

**HT48R10A-1, HT48R30A-1,
HT48R50A-1, HT48R70A-1,
HT48RU80
I/O Type MCU
Handbook**

Third Edition

June 2006

Copyright © 2006 by HOLTEK SEMICONDUCTOR INC. All rights reserved. Printed in Taiwan. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form by any means, electronic, mechanical photocopying, recording or otherwise without the prior written permission of HOLTEK SEMICONDUCTOR INC.

Contents

| | |
|---|----------|
| Part I Microcontroller Profile | 1 |
| Chapter 1 Hardware Structure | 3 |
| Introduction | 3 |
| Features | 4 |
| Technology Features | 4 |
| Kernel Features | 4 |
| Peripheral Features | 5 |
| Selection Table | 5 |
| Block Diagram | 6 |
| Pin Assignment | 7 |
| Pin Description | 8 |
| Absolute Maximum Ratings | 14 |
| D.C. Characteristics | 14 |
| A.C. Characteristics | 15 |
| System Architecture | 16 |
| Clocking and Pipelining | 16 |
| Program Counter | 17 |
| Stack | 19 |
| Arithmetic and Logic Unit – ALU | 20 |
| Program Memory | 20 |
| Organization | 20 |
| Special Vectors | 21 |
| Managing Multiple Banks | 22 |
| Look-up Table | 24 |
| Table Program Example | 25 |
| Data Memory | 28 |
| Organization | 28 |
| General Purpose Data Memory | 28 |
| Special Purpose Data Memory | 30 |

| | |
|---|----|
| Special Function Registers | 31 |
| Indirect Addressing Registers – IAR, IAR0, IAR1 | 31 |
| Memory Pointers – MP, MP0, MP1 | 31 |
| Bank Pointer – BP | 32 |
| Accumulator – ACC..... | 33 |
| Program Counter Low Register – PCL | 33 |
| Look-up Table Registers – TBLP, TBHP, TBLH | 33 |
| Watchdog Timer Register – WDTS | 33 |
| Status Register – STATUS | 34 |
| Interrupt Control Registers – INTC, INTC0, INTC1 | 35 |
| Timer/Event Counter Registers | 35 |
| Input/Output Ports and Control Registers | 35 |
| UART Registers – USR, UCR1, UCR2, TXR/RXR, BRG | 36 |
| Input/Output Ports | 36 |
| Pull-high Resistors | 36 |
| Port A Wake-up | 36 |
| I/O Port Control Registers | 37 |
| Pin-shared Functions | 37 |
| Programming Considerations | 40 |
| Timer/Event Counters | 41 |
| Configuring the Timer/Event Counter Input Clock Source | 41 |
| Timer Registers – TMR, TMR0, TMR0L/TMR0H, TMR1L/TMR1H, TMR2 | 43 |
| Timer Control Registers – TMRC, TMR0C, TMR1C, TMR2C | 44 |
| Configuring the Timer Mode | 47 |
| Configuring the Event Counter Mode | 47 |
| Configuring the Pulse Width Measurement Mode | 48 |
| Programmable Frequency Divider (PFD) and Buzzer Application | 49 |
| Prescaler | 50 |
| I/O Interfacing | 50 |
| Programming Considerations | 50 |
| Timer Program Example | 50 |
| Interrupts | 52 |
| Interrupt Registers | 52 |
| Interrupt Priority | 56 |
| External Interrupt | 56 |
| Timer/Event Counter Interrupt | 57 |
| UART Interrupt | 57 |
| Programming Considerations | 58 |
| Reset and Initialization | 58 |
| Reset | 58 |
| Universal Asynchronous Receiver/Transmitter – UART | 66 |
| UART Features | 66 |
| UART External Pin Interfacing | 66 |
| UART Data Transfer Scheme | 67 |
| UART Status and Control Registers | 67 |
| Baud Rate Generator | 73 |
| Setting Up and Controlling the UART | 75 |
| UART Transmitter | 76 |

| | |
|--|------------|
| UART Receiver | 78 |
| Managing Receiver Errors | 80 |
| UART Interrupt Scheme | 81 |
| Address Detect Mode | 82 |
| UART Operation in Power Down Mode | 82 |
| UART Sample Program | 83 |
| Oscillator | 83 |
| System Clock Configurations | 83 |
| System Crystal/Ceramic Oscillator | 83 |
| System RC Oscillator | 84 |
| Internal System RC Oscillator | 84 |
| RTC Oscillator | 85 |
| Watchdog Timer Oscillator | 86 |
| Power Down Mode and Wake-up | 86 |
| Power Down Mode | 86 |
| Entering the Power Down Mode | 86 |
| Standby Current Considerations | 86 |
| Wake-up | 87 |
| Watchdog Timer | 88 |
| Configuration Options | 89 |
| Application Circuits | 90 |
| | |
| Part II Programming Language | 91 |
| Chapter 2 Instruction Set Introduction | 93 |
| Instruction Set | 93 |
| Instruction Timing | 93 |
| Moving and Transferring Data | 94 |
| Arithmetic Operations | 94 |
| Logical and Rotate Operations | 94 |
| Branches and Control Transfer | 94 |
| Bit Operations | 95 |
| Table Read Operations | 95 |
| Other Operations | 95 |
| Instruction Set Summary | 95 |
| Convention | 95 |
| Chapter 3 Instruction Definition | 99 |
| Chapter 4 Assembly Language and Cross Assembler | 111 |
| Notational Conventions | 111 |
| Statement Syntax | 112 |
| Name | 112 |
| Operation | 112 |

| | |
|--|-----|
| Operand | 112 |
| Comment | 113 |
| Assembly Directives | 113 |
| Conditional Assembly Directives | 113 |
| File Control Directives | 114 |
| Program Directives | 115 |
| Data Definition Directives | 118 |
| Macro Directives | 120 |
| Assembly Instructions | 122 |
| Name | 122 |
| Mnemonic | 122 |
| Operand, Operator and Expression | 122 |
| Miscellaneous | 124 |
| Forward References | 124 |
| Local Labels | 124 |
| Reserved Assembly Language Words | 125 |
| Cross Assembler Options | 126 |
| Assembly Listing File Format | 126 |
| Source Program Listing | 126 |
| Summary of Assembly | 127 |
| Miscellaneous | 127 |

Part III Development Tools 129

Chapter 5 MCU Programming Tools 131

| | |
|---|-----|
| HT-IDE Development Environment | 131 |
| Holtek In-Circuit Emulator – HT-ICE | 132 |
| HT-ICE Interface Card | 132 |
| OTP Programmer | 133 |
| OTP Adapter Card | 133 |
| System Configuration | 133 |
| HT-ICE Interface Card Settings | 134 |
| Installation | 135 |
| System Requirement | 135 |
| Hardware Installation | 135 |
| Software Installation | 135 |

Chapter 6 Quick Start 141

| | |
|--|-----|
| Step 1 – Create a New Project | 141 |
| Step 2 – Add Source Program Files to the Project | 141 |
| Step 3 – Build the Project | 141 |
| Step 4 – Programming the OTP Device | 141 |
| Step 5 – Transmit Code to Holtek | 142 |

| | |
|--|------------|
| Appendix | 143 |
| Appendix A Device Characteristic Graphics | 145 |
| Appendix B Package Information | 155 |

Preface

Since the founding of the company, Holtek Semiconductor has concentrated much of its design efforts in the area of microcontroller development. Although supplying a wide range of semiconductor devices, the microcontroller category has always been a key product category within the Holtek range, and one which will continue to expand as their devices increase in functionality and maturity. By capitalizing on the substantial accumulated skills within its dedicated microcontroller development department, Holtek has been able to release a comprehensive range of high quality low-cost microcontroller devices for a wide range of application areas. On some of the devices a full-duplex asynchronous serial communications UART function is integrated, giving these devices the ability to easily communicate with external serial interfaces. Holtek's high quality embedded I/O microcontroller solutions provide a means for customers to greatly enhance the functional contents of their products, which when combined with Holtek's comprehensive range of development tools provide designers with the means to reduce their design to market times and greatly increasing their added value.

This handbook is divided into three parts for user convenience. Most details regarding general datasheet information and device specification is located within Part I. Information related to microcontroller programming such as device instruction set, instruction definition, and assembly language directives is found within Part II. Part III relates to the Holtek range of Development Tools where information can be found on their installation and use.

By compiling all relevant data together in one handbook, we hope users of the Holtek range of I/O Type microcontroller devices will have at their fingertips a useful, complete and simple means to efficiently implement their microcontroller applications. Holtek's efforts to combine information on device specifications, programming and development tools into one publication have produced a handbook which with careful use by the user should result in trouble free designs and the maximum benefit being gained from the many features of Holtek microcontroller devices. We recommend that users regularly check our website for the latest updates to our handbook and also welcome feedback and comments from our customers regarding further improvements.

Part I

Microcontroller Profile

Chapter 1**Hardware Structure****1**

This section is the main datasheet section of the I/O Type microcontroller handbook and contains all the parameters and information related to the hardware. The information contained provides designers with details on all the main hardware features of the I/O Type microcontroller range which together with the programming section contains the information to enable swift and successful implementation of user microcontroller applications. By proper consultation of the relevant parts of this section, users can ensure that they make the most efficient use of the flexible and multi-function features within the I/O Type microcontroller series.

Introduction

The HT48R10A-1/HT48C10-1, HT48R30A-1/HT48C30-1, HT48R50A-1/HT48C50-1, HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80 are 8-bit high performance, RISC architecture microcontroller devices specifically designed for multiple I/O control product applications. Device flexibility is enhanced with their internal special features such as HALT and wake-up functions, oscillator options, buzzer driver, UART, etc. These features combine to ensure applications require a minimum of external components and therefore reduce overall product costs. Having the advantages of low power consumption, high performance, I/O flexibility as well as low cost, these devices have the versatility to suit a wide range of application possibilities such as industrial control, consumer products, subsystem controllers, etc. Many features are common to all devices, however, they differ in areas such as I/O pin count, RAM and ROM capacity, timer number and size etc. An additional feature that is incorporated within the HT48RU80/HT48CU80 devices is a full-duplex asynchronous serial communications UART function.

The HT48R10A-1, HT48R30A-1, HT48R50A-1, HT48R70A-1 and HT48RU80 are OTP devices offering the advantages of easy and effective program updates, using the Holtek range of development and programming tools. These devices provide the designer with the means for fast and low cost product development cycles. However, for applications that are at a mature state in their design process, the HT48C10-1, HT48C30-1, HT48C50-1, HT48C70-1 and HT48CU80 mask version devices offer a complementary device for products with high volume and low cost demands. Fully pin and functionally compatible with their OTP sister devices, such mask version devices provide the ideal substitute for products which have gone beyond their development cycle and are facing cost down demands.

Features

Technology Features

- High-performance RISC Architecture
- Low-power Fully Static CMOS Design
- Operating Voltage:
f_{SYS}=4MHz: 2.2V~5.5V
f_{SYS}=8MHz: 3.3V~5.5V
- Power Consumption:
2mA Typical at 5V 4MHz
Maximum of 1μA Standby Current at 3V with WDT and RTC Disabled
- Temperature Range:
Operating Temperature -40°C to 85°C (Industrial Grade)
Storage Temperature -50°C to 125°C

Kernel Features

- Program Memory:
1K×14 OTP/Mask ROM (HT48R10A-1/HT48C10-1)
2K×14 OTP/Mask ROM (HT48R30A-1/HT48C30-1)
4K×15 OTP/Mask ROM (HT48R50A-1/HT48C50-1)
8K×16 OTP/Mask ROM (HT48R70A-1/HT48C70-1)
16K×16 OTP/Mask ROM (HT48RU80/HT48CU80)
- Data Memory:
64×8 SRAM (HT48R10A-1/HT48C10-1)
96×8 SRAM (HT48R30A-1/HT48C30-1)
160×8 SRAM (HT48R50A-1/HT48C50-1)
224×8 SRAM (HT48R70A-1/HT48C70-1)
576×8 SRAM (HT48RU80/HT48CU80)
- Table Read Function
- Multi-level Hardware Stack:
4-level (HT48R10A-1/HT48C10-1, HT48R30A-1/HT48C30-1)
6-level (HT48R50A-1/HT48C50-1)
16-level (HT48R70A-1/HT48C70-1, HT48RU80/HT48CU80)
- Direct and Indirect Data Addressing Mode
- Bit Manipulation Instructions
- 63 Powerful Instructions
- Most Instructions Implemented in 1 Machine Cycle

