock pulses, which are compared to the TREFn value as TCNTn is incremented. When there is a match ( $TREFn = TCNTn$ ), an interrupt request is generated.

To program timer/counter to generate interrupt requests at specific intervals, choose one of four internal clock frequencies (divisions of the system clock,  $fx$ ) and load a counter reference value into the reference register. The count register is incremented each time an internal counter pulse is detected with the reference clock frequency specified by TMODn.4–TMODn.6 settings. To generate an interrupt request, the TC interrupt request flag (IRQTn) should be set to logic one, the status of TOLn is inverted, and the interrupt is generated. The content of the counter register is then cleared to 00H and TC continues counting. The interrupt request mechanism for TC includes an interrupt enable flag (IETn) and an interrupt request flag (IRQTn).

## TC OPERATION SEQUENCE

The general sequence of operations for using TC can be summarized as follows:

1. Set TMODn.2 to "1" to enable TC0 and TC1
2. Set TMODn.6 to "1" to enable the system clock ( $fx$ ) input
3. Set TMODn.5 and TMODn.4 bits to desired internal frequency ( $fx/2^n$ )
4. Load a value to TREFn to specify the interval between interrupt requests
5. Set the TC interrupt enable flag (IETn) to "1"
6. Set TMODn.3 bit to "1" to clear TCNTn and IRQTn, and start counting
7. TCNTn increments with each internal clock pulse
8. When the comparator shows  $TCNTn = TREFn$ , the IRQTn flag is set to "1"
9. Output latch (TOLn) logic toggles high or low
10. Interrupt request is generated
11. TCNTn is cleared to 00H and counting resumes
12. Programmable timer/counter operation continues until TMODn.2 is cleared to "0".

**TC EVENT COUNTER FUNCTION**

Timer/counters can monitor or detect system 'events' by using the external clock input at the TCLn pin as the counter source. The TC mode register selects rising or falling edge detection for incoming clock signals. The counter register is incremented each time the selected state transition of the external clock signal occurs.

With the exception of the different TMODn.4–TMODn.6 settings, the operation sequence for TC's event counter function is identical to its programmable timer/counter function. To activate the TC event counter function,

- Set TMODn.2 to "1" to enable TC;
- Clear TMODn.6 to "0" to select the external clock source at the TCLn pin;
- Select TCLn edge detection for rising or falling signal edges by loading the appropriate values to TMODn.5 and TMODn.4.
- P3.0 and P3.1 must be set to input mode.

**Table 12-5. TMODn Settings for TCLn Edge Detection**

<b>TMODn.5</b>	<b>TMODn.4</b>	<b>TCLn Edge Detection</b>
0	0	Rising edges
0	1	Falling edges

## TC CLOCK FREQUENCY OUTPUT

Using timer/counters, a modifiable clock frequency can be output to the TC clock output pin, TClOn. To select the clock frequency, load the appropriate values to the TC mode register, TModn. The clock interval is selected by loading the desired reference value into the reference register TREFn. In summary, the operational sequence required to output a TC-generated clock signal to the TClOn pin is as follows:

1. Load a reference value to TREFn.
2. Set the internal clock frequency in TModn.
3. Initiate TCn clock output to TClOn (TModn.2 = "1").
4. Set port 2, port9 mode flag (PM2.0 and PM 9.1) to "1".
5. Set P2.0 and P9.1 output latches to "0".
6. Set TOEn flag to "1".

Each time the contents of TCNTn and TREFn coincide and an interrupt request is generated, the state of the output latch TOLn is inverted and the TC-generated clock signal is output to the TClOn pin.

### PROGRAMMING TIP — TC0 Signal Output to the TCLO0 Pin

Output a 30 ms pulse width signal to the TCLO0 pin:

BITS	EMB	
SMB	15	
LD	EA,#68H	
LD	TREF0,EA	
LD	EA,#4CH	
LD	TMOD0,EA	
LD	EA,#01H	
LD	PMG1,EA	; P2.0 ← output mode
BITR	P2.0	; P2.0 clear
BITS	TOE0	



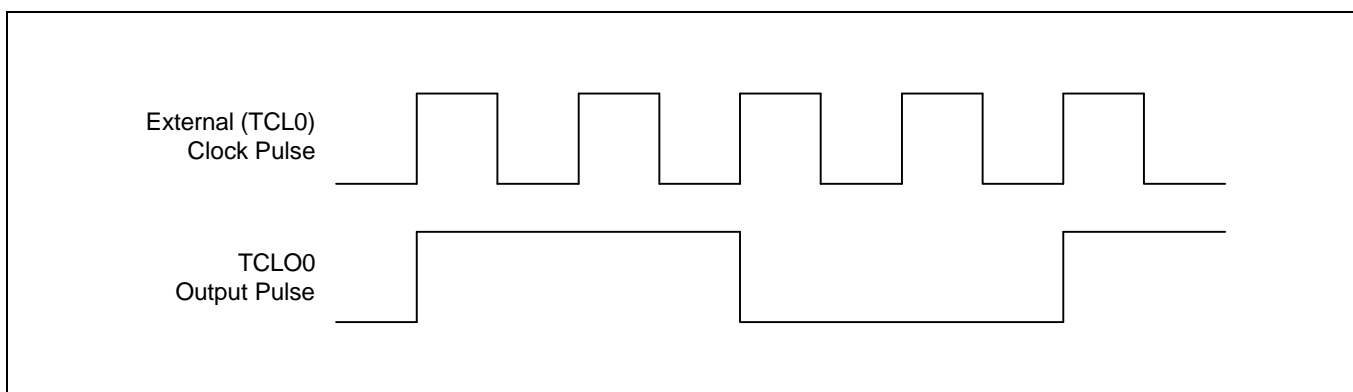
**TC EXTERNAL INPUT SIGNAL DIVIDER**

By selecting an external clock source and loading a reference value into the TC reference register, TREFn, you can divide the incoming clock signal by the TREFn value and then output this modified clock frequency to the TClOn pin. The sequence of operations used to divide external clock input can be summarized as follows:

1. Load a signal divider value to the TREFn register
2. Clear TMODn.6 to "0" to enable external clock input at the TClN pin
3. Set TMODn.5 and TMODn.4 to desired TClN signal edge detection
4. Set port 2, port 9 mode flag (PM2.0, PM9.1) to output ("1")
5. Set P2.0 and P9.1 output latches to "0"
6. Set TOEn flag to "1" to enable output of the divided frequency to the TClOn pin

 **PROGRAMMING TIP — External TCl0 Clock Output to the TClO0 Pin**

Output external TCl0 clock pulse to the TClO0 pin (divide by four):



```

BITS      EMB
SMB       15
LD        EA,#01H
LD        TREF0,EA
LD        EA,#0CH
LD        TMOD0,EA
LD        EA,#01H
LD        PMG1,EA      ; P2.0 ← output mode
BITR      P2.0         ; P2.0 clear
BITS      TOE0

```

**TC MODE REGISTER (TMODn)**

TMODn are the 8-bit mode control registers for timer/counter 0 and 1. They are addressable by 8-bit write instructions. One bit, TMODn.3, is also 1-bit writeable. RESET clears all TMODn bits to logic zero and disables TC operations.

F90H	TMOD0.3	TMOD0.2	"0"	"0"	TMOD0
F91H	"0"	TMOD0.6	TMOD0.5	TMOD0.4	
FA0H	TMOD1.3	TMOD1.2	"0"	"0"	TMOD1
FA1H	"0"	TMOD1.6	TMOD1.5	TMOD1.4	

TMODn.2 is the enable/disable bit for timer/counter 0 and 1. When TMODn.3 is set to "1", the contents of TCNTn and IRQTn are cleared, counting starts from 00H, and TMODn.3 is automatically reset to "0" for normal TC operation. When TC operation stops (TMODn.2 = "0"), the contents of the counter register TCNTn are retained until TC is re-enabled.

The TMODn.6, TMODn.5, and TMODn.4 bit settings are used together to select the TC clock source. This selection involves two variables:

- Synchronization of timer/counter operations with either the rising edge or the falling edge of the clock signal input at the TCLn pin, and
- Selection of one of four frequencies, based on division of the incoming system clock frequency, for use in internal TC operation.

**Table 12-6. TC Mode Register (TMODn) Organization**

Bit Name	Setting	Resulting TC0 Function	Address
TMODn.7	0	Always logic zero	F91H (TMOD0)
TMODn.6 TMODn.5 TMODn.4	0,1	Specify input clock edge and internal frequency	FA1H (TMOD1)
TMODn.3	1	Clear TCNTn and IRQTn. TOLn is remained and resume counting immediately (This bit is automatically cleared to logic zero immediately after counting resumes.)	F90H (TMOD0) FA0H (TMOD1)
TMODn.2	0	Disable timer/counter; retain TCNTn contents	
	1	Enable timer/counter	
TMODn.1	0	Always logic zero	
TMODn.0	0	Always logic zero	

Table 12-7. TMODn.6, TMODn.5, and TMODn.4 Bit Settings

TMODn.6	TMODn.5	TMODn.4	TC0 Counter Source	TC1 Counter Source
0	0	0	External clock input (TCL0) on rising edges	External clock input (TCL1) on rising edges
0	0	1	External clock input (TCL0) on falling edges	External clock input (TCL1) on falling edges
1	0	0	$f_x/2^{10}$ (4.09kHz)	$f_x/2^{12}$ (1.02kHz)
1	0	1	$f_x/2^6$ (65.5kHz)	$f_x/2^{10}$ (4.09kHz)
1	1	0	$f_x/2^4$ (262kHz)	$f_x/2^8$ (16.4kHz)
1	1	1	$f_x = 4.19\text{MHz}$	$f_x/2^6$ (65.5kHz)

**NOTE:** 'fx' = system clock of 4.19MHz.

### PROGRAMMING TIP — Restarting TC0 Counting Operation

1. Set TC0 timer interval to 4.09kHz:

```

BITS      EMB
SMB      15
LD        EA,#4CH
LD        TMOD0,EA
EI
BITS      IET0

```

2. Clear TCNT0 and IRQT0, TOL0 is remained and restart TC0 counting operation:

```

BITS      EMB
SMB      15
BITS      TMOD0.3

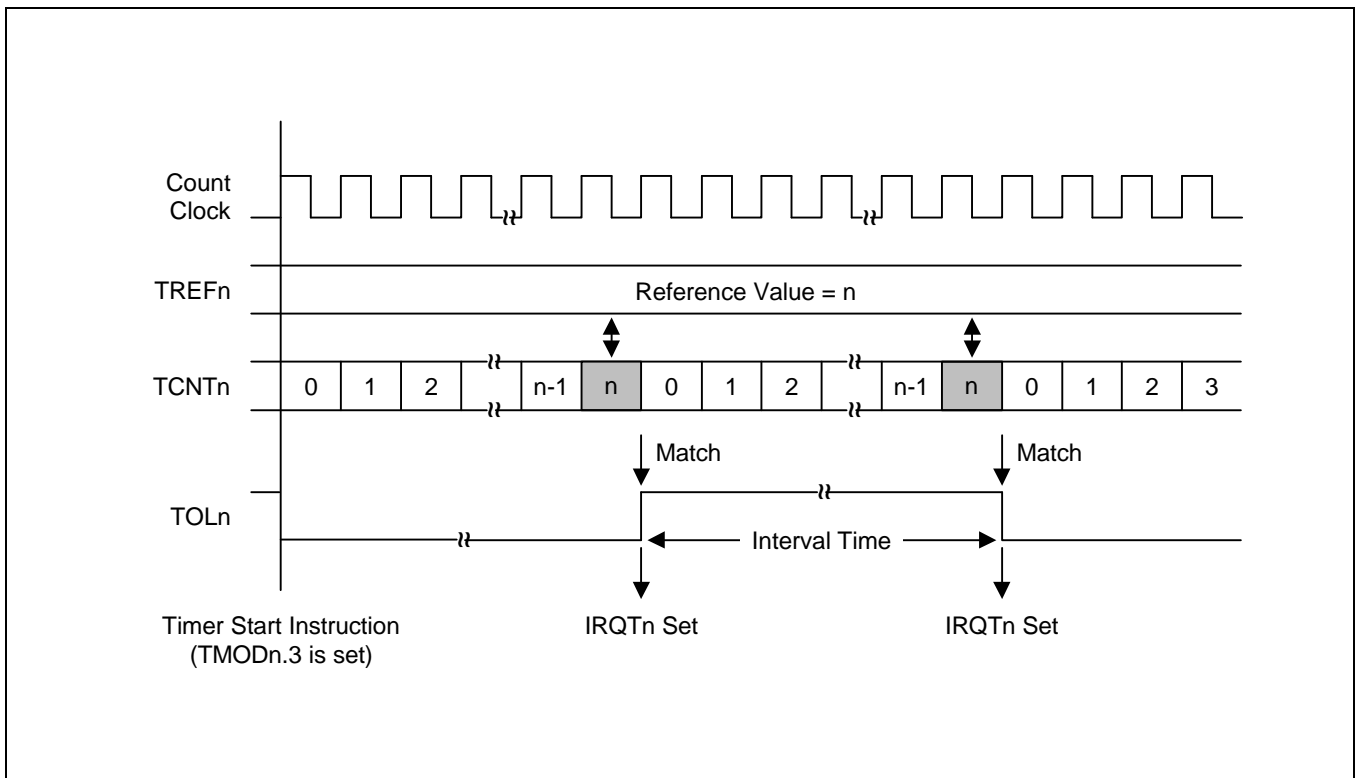
```

**TC COUNTER REGISTER (TCNTn)**

The 8-bit counter register for TC, TCNTn, is read-only and can be addressed by 8-bit RAM control instructions. RESET sets all counter register values to logic zero (00H).

Whenever TMODn.3 is enabled, TCNTn is cleared to logic zero and counting resumes. The TCNTn register value is incremented each time an incoming clock signal is detected that matches the signal edge and frequency setting of the TMODn register (specifically, TMODn.6–TMODn.4).

Each time TCNTn is incremented, the new value is compared to the reference value stored in the reference register, TREFn. When TCNTn = TREFn, an overflow occurs in the counter register, the interrupt request flag, IRQTn, is set to logic one, and an interrupt request is generated to indicate that the specified timer/counter interval has elapsed.