Peripheral Features

- From 21 to 56 Bidirectional I/O with Pull-high Options
- Port A Wake-up Options
- One or Two External Interrupt Inputs
- Event Counter Input
- Full Timer Functions with Prescaler and Interrupt
- Watchdog Timer (WDT)
- HALT and Wake-up Feature for Power Saving Operation
- PFD/Buzzer Driver Outputs
- On-chip Crystal and RC Oscillator
- 32768Hz Real Time Clock (RTC) Function
- Universal Asynchronous Receiver Transmitter – UART (HT48RU80/HT48CU80)
- Low Voltage Reset (LVR) Feature for Brown-out Protection
- Programming Interface with Code Protection
- Mask Version Devices Available for High Volume Production
- Full Suite of Supported Hardware and Software Tools Available

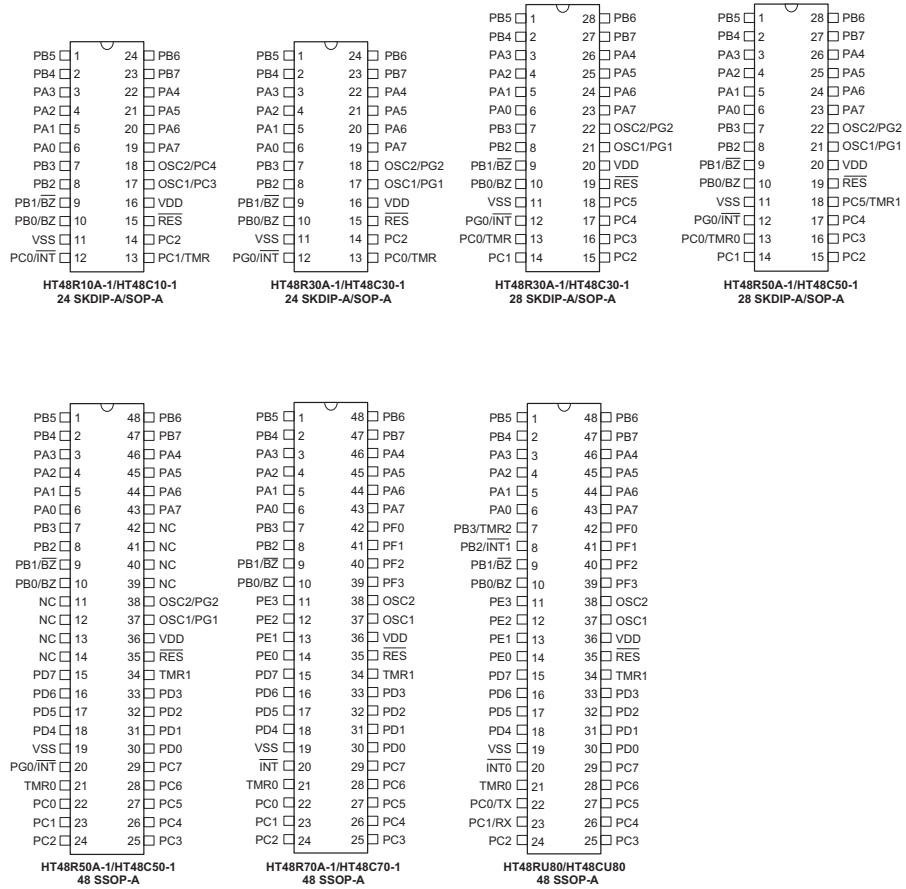
Selection Table

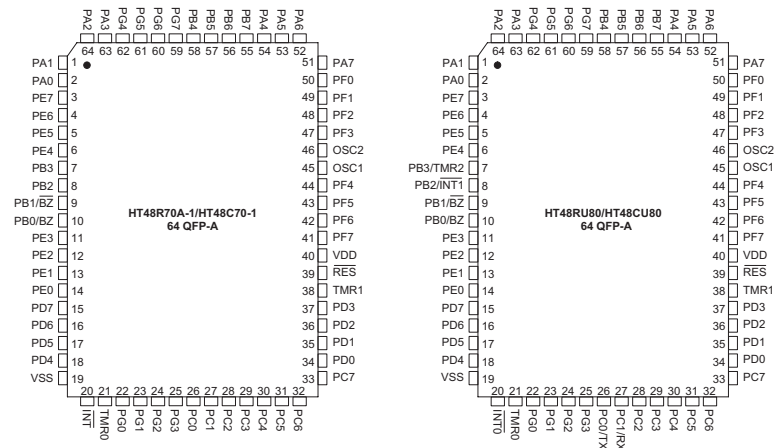
The series of I/O microcontrollers include a comprehensive range of features, some of which are standard and some of which are device dependent. Most features are common to all devices, the main features distinguishing them are Program Memory, Data Memory capacity, I/O count, timer and UART functions. To assist users in their selection of the most appropriate device for their application, the following table, which summarizes the main features of each device, is provided.

| Part No. | VDD | Program Memory | Data Memory | I/O | Timer | Interrupt | UART | Stack | Package Types |
|-------------------------|-----------|----------------|-------------|-----|---------------------|-----------|------|-------|-----------------------------|
| HT48R10A-1 HT48C10-1 | 2.2V~5.5V | 1K×14 | 64×8 | 21 | 8-bit×1 | 2 | — | 4 | 24SKDIP/SOP |
| HT48R30A-1 HT48C30-1 | 2.2V~5.5V | 2K×14 | 96×8 | 25 | 8-bit×1 | 2 | — | 4 | 24SKDIP/SOP, 28SKDIP/SOP |
| HT48R50A-1 HT48C50-1 | 2.2V~5.5V | 4K×15 | 160×8 | 35 | 8-bit×1 16-bit×1 | 3 | — | 6 | 28SKDIP/SOP, 48SSOP |
| HT48R70A-1 HT48C70-1 | 2.2V~5.5V | 8K×16 | 224×8 | 56 | 16-bit×2 | 3 | — | 16 | 48SSOP, 64QFP |
| HT48RU80 HT48CU80 | 2.2V~5.5V | 16K×16 | 576×8 | 56 | 8-bit×1 16-bit×2 | 6 | √ | 16 | 48SSOP, 64QFP |

- Note**
1. Part numbers including "C" are mask version devices while "R" are OTP devices.
 2. For devices that exist in two package formats, the table reflects the situation for the larger package.

Pin Assignment





Note The pin compatibility features of the microcontroller packages allow for straightforward upgrading to devices of higher functionality with minimal changes to application hardware.

Pin Description

HT48R10A-1/HT48C10-1

| Pin Name | I/O | Configuration Option | Description |
|-----------------------------|-----|---|--|
| PA0~PA7 | I/O | Pull-high Wake-up Schmitt Trigger | Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input by configuration option. Software instructions determine if the pin is a CMOS output or input. Configuration options determine if all pins on this port have pull-high resistors and if the inputs are Schmitt Trigger or non Schmitt Trigger. |
| PB0/BZ PB1/BZ PB2~PB7 | I/O | Pull-high I/O or BZ/BZ | Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. Pins PB0 and PB1 are pin-shared with BZ and BZ, respectively. |
| PC0/INT PC1/TMR PC2 | I/O | Pull-high | Bidirectional 3-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. Pin PC0 is pin-shared with external interrupt pin INT and PC1 shared with external timer pin TMR. The external interrupt is activated on a high to low transition. |

| Pin Name | I/O | Configuration Option | Description |
|----------------------|--------|--|--|
| OSC1/PC3 OSC2/PC4 | I O | Pull-high Crystal or RC or Int. RC+I/O or Int. RC+RTC | OSC1, OSC2 are connected to an external RC network or external Crystal (determined by configuration option) for the internal system clock. For external RC system clock operation, OSC2 is an output pin for 1/4 system clock. These two pins can also be optioned as an RTC oscillator (32768Hz) or I/O lines. In these two cases, the system clock comes from an internal RC oscillator whose nominal frequency at 5V has 4 options, 3.2MHz, 1.6MHz, 800kHz, 400kHz. If the pins are used as normal I/O pins, then pull-high options are available. If used as oscillator pins, bits PC3 and PC4 will be free for use by the application program. In this case the pull-high options are disabled. |
| RES | I | — | Schmitt Trigger reset input. Active low. |
| VDD | — | — | Positive power supply |
| VSS | — | — | Negative power supply, ground |

- Note**
1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
 2. Individual pins cannot be selected to have pull-high resistors. If the pull-high configuration is chosen for a particular port, then all input pins on this port will be connected to pull-high resistors.

HT48R30A-1/HT48C30-1

| Pin Name | I/O | Configuration Option | Description |
|-----------------------------|-----|---|---|
| PA0~PA7 | I/O | Pull-high Wake-up Schmitt Trigger | Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input by configuration option. Software instructions determine if the pin is a CMOS output or input. Configuration options determine if all pins on this port have pull-high resistors and if the inputs are Schmitt Trigger or non Schmitt Trigger. |
| PB0/BZ PB1/BZ PB2~PB7 | I/O | Pull-high I/O or BZ/BZ | Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. Pins PB0 and PB1 are pin-shared with BZ and BZ, respectively. |
| PC0/TMR PC1~PC5 | I/O | Pull-high | Bidirectional 6-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. PC0 is pin-shared with external timer pin TMR. |

| Pin Name | I/O | Configuration Option | Description |
|------------------------------|--------|--|--|
| PG0/ $\overline{\text{INT}}$ | I/O | Pull-high | Bidirectional 1-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if the pin has a pull-high resistor. PG0 is pin-shared with external interrupt pin INT. The external interrupt input is activated on a high to low transition. |
| OSC1/PG1 OSC2/PG2 | I O | Pull-high Crystal or RC or Int. RC+I/O or Int. RC+RTC | OSC1, OSC2 are connected to an external RC network or external Crystal (determined by configuration option) for the internal system clock. For external RC system clock operation, OSC2 is an output pin for 1/4 system clock. These two pins can also be optioned as an RTC oscillator (32768Hz) or I/O lines. In these two cases, the system clock comes from an internal RC oscillator whose nominal frequency at 5V has 4 options, 3.2MHz, 1.6MHz, 800kHz, 400kHz. If the pins are used as normal I/O pins, then pull-high options are available. If used as oscillator pins, bits PG1 and PG2 will be free for use by the application program. In this case the pull-high options are disabled. |
| $\overline{\text{RES}}$ | I | — | Schmitt Trigger reset input. Active low. |
| VDD | — | — | Positive power supply |
| VSS | — | — | Negative power supply, ground |

- Note**
1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
 2. Individual pins cannot be selected to have pull-high resistors. If the pull-high configuration is chosen for a particular port, then all input pins on this port will be connected to pull-high resistors.
 3. Pins PC1 and PC3~PC5 only exist on the 28-pin package. On the 24-pin package, these pins are not available.

HT48R50A-1/HT48C50-1

| Pin Name | I/O | Configuration Option | Description |
|----------|-----|---|---|
| PA0~PA7 | I/O | Pull-high Wake-up Schmitt Trigger | Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input by configuration option. Software instructions determine if the pin is a CMOS output or input. Configuration options determine if all pins on this port have pull-high resistors and if the inputs are Schmitt Trigger or non Schmitt Trigger. |

| Pin Name | I/O | Configuration Option | Description |
|--|--------|--|--|
| PB0/BZ PB1/BZ PB2~PB7 | I/O | Pull-high I/O or BZ/BZ | Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. Pins PB0 and PB1 are pin-shared with BZ and BZ, respectively. |
| PC0/TMR0 PC5/TMR1 PC1~PC4 PC6~PC7 | I/O | Pull-high | Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. TMR0 and TMR1 are pin-shared with PC0 and PC5 respectively in the 28-pin package. |
| PD0~PD7 | I/O | Pull-high | Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if all pins on this port have pull-high resistors. |
| PG0/INT | I/O | Pull-high | Bidirectional 1-bit input/output ports. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if the pin has a pull-high resistor. PG0 is pin-shared with external interrupt pin INT. |
| OSC1/PG1 OSC2/PG2 | I O | Pull-high Crystal or RC or Int. RC+I/O or Int. RC+RTC | OSC1, OSC2 are connected to an external RC network or external Crystal (determined by configuration option) for the internal system clock. For external RC system clock operation, OSC2 is an output pin for 1/4 system clock. These two pins can also be optioned as an RTC oscillator (32768Hz) or I/O lines. In these two cases, the system clock comes from an internal RC oscillator whose nominal frequency at 5V has 4 options, 3.2MHz, 1.6MHz, 800kHz, 400kHz. If the pins are used as normal I/O pins, then pull-high options are available. If used as oscillator pins, bits PG1 and PG2 will be free for use by the application program. In this case the pull-high options are disabled. |
| RES | I | — | Schmitt Trigger reset input. Active low. |
| VDD | — | — | Positive power supply |
| VSS | — | — | Negative power supply, ground |

- Note**
1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
 2. Individual pins cannot be selected to have pull-high resistors. If the pull-high configuration is chosen for a particular port, then all input pins on this port will be connected to pull-high resistors.
 3. On the 48-pin package Port C has no shared pins. All of Port C pins exist as I/Os as the TMR0 and TMR1 are independent pins.
 4. Pins PC6 and PC7 only exist on the 48-pin package.
 5. Port D is only present on the 48-pin package.

HT48R70A-1/HT48C70-1

| Pin Name | I/O | Configuration Option | Description |
|--|--------|---|---|
| PA0~PA7 | I/O | Pull-high Wake-up Schmitt Trigger | Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input by configuration option. Software instructions determine if the pin is a CMOS output or input. Configuration options determine if all pins on this port have pull-high resistors and if the inputs are Schmitt Trigger or non Schmitt Trigger. |
| PB0/BZ PB1/BZ PB2~PB7 PC0~PC7 PD0~PD7 PE0~PE7 PF0~PF7 PG0~PG7 | I/O | Pull-high I/O or BZ/BZ | Bidirectional 8-bit input/output ports. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option for each port determines if all pins on the relevant port have pull-high resistors. Pins PB0 and PB1 are pin-shared with BZ and BZ respectively. |
| INT | I | — | External interrupt Schmitt Trigger input. Edge triggered on high to low transition. |
| TMR0 | I | — | Schmitt Trigger input for Timer/Event Counter 0 |
| TMR1 | I | — | Schmitt Trigger input for Timer/Event Counter 1 |
| OSC1 OSC2 | I O | Crystal or RC or Int. RC+RTC | OSC1, OSC2 are connected to an external RC network or external Crystal (determined by configuration option) for the internal system clock. For external RC system clock operation, OSC2 is an output pin for 1/4 system clock. These two pins also can be optioned as an RTC oscillator (32768Hz). In this case, the system clock comes from an internal RC oscillator whose nominal frequency at 5V has 4 options, 3.2MHz, 1.6MHz, 800kHz, 400kHz. |
| RES | I | — | Schmitt Trigger reset input. Active low. |
| VDD | — | — | Positive power supply |
| VSS | — | — | Negative power supply, ground |

- Note**
1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
 2. Individual pins cannot be selected to have pull-high resistors. If the pull-high configuration is chosen for a particular port, then all input pins on this port will be connected to pull-high resistors.
 3. Pins PE4~PE7 and pins PF4~PF7 only exist on the 64-pin package.
 4. Port G only exists on the 64-pin package.

HT48RU80/HT48CU80

| Pin Name | I/O | Configuration Option | Description |
|--|--------|---|---|
| PA0~PA7 | I/O | Pull-high Wake-up Schmitt Trigger | Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input by configuration option. Software instructions determine if the pin is a CMOS output or input. Configuration options determine if all pins on this port have pull-high resistors and if the inputs are Schmitt Trigger or non Schmitt Trigger. |
| PB0/BZ PB1/ \overline{BZ} PB2/ $\overline{INT1}$ PB3/TMR2 PB4~PB7 PC0/TX PC1/RX PC2~PC7 PD0~PD7 PE0~PE7 PF0~PF7 PG0~PG7 | I/O | Pull-high I/O or BZ/ \overline{BZ} | Bidirectional 8-bit input/output ports. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option for each port determines if all pins on the relevant port have pull-high resistors. Pins PB0, PB1, PB2 and PB3 are pin-shared with BZ, \overline{BZ} , $\overline{INT1}$ and TMR2 respectively. Pins PC0 and PC1 are pin-shared with the UART pins TX and RX. |
| $\overline{INT0}$ | I | — | External interrupt Schmitt Trigger input. Edge triggered on high to low transition. |
| TMR0 | I | — | Schmitt Trigger input for Timer/Event Counter 0 |
| TMR1 | I | — | Schmitt Trigger input for Timer/Event Counter 1 |
| OSC1 OSC2 | I O | Crystal or RC or Int. RC+RTC | OSC1, OSC2 are connected to an external RC network or external Crystal (determined by configuration option) for the internal system clock. For external RC system clock operation, OSC2 is an output pin for 1/4 system clock. These two pins also can be optioned as an RTC oscillator (32768Hz). In this case, the system clock comes from an internal RC oscillator whose nominal frequency at 5V has 4 options, 3.2MHz, 1.6MHz, 800kHz, 400kHz. |
| \overline{RES} | I | — | Schmitt Trigger reset input. Active low. |
| VDD | — | — | Positive power supply |
| VSS | — | — | Negative power supply, ground |

- Note**
1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
 2. Individual pins cannot be selected to have pull-high resistors. If the pull-high configuration is chosen for a particular port, then all input pins on this port will be connected to pull-high resistors.
 3. Pins PE4~PE7 and pins PF4~PF7 only exist on the 64-pin package.
 4. Port G only exists on the 64-pin package.

Absolute Maximum Ratings

| | |
|----------------------------|----------------------------------|
| Supply Voltage..... | $V_{SS}-0.3V$ to $V_{SS}+6.0V$ |
| Input Voltage | $V_{SS}-0.3V$ to $V_{DD}+0.3V$ |
| Storage Temperature..... | $-50^{\circ}C$ to $125^{\circ}C$ |
| Operating Temperature..... | $-40^{\circ}C$ to $85^{\circ}C$ |

These are stress ratings only. Stresses exceeding the range specified under Absolute Maximum Ratings may cause substantial damage to the device. Functional operation of this device at other conditions beyond those listed in the specification is not implied and prolonged exposure to extreme conditions may affect device reliability.

D.C. Characteristics

 $T_a=25^{\circ}C$

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|------------|--|-----------------|----------------------------------|-------------|------|-------------|-----------|
| | | V_{DD} | Conditions | | | | |
| V_{DD} | Operating Voltage | — | $f_{SYS}=4MHz$ | 2.2 | — | 5.5 | V |
| | | — | $f_{SYS}=8MHz$ | 3.3 | — | 5.5 | V |
| I_{DD1} | Operating Current (Crystal OSC) | 3V | No load, $f_{SYS}=4MHz$ | — | 0.6 | 1.5 | mA |
| | | 5V | | — | 2 | 4 | mA |
| I_{DD2} | Operating Current (RC OSC) | 3V | No load, $f_{SYS}=4MHz$ | — | 0.8 | 1.5 | mA |
| | | 5V | | — | 2.5 | 4 | mA |
| I_{DD3} | Operating Current (Crystal OSC, RC OSC) | 5V | No load, $f_{SYS}=8MHz$ | — | 4 | 8 | mA |
| I_{STB1} | Standby Current (WDT Enabled) | 3V | No load, system HALT, RTC Off | — | — | 5 | μA |
| | | 5V | | — | — | 10 | μA |
| I_{STB2} | Standby Current (WDT Disabled) | 3V | No load, system HALT, RTC Off | — | — | 1 | μA |
| | | 5V | | — | — | 2 | μA |
| I_{STB3} | Standby Current (WDT Disabled) | 3V | No load, system HALT, RTC On | — | — | 5 | μA |
| | | 5V | | — | — | 10 | μA |
| V_{IL1} | Input Low Voltage for I/O Ports | — | — | 0 | — | $0.3V_{DD}$ | V |
| V_{IH1} | Input High Voltage for I/O Ports | — | — | $0.7V_{DD}$ | — | V_{DD} | V |
| V_{IL2} | Input Low Voltage (\overline{RES}) | — | — | 0 | — | $0.4V_{DD}$ | V |
| V_{IH2} | Input High Voltage (\overline{RES}) | — | — | $0.9V_{DD}$ | — | V_{DD} | V |
| V_{LVR} | Low Voltage Reset | — | LVR enabled | 2.7 | 3 | 3.3 | V |
| I_{OL} | I/O Port Sink Current | 3V | $V_{OL}=0.1V_{DD}$ | 4 | 8 | — | mA |
| | | 5V | | 10 | 20 | — | mA |
| I_{OH} | I/O Port Source Current | 3V | $V_{OH}=0.9V_{DD}$ | -2 | -4 | — | mA |
| | | 5V | | -5 | -10 | — | mA |
| R_{PH} | Pull-high Resistance | 3V | — | 20 | 60 | 100 | $k\Omega$ |
| | | 5V | — | 10 | 30 | 50 | $k\Omega$ |

A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---------------------|---|-----------------|-----------------------|------|-------|------|--------------------|
| | | V _{DD} | Conditions | | | | |
| f _{SYS1} | System Clock (Crystal OSC) | — | 2.2V~5.5V | 400 | — | 4000 | kHz |
| | | — | 3.3V~5.5V | 400 | — | 8000 | kHz |
| f _{SYS2} | System Clock (RC OSC) | — | 2.2V~5.5V | 400 | — | 4000 | kHz |
| | | — | 3.3V~5.5V | 400 | — | 8000 | kHz |
| f _{SYS3} | System Clock (Internal RC OSC) | 5V | 3.2MHz | 1800 | — | 5400 | kHz |
| | | | 1.6MHz | 900 | — | 2700 | kHz |
| | | | 800kHz | 450 | — | 1350 | kHz |
| | | | 400kHz | 225 | — | 675 | kHz |
| f _{TIMER} | Timer I/P Frequency (TMR) | — | 2.2V~5.5V | 0 | — | 4000 | kHz |
| | | — | 3.3V~5.5V | 0 | — | 8000 | kHz |
| t _{WDTOSC} | Watchdog Oscillator Period | 3V | — | 45 | 90 | 180 | μs |
| | | 5V | — | 32 | 65 | 130 | μs |
| t _{WDT1} | Watchdog Time-out Period (WDT OSC) | 3V | Without WDT prescaler | 11 | 23 | 46 | ms |
| | | 5V | | 8 | 17 | 33 | ms |
| t _{WDT2} | Watchdog Time-out Period (System Clock) | — | Without WDT prescaler | — | 1024 | — | t _{sys} * |
| t _{WDT3} | Watchdog Time-out Period (RTC OSC) | — | Without WDT prescaler | — | 7.812 | — | ms |
| t _{RES} | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| t _{SST} | System Start-up Timer Period | — | Wake-up from HALT | — | 1024 | — | t _{sys} * |
| t _{LVR} | Low Voltage Width to Reset | — | — | 0.25 | 1 | 2 | ms |
| t _{INT} | Interrupt Pulse Width | — | — | 1 | — | — | μs |

 *t_{sys} = 1/f_{SYS1}, 1/f_{SYS2} or 1/f_{SYS3}

Note The internal RC system clock has a typical base frequency of 3.2MHz at 5V. The other internal RC system typical clock frequencies of 1.6MHz, 800kHz and 400kHz at 5V are division ratios of this 3.2MHz base frequency.

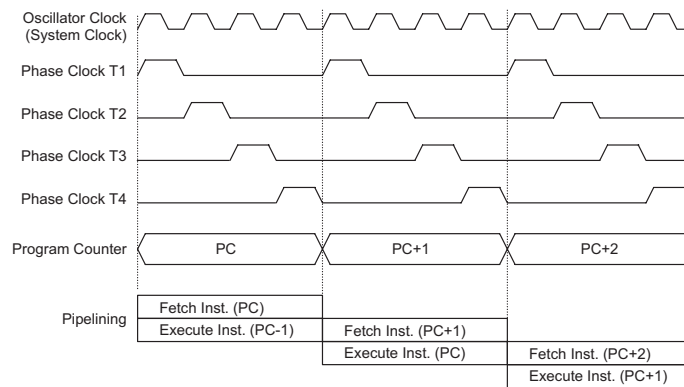
System Architecture

A key factor in the high performance features of the range of I/O Type microcontrollers is attributed to the internal system architecture. The range of devices take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one cycle, with the exception of branch or call instructions. An 8-bit wide ALU is used in practically all operations of the instruction set. It carries out arithmetic operations, logic operations, rotation, increment, decrement, branch decisions, etc. The internal data path is simplified by moving data through the Accumulator and the ALU. Certain internal registers are implemented in the Data Memory and can be directly or indirectly addressed. The simple addressing methods of these registers along with additional architectural features ensure that a minimum of external components is required to provide a functional I/O control system with maximum reliability and flexibility. This makes these devices suitable for low cost, high-volume production for controller applications requiring from 1K up to 16K words of Program Memory and from 64 to 576 bytes of data storage.

Clocking and Pipelining

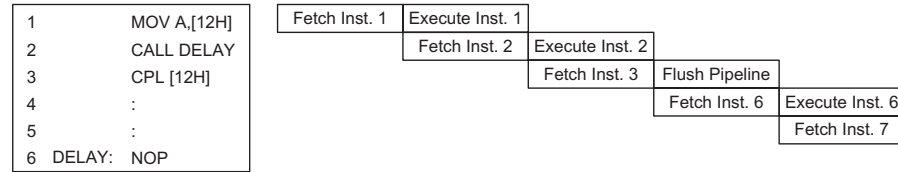
The main system clock, derived from either a Crystal/Resonator or RC oscillator is subdivided into four internally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. Although the fetching and execution of instructions takes place in consecutive instruction cycles, the pipelining structure of the microcontroller ensures that instructions are effectively executed in one instruction cycle. The exception to this are instructions where the contents of the Program Counter are changed, such as subroutine calls or jumps, in which case the instruction will take one more instruction cycle to execute.