**Figure 12-3. TC Timing Diagram**

### TC REFERENCE REGISTER (TREFn)

The TC reference register, TREFn, is an 8-bit write-only register. RESET initializes the TREFn value to 'FFH'.

TREFn is used to store a reference value to be compared to the incrementing TCNTn register in order to identify an elapsed time interval. Reference values will differ depending upon the specific function that TC is being used to perform — as a programmable timer/counter, event counter, clock signal divider, or arbitrary frequency output source.

During timer/counter operation, the value loaded into the reference register is compared to the counter value. When TCNTn = TREFn, the TC output latch (TOLn) is inverted and an interrupt request is generated to signal the interval or event. The TREFn value, together with the TMODn clock frequency selection, determines the specific TC timer interval. Use the following formula to calculate the correct value to load to the TREFn reference register:

$$\text{TC timer interval} = (\text{TREFn value} + 1) \times \frac{1}{\text{TMODn frequency setting}}$$

(assuming a TREFn value  $\neq$  0)

The 1-bit timer/counter output enable flag TOEn controls output from timer/counter to the TCLOn pin. TOEn is addressable by 1-bit read and write instructions.


	Bit 3	Bit 2	Bit 1	Bit 0
F92H	TOE1	TOE0	BOE	0

When you set the TOEn flag to "1", the contents of TOLn can be output to the TCLOn pin. Whenever a RESET occurs, TOEn is automatically set to logic zero, disabling all TC output.

### TC OUTPUT LATCH (TOLn)

TOLn is the output latch for timer/counter 0 and 1. When the 8-bit comparator detects a correspondence between the value of the counter register TCNTn and the reference value stored in the TREFn register, the TOLn value is inverted — the latch toggles high-to-low or low-to-high. Whenever the state of TOLn is switched, the TC signal is output. TC output is directed to the TCLOn pin.

Assuming TC is enabled, when bit 3 of the TMODn register is set to "1", the TOLn latch is remained, the counter register, TCNTn and the interrupt request flag, IRQTn are cleared, and counting resumes immediately. When TCn is disabled (TMODn.2 = "0"), the contents of the TOLn latch are retained and can be read, if necessary.

 **PROGRAMMING TIP — Setting a TC0 Timer Interval**

To set a 30ms timer interval for TC0, given  $f_x = 3.58\text{MHz}$ , follow these steps.

1. Select the timer/counter 0 mode register with a maximum setup time of 73.3ms (assume the TC0 counter clock =  $f_x/2^{10}$ , and TREF0 is set to FFH):

2. Calculate the TREF0 value:

$$30 \text{ ms} = \frac{\text{TREF0 value} + 1}{3.49 \text{ kHz}}$$

$$\text{TREF0} + 1 = \frac{30 \text{ ms}}{286 \mu\text{s}} = 104.8 = 69\text{H}$$

$$\text{TREF0 value} = 69\text{H} - 1 = 68\text{H}$$

3. Load the value 68H to the TREF0 register:

```

BITS      EMB
SMB      15
LD        EA,#68H
LD        TREF0,EA
LD        EA,#4CH
LD        TMOD0,EA

```

## WATCH TIMER

### OVERVIEW

The watch timer is a multi-purpose timer consisting of three basic components:

- 8-bit watch timer mode register (WMOD)
- Clock selector
- Frequency divider circuit

Watch timer functions include real-time and watch-time measurement and interval timing for the system clock. It is also used as a clock source for generating buzzer output.

### Real-Time and Watch-Time Measurement

To start watch timer operation, set bit 2 of the watch timer mode register, WMOD.2, to logic one. The watch timer starts, the interrupt request flag IRQW is automatically set to logic one, and interrupt requests commence in 0.5-second intervals.

Since the watch timer functions as a quasi-interrupt instead of a vectored interrupt, the IRQW flag should be cleared to logic zero by program software as soon as a requested interrupt service routine has been executed.

### Using a System Clock Source

The watch timer can generate interrupts based on the system clock frequency. The system clock (fx) is used as the signal source, according to the following formula:

$$\text{Watch timer clock}(fw) = \frac{\text{Main system clock}(fx)}{128} = 32.768 \text{ kHz}$$

(assuming fx = 4.19 MHz)

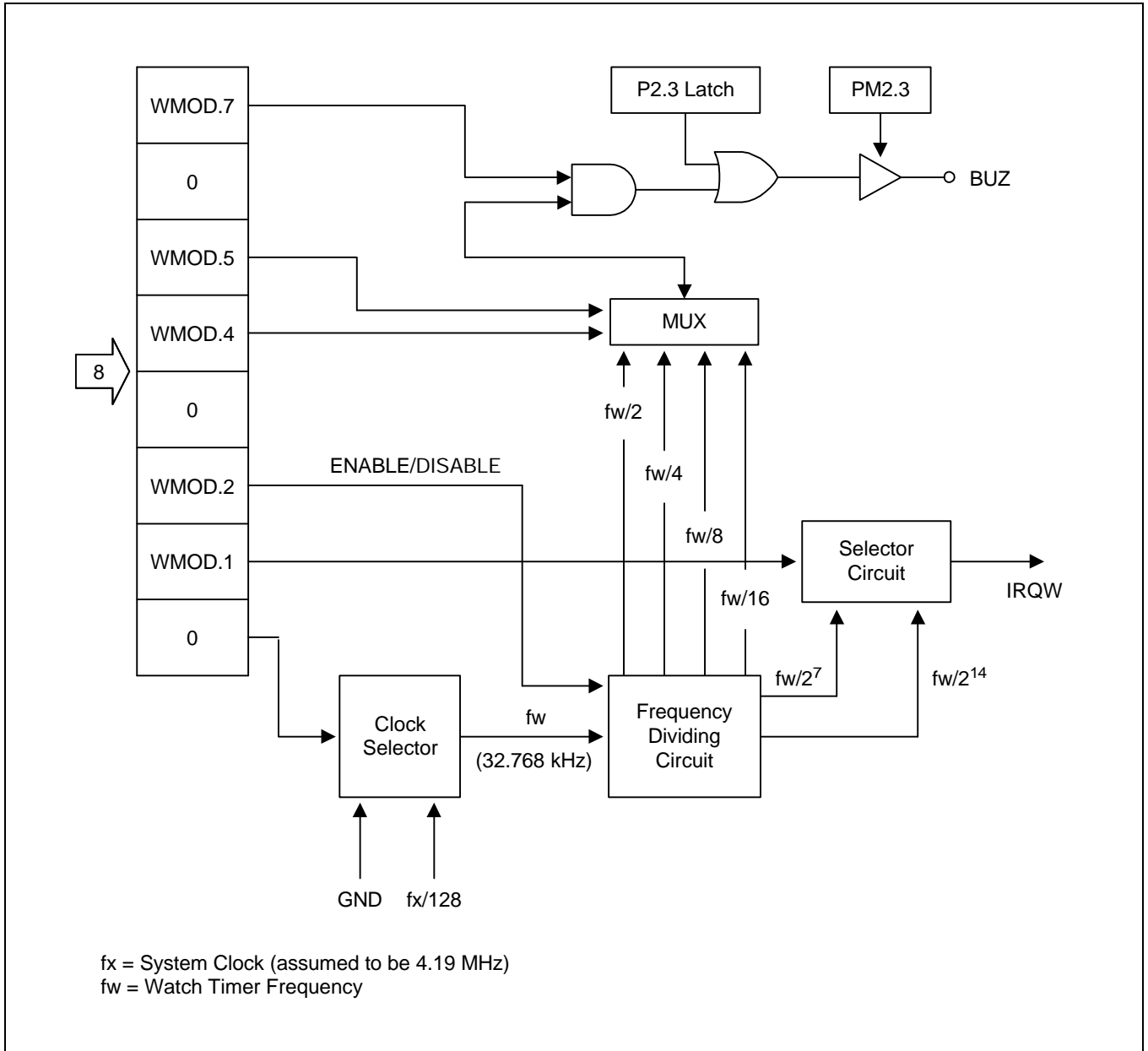
### Buzzer Output Frequency Generator

The watch timer can generate a steady 2kHz, 4kHz, 8kHz, or 16kHz signal at 4.19MHz to the BUZ pin. To select the BUZ frequency you want, load the appropriate value to the WMOD register. This output can then be used to actuate an external buzzer sound. To generate a BUZ signal, three conditions must be met:

- The WMOD.7 register bit is set to "1"
- The output latch for I/O port 2.3 is cleared to "0"
- The port 2.3 output mode flag (PM2.3) set to 'output' mode

**Timing Tests in High-Speed Mode**

By setting WMOD.1 to "1", the watch timer will function in high-speed mode, generating an interrupt every 3.91ms at 4.19MHz. At its normal speed (WMOD.1 = "0"), the watch timer generates an interrupt request every 0.5 seconds. High-speed mode is useful for timing events for program debugging sequences.



**Figure 12-4. Watch Timer Circuit Diagram**



**WATCH TIMER MODE REGISTER (WMOD)**

The watch timer mode register WMOD is used to select specific watch timer operations. It is 8-bit write-only addressable.

F88H	"0"	WMOD.2	WMOD.1	"0"
F89H	WMOD.7	"0"	WMOD.5	WMOD.4

WMOD settings control the following watch timer functions:

- Watch timer speed control (WMOD.1)
- Enable/disable watch timer (WMOD.2)
- Buzzer frequency selection (WMOD.4 and WMOD.5)
- Enable/disable buzzer output (WMOD.7)

**Table 12-8. Watch Timer Mode Register (WMOD) Organization**

Bit Name	Values		Function	Address
WMOD.7	0		Disable buzzer (BUZ) signal output	F89H
	1		Enable buzzer (BUZ) signal output	
WMOD.6	0		Always logic zero	
WMOD.5 – .4	0	0	fw/16 buzzer (BUZ) signal output (2kHz)	
	0	1	fw/8 buzzer (BUZ) signal output (4kHz)	
	1	0	fw/4 buzzer (BUZ) signal output (8kHz)	
	1	1	fw/2 buzzer (BUZ) signal output (16kHz)	
WMOD.3	0		Always logic zero	
WMOD.2	0		Disable watch timer; clear frequency dividing circuits	
	1		Enable watch timer	
WMOD.1	0		Normal mode; sets IRQW to 0.5 seconds	
	1		High-speed mode; sets IRQW to 3.91 ms	
WMOD.0	0		Always logic zero	

**NOTE:** System clock frequency (fx) is assumed to be 4.19MHz. 'fw' = watch timer clock frequency.

 **PROGRAMMING TIP — Using the Watch Timer**

1. Select a 0.5 second interrupt, and 2kHz buzzer enable:

```

BITS      EMB
SMB       15
LD        EA,#08H
LD        PMG1,EA      ; P2.3 ← output mode
BITR      P2.3         ; Clear P2.3 output latch
LD        EA,#84H
LD        WMOD,EA
BITS      IEW

```

2. Sample real-time clock processing method:

```

CLOCK     BTSTZ      IRQW      ; 0.5 second check
          RET        ; No, return
          •          ; Yes, 0.5 second interrupt generation
          •
          •          ; Increment HOUR, MINUTE, SECOND

```

NOTES

# 13

## DTMF GENERATOR

### OVERVIEW

The dual-tone multi-frequency (DTMF) output circuit is used to generate 16 dual-tone multiple frequency signals for tone dialing. This function is controlled by the DTMF mode register(DTMR) or by writing caller id register with serial interfacing. That is, there are two method to control DTMF generator, one is to use caller id register and the other is to use memory mapped register of DTMF generator. There are only memory mapped registers described in this chapter. Please refer to chapter 7 to know about the caller id registers.

It is recommended to use caller id's register because you'd better to use S3P7588X's DTMF output rather than KS57C5208/KS57C5308 SMDS's DTMF output. When you develop your own application system by setting S3P7588X as Caller ID Mode (Refer to chapter 7), all registers except that of caller id are unable to be used, so if you use memory mapped register of DTMF generator, SMDS's DTMF generator is activated instead of S3P7588X's one.

By writing the contents of the output latch for DTMF circuit with output instructions, 16 dual or single tones can be output to the DTMF output pin. The tone output frequency is selected by the DTMF mode register, and tone output amplitude is controlled by DTMF gain register. Figure 13-1 shows the DTMF block diagram. A frequency of 3.58 MHz is used for DTMF generator. Clock output is inhibited when DTMR.0 (DTMF Enable Bit) goes low. The tone output has a PDM format, so RC filter is required to get a real DTMF tone wave form.

The decoder receives data from the data latch and outputs the result to the row and column tone counter. The row and column tone counter are incremented until new data is latched. When DTMR.0 is logic one, data is latched, and the tone output is changed. Table 13-2 shows the 16 available keyboard frequencies.

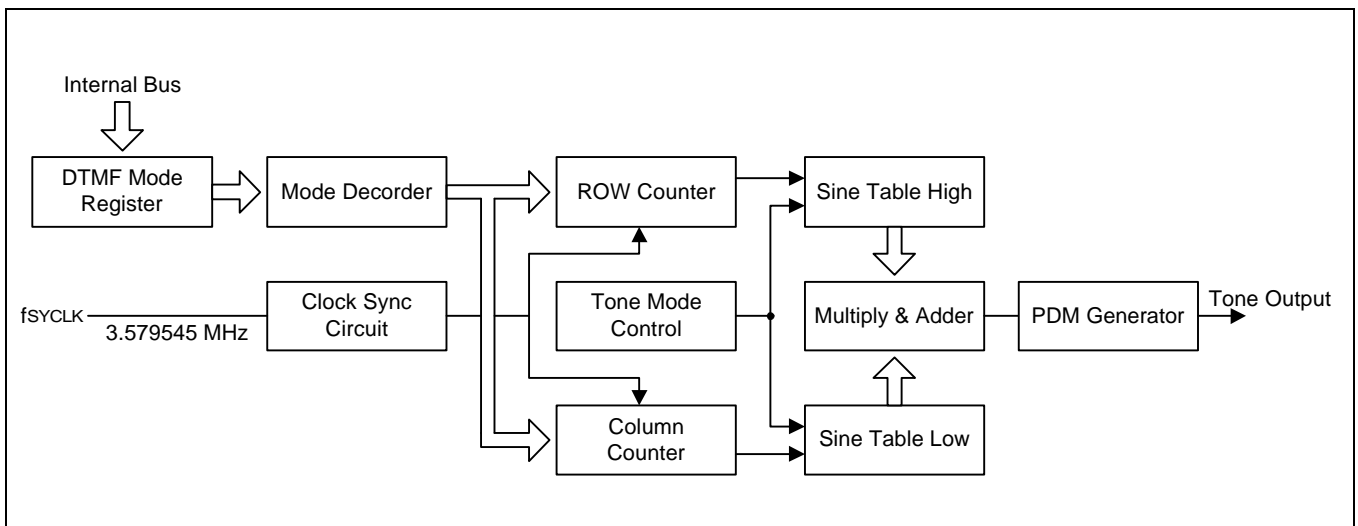


Figure 13-1. Block Diagram of DTMF Generator

Table 13-1. Keyboard Arrangement

1	2	3	A	ROW1
4	5	6	B	ROW2
7	8	9	C	ROW3
*	0	#	D	ROW4
COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	

Table 13-2. Tone Output Frequencies

Input	Specified Frequency (Hz)	Actual Frequency (Hz)	% Error
Row1	697	699.1	+ 0.31
Row2	770	766.2	- 0.49
Row3	852	847.4	- 0.54
Row4	941	948.0	+ 0.74
Column 1	1209	1215.7	+ 0.57
Column 2	1336	1331.7	- 0.32
Column 3	1477	1471.7	- 0.35
Column 4	1633	1645.0	+ 0.73

### DTMF MODE REGISTER

DTMF output is controlled by the DTMF mode register. Bit position DTMR.0 enables or disables DTMF operation. If DTMR.0 = 1, DTMF operation is enabled.