Note When the RC oscillator is used, OSC2 is freed for use as a T1 phase clock synchronizing pin. This T1 phase clock has a frequency of $f_{SYS}/4$ with a 1:3 high/low duty cycle.



System Clocking and Pipelining

For instructions involving branches, such as jump or call instructions, two machine cycles are required to complete instruction execution. An extra cycle is required as the program takes one cycle to first obtain the actual jump or call address and then another cycle to actually execute the branch. The requirement for this extra cycle should be taken into account by programmers in timing sensitive applications.



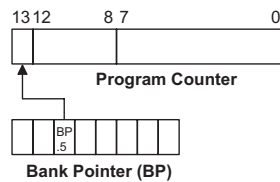
Program Counter

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions such as JMP or CALL that demand a jump to a non-consecutive Program Memory address. For the I/O series of microcontrollers, note that the Program Counter width varies with the Program Memory capacity depending upon which device is selected. However, it must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by the user.

When executing instructions requiring jumps to non-consecutive addresses such as a jump instruction, a subroutine call, interrupt or reset, etc., the microcontroller manages program control by loading the required address into the Program Counter. For conditional skip instructions, once the condition has been met, the next instruction, which has already been fetched during the present instruction execution, is discarded and a dummy cycle takes its place while the correct instruction is obtained.

The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writable register. By transferring data directly into this register, a short program jump can be executed directly, however, as only this low byte is available for manipulation, the jumps are limited to the present page of memory, that is 256 locations. When such program jumps are executed it should also be noted that a dummy cycle will be inserted.

For the HT48RU80/HT48CU80 devices, whose Program Memory is stored in two Banks, note that the bank selection is under the control of bit 5 of the Bank Pointer. It is this Bank Pointer bit that controls the highest address of the Program Counter as shown in the following diagram:



Note The lower byte of the Program Counter is fully accessible under program control. The use of the PCL might cause program branching, so an extra cycle is needed to pre-fetch. Further information on the PCL register can be found in the Special Function Register section.

Except HT48RU80/HT48CU80

| Mode | Program Counter Bits | | | | | | | | | | | | |
|--------------------------------|----------------------|------|------|-----|-----|----|----|----|----|----|----|----|----|
| | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| Initial Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| External Interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Timer/Event Counter 0 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Timer/Event Counter 1 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Skip | Program Counter + 2 | | | | | | | | | | | | |
| Loading PCL | PC12 | PC11 | PC10 | PC9 | PC8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, Call Branch | #12 | #11 | #10 | #9 | #8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return from Subroutine | S12 | S11 | S10 | S9 | S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

HT48RU80/HT48CU80

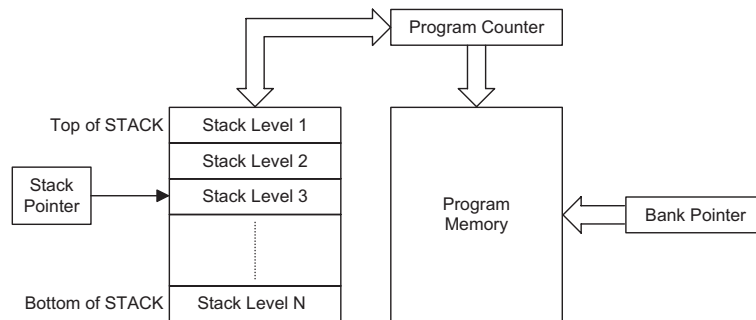
| Mode | Program Counter Bits | | | | | | | | | | | | | |
|--------------------------------|----------------------|------|------|------|-----|-----|----|----|----|----|----|----|----|----|
| | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| Initial Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| External Interrupt 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Timer/Event Counter 0 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Timer/Event Counter 1 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Timer/Event Counter 2 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| External Interrupt 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| UART Bus Interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Skip | Program Counter + 2 | | | | | | | | | | | | | |
| Loading PCL | PC13 | PC12 | PC11 | PC10 | PC9 | PC8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, Call Branch | BP.5 | #12 | #11 | #10 | #9 | #8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return from Subroutine | S13 | S12 | S11 | S10 | S9 | S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

- Note**
1. PC13~PC8: Current Program Counter bits.
 2. @7~@0: PCL bits.
 3. BP.5: Bank Pointer bit.
 4. #12~#0: Instruction code bits.
 5. S13~S0: Stack register bits.
 6. For the HT48RU80/HT48CU80, the Program Counter is 14 bits wide, i.e. from b13~b0.
 7. For the HT48R70A-1/HT48C70-1, since their Program Counter is 13 bits wide, the b13 column in the table is not applicable.
 8. For the HT48R50A-1/HT48C50-1, since their Program Counter is 12 bits wide, the b12 and b13 columns in the table are not applicable.
 9. For the HT48R30A-1/HT48C30-1, since their Program Counter is 11 bits wide, the b11, b12 and b13 columns in the table are not applicable.
 10. For the HT48R10A-1/HT48C10-1, since their Program Counter is 10 bits wide, the b10, b11, b12 and b13 columns in the table are not applicable.
 11. The Timer/Event Counter 2 Overflow row is available only for the HT48RU80/HT48CU80.
 12. The Timer/Event Counter 1 Overflow row is available only for the HT48R50A-1/HT48C50-1, HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80.
 13. For the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1, the Timer/Event Counter 0 represents the single timer.
 14. The UART Bus interrupt is available only for the HT48RU80/HT48CU80 devices.

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack can have between 4, 6 or 16 levels depending upon which device is selected and is neither part of the data nor part of the program space, and is neither readable nor writable. The activated level is indexed by the Stack Pointer (SP) and is neither readable nor writable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction (RET or RETI), the Program Counter is restored to its previous value from the stack. After a chip reset, the Stack Pointer will point to the top of the stack.

If the stack is full and an enabled interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal will be inhibited. When the Stack Pointer is decremented (by RET or RETI), the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. However, when the stack is full, a CALL subroutine instruc-



tion can still be executed which will result in a stack overflow. Precautions should be taken to avoid such cases which might cause unpredictable program branching.

-
- Note**
1. For the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1, N=4, i.e. 4 levels of stack available.
 2. For the HT48R50A-1/HT48C50-1, N=6, i.e. 6 levels of stack available.
 3. For the HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80, N=16, i.e. 16 levels of stack available.
-

Arithmetic and Logic Unit – ALU

The arithmetic-logic unit or ALU is a critical area of the microcontroller that carries out arithmetic and logic operations of the instruction set. Connected to the main microcontroller data bus, the ALU receives related instruction codes and performs the required arithmetic or logical operations after which the result will be placed in the specified register. As these ALU calculation or operations may result in carry, borrow or other status changes, the status register will be correspondingly updated to reflect these changes. The ALU supports the following functions:

- Arithmetic operations ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA
- Logic operations AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA
- Rotation RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC
- Increment and Decrement INCA, INC, DECA, DEC
- Branch decision JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET, RETI

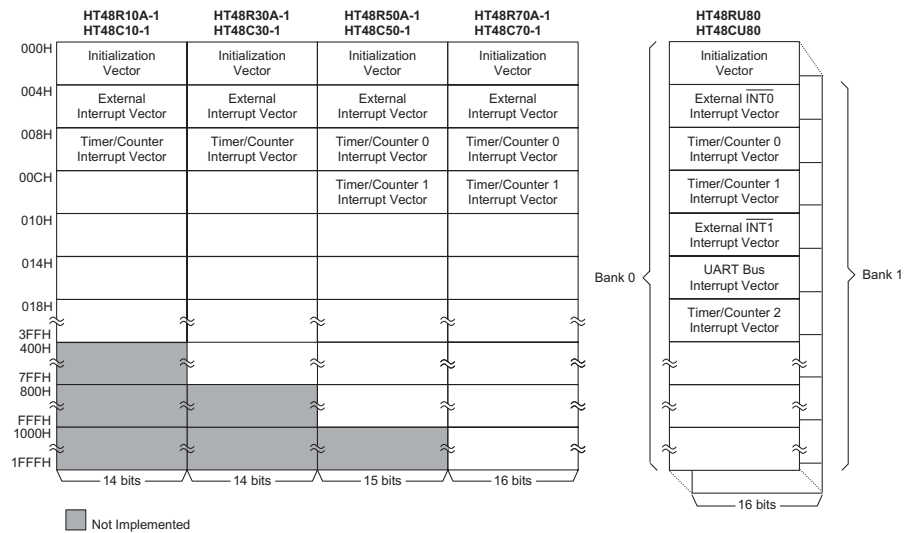
Program Memory

The Program Memory is the location where the user code or program is stored. For microcontrollers, two types of Program Memory are usually supplied. The first type is the One-Time Programmable (OTP) Memory where users can program their application code into the device. Devices with OTP memory are denoted by having an "R" within their device name. By using the appropriate programming tools, OTP devices offer users the flexibility to freely develop their applications which may be useful during debug or for products requiring frequent upgrades or program changes. OTP devices are also applicable for use in applications that require low or medium volume production runs. The other type of memory is the mask ROM memory, denoted by having a "C" within the device name. These devices offer the most cost effective solutions for high volume products.

Organization

The Program Memory has a capacity of 1K by 14 to 16K by 16 bits depending upon which device is selected. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by a separate table pointer register. In the case of the HT48RU80/HT48CU80 devices, the Program Memory is divided into two Banks, Bank 0 and Bank 1, each with a capacity of 8K. The Program Memory Bank is selected by controlling Bit 5 of the Bank Pointer Register. Clearing this bit selects Bank 0 while setting the bit selects Bank 1. Care must be exercised when using the Bank Pointer Register as it is also used to control the Data Memory Bank Pointer.

The following diagram shows the Program Memory for the I/O Type MCU series.



Special Vectors

Within the Program Memory, certain locations are reserved for special usage such as reset and interrupts.

- Location 000H**
 This vector is reserved for use by the chip reset for program initialization. After a chip reset is initiated, the program will jump to this location and begin execution.
- Location 004H**
 This vector is used by the external interrupt. If the external interrupt pin on the device receives a high to low transition, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full. For the HT48RU80/HT48CU80, the external interrupt is known as $\overline{\text{INT0}}$.
- Location 008H**
 This internal vector is used by the Timer/Event Counter. If a counter overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full. For the HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1 devices, which have dual timers, and the HT48RU80/HT48CU80, which have triple timers, this timer is known as Timer/Event Counter 0.
- Location 00CH**
 This internal vector is used by the Timer/Event Counter. If a counter overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full. This vector is available for the HT48R50A-1/HT48C50-1, HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80 only. The Timer/Event Counter is known as Timer/Event Counter 1. Note that the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices have only one timer, therefore, this interrupt vector is not used.

- Location 010H
This vector is used by the external interrupt $\overline{\text{INT1}}$. If the external interrupt pin on the device goes low, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full. This vector is available for the HT48RU80/HT48CU80 only. For the other devices, this interrupt is not used.
- Location 014H
For the HT48RU80/HT48CU80 devices, this internal vector is reserved for the UART Bus interrupt service routine. When a UART Bus interrupt resulting from a transmission flag or reception is completed, the program will jump to this location and begin execution if the interrupt is enabled and the stack is not full.
- Location 018H
For the HT48RU80/HT48CU80 devices, this internal vector is reserved for the Timer/Event Counter 2 interrupt service routine. If a timer interrupt results from a Timer/Event Counter 2 overflow, the program will jump to this location and begin execution if the interrupt is enabled and the stack is not full.

Managing Multiple Banks

For the HT48RU80/HT48CU80 devices, which have multiple Program Memory banks, there are some special considerations that have to be taken into account. First, the sections of program which are to be located into different banks are placed using the ROMBANK directive. When using the "CALL" instruction to call routines located in a different bank, or when using the "JMP" instructions to directly jump to a location in a different bank, the target bank must be first selected by correctly setting up the Bank Pointer prior to executing the "CALL" or "JMP" instruction. This of course can be achieved by directly controlling Bit 5 of the Bank Pointer, BP, but can also be done by using the BANK directive as shown in the example. Then, when a "CALL" or "JMP" instruction is executed, the Bank Pointer value stored in the BP register will be automatically loaded into the Program Counter. When the "RET" instruction is encountered in a subroutine called from a different bank, the program will automatically return to the original bank, however, the BP value will not be changed and will remain at the value where the subroutine is located. For this reason the BP must be carefully managed when moving between banks. The following example for the HT48RU80/HT48CU80 devices illustrates how to use the "CALL" and "JMP" instructions between different banks:

```
include HT48RU80.inc
:
rombank 0 codesec0           ; define rombank 0
rombank 1 codesec1           ; define rombank 1
:
:
codesec0 .section at 000h 'code' ; locates the following program section
:                               ; into Bank 0
    clr bp                       ; re-initializing the BP
    jmp start
:
:
start:
:
:
lab0:
:
:
```

```

mov a, BANK routbl          ; routine "routbl" is located in Bank 1
mov bp,a                   ; load bank number for routbl into BP
call routbl                ; call subroutine located in Bank 1
clr bp.5                   ; program will return to this location
                           ; after RET in Bank 1
                           ; but BP will retain Bank 1 value
                           ; so clear the BP
:
:
codesecl .section at 000h 'code' ; locates following program section
                           ; into Bank 1
:
:
routbl proc
:
:
ret                        ; return program to Bank 0 but BP will
                           ; retain Bank 1 value
routbl endp
:
:

```

When managing interrupts, care has to be exercised in supervising the Bank Pointer. Irrespective of what Bank the program is presently running in, when an interrupt occurs, whether it be an external interrupt or internal interrupt, the program will immediately jump to its respective interrupt vector located in Bank 0. Note however that, although in all cases the program will jump to Bank 0, the Bank Pointer will retain its original value and not indicate Bank 0. For this reason, after entering the interrupt subroutine, in addition to the usual backup of the accumulator and status register, it is important to backup its original value immediately and also clear the Bank Pointer to indicate Bank 0 especially if other calls or jumps are encountered within Bank 0. Before the "RET!" instruction in the interrupt subroutine is executed, the Bank Pointer, along with the accumulator and status register, must be restored to ensure the program returns to the same status from where the subroutine was called. The following example illustrates how interrupts can be managed:

```

include HT48RU80.inc
:
:
rombank 0 codesecl         ; define rombank 0
rombank 1 codesecl         ; define rombank 1
:
:
codesecl .section at 000h 'code' ; locates the following program section
                           ; into Bank 0
clr bp                    ; clear bank pointer after power-on reset
:
:
org 004h                  ; jump here from any bank when ext. int.
                           ; occurs - BP retains original value

mov accbuf0,a             ; backup accumulator
mov a,bp                 ; backup bank pointer
clr bp                   ; clear bp to indicate Bank 0 otherwise
                           ; original BP value will remain and give
                           ; rise to false jmp or call addresses
jmp ext_int              ; jump to external interrupt subroutine
:
:
org 008h                  ; jump here from any bank when timer 0 int.
                           ; occurs - BP retains original value
mov accbuf1,a            ; backup accumulator
mov a,bp                 ; backup bank pointer
clr bp                   ; clear bp to indicate Bank 0 otherwise
                           ; original BP value will remain and give
                           ; rise to false jmp or call addresses
jmp tim0_int            ; jump to timer 0 interrupt subroutine
:
:

```

```

    org 00Ch                ; jump here from any bank when timer 1 int.
                            ; occurs - BP retains original value
    :
    :
ext_int:                   ; external interrupt subroutine
    mov bp_exti,a          ; backup bank pointer
    mov a,status           ; backup status register
    mov statusbuf0,a      ; backup status register
    :
    :
    mov a,statusbuf0      ; restore status register
    mov status,a          ; restore status register
    mov a,bp_exti         ; restore bank pointer
    mov bp,a              ; restore bank pointer
    mov a,accbuf0        ; restore accumulator
                            ; restore accumulator
    reti                  ; return to main program and original
                            ; calling bank
    :
    :
tim0_int:                  ; timer 0 interrupt subroutine
    mov bp_tmr0,a         ; backup bank pointer
    mov a,status         ; backup status register
    mov statusbuf1,a     ; backup status register
    :
    :
    mov a,statusbuf1     ; restore status register
    mov status,a         ; restore status register
    mov a,bp_tmr0        ; restore bank pointer
    mov bp,a             ; restore bank pointer
    mov a,accbuf1        ; restore accumulator
                            ; restore accumulator
    reti                  ; return to main program and original
                            ; calling bank
    :

```

Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, table pointers are used to setup the address of the data that is to be accessed from the Program Memory. However, as some devices possess only a low byte table pointer and other devices possess both a high and low byte pointer it should be noted that depending upon which device is used, accessing look-up table data is implemented in slightly different ways.

With the exception of the HT48RU80/HT48CU80 devices, there is only one Table Pointer Register known as TBLP, in which the lower byte address of the look-up data to be retrieved must be first written. This register defines the lower 8-bit address of the look-up table. For these devices, after setting up the table pointer, the table data can be retrieved from the current Program Memory page or last Program Memory page using the "TABRDC [m]" or "TABRDL [m]" instructions respectively. When these instructions are executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register. Any unused bits in this transferred higher order byte will be read as "0".

With the exception of the HT48RU80/HT48CU80 devices, the following diagram illustrates the addressing/data flow of the look-up table:

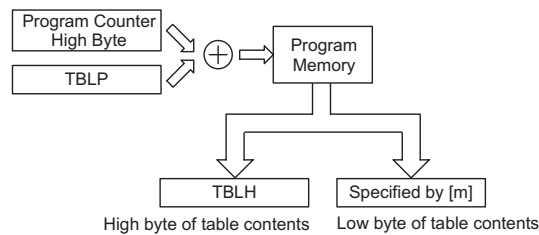


Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the HT48R10A-1/HT48C10-1 I/O microcontroller. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "300H" which refers to the start address of the last page within the 1K Program Memory of the HT48R10A-1/HT48C10-1 microcontroller. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "306H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address of the present page if the "TABRDC [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRDL [m]" instruction is executed.

```

tempreg1 db ? ; temporary register #1
tempreg2 db ? ; temporary register #2
:
:
mov a,06h ; initialize table pointer - note that this address is
; referenced
mov tblp,a ; to the last page or present page
:
:
tabrdl tempreg1 ; transfers value in table referenced by table pointer
; to tempreg1
; data at prog. memory address 306H transferred to
; tempreg1 and TBLH

dec tblp ; reduce value of table pointer by one

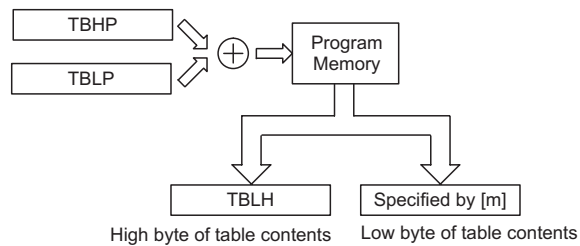
tabrdl tempreg2 ; transfers value in table referenced by table pointer
; to tempreg2
; data at prog. memory address 305H transferred to
; tempreg2 and TBLH
; in this example the data "1AH" is transferred to
; tempreg1 and data "0FH" to register tempreg2
; the value "00H" will be transferred to the high byte
; register TBLH
:
:
org 300h ; sets initial address of last page (for HT48R10A-1)

dc 00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
:
:

```

For the HT48RU80/HT48CU80 devices, there are two Table Pointer Registers known as TBLP and TBHP in which the lower order and higher order address of the look-up data to be retrieved must be respectively first written. Unlike the other devices in which only the low address byte is defined using the TBLP register, the additional TBHP register allows the complete address of the look-up table to be defined and consequently allow table data from any address and any page to be directly accessed. For these devices, after setting up both the low and high byte table pointers, the table data can then be retrieved from any area of Program Memory using the "TABRDC [m]" instruction or from the last page of the current Program Memory Bank using the "TABRDL [m]" instruction. When either of these instructions are executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register.

The following diagram illustrates the addressing/data flow of the look-up table for the HT48RU80/HT48CU80 devices:



The following example shows how the table pointer is defined and table data retrieved from the HT48RU80/HT48CU80 devices. This example uses raw table data which is located and stored in the Program Memory using the ORG statement. The value at this ORG statement is "000H", however, this only indicates the offset value from the start address of Bank 1 which in this case is "2000H". The table pointer high byte is setup to have a value of "20H" while the value of the table pointer low byte is setup here to have an initial value of "05H". This will ensure that the data byte read from the data table will be located at the Program Memory address "2005H" or 5 locations after the first address defined by the ORG statement. When the "TABRDC [m]" instruction is executed, the table data low byte which has a value of "FFH", will be transferred to the user defined temp register, while the table data high byte, which has a value of "55H", will be transferred to the TBLH register.

```

include HT48RU80.inc
:
:
data .section 'data'
temp db ?
:
:
rombank 0 codesec0 ; Bank 0 definition
rombank 1 codesec1 ; Bank 1 definition
:
:
codesec0 .section at 0 'code'
jmp start
:

```

```

    org 010h
start:
    :
    :
    mov     a,020h           ; setup table high byte address
    mov     tbhp,a
    mov     a,005h         ; setup table low byte address
    mov     tblp,a         ; table pointer address is now 2005H
    tabrdc temp           ; read table data from PC address 2005H
    nop                ; "FFH" will be placed in temp register
                    ; and "55H" will be placed in TBLH
                    ; register

codesec1 .section at 000h 'code' ; Bank 1 code located here
    org 0000h           ; this defines the offset from the
                    ; start address of Bank 1 which is
                    ; 2000H

dc 000AAh, 011BBh, 022CCh, 033DDh, 044EEh, 055FFh
    :
    :

```

Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use the table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of the TBLH and subsequently cause errors if used again by the main routine. As a rule it is recommended that simultaneous use of the table read instructions should be avoided. However, in situations where simultaneous use cannot be avoided, the interrupts should be disabled prior to the execution of any main routine table-read instructions. Note that all table related instructions require two instruction cycles to complete their operation.

Except HT48RU80/HT48CU80

| Instruction | Table Location Bits | | | | | | | | | | | | |
|-------------|---------------------|------|------|-----|-----|----|----|----|----|----|----|----|----|
| | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| TABRDC [m] | PC12 | PC11 | PC10 | PC9 | PC8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL [m] | 1 | 1 | 1 | 1 | 1 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

HT48RU80/HT48CU80

| Instruction | Table Location Bits | | | | | | | | | | | | | |
|-------------|---------------------|------|------|------|------|------|----|----|----|----|----|----|----|----|
| | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| TABRDC [m] | TBHP | TBHP | TBHP | TBHP | TBHP | TBHP | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL [m] | 1 | 1 | 1 | 1 | 1 | 1 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

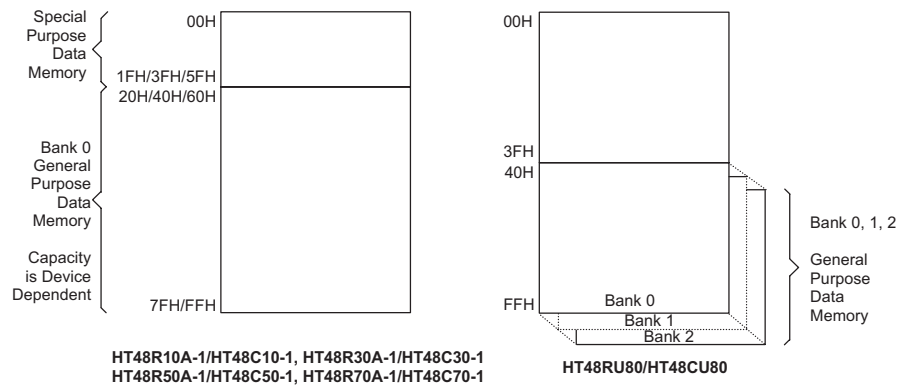
- Note**
1. PC12~PC8: Current Program Counter bits.
 2. @7~@0: Table Pointer TBLP bits.
 3. For the HT48RU80/HT48CU80, the Table address location is 14 bits, i.e. from b13~b0.
 4. For the HT48R70A-1/HT48C70-1, the Table address location is 13 bits, i.e. from b12~b0.
 5. For the HT48R50A-1/HT48C50-1, the Table address location is 12 bits, i.e. from b11~b0.
 6. For the HT48R30A-1/HT48C30-1, the Table address location is 11 bits, i.e. from b10~b0.
 7. For the HT48R10A-1/HT48C10-1, the Table address location is 10 bits, i.e. from b9~b0.

Data Memory

The Data Memory is a volatile area of 8-bit wide RAM internal memory and is the location where temporary information is stored. Divided into two sections, the first of these is an area of RAM where special function registers are located. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is reserved for general purpose use. All locations within this area are read and write accessible under program control. For the HT48RU80/HT48CU80 devices, this General Purpose Data Memory is divided into three banks, Bank 0, Bank 1 and Bank 2.

Organization

The two sections of Data Memory, the Special Purpose and General Purpose Data Memory are located at consecutive locations. All are implemented in RAM and are 8 bits wide but the length of each memory section is dictated by the type of microcontroller chosen. The start address of the Data Memory for all devices is the address 00H. The last Data Memory address is 7FH for the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices, and FFH for the HT48R50A-1/HT48C50-1, HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80 devices. Registers which are common to all microcontrollers, such as ACC, PCL, etc., have the same Data Memory address.