Programmers should write zeros or ones to bit positions DTMR.4–DTMR.7 according to the keyboard input specification. Writing the data in a look-up table is useful for program efficiency. The DTMR register is a write-only register, and is manipulated using 8-bit RAM control instructions.

Table 13-3. DTMF Mode Register (DTMR) Organization

Bit Name	Setting	Resulting DTMF Function	Address
DTMR.7–.4	0,1	Specify according to keyboard	FD3H
DTMR.3	–	Not Applicable	FD2H
DTMR.2–.1	0   0	Dual-tone enable	
	1   0		
	0   1	Single-column tone enable	
	1   1	Single-low tone enable	
DTMR.0	0	Disable DTMF operation	
	1	Enable DTMF operation	

Table 13-4. DTMR.7–DTMR.4 key Input Control Settings

DTMR.7	DTMR.6	DTMR.5	DTMR.4	Keyboard
0	0	0	0	D
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	0
1	0	1	1	*
1	1	0	0	#
1	1	0	1	A
1	1	1	0	B
1	1	1	1	C

When you want to use the caller id register, DTMR register should be zero (the reset value of DTMR) ahead.

### DTMF GAIN REGISTER

DTMF output amplitude is controlled by the DTMF gain register. Reset value is 10000b and this means that DTMF output is amplified by 1.

The DTGR register is a write-only register, and is manipulated using 8-bit RAM control instructions. When you want to use the caller id register, DTGR register should be 10000b (the reset value of DTGR) ahead.

Table 13-5. DTMF Gain Register (DTGR) Organization

Bit Name	Setting	Resulting DTMF Function	Address
DTGR.4–.0	0,1	Specify amplification factor. DTMF tone output is amplified by (DTGR.4-0 / 16)	FD5, FD4H

### RC FILTERING

DTMF output has a PDM format, so RC filtering is needed to make real DTMF tone wave. Recommended value of R and C is as follows.

R: 2k $\Omega$ , C: 10nF

To see additional information about DTMF application, please refer to chapter 7.

NOTES

# 14 ELECTRICAL DATA

## OVERVIEW

In this section, information on S3P7588X electrical characteristics is presented as tables and graphics. The information is arranged in the following order:

### Standard Electrical Characteristics

- Absolute maximum ratings
- D.C. electrical characteristics
- System clock oscillator characteristics
- I/O capacitance
- A.C. electrical characteristics
- Operating voltage range

### Miscellaneous Timing Waveforms

- A.C timing measurement point
- Clock timing measurement at  $X_{in}$  and  $X_{out}$
- TCL timing
- Input timing for RESET
- Input timing for external interrupts
- Serial data transfer timing

### Stop Mode Characteristics and Timing Waveforms

- RAM data retention supply voltage in stop mode
- Stop mode release timing when initiated by RESET
- Stop mode release timing when initiated by an interrupt request



Table 14-1. Absolute Maximum Ratings

(T<sub>A</sub> = 25°C)

Parameter	Symbol	Conditions	Rating	Units
Supply Voltage	V <sub>DD</sub>	–	– 0.3 to + 6.5	V
Input Voltage	V <sub>I1</sub>	All I/O ports	– 0.3 to V <sub>DD</sub> + 0.3	V
Output Voltage	V <sub>O</sub>	–	– 0.3 to V <sub>DD</sub> + 0.3	V
Output Current High	I <sub>OH</sub>	One I/O port active	– 15	mA
		All I/O ports active	– 35	
Output Current Low	I <sub>OL</sub>	One I/O port active	+ 30 (Peak value) + 15 (NOTE)	mA
		All I/O ports active	+ 100 (Peak value) + 60 (NOTE)	
Operating Temperature	T <sub>A</sub>	–	0 to + 70	°C
Storage Temperature	T <sub>STG</sub>	–	0 to + 70	°C

**NOTE:** The values for output current low (I<sub>OL</sub>) are calculated as peak value × √Duty.

Table 14-2. D.C. Electrical Characteristics

(T<sub>A</sub> = 0°C to +70°C, V<sub>DD</sub> = 2.7V to 5.5V)

Parameter	Symbol	Conditions	Min	Typ	Max	Units
Input High Voltage	V <sub>IH1</sub>	All input pins except those specified below for V <sub>IH2</sub> – V <sub>IH3</sub>	0.7 V <sub>DD</sub>	–	V <sub>DD</sub>	V
	V <sub>IH2</sub>	Ports 1, 3, 6, 7, and RESET	0.8 V <sub>DD</sub>		V <sub>DD</sub>	
	V <sub>IH3</sub>	Xin and Xout	V <sub>DD</sub> – 0.1		V <sub>DD</sub>	
Input Low Voltage	V <sub>IL1</sub>	All input pins except those specified below for V <sub>IL2</sub> – V <sub>IL3</sub>	–	–	0.3 V <sub>DD</sub>	V
	V <sub>IL2</sub>	Ports 1, 3, 6, 7, and RESET			0.2 V <sub>DD</sub>	
	V <sub>IL3</sub>	X <sub>in</sub> and X <sub>out</sub>			0.1	

Table 14-2. D.C. Electrical Characteristics (Continued)

(T<sub>A</sub> = 0°C to + 70°C, V<sub>DD</sub> = 2.7V to 5.5V)

Parameter	Symbol	Conditions	Min	Typ	Max	Units
Output high voltage	V <sub>OH</sub>	I <sub>OH</sub> = - 1mA Ports except 1	V <sub>DD</sub> - 1.0	-	-	V
Output low voltage	V <sub>OL1</sub>	V <sub>DD</sub> = 4.5V to 5.5V I <sub>OL</sub> = 6mA, Ports 4 and 5 only	-	0.4	2	V
		V <sub>DD</sub> = 2.7 to 5.5V, I <sub>OL</sub> = 1.6mA	-	-	0.4	
	V <sub>OL2</sub>	V <sub>DD</sub> = 4.5V to 5.5V I <sub>OL</sub> = 4mA, all out ports except 4, 5	-	-	2	V
		V <sub>DD</sub> = 2.7 to 5.5 V, I <sub>OL</sub> = 1.6mA	-	-	0.4	
	V <sub>OL3</sub>	V <sub>DD</sub> = 4.5V to 5.5V I <sub>OL</sub> = 1mA, DTMF	-	-	2	V
		V <sub>DD</sub> = 2.7 to 5.5V, I <sub>OL</sub> = 1.6mA	-	-	0.4	
Input high leakage current	I <sub>LIH1</sub>	V <sub>I</sub> = V <sub>DD</sub> All input pins except those specified below	-	-	3	μA
	I <sub>LIH2</sub>	V <sub>I</sub> = V <sub>DD</sub> Xin and Xout	-	-	20	
Input low leakage current	I <sub>LIL1</sub>	V <sub>I</sub> = 0V All input pins except below and RESET	-	-	- 3	μA
	I <sub>LIL2</sub>	V <sub>I</sub> = 0V X <sub>in</sub> and X <sub>out</sub> only	-	-	- 20	
Output high leakage current	I <sub>LOH</sub>	V <sub>O</sub> = V <sub>DD</sub> All out pins	-	-	3	μA
Output low leakage current	I <sub>LOL</sub>	V <sub>O</sub> = 0V Xin and Xout only	-	-	- 3	μA
Pull-up resistor	R <sub>L1</sub>	V <sub>DD</sub> = 5V; V <sub>I</sub> = 0V Except RESET	25	47	100	kΩ
		V <sub>DD</sub> = 3V	50	95	200	
	R <sub>L2</sub>	V <sub>DD</sub> = 5V; V <sub>I</sub> = 0V; RESET	100	220	400	
		V <sub>DD</sub> = 3V	200	450	800	

Table 14-2. D.C. Electrical Characteristics (Continued)

(T<sub>A</sub> = 0°C to + 70°C, V<sub>DD</sub> = 2.7V to 5.5V)

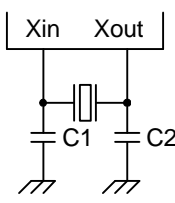
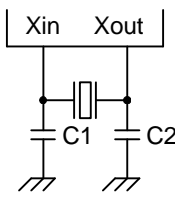
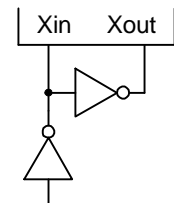
Parameter	Symbol	Conditions		Min	Typ	Max	Units
Supply current (note 1)	I <sub>DD1</sub> (FSK on)	Run mode; V <sub>DD</sub> = 5V ± 10% (note 2) 3.58MHz crystal oscillator, C1 = C2 = 22pF		–	9.0	11.0	mA
		V <sub>DD</sub> = 3V ± 10%			6.0	8.0	
	I <sub>DD2</sub> (CAS on)	Run mode; V <sub>DD</sub> = 5V ± 10% (note 2) 3.58MHz crystal oscillator, C1 = C2 = 22pF		–	9.9	12.1	mA
		V <sub>DD</sub> = 3V ± 10%			6.6	8.8	
	I <sub>DD3</sub> (CAS/FSK off)	Run mode; V <sub>DD</sub> = 5V ± 10% crystal oscillator, C1 = C2 = 22pF	3.58MHz		7.2	8.5	mA
		V <sub>DD</sub> = 3V ± 10%	3.58MHz		5.2	6.9	
	I <sub>DD4</sub>	Idle mode; V <sub>DD</sub> = 5V ± 10% crystal oscillator, C1 = C2 = 22pF	3.58MHz		2.5	3.5	mA
		V <sub>DD</sub> = 3V ± 10%	3.58MHz		1.2	2.2	
I <sub>DD5</sub>	Stop mode; V <sub>DD</sub> = 5V ± 10%		–	0.1	3	μA	
	Stop mode; V <sub>DD</sub> = 3V ± 10%			0.1	2		

**NOTES:**

- D.C. electrical values for Supply Current (I<sub>DD1</sub> to I<sub>DD3</sub>) do not include current drawn through internal pull-up registers.
- For D.C. electrical values, the power control register (PCON) must be set to 0011B.

Table 14-3. Main System Clock Oscillator Characteristics

(T<sub>A</sub> = 0°C to + 70°C, V<sub>DD</sub> = 2.7V to 5.5V)

Oscillator	Clock Configuration	Parameter	Test Condition	Min	Typ	Max	Units
Ceramic Oscillator		Oscillation frequency (note 1)	V <sub>DD</sub> = 2.7V to 5.5V	0.4	–	6.0	MHz
		Stabilization time (note 2)	V <sub>DD</sub> = 2.7V	–	–	4	ms
Crystal Oscillator		Oscillation frequency (note 1)	V <sub>DD</sub> = 2.7V to 5.5V	0.4	–	6.0	MHz
		Stabilization time (note 2)	V <sub>DD</sub> = 2.7V	–	–	10	ms
External Clock		X <sub>in</sub> input frequency (note 1)	V <sub>DD</sub> = 2.7V to 5.5V	0.4	–	6.0	MHz
		X <sub>in</sub> input high and low level width (t <sub>XH</sub> , t <sub>XL</sub> )	–	83.3	–	1250	ns

**NOTES:**

- Oscillation frequency and X<sub>in</sub> input frequency data are for oscillator characteristics only.
- Stabilization time is the interval required for oscillating stabilization after a power-on occurs, or when stop mode is terminated.

Table 14-4. Input/Output Capacitance

(T<sub>A</sub> = 25°C, V<sub>DD</sub> = 0V)

Parameter	Symbol	Condition	Min	Typ	Max	Units
Input Capacitance	C <sub>IN</sub>	f = 1MHz; Unmeasured pins are returned to V <sub>SS</sub>	–	–	15	pF
Output Capacitance	C <sub>OUT</sub>		–	–	15	pF
I/O Capacitance	C <sub>IO</sub>		–	–	15	pF

Table 14-5. A.C. Electrical Characteristics

(T<sub>A</sub> = 0°C to +70°C, V<sub>DD</sub> = 2.7V to 5.5V)

Parameter	Symbol	Conditions	Min	Typ	Max	Units
Instruction Cycle Time (note 1)	t <sub>CY</sub>	V <sub>DD</sub> = 2.7V to 5.5V	0.67	–	64	μs
TCL0, TCL1 Input Frequency	f <sub>TI0</sub> , f <sub>TI1</sub>	V <sub>DD</sub> = 2.7V to 5.5V	0	–	1.5	MHz
TCL0, TCL1 Input High, Low Width	t <sub>TIH0</sub> , t <sub>TIL0</sub> t <sub>TIH1</sub> , t <sub>TIL1</sub>	V <sub>DD</sub> = 2.7V to 5.5V	0.48	–	–	μs
Interrupt Input High, Low Width	t <sub>INTH</sub> , t <sub>INTL</sub>	INT1, INT2, INT4, KS0–KS7	0.1	–	–	μs
RESET Input Low Width	t <sub>RSL</sub>	Input	0.5	–	–	μs

Table 14-6. RAM Data Retention Supply Voltage in Stop Mode

(T<sub>A</sub> = 0°C to + 70°C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V <sub>DDDR</sub>	–	1.5	–	5.5	V
Data retention supply current	I <sub>DDDR</sub>	V <sub>DDDR</sub> = 1.5V	–	0.1	10	μA
Release signal set time	t <sub>SREL</sub>	–	0	–	–	μs
Oscillator stabilization wait time (note 1)	t <sub>WAIT</sub>	Released by RESET Released by interrupt	–	2 <sup>17</sup> /fx (note 2)	–	ms ms

**NOTES:**

1. During oscillator stabilization wait time, all CPU operations must be stopped to avoid instability during oscillator start-up.
2. Use the basic timer mode register (BMOD) interval timer to delay execution of CPU instructions during the wait time.