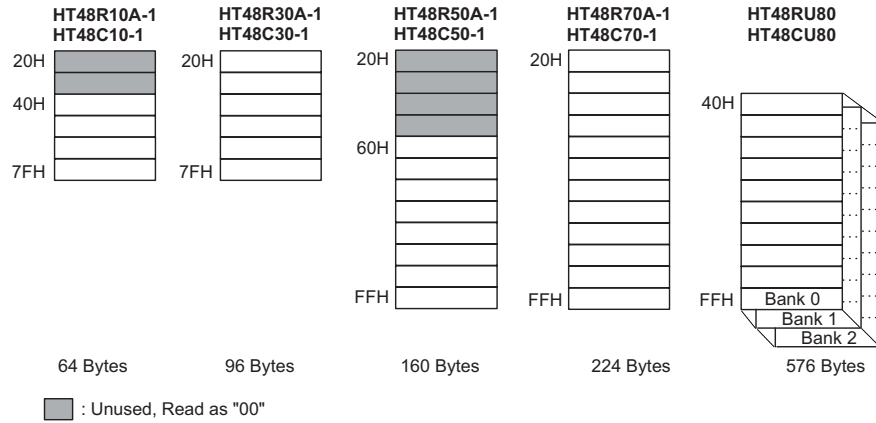
Note Most of the Data Memory bits can be directly manipulated using the "SET [m].i" and "CLR [m].i" with the exception of a few dedicated bits. The Data Memory can also be accessed through the Memory Pointer register MP for the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1. For the other devices there are two Memory Pointer registers MP0 and MP1.

General Purpose Data Memory

All microcontroller programs require an area of read/write memory where temporary data can be stored and retrieved for use later. It is this area of RAM memory that is known as General Purpose Data Memory. This area of Data Memory is fully accessible by the user program for both read and write operations. By using the "SET [m].i" and "CLR [m].i" instructions individual bits can be set or reset under program control giving the user a large range of flexibility for bit manipulation in the Data Memory. As the General Purpose Data Memory exists in three separate Banks for the

HT48RU80/HT48CU80 devices, known as Bank 0, Bank 1 and Bank 2, it is necessary to first ensure that the Bank Pointer is properly set to the correct value before accessing the General Purpose Data Memory. Bank 1 or Bank 2 must be addressed indirectly using the Memory Pointer MP1 and the indirect addressing register IAR1. Any direct addressing or any indirect addressing using MP0 and IAR0 will always result in data from Bank 0 being accessed.

The following diagram shows the General Purpose Data Memory Organization Map for the I/O Type microcontrollers:



Note The 576 bytes of the General Purpose Data Memory in the HT48RU80/HT48CU80 devices are stored in three individual memory banks, known as Bank 0, Bank 1 and Bank 2. Before reading or writing to the General Purpose Data Memory it is essential to first ensure that the correct Data Memory bank is selected by setting up the Bank Pointer.

Special Purpose Data Memory

This area of Data Memory is where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers are both readable and writable but some are protected and are readable only, the details of which are located under the relevant Special Function Register section. Note that for locations that are unused, any read instruction to these addresses will return the value "00H".

The following diagram shows a detailed Special Purpose Data Memory Organization Map of the I/O Type microcontrollers:

| | HT48R10A-1 HT48C10-1 | HT48R30A-1 HT48C30-1 | HT48R50A-1 HT48C50-1 | HT48R70A-1 HT48C70-1 | HT48RU80 HT48CU80 |
|-----|-------------------------|-------------------------|-------------------------|-------------------------|----------------------|
| 00H | IAR | IAR | IAR0 | IAR0 | IAR0 |
| 01H | MP | MP | MP0 | MP0 | MP0 |
| 02H | | | IAR1 | IAR1 | IAR1 |
| 03H | | | MP1 | MP1 | MP1 |
| 04H | | | | | BP |
| 05H | ACC | ACC | ACC | ACC | ACC |
| 06H | PCL | PCL | PCL | PCL | PCL |
| 07H | TBLP | TBLP | TBLP | TBLP | TBLP |
| 08H | TBLH | TBLH | TBLH | TBLH | TBLH |
| 09H | WDTS | WDTS | WDTS | WDTS | WDTS |
| 0AH | STATUS | STATUS | STATUS | STATUS | STATUS |
| 0BH | INTC | INTC | INTC | INTC | INTC0 |
| 0CH | | | | TMR0H | TMR0H |
| 0DH | TMR | TMR | TMR0 | TMR0L | TMR0L |
| 0EH | TMRC | TMRC | TMR0C | TMR0C | TMR0C |
| 0FH | | | TMR1H | TMR1H | TMR1H |
| 10H | | | TMR1L | TMR1L | TMR1L |
| 11H | | | TMR1C | TMR1C | TMR1C |
| 12H | PA | PA | PA | PA | PA |
| 13H | PAC | PAC | PAC | PAC | PAC |
| 14H | PB | PB | PB | PB | PB |
| 15H | PBC | PBC | PBC | PBC | PBC |
| 16H | PC | PC | PC | PC | PC |
| 17H | PCC | PCC | PCC | PCC | PCC |
| 18H | | | PD | PD | PD |
| 19H | | | PDC | PDC | PDC |
| 1AH | | | | PE | PE |
| 1BH | | | | PEC | PEC |
| 1CH | | | | PF | PF |
| 1DH | | | | PFC | PFC |
| 1EH | | PG | PG | PG | INTC1 |
| 1FH | | PGC | PGC | PGC | TBHP |
| 20H | | | | | |
| 21H | | | | | TMR2 |
| 22H | | | | | TMR2C |
| 23H | | | | | |
| 24H | | | | | |
| 25H | | | | | PG |
| 26H | | | | | PGC |
| 27H | | | | | |
| 28H | | | | | USR |
| 29H | | | | | UCR1 |
| 2AH | | | | | UCR2 |
| 2BH | | | | | TXR/RXR |
| 2CH | | | | | BRG |
| 2DH | | | | | |
| 3FH | | | | | |

■ : Unused, Read as "00"

Special Function Registers

To ensure successful operation of the microcontroller, certain internal registers are implemented in the Data Memory area. These registers ensure correct operation of internal functions such as timers, interrupts, watchdog, etc., as well as external functions such as I/O data control. The location of these registers within the Data Memory begins at the address "00H". Any unused Data Memory locations between these special function registers and the point where the General Purpose Memory begins is reserved for future expansion purposes, attempting to read data from these locations will return a value of "00H".

Indirect Addressing Registers – IAR, IAR0, IAR1

The method of indirect addressing allows data manipulation using memory pointers instead of the usual direct memory addressing method where the actual memory address is defined. Any action on the Indirect Addressing Registers will result in corresponding read/write operations to the memory location specified by the corresponding memory pointer. For the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices, one Indirect Addressing Register, IAR, and one Memory Pointer, MP, is provided. For the HT48R50A-1/HT48C50-1, HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80 devices, two Indirect Addressing Registers, IAR0 and IAR1, and two Memory Pointers, MP0 and MP1, are provided. Note that these Indirect Addressing Registers are not physically implemented and that reading the Indirect Addressing Registers directly will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointers – MP, MP0, MP1

For the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices, one memory pointer known as MP is provided, whereas for the HT48R50A-1/HT48C50-1, HT48R70A-1/ HT48C70-1 and HT48RU80/ HT48CU80 devices, two memory pointers known as MP0 and MP1 are provided. These memory pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to is the address specified by the related Memory Pointer.

Note For the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices, bit 7 of the memory pointers are not implemented. However, it must be noted that when the memory pointers in these devices are read, a value of "1" will be read.

The following example for the HT48R10A-1/HT48C10-1 or HT48R30A-1/HT48C30-1 devices shows how to clear a section of four RAM locations already defined as locations adres1 to adres4.

```
data .section 'data'
adres1 db ?
adres2 db ?
adres3 db ?
adres4 db ?
block db ?
code .section at 0 'code'
org 00h
```

```

start:
  mov a,04h           ; setup size of block
  mov block,a
  mov a,offset adre1 ; Accumulator loaded with first RAM address
  mov mp,a           ; setup memory pointer with first RAM address

loop:
  clr IAR            ; clear the data at address defined by MP
  inc mp            ; increment memory pointer
  sdz block         ; check if last memory location has been cleared
  jmp loop

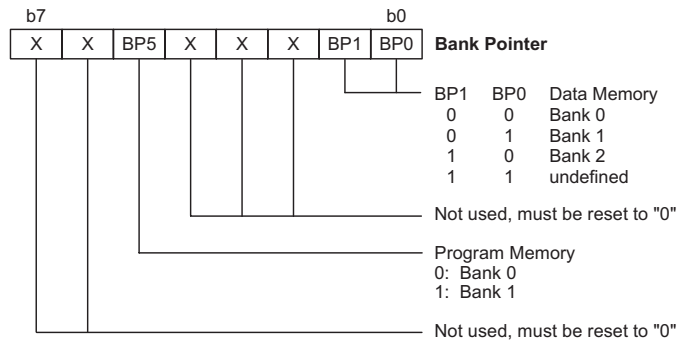
continue:

```

The important point to note here is that in the example shown above, no reference is made to specific RAM addresses.

Bank Pointer – BP

As both the Data Memory and Program Memory are divided into several banks for the HT48RU80/HT48CU80 devices, the Bank Pointer only exists in these devices. As the Data Memory for these devices is subdivided into three banks, selecting the correct Data Memory area is achieved by using the Bank Pointer. If data in either Bank 1 or Bank 2 is to be accessed, the lowest two bits of the BP must be set to the binary values 01 or 10 respectively, however, it must be noted that data in these two banks can only be addressed indirectly using the MP1 memory pointer and the IAR1 indirect addressing register. Any direct addressing or any indirect addressing using MP0 and IAR0 will always result in data from Bank 0 being accessed. The Data Memory is initialized to Bank 0 after a reset, except for the WDT time-out reset in the Power Down Mode, in which case, the Data Memory Bank remains unchanged. It should be noted that the Special Function Data Memory is not affected by the bank selection, which means that the Special Function Registers can be accessed from within either Bank 0, Bank 1 or Bank 2. For these devices, whose 16K of Program Memory is divided into two 8K banks, known as Bank 0 and Bank 1, bit 5 of the Bank Pointer is used to control which Program Memory Bank is selected. Although only some of the Bank Pointer register bits are actually used for Data Memory and Program Memory Bank indicating purposes, note that all 8 bits of the BP register are actually implemented. Any unused bits must be reset to "0".



Accumulator – ACC

The Accumulator is central to the operation of any microcontroller and is closely related with operations carried out by the ALU. The Accumulator is the place where all intermediate results from the ALU are stored. Without the Accumulator it would be necessary to write the result of each calculation or logical operation, such as addition, subtraction, shift, etc., to the Data Memory resulting in higher programming and timing overheads. Data transfer operations usually involve the temporary storage function of the Accumulator; for example, when transferring data between one user defined register and another, it is necessary to do this by passing the data through the Accumulator as no direct transfer between two registers is permitted.

Program Counter Low Register – PCL

To provide additional program control functions, the low byte of the Program Counter is made accessible to programmers by locating it within the Special Purpose area of the Data Memory. By manipulating this register, direct jumps to other program locations are easily implemented. Loading a value directly into this PCL register will cause a jump to the specified Program Memory location, however as the register is only 8-bit wide, only jumps within the current Program Memory page are permitted. When such operations are used, note that a dummy cycle will be inserted.

Look-up Table Registers – TBLP, TBHP, TBLH

These three special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP is the table low byte pointer and indicates the location where the table data is located. Its value must be setup before any table read commands are executed. Its value can be changed, for example using the "INC" or "DEC" instructions, allowing for easy table data pointing and reading. TBHP is the table high byte pointer but is only available in the HT48RU80/HT48CU80 devices. TBLH is the location where the high order byte of the table data is stored after a table read data instruction has been executed. Note that the lower order table data byte is transferred to a user defined location.

Watchdog Timer Register – WDTS

The Watchdog feature of the microcontroller provides an automatic reset function giving the microcontroller a means of protection against spurious jumps to incorrect Program Memory addresses. To implement this, a timer is provided within the microcontroller which will issue a reset command when its value overflows. To provide variable Watchdog Timer reset times, the Watchdog Timer clock source can be divided by various division ratios, the value of which is set using the WDTS register. By writing directly to this register, the appropriate division ratio for the Watchdog Timer clock source can be setup. Note that only the lower 3 bits are used to set division ratios between 1 and 128, the remaining 5 bits of the 8-bit register can be used by programmers for other purposes.

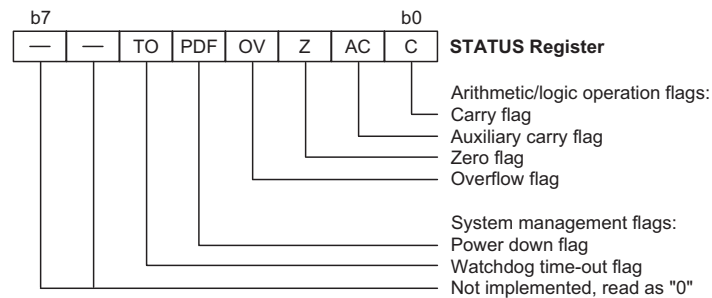
Status Register – STATUS

This 8-bit register (0AH) contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), power down flag (PDF), and watchdog time-out flag (TO). It also records the status information and controls the operation sequence.

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most other registers. Any data written into the status register will not change the TO or PDF flag. In addition, operations related to the status register may give different results due to the different instruction operations. The TO flag can be affected only by a system power-up, a WDT time-out or by executing the "CLR WDT" or "HALT" instruction. The PDF flag is affected only by executing the "HALT" or "CLR WDT" instruction or during a system power-up.

The Z, OV, AC and C flags generally reflect the status of the latest operations.

- **C** is set if an operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction.
- **AC** is set if an operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction; otherwise AC is cleared.
- **Z** is set if the result of an arithmetic or logical operation is zero; otherwise Z is cleared.
- **OV** is set if an operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared.
- **PDF** is cleared by a system power-up or executing the "CLR WDT" instruction. PDF is set by executing the "HALT" instruction.
- **TO** is cleared by a system power-up or executing the "CLR WDT" or "HALT" instruction. TO is set by a WDT time-out.



In addition, on entering an interrupt sequence or executing a subroutine call, the status register will not be pushed onto the stack automatically. If the contents of the status registers are important and if the subroutine can corrupt the status register, precautions must be taken to correctly save it.

Interrupt Control Registers – INTC, INTC0, INTC1

These 8-bit registers control the operation of both the external and internal interrupts. For the HT48RU80/HT48CU80 devices there are two interrupt control registers, known as INTC0 and INTC1, while for the other devices there is only one interrupt control register, known as INTC. By setting various bits within these registers using standard bit manipulation instructions, the enable/disable function of the external interrupts and each of the internal interrupts can be independently controlled. A master interrupt bit within these registers, the EMI bit, acts like a global enable/disable and is used to set all of the interrupt enable bits on or off. This bit is cleared when an interrupt routine is entered to disable further interrupt and is set by executing the "RETI" instruction.

Note In situations where other interrupts may require servicing within present interrupt service routines, the EMI bit can be manually set by the program after the present interrupt service routine has been entered.

Timer/Event Counter Registers

Depending upon which device is selected, all devices contain one, two or three integrated Timer/Event Counters of either 8-bit or 16-bit size. For the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices, which have a single 8-bit Timer/Event Counter, an associated register known as TMR is the location where the timer's 8-bit value is located. An associated control register, known as TMRC, contains the setup information for this timer. The HT48R50A-1/HT48C50-1 devices contain a single 8-bit Timer/Event Counter with an associated register known as TMR0, and a single 16-bit Timer/Event Counter with an associated register pair known as TMR1L/TMR1H, where the timer's values are located. Two associated control registers, known as TMR0C and TMR1C contain the setup information for these two timers. The HT48R70A-1/HT48C70-1 devices contain two 16-bit Timer/Event Counters with two associated register pairs, known as TMR0L/TMR0H and TMR1L/TMR1H, where the timer's 16-bit values are located. Two associated control registers, known as TMR0C and TMR1C contain the setup information for these two timers. Like the HT48R70A-1/HT48C70-1 devices, the HT48RU80/HT48CU80 devices also contain two 16-bit Timer/Event Counters with the same register names. However, they also contain an additional 8-bit Timer/Event Counter with an associated timer register known as TMR2 and timer control register known as TMR2C. Note that all timer registers can be directly written to in order to preload their contents with fixed data to allow different time intervals to be setup.

Input/Output Ports and Control Registers

Within the area of Special Function Registers, the I/O registers and their associated control registers play a prominent role. All I/O ports have a designated register correspondingly labeled as PA, PB, PC, etc. These labeled I/O registers are mapped to specific addresses within the Data Memory as shown in the Data Memory table which are used to transfer the appropriate output or input data on that port. With each I/O port there is an associated control register labeled PAC, PBC, PCC, etc., also mapped to specific addresses with the Data Memory. The control register specifies which pins of that port are set as inputs and which are set as outputs. To setup a pin as an input, the corresponding bit of the control register must be set high, for an output it must be set low. During program initialization, it is important to first setup the control registers to specify which pins are outputs and which are inputs before reading data from or writing data to the I/O ports. One flexible

feature of these registers is the ability to directly program single bits using the "SET [m].i" and "CLR [m].i" instructions. The ability to change I/O pins from output to input and vice-versa by manipulating specific bits of the I/O control registers during normal program operation is a useful feature of these devices.

UART Registers – USR, UCR1, UCR2, TXR/RXR, BRG

The HT48RU80/HT48CU80 devices each contain an internal UART function which is controlled via these five registers. The USR is the status register for the UART while UCR1 and UCR2 are the two control registers. The actual data that is to be transmitted or that is received on the serial interface is stored in the TXR/RXR register while the Baud Rate for the UART is setup via the BRG register.

Input/Output Ports

Holtek microcontrollers offer considerable flexibility on their I/O ports. With the input or output designation of every pin fully under user program control, pull-high options for all pins and wake-up options on certain pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

Depending upon which device or package is chosen, the microcontroller range provides from 21 to 56 bidirectional input/output lines labeled with port names PA, PB, PC, etc. These I/O ports are mapped to the Data Memory with specific addresses as shown in the Special Purpose Data Memory table. All of these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction "MOV A,[m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Pull-high Resistors

Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, all I/O pins, when configured as an input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selectable via configuration options and are implemented using a weak PMOS transistor. Note that if the pull-high option is selected, then all I/O pins on that port will be connected to pull-high resistors, individual pins cannot be selected for pull-high resistor options.

Port A Wake-up

Each device has a HALT feature enabling the microcontroller to enter a Power Down Mode and preserve power, a feature that is important for battery and other low power applications. Various methods exist to wake-up the microcontroller, one of which is to change the logic condition on one of the Port A pins from high to low. After a "HALT" instruction forces the microcontroller into entering a HALT condition, the processor will remain idle or in a low-power state until the logic condition of the selected wake-up pin on Port A changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Note that each pin on Port A can be selected individually to have this wake-up feature.

I/O Port Control Registers

Each I/O line has its own control register (PAC, PBC, PCC, etc.) to control the input/output configuration. With this control register, each CMOS output or Schmitt Trigger input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each pin of the I/O ports is directly mapped to a bit in its associated port control register. For the I/O pin to function as an input, the corresponding bit of the control register must be written as a "1". This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a "0", the I/O pin will be setup as a CMOS output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

Pin-shared Functions

The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For some pins, the chosen function of the multi-function I/O pins is set by configuration options while for others the function is set by application program control.

Buzzer

The buzzer pins BZ and $\overline{\text{BZ}}$ are pin-shared with I/O pins PB0 and PB1. If configured as buzzer pins, the correct hardware and software options must be selected.

External Interrupt Input

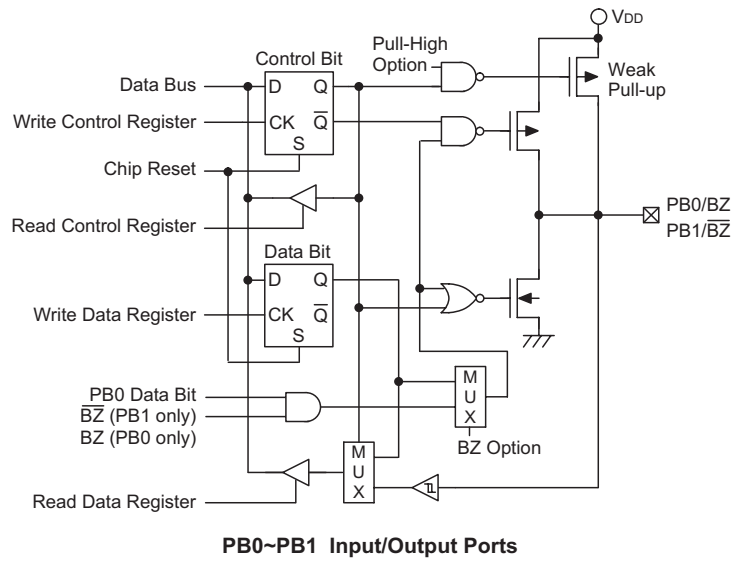
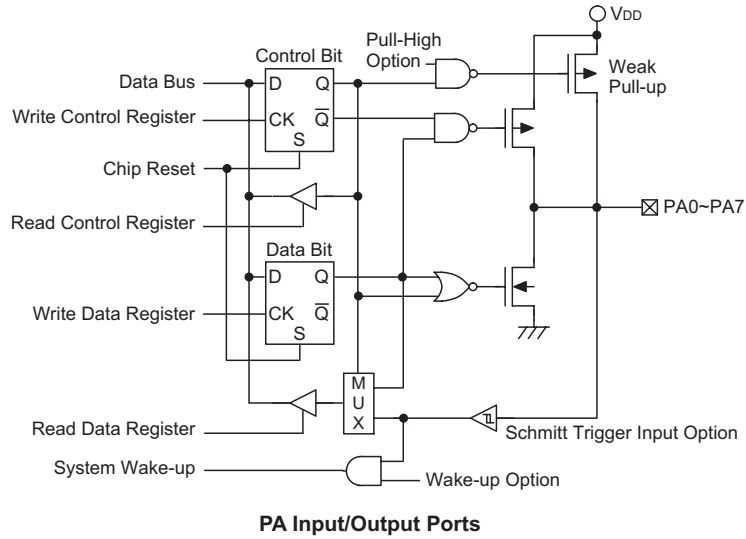
The external interrupt pin $\overline{\text{INT}}$ is pin-shared with the I/O pin PC0 or PG0 depending upon which device is used. However, for the HT48R70A-1/HT48C70-1 devices, the external interrupt pin INT is an independent non-shared pin. For the HT48RU80/HT48CU80 devices only, there are two external interrupt pins, the INT0 pin, which exists as an independent pin and the INT1 pin which is pin-shared with the I/O pin PB2. For these pins to operate as external interrupt pins and not as a normal I/O pin, the corresponding external interrupt enable bits in the INTC, INTC0 and INTC1 interrupt control registers must be correctly set. For applications not requiring an external interrupt input, the pin-shared external interrupt pins can be used as normal I/O pins, however to do this, the external interrupt enable bits in the INTC register must be disabled.

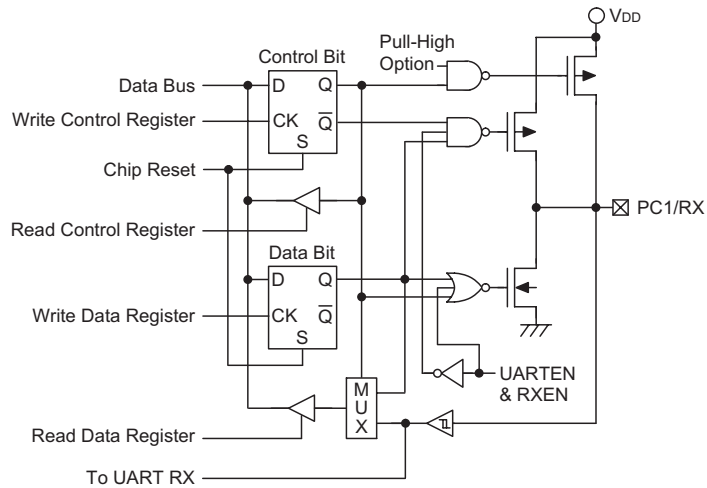
External Timer Clock Input

Each device contains either one or two timers depending upon which one is chosen. Each timer has an external input pin, which in the case of devices with a single timer, is known as TMR and in the case of devices with two timers are known as TMR0 and TMR1. For all devices with a single timer, the external input pin TMR is pin-shared with I/O pin PC0 or PC1. For devices with two timers, the external input pins TMR0 and TMR1 are pin-shared with pins PC0 and PC5 respectively or exist as independent non-shared pins depending upon which device and which package is selected. The HT48RU80/HT48CU80 devices contain three timers, all of which have external timer pins, TMR0, TMR1 and TMR2. The TMR0 and TMR1 pins are independent pins while the TMR2 pin is pin-shared with I/O pin PB3. These external timer pins, if they are shared pins, can be used as normal I/O pins for applications that do not require external timer inputs. For such applications the timer mode control bits in the timer control register must select the timer mode which has an internal clock source, to prevent the I/O from interfering with the timer counter operation.

Oscillator

The system oscillator pins OSC1 and OSC2 are pin-shared with PC3 and PC4 on the HT48R10A-1/HT48C10-1 and pins PG1 and PG2 on the HT48R30A-1/HT48C30-1 and HT48R50A-1/HT48C50-1. On the HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80, the oscillator pins are not pin-shared. The pin-shared functions are selected via configuration option. If chosen to function as I/O pins, then full pull-high options remain.