Timing Waveforms

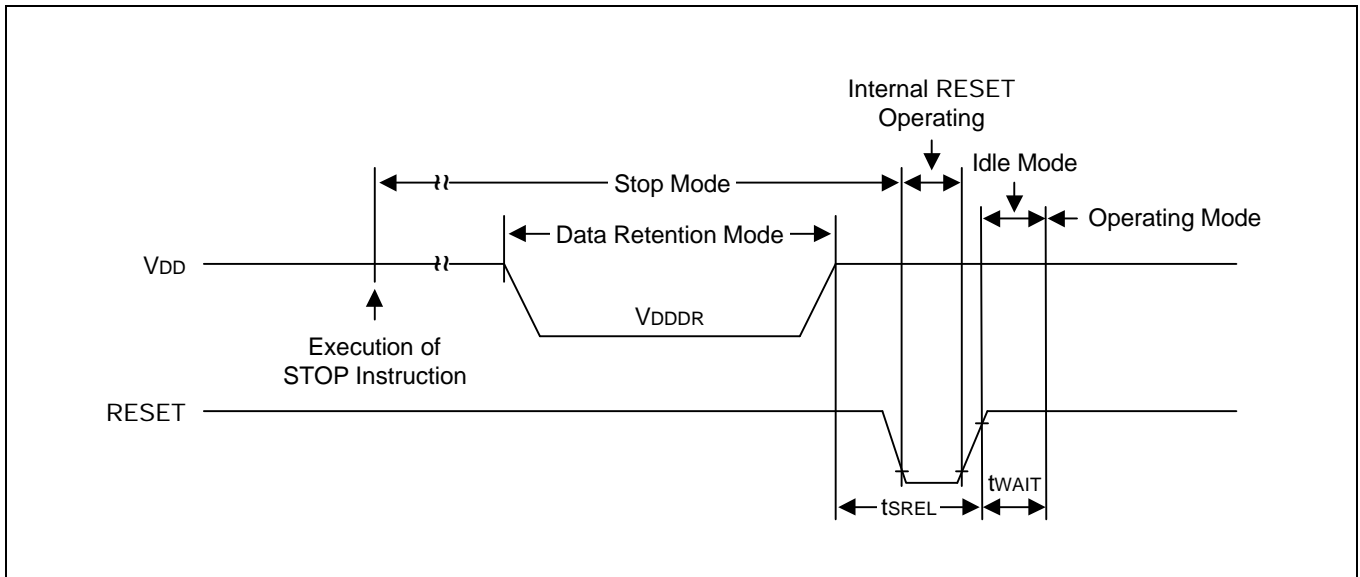


Figure 14-1. Stop Mode Release Timing When Initiated By RESET

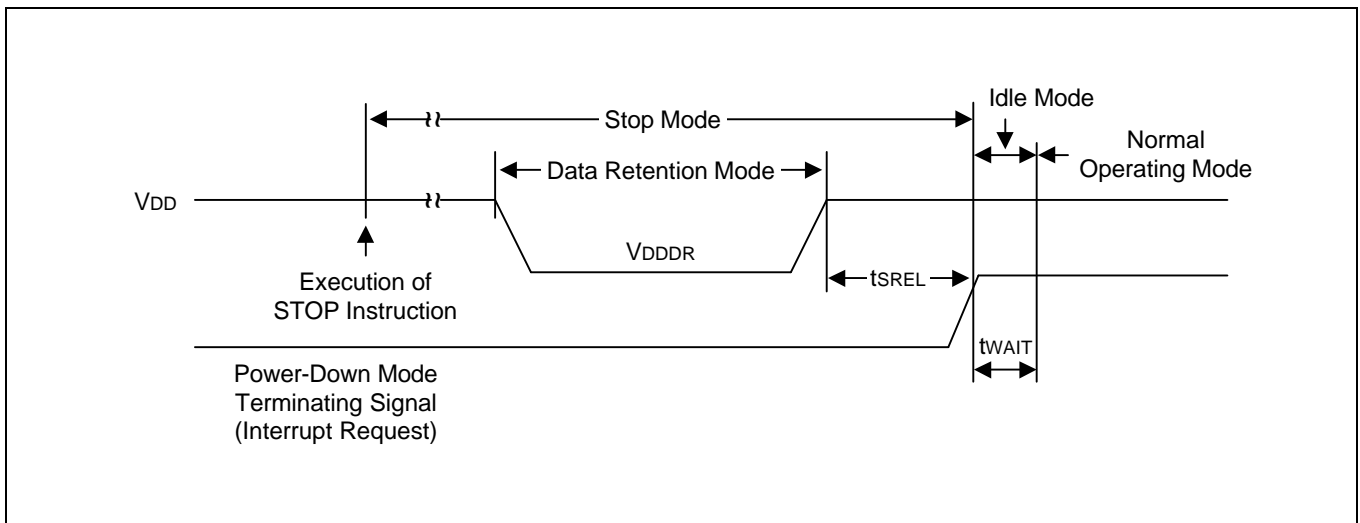


Figure 14-2. Stop Mode Release Timing When Initiated By Interrupt Request

Timing Waveforms (Continued)

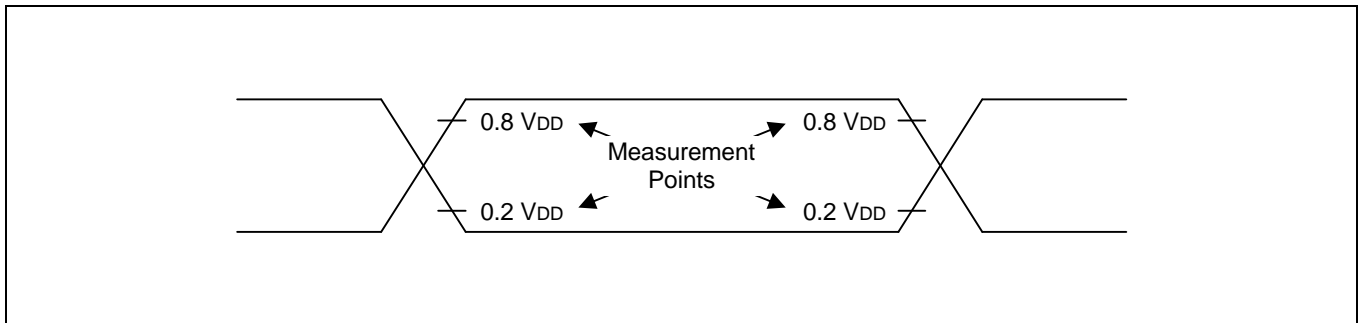


Figure 14-3. A.C. Timing Measurement Points (Except for Xin)

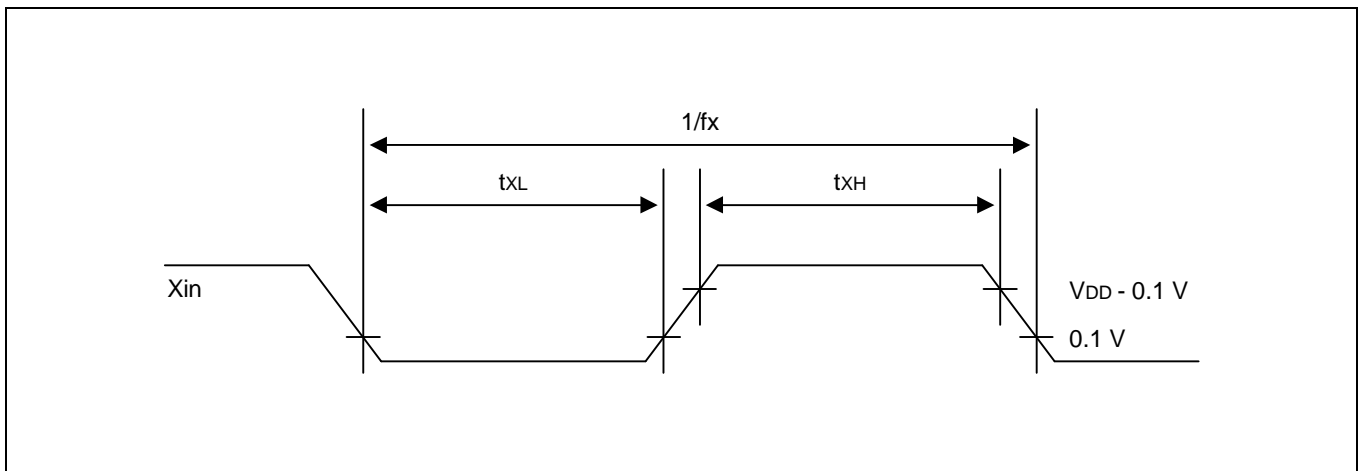


Figure 14-4. Clock Timing Measurement at Xin

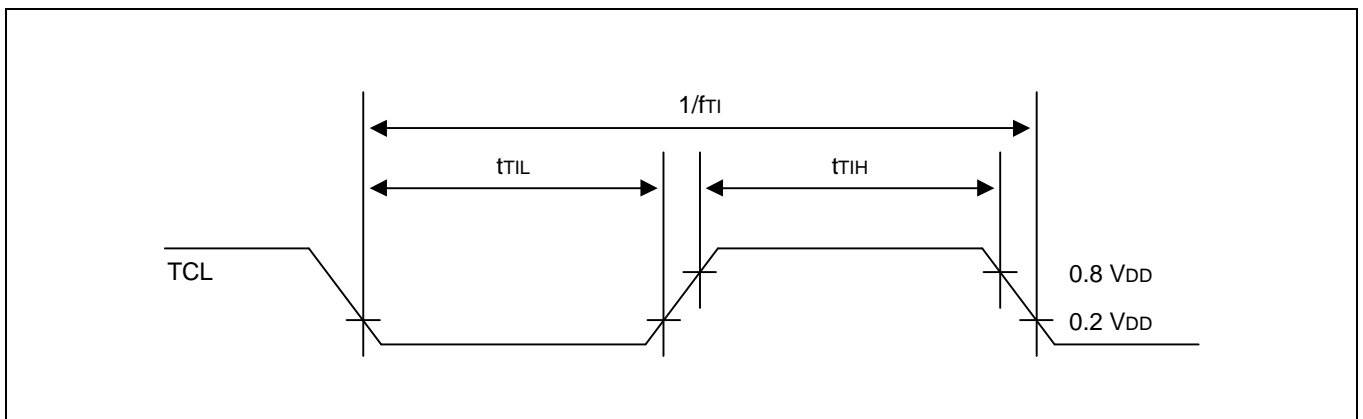


Figure 14-5. TCL Timing



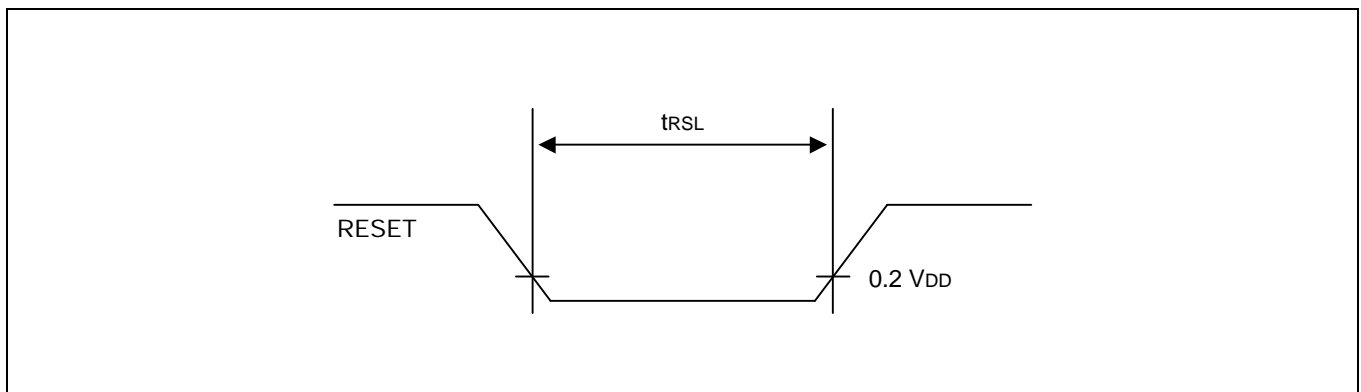


Figure 14-6. Input Timing for RESET Signal

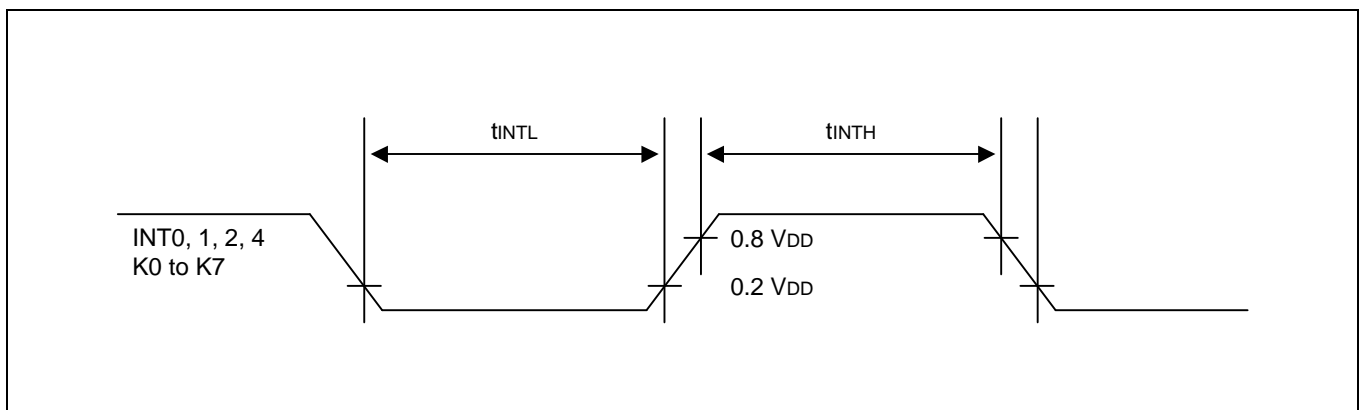


Figure 14-7. Input Timing for External Interrupts and Quasi-Interrupts

Table 14-7. Electrical Characteristics of CID Block

( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{DD} = 5.0\text{V} \pm 5\%$ ,  $X_{IN} = 3.579545\text{MHz} \pm 0.1\%$ )