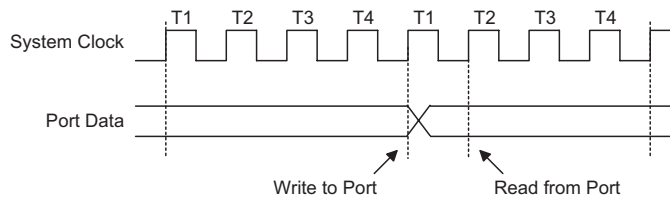




PC1/RX Input/Output Port – HT48RU80/HT48CU80

Programming Considerations

Within the user program, one of the first things to consider is port initialization. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high options have been selected. If the port control registers, PAC, PBC, PCC, etc., are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated port data registers, PA, PB, PC, etc., are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct values into the appropriate port control register or by programming individual bits in the port control register using the "SET [m].i" and "CLR [m].i" instructions. Note that when using these bit control instructions, a read-modify-write operation takes place. The microcontroller must first read in the data on the entire port, modify it to the required new bit values and then rewrite this data back to the output ports.



Port A has the additional capability of providing wake-up functions. When the chip is in the HALT state, various methods are available to wake the device up. One of these is a high to low transition of any of the Port A pins. Single or multiple pins on Port A can be setup to have this function.

Timer/Event Counters

The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. The devices in the I/O Type MCU series contain either one, two or three count up timers of either 8 or 16-bit capacity depending upon which device is selected. As each timer has three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width measurement device. The provision of an internal prescaler to the clock circuitry of some of the timer/event counters gives added range to the timer.

There are two types of registers related to the Timer/Event Counters. The first is the register that contains the actual value of the timer and into which an initial value can be preloaded. Reading from this register retrieves the contents of the Timer/Event Counter. The second type of associated register is the timer control register which defines the timer options and determines how the timer is to be used. All devices can have the timer clock configured to come from the internal clock source. In addition, the timer clock source can also be configured to come from an external timer pin. The accompanying table lists the associated timer register names.

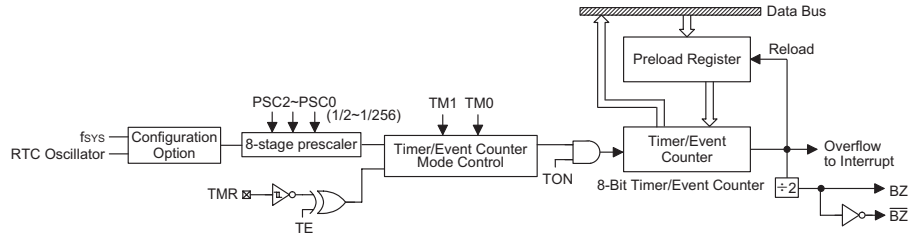
| | HT48R10A-1 HT48C10-1 | HT48R30A-1 HT48C30-1 | HT48R50A-1 HT48C50-1 | HT48R70A-1 HT48C70-1 | HT48RU80 HT48CU80 |
|-----------------------------|-------------------------|-------------------------|-------------------------|----------------------------|----------------------------|
| No. of 8-bit Timers | 1 | 1 | 1 | 0 | 1 |
| Timer Register Name | TMR | TMR | TMR0 | — | TMR2 |
| Timer Control Register | TMRC | TMRC | TMR0C | — | TMR2C |
| No. of 16-bit Timers | 0 | 0 | 1 | 2 | 2 |
| Timer Register Name | — | — | TMR1L/TMR1H | TMR0L/TMR0H TMR1L/TMR1H | TMR0L/TMR0H TMR1L/TMR1H |
| Timer Control Register | — | — | TMR1C | TMR0C TMR1C | TMR0C TMR1C |

An external clock source is used when the timer is in the event counting mode, the clock source being provided on the external timer pin known as TMR, TMR0, TMR1 or TMR2 depending on which device is selected. These external pins may be pin-shared with other I/O pins depending upon which device and package is chosen. Depending upon the condition of the TE, T0E, T1E or T2E bit in the corresponding timer control register, each high to low, or low to high transition on the external timer input pin will increment the counter by one.

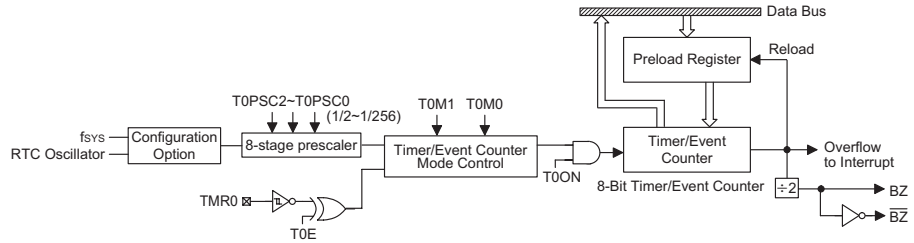
Configuring the Timer/Event Counter Input Clock Source

The internal timer's clock can originate from various sources, depending upon which device and which timer is chosen. The system clock input timer source is used when the timer is in the timer mode or in the pulse width measurement mode. Depending upon which timer and which device is chosen this system clock timer source may be first divided by a prescaler, the division ratio of which is conditioned by the timer control register bits PSC2~PSC0, T0PSC2~T0PSC0 or T2PSC2~T2PSC0.

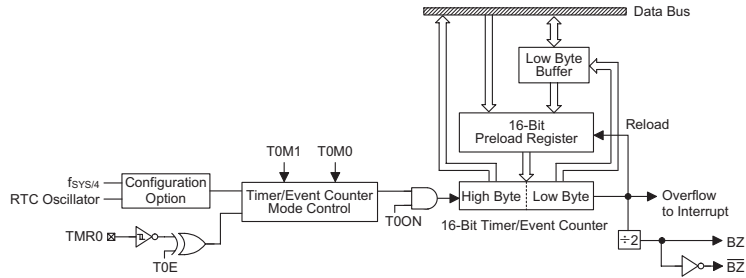
An external clock source is used when the timer is in the event counting mode, the clock source being provided on an external timer pin, TMR, TMR0, TMR1 or TMR2 depending upon which device and which timer is used. Depending upon the condition of the TE, T0E, T1E or T2E bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.



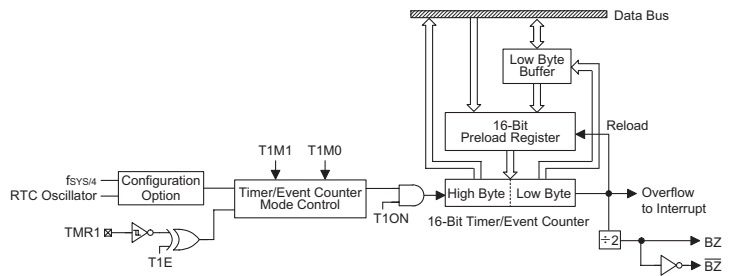
8-bit Timer/Event Counter Structure – HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1



8-bit Timer/Event Counter 0 Structure – HT48R50A-1/HT48C50-1

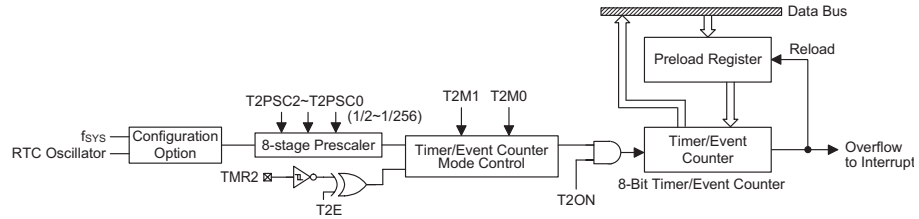


16-bit Timer/Event Counter 0 Structure – HT48R70A-1/HT48C70-1, HT48RU80/HT48CU80



Note: No buzzer output on HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1

16-bit Timer/Event Counter 1 Structure – HT48R50A-1/HT48C50-1, HT48R70A-1/HT48C70-1, HT48RU80/HT48CU80



8-bit Timer/Event Counter 2 Structure – HT48RU80/HT48CU80

Timer Registers – TMR, TMR0, TMR0L/TMR0H, TMR1L/TMR1H, TMR2

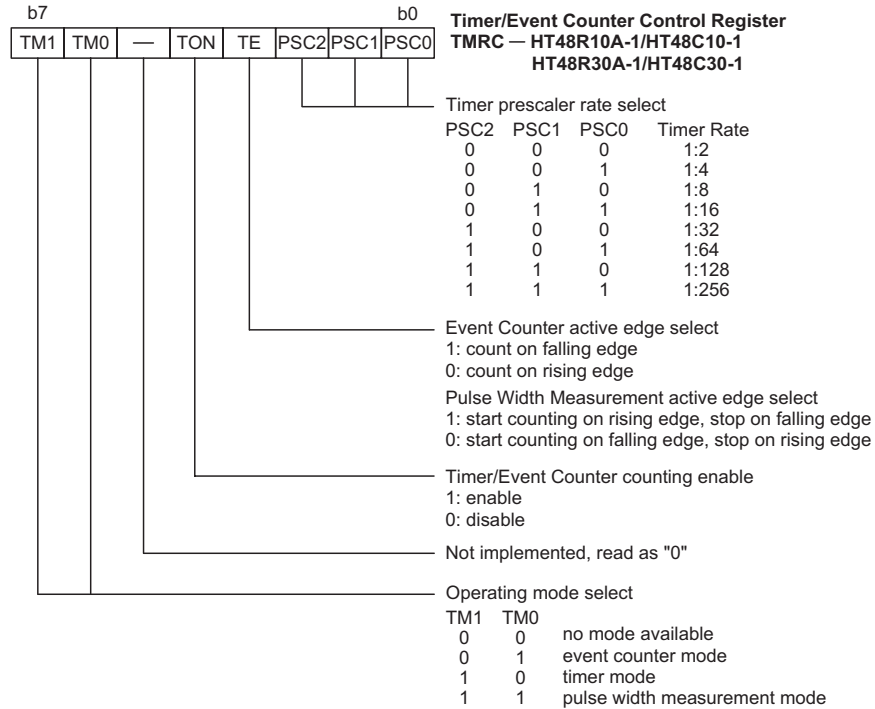
The timer registers are special function registers located in the special purpose Data Memory and is the place where the actual timer value is stored. For the 8-bit timer, this register is known as TMR for the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices, TMR0 for the HT48R50A-1/HT48C50-1 devices and TMR2 for the HT48RU80/HT48CU80 devices. In the case of the 16-bit timer, a pair of 8-bit registers is required to store the 16-bit timer value. These register pairs are known as TMR0L/TMR0H or TMR1L/TMR1H depending upon which device and timer is used. The value in the timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFH for the 8-bit timer or FFFFH for the 16-bit timers at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting.

Note that to achieve a maximum full range count of FFH for the 8-bit timer or FFFFH for the 16-bit timers, the preload registers must first be cleared to all zeros. It should be noted that after power-on, the preload registers will be in an unknown condition. Note that if the Timer/Event Counters are in an OFF condition and data is written to their preload registers, this data will be immediately written into the actual counter. However, if the counter is enabled and counting, any new data written into the preload data register during this period will remain in the preload register and will only be written into the actual counter the next time an overflow occurs. Note also that when the timer registers are read, the timer clock will be blocked to avoid errors, however, as this may result in certain timing errors, programmers must take this into account.

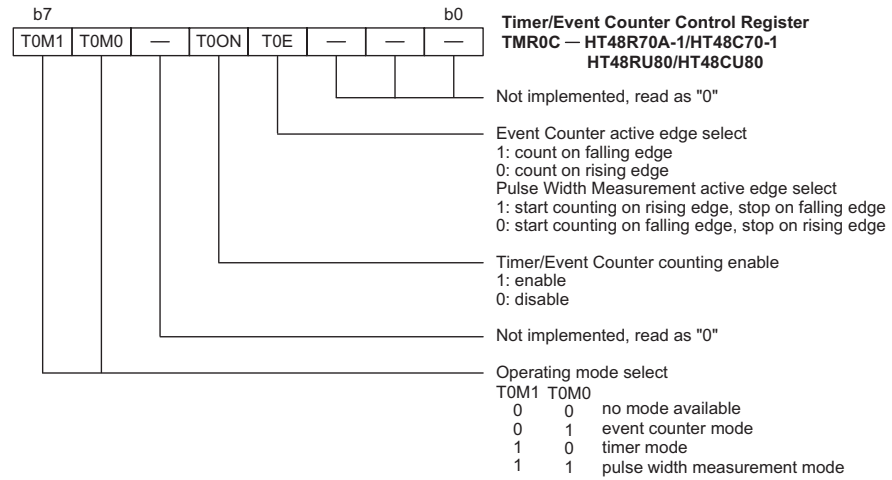