Symbol	Parameter	Min	Typ	Max	Unit
<b>Voltage reference</b>					
VREF	Reference voltage output		2.25		V
<b>CAS detector</b>					
THac	Input accept threshold (in 600Ω load)	-38			dBm
Pic	Input signal power (in 600Ω load)	-37		-6	dBm
flc	Low tone frequency		2130		Hz
fhc	High tone frequency		2750		Hz
Δfmaxc	Maximum frequency deviation	-0.6		+0.6	%
Twc	Twist	-6		6	dB
<b>FSK receiver</b>					
Pif	Input signal power (in 600Ω load)	-38		0	dBm
fD	Data transmission rate frequency	1188	1200	1212	Baud
fmb	Mark frequency (Bell202)	1188	1200	1212	Hz
fsb	Space frequency (Bell202)	2178	2200	2222	Hz
fmv	Mark frequency (CCITT/V23)		1300		Hz
fsv	Space frequency (CCITT/V23)		2100		Hz
Twf	Twist	-10		10	dB
S/N0	Signal to noise ratio (0Hz – 200Hz)	-25			dB
S/N1	Signal to noise ratio (200Hz – 3.2kHz)	6			dB
S/N3	Signal to noise ratio (3.2kHz – 15kHz)	-25			dB
<b>Stutter dial tone detector</b>					
BW	Detection bandwidth	330		440	Hz
THap	Input accept threshold (in 600Ω load )	-36		-10	dBm
<b>DTMF generation</b>					
Pod	Output signal power (for high tone)	-	-	-7.3	dBm
Δfmaxd	Maximum frequency deviation	-0.1		+0.1	%
Thdd	Total harmonic distortion (0 ~ 6kHz)	-	-	2.5	%
S/Nd	Signal to noise ratio (0 ~ 6kHz)	-35			dBm

Table 14-8. CAS Timing Characteristics

( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{DD} = 2.7\text{V}$  to  $5.5\text{V}$ ,  $X_{IN} = 3.579545\text{MHz} \pm 0.1\%$ )

Parameter	Symbol	Min	Typ	Max	Unit
CAS detection time from CAS start	$T_{DETC}$		67		ms
Detection off time from CAS end	$T_{OFFC}$		30		ms
CAS detection time width	$T_{WIDTHC}$	8			ms

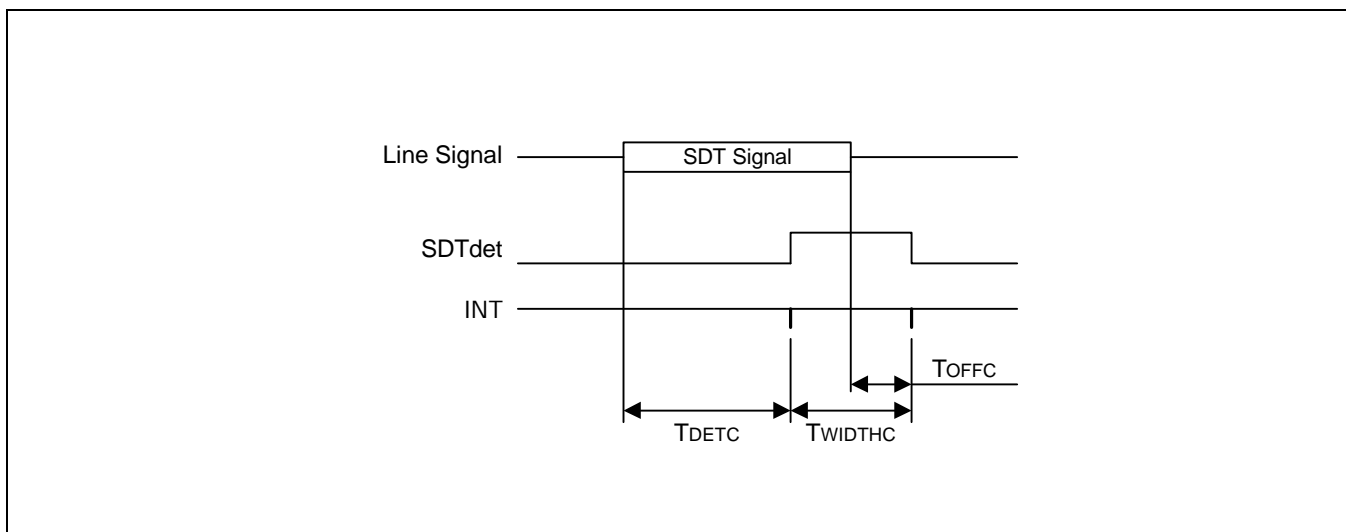


Figure 14-8. Waveform for CAS Timing Characteristics

Table 14-9. SDT Timing Characteristics

( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{DD} = 2.7\text{V}$  to  $5.5\text{V}$ ,  $X_{IN} = 3.579545\text{MHz} \pm 0.1\%$ )

Parameter	Symbol	Min	Typ	Max	Unit
SDT detection time from SDT start	$T_{DETS}$		60		ms
Detection off time from SDT end	$T_{OFFC}$		30		ms

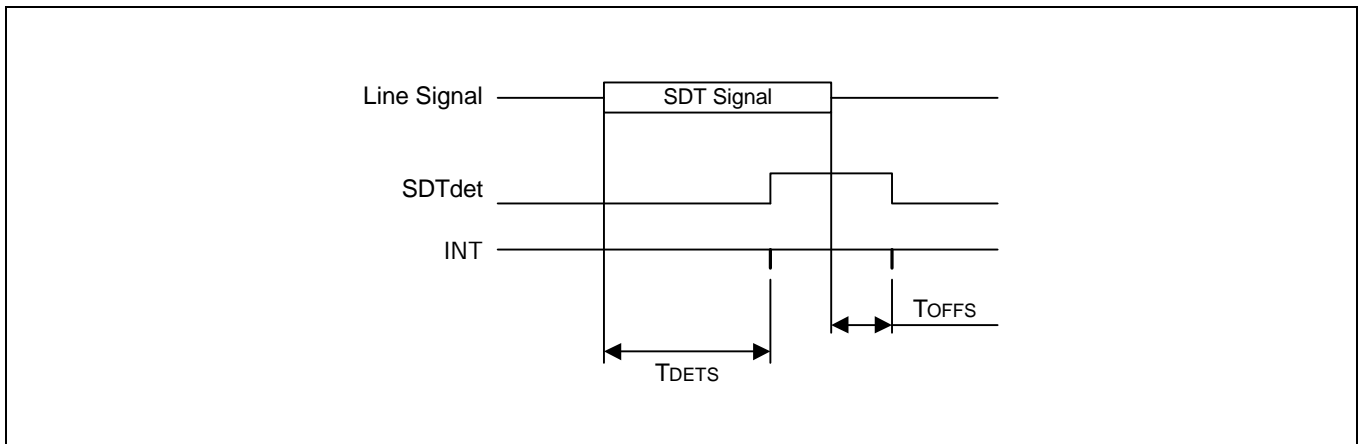


Figure 14-9. Waveform for SDT Timing Characteristics

Table 14-10. Serial Interface Timing Characteristics

( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{DD} = 2.7\text{V}$  to  $5.5\text{V}$ ,  $X_{IN} = 3.579545\text{MHz} \pm 0.1\%$ )

Parameter	Symbol	Min	Typ	Max	Unit
SDT to SCK time to start serial interface	$T_{\text{START}}$		50		ns
SCK to SDT time to stop serial interface	$T_{\text{STOP}}$		50		ns
SCK low time period	$T_{\text{SCKL}}$		500		ns
SCK high time period	$T_{\text{SCKH}}$		500		ns
SDT set-up time	$T_{\text{SU}}$		50		ns
SDT hold time	$T_{\text{HD}}$		50		ns

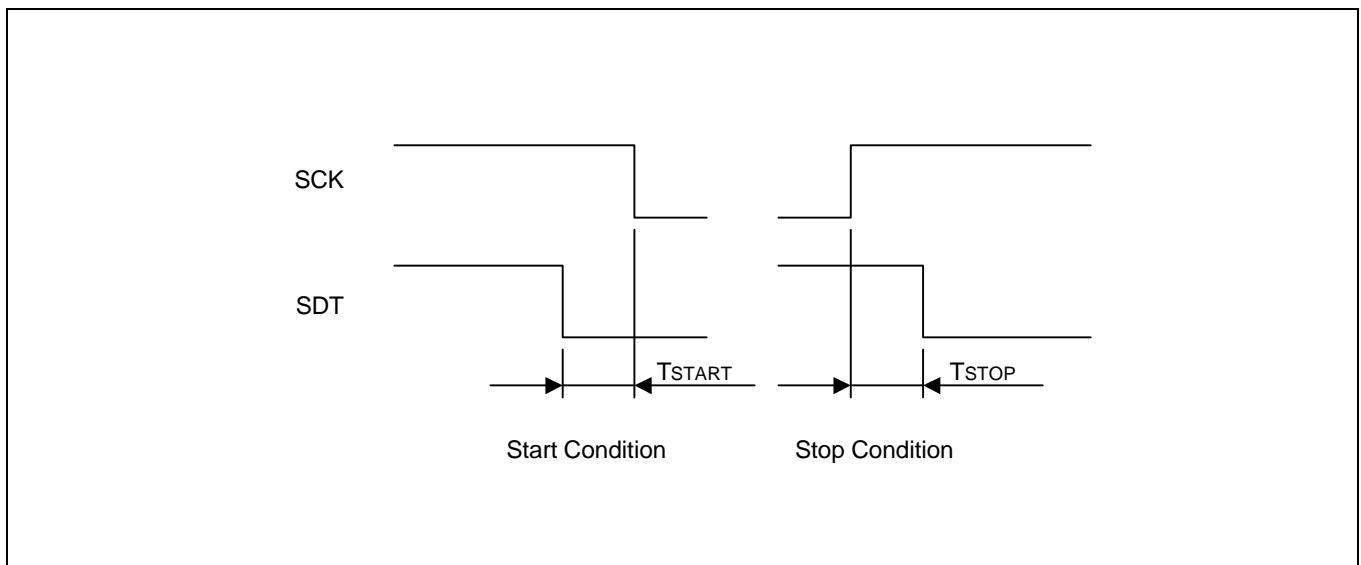


Figure 14-10. Timing Constraints of Start and Stop Condition

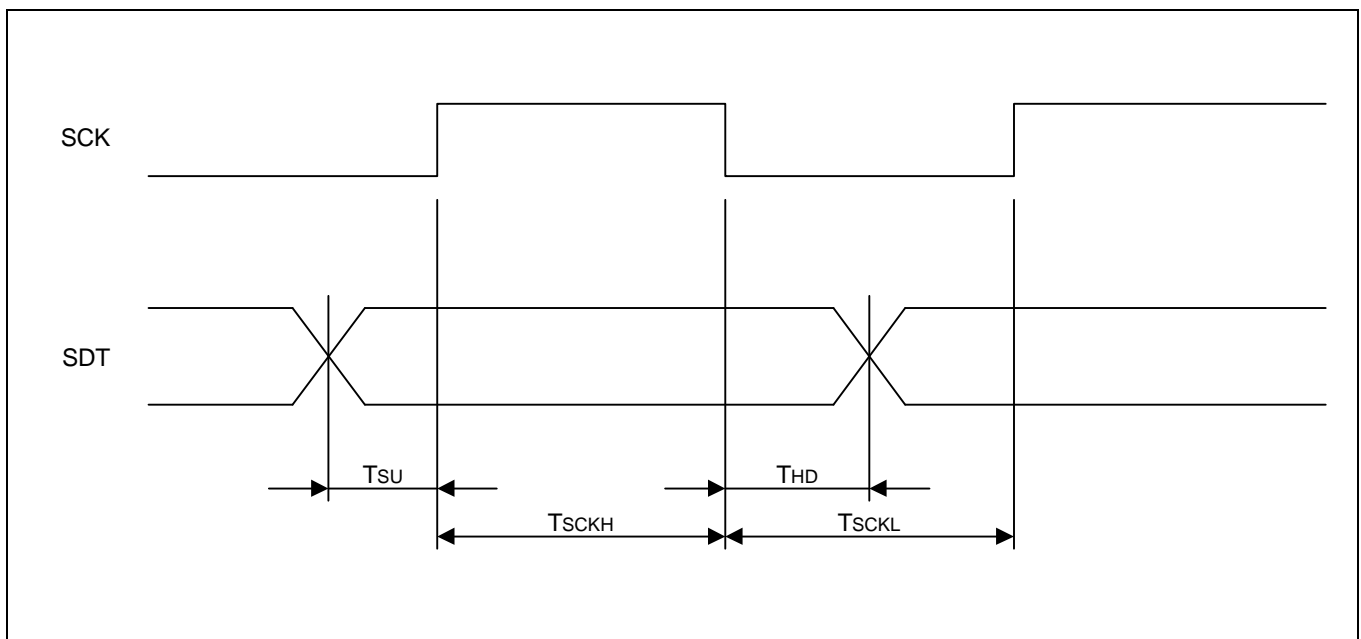


Figure 14-11. Timing of SCK and SDT During Byte Transmission

# 15

## MECHANICAL DATA

### OVERVIEW

The S3P7588X microcontroller are available in a 100-pin TQFP package (100-TQFP-1414), and a pellet type.

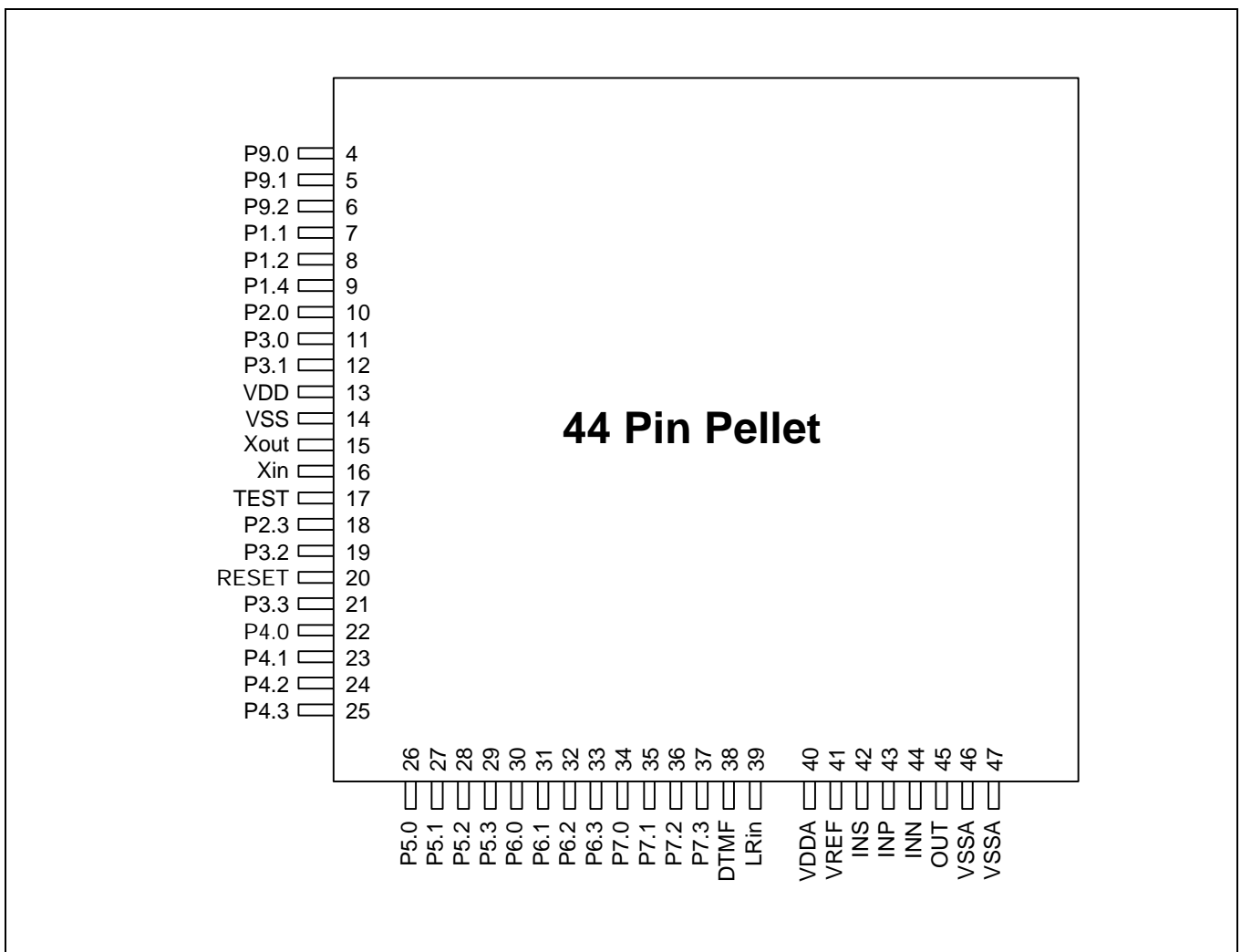


Figure 15-1. Pin Diagram of Pellet Type



# 16

## OTP

### OVERVIEW

The S3P7588X single-chip CMOS microcontroller is the OTP (One Time Programmable) version. It has an on-chip EPROM instead of masked ROM. The EPROM is accessed by a serial data format.



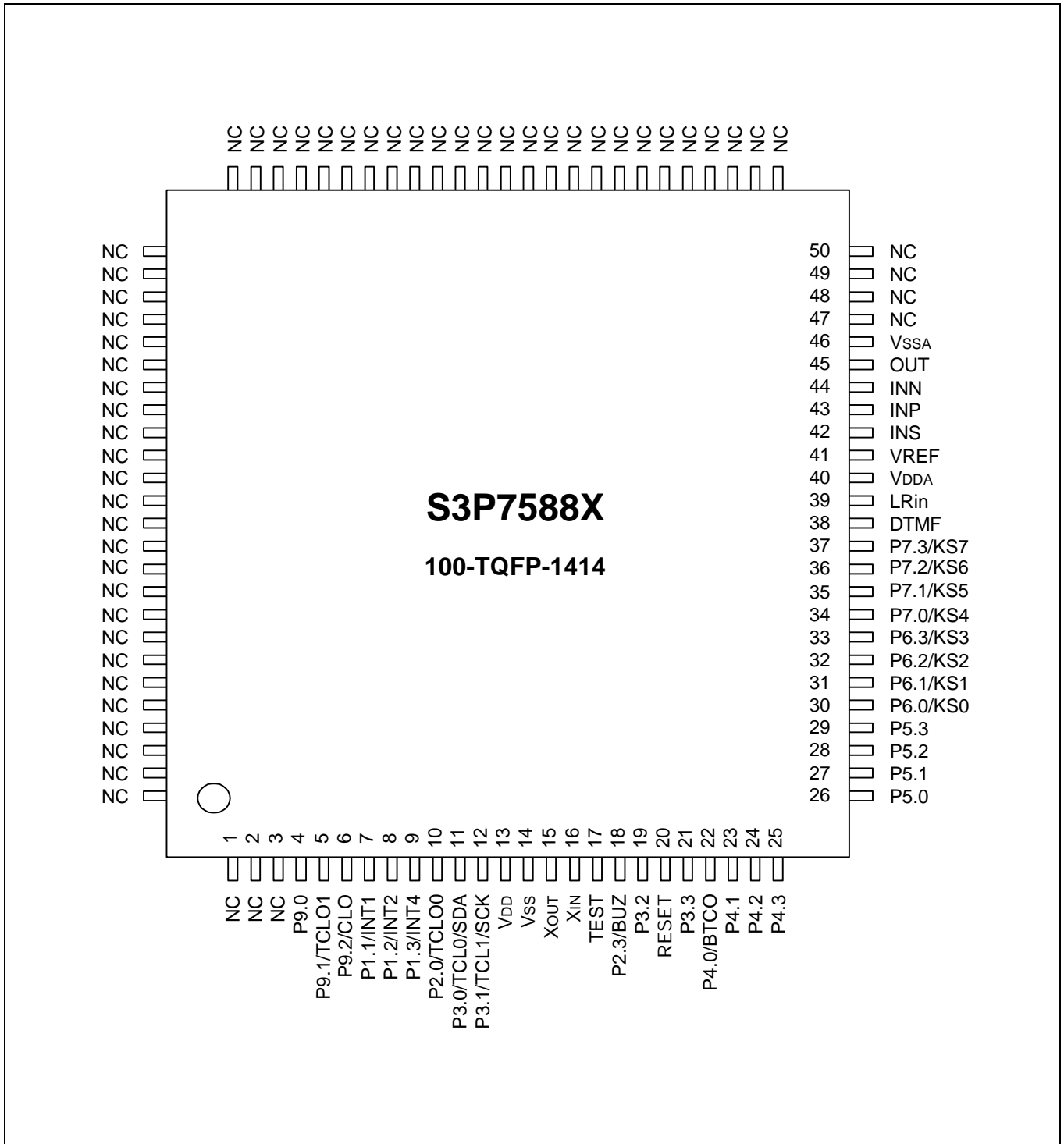


Figure 16-1. S3P7588X Pin Assignments (100-TQFP-1414)

Table 16-1. S3P7588X Pin Descriptions Used to Read/Write the EPROM

Main Chip	During Programming			
Pin Name	Pin Name	Pin No.	I/O	Function
P3.0	SDAT	11	I/O	Serial data pin. Output port when reading and input port when writing. Can be assigned as a Input / push-pull output port.
P3.1	SCLK	12	I/O	Serial clock pin. Input only pin.
TEST	V <sub>PP</sub> (TEST)	17	I	Power supply pin for EPROM cell writing (indicates that OTP enters into the writing mode). When 12.5V is applied, OTP is in writing mode and when 5V is applied, OTP is in reading mode. (Option)
RESET	RESET	20	I	Chip initialization
V <sub>DD</sub> /V <sub>SS</sub>	V <sub>DD</sub> /V <sub>SS</sub>	13/14	I	Logic power supply pin. V <sub>DD</sub> should be tied to +5V during programming.

Table 16-2. S3P7588X Features

Characteristic	S3P7588X
Program Memory	8K byte EPROM
Operating Voltage (V <sub>DD</sub> )	2.4V to 5.5V
OTP Programming Mode	V <sub>DD</sub> = 5V, V <sub>PP</sub> (TEST) = 12.5V
Pin Configuration	100-TQFP-1414
EPROM Programmability	User Program 1 time

### OPERATING MODE CHARACTERISTICS

When 12.5V is supplied to the V<sub>PP</sub> (TEST) pin of the S3P7588X, the EPROM programming mode is entered. The operating mode (read, write, or read protection) is selected according to the input signals to the pins listed in Table 16-3 below.

Table 16-3. Operating Mode Selection Criteria

V <sub>DD</sub>	V <sub>PP</sub> (TEST)	REG/MEM	Address (A15-A0)	R/W	Mode
5V	5V	0	0000H	1	EPROM read
	12.5V	0	0000H	0	EPROM program
	12.5V	0	0000H	1	EPROM verify
	12.5V	1	0E3FH	0	EPROM read protection

**NOTE:** "0" means Low level; "1" means High level.

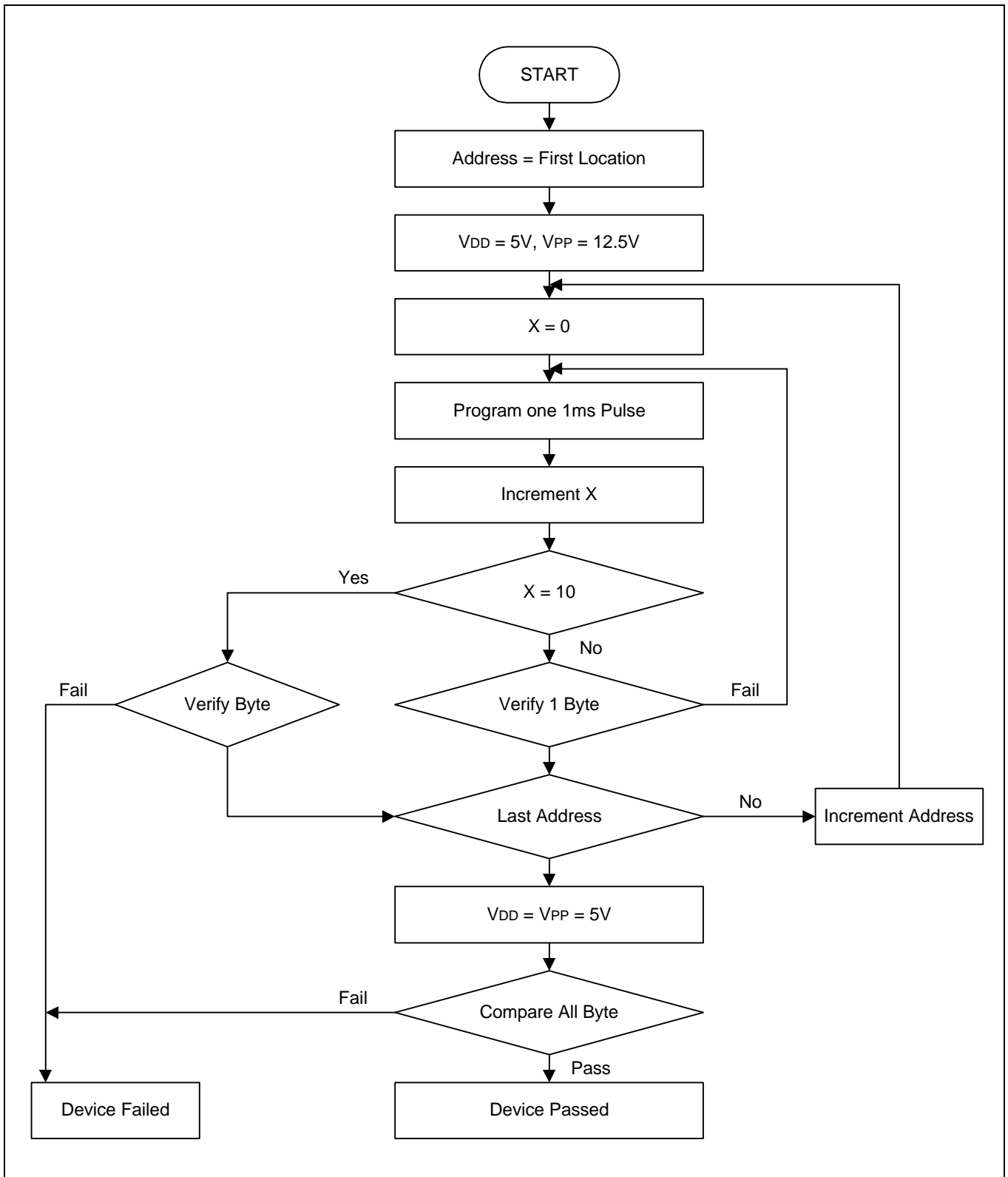


Figure 16-2. OTP Programming Algorithm

# 17

## DEVELOPMENT TOOLS

### OVERVIEW

S3P7588X contains the 4-bit micom of Samsung, and all of Samsung's development system for 4-bit micom is applicable to S3P7588X. The development support system is configured with a host system, debugging tools, and support software. For the host system, any standard computer that operates with MS-Windows as its operating system can be used. One type of debugging tool including hardware and software is provided: the in-circuit emulator, OPENICE i500, for 4-bit, 8-bit families of Samsung microcontrollers. There are other support softwares that includes debugger, assembler, and a program for setting options.

### OTPs

One time programmable microcontroller (OTP) for the S3P7588X microcontroller and OTP programmer (Gang) are now available.

### Development System Configuration

There are four possible configurations of the S3P7588X development system configuration.

To Use Probe

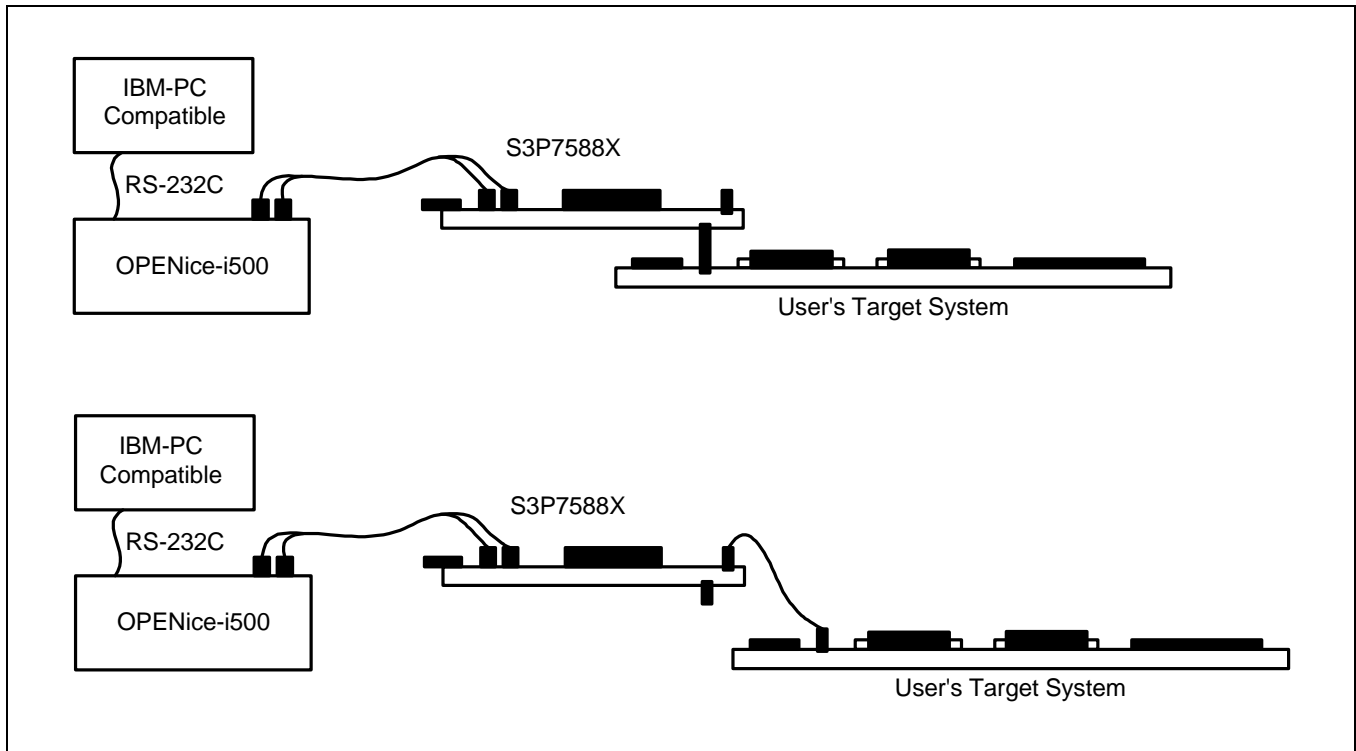


Figure 17-1. S3P7588X Development System Configuration

To Use Target Board

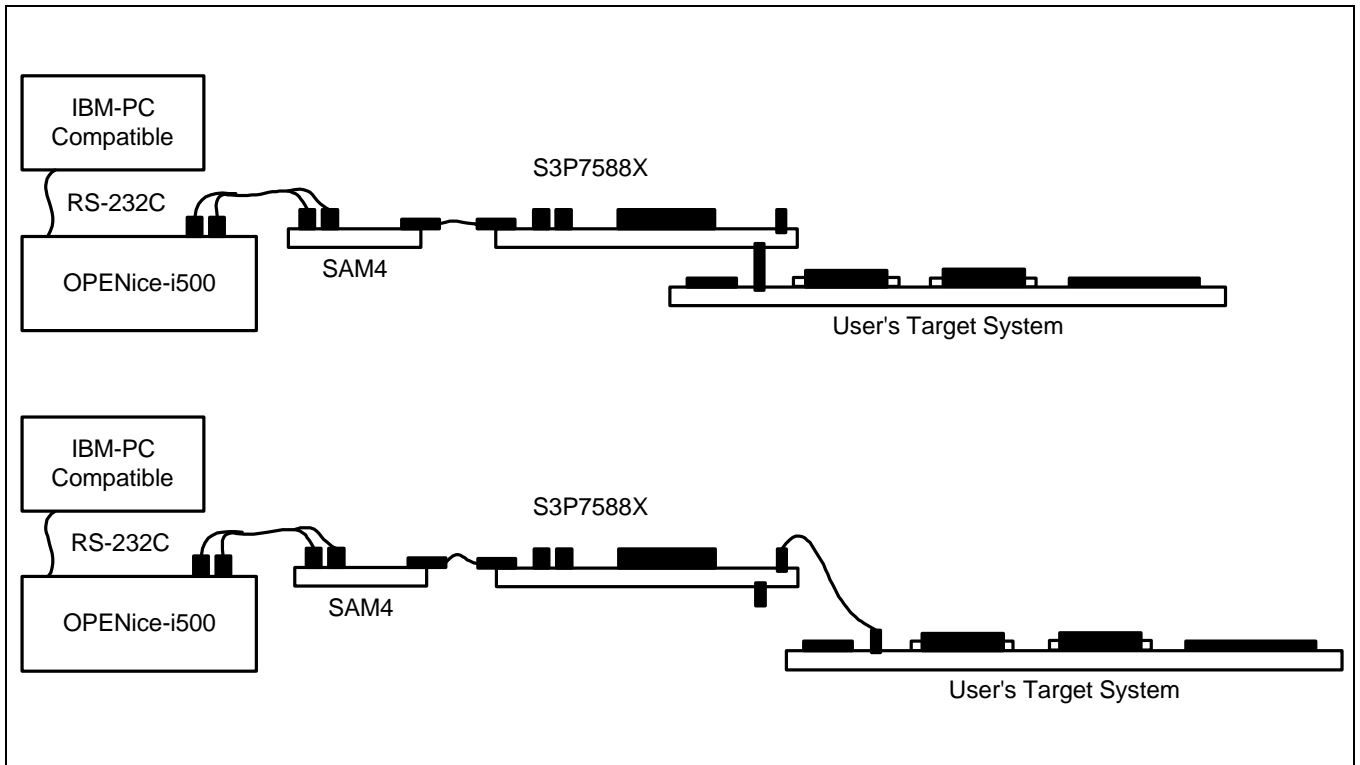


Figure 17-1. S3P7588X Development System Configuration (Continued)

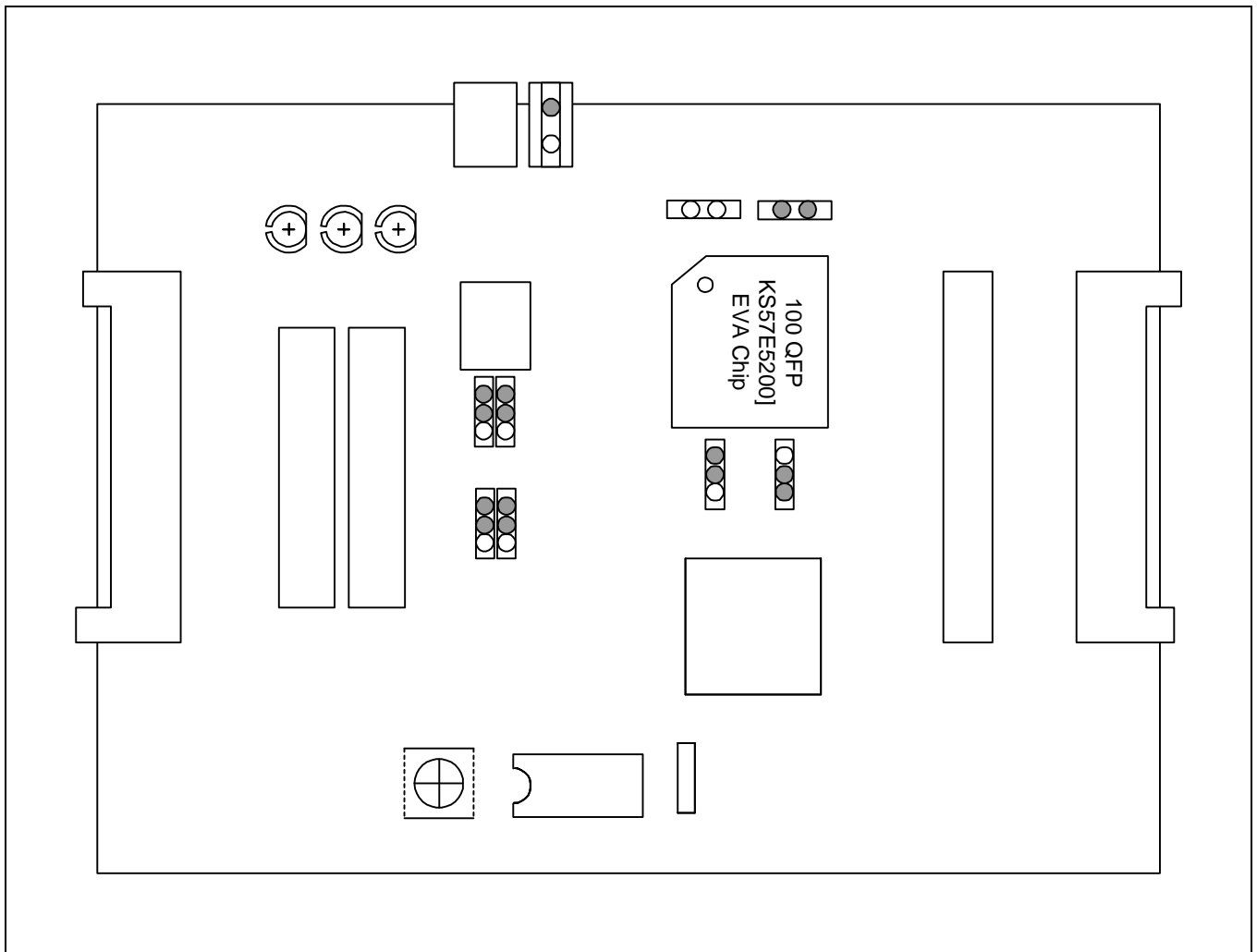


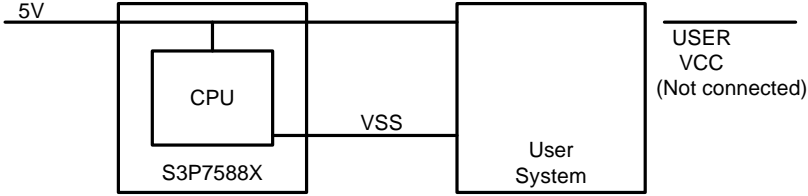


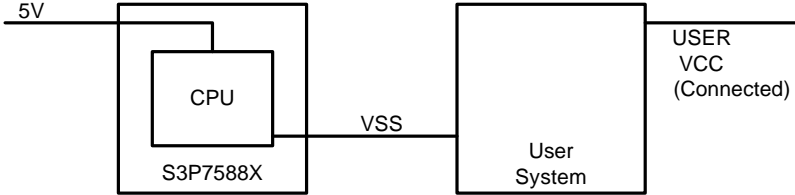


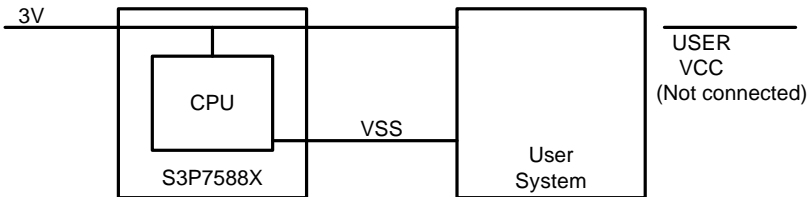


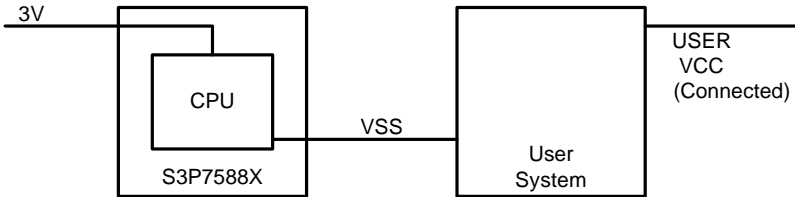


Figure 17-2. S3P7588X Target Board Diagram

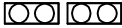

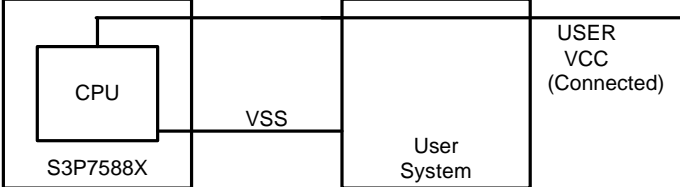
**Power Configuration**



The power configuration of S3P7588X development system is selected by JP1, JP2, JP3, and S1 as following table.

**Table 17-1. Switch Settings for Power Configuration**

Jumper State		Description
JP8 JP6 	JP3 	- S3P7588X target board use 5V (from MDS or Adapter) - User system use same power source as S3P7588X target board 
JP8 JP6 	JP3 	- S3P7588X target board use 5V (from MDS or Adapter) - User system use different power source from S3P7588X target board 
JP8 JP6 	JP3 	- S3P7588X target board use 3V (from MDS or Adapter) - User system use same power source as S3P7588X target board 
JP8 JP6 	JP3 	- S3P7588X target board use 3V (from MDS or Adapter) - User system use different power source from S3P7588X target board 



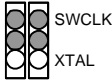
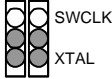
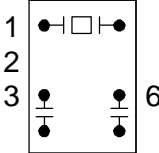
Jumper State		Description
	<p>JP3</p>  <p>ON OFF</p>	<p>- S3P7588X target board use the power source from user system.</p> 

Jumper State		Description
<p>S1</p>  <p>ON OFF</p>	<p>When the probe system (68 pin connector - CN1, CN2) is used: Switch on S1, and Connect J1 jack to 5V adapter, and you can select the 3V or 5V by setting JP6, JP8 appropriately.</p>	
<p>S1</p>  <p>ON</p>	<p>When the target board system (100 pin connector - U3) is used: Switch off S1, and don't supply power from J1 jack. You can select the 3V or 5V by setting JP6, JP8 appropriately.</p>	

**User Clock Selection**

The user clock (Xin, Xout) for target system can be selected as following table.

**Table 17-2. Switch Settings for User Clock Selection**

Jumper State	Description
<p style="text-align: center;"><b>JP4</b></p> 	<p>To use MDS(OPENice-i500) clock as Xin, Xout</p>
<p style="text-align: center;"><b>JP4</b></p> 	<p>To use crystal as Xin, Xout                      You can use appropriate crystal and capacitors by mounting them at DIP1 as follows.                      When this configuration is used, It is strongly recommended to connect the S3P7588X target board directly to user board (not with cable).</p> 

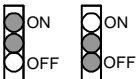
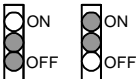
**NOTE:** Figure 17-2 shows default settings as follows.

- Use probe(CN1/CN2) and 5V power adapter and target system uses this same power.
- Use P8.0 as Caller ID reset signal.
- Use MDS(OPENice-i500) clock as Xin, Xout

**Caller ID Reset Selection**

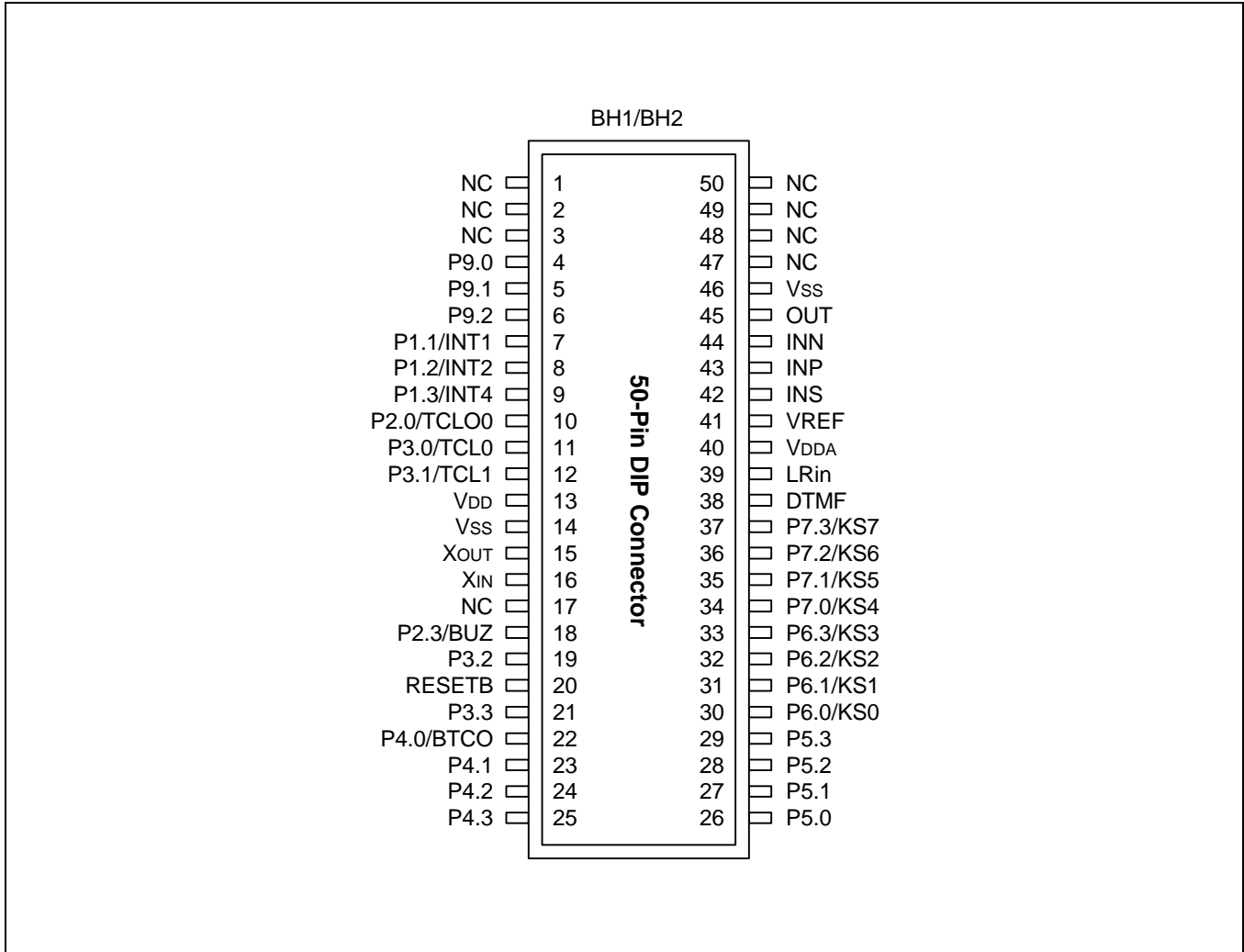
There are two options for reset signal of Caller ID block in S3P7588X chip, and it can be also configured in S3P7588X target board as following table.

**Table 17-3. Switch Settings for Reset Signal of Caller ID**

<p style="text-align: center;"><b>JP1    JP2</b></p> 	<p>Select P8.0 as Caller ID reset (JP1, JP2 must be used exclusively)</p>
<p style="text-align: center;"><b>JP1    JP2</b></p> 	<p>Select P3.1 as Caller ID reset (JP1, JP2 must be used exclusively)</p>

**Pin Assignment**

S3P7588X target board has two 50-DIP connectors (BH1/BH2) for connecting to user system. These connectors have same pin assignments except that TCLO1 and CLO output is not monitored at P9.1, P9.2 pin respectively. These signals can be only seen in S3P7588X of OTP or mask ROM version. This mismatch comes from the compatibility issue between the former MCU version (KS57C5208) and S3P7588X.



**Figure 17-3. Pin Assignment of 50-Pin DIP Connector**

# 18

## ERRATA

### REVISION 1.0

This documentation have been released first at 2001. 8. 16.

### ERRATA List

From the first released documentation, the followings have been revised.

- |              |   |
|--------------|---|
| 2001. 10. 11 | It is refined that the called id receiver block has no hardware reset input.<br>The reset signal must be made by software to release the caller id receiver from reset state. (Refer Chapter 7, 10) |
| 2001. 10. 13 | This ERRATA has released.<br>The pin description of 'LRin' pin is added, and that of 'RESETB' is revised.   |
| 2001. 12. 28 | The package options are mentioned that only pellet type can be mass-produced.<br>The electrical data has been revised.<br>The pin mismatch between MDS board and S3P7588X is mentioned.             |
| 2002. 01. 12 | The description for TCLO1, CLO output are corrected that these output come from P9.1, P9.2 instead of P2.1, P2.2 respectively.  |

NOTES

# S3P7 SERIES MASK ROM ORDER FORM

**Product description:**

Device Number: S3P7 \_\_\_\_\_ - \_\_\_\_\_ (write down the ROM code number)

Product Order Form:  Package  Pellet  Wafer Package Type: \_\_\_\_\_

**Package Marking (Check One):**

Standard

Custom A  
(Max 10 chars)

Custom B  
(Max 10 chars each line)

<b>SEC</b> @ YWW Device Name
---------------------------------

@ YWW Device Name  <div style="border: 1px solid black; width: 100%; height: 15px;"></div>
---

@ YWW  <div style="border: 1px solid black; width: 100%; height: 15px;"></div> <div style="border: 1px solid black; width: 100%; height: 15px;"></div>
---

@ : Assembly site code, Y : Last number of assembly year, WW : Week of assembly

**Delivery Dates and Quantities:**

Deliverable	Required Delivery Date	Quantity	Comments
ROM code	-	Not applicable	See ROM Selection Form
Customer sample			
Risk order			See Risk Order Sheet

Please answer the following questions:

**For what kind of product will you be using this order?**

- New product
  Upgrade of an existing product  
 Replacement of an existing product
  Other

If you are replacing an existing product, please indicate the former product name  
( \_\_\_\_\_ )

**What are the main reasons you decided to use a Samsung microcontroller in your product?**

**Please check all that apply.**

- Price
  Product quality
 Features and functions  
 Development system
  Technical support
 Delivery on time  
 Used same micom before
  Quality of documentation
 Samsung reputation

**Mask Charge (US\$ / Won):** \_\_\_\_\_

**Customer Information:**

Company Name: \_\_\_\_\_ Telephone number \_\_\_\_\_

**Signatures:** \_\_\_\_\_  
(Person placing the order)
(Technical Manager)



# S3P7 SERIES REQUEST FOR PRODUCTION AT CUSTOMER RISK

**Customer Information:**

Company Name: \_\_\_\_\_  
Department: \_\_\_\_\_  
Telephone Number: \_\_\_\_\_ Fax: \_\_\_\_\_  
Date: \_\_\_\_\_

**Risk Order Information:**

Device Number: S3P7 \_\_\_\_\_ - \_\_\_\_\_ (write down the ROM code number)  
Package: \_\_\_\_\_ Number of Pins: \_\_\_\_\_ Package Type: \_\_\_\_\_  
Intended Application: \_\_\_\_\_  
Product Model Number: \_\_\_\_\_

**Customer Risk Order Agreement:**

We hereby request SEC to produce the above named product in the quantity stated below. We believe our risk order product to be in full compliance with all SEC production specifications and, to this extent, agree to assume responsibility for any and all production risks involved.

**Order Quantity and Delivery Schedule:**

Risk Order Quantity: \_\_\_\_\_ PCS

Delivery Schedule:

Delivery Date (s)	Quantity	Comments

**Signatures:**

\_\_\_\_\_

(Person Placing the Risk Order)

\_\_\_\_\_

(SEC Sales Representative)





# S3P7588X MASK OPTION SELECTION FORM

**Device Number:** S3P7588X-\_\_\_\_\_ (write down the ROM code number)

**Attachment (Check one):**  Diskette  PROM

**Customer Checksum:** \_\_\_\_\_

**Company Name:** \_\_\_\_\_

**Signature (Engineer):** \_\_\_\_\_

Please answer the following questions:



**Application** (Product Model ID: \_\_\_\_\_)

- |                                       |   |  |
|---------------------------------------|---|--|
| <input type="checkbox"/> Audio        | <input type="checkbox"/> Video          | <input type="checkbox"/> Telecom           |
| <input type="checkbox"/> LCD Databank | <input type="checkbox"/> Caller ID      | <input type="checkbox"/> LCD Game          |
| <input type="checkbox"/> Industrials  | <input type="checkbox"/> Home Appliance | <input type="checkbox"/> Office Automation |
| <input type="checkbox"/> Remocon      | <input type="checkbox"/> Other          |  |

Please describe in detail its application

---



# S3P7 SERIES OTP FACTORY WRITING ORDER FORM (1/2)

**Product Description:**

Device Number: S3P7 \_\_\_\_\_ - \_\_\_\_\_ (write down the ROM code number)

Product Order Form:  Package  Pellet  Wafer

If the product order form is package: Package Type: \_\_\_\_\_

**Package Marking (Check One):**

Standard  Custom A (Max 10 chars)  Custom B (Max 10 chars each line)

<b>SEC</b> @ YWW Device Name
---------------------------------

@ YWW Device Name _____
-------------------------------

@ YWW _____ _____
-------------------------

@ : Assembly site code, Y : Last number of assembly year, WW : Week of assembly

**Delivery Dates and Quantity:**

ROM Code Release Date	Required Delivery Date of Device	Quantity

Please answer the following questions:

**What is the purpose of this order?**

- New product development  Upgrade of an existing product  
 Replacement of an existing microcontroller  Other

If you are replacing an existing microcontroller, please indicate the former microcontroller name ( \_\_\_\_\_ )

**What are the main reasons you decided to use a Samsung microcontroller in your product?**

Please check all that apply.

- Price  Product quality  Features and functions  
 Development system  Technical support  Delivery on time  
 Used same micom before  Quality of documentation  Samsung reputation

**Customer Information:**

Company Name: \_\_\_\_\_ Telephone number \_\_\_\_\_

Signatures: \_\_\_\_\_ (Person placing the order) \_\_\_\_\_ (Technical Manager)



# S3P7588X OTP FACTORY WRITING ORDER FORM (2/2)

**Device Number:** S3P7588X-\_\_\_\_\_ (write down the ROM code number)

**Customer Checksums:** \_\_\_\_\_

**Company Name:** \_\_\_\_\_

**Signature (Engineer):** \_\_\_\_\_


**Read Protection<sup>(1)</sup>:**       **Yes**                       **No**

Please answer the following questions:

 **Are you going to continue ordering this device?**

**Yes**                       **No**

**If so, how much will you be ordering?** \_\_\_\_\_ pcs

 **Application** (Product Model ID: \_\_\_\_\_)

- |                                       |   |  |
|---------------------------------------|---|--|
| <input type="checkbox"/> Audio        | <input type="checkbox"/> Video          | <input type="checkbox"/> Telecom           |
| <input type="checkbox"/> LCD Databank | <input type="checkbox"/> Caller ID      | <input type="checkbox"/> LCD Game          |
| <input type="checkbox"/> Industrials  | <input type="checkbox"/> Home Appliance | <input type="checkbox"/> Office Automation |
| <input type="checkbox"/> Remocon      | <input type="checkbox"/> Other          |  |

Please describe in detail its application

---

**NOTES:**

1. Once you choose a read protection, you cannot read again the programming code from the EPROM.
2. OTP Writing will be executed in our manufacturing site.
3. The writing program is completely verified by a customer. Samsung does not take on any responsibility for errors occurred from the writing program.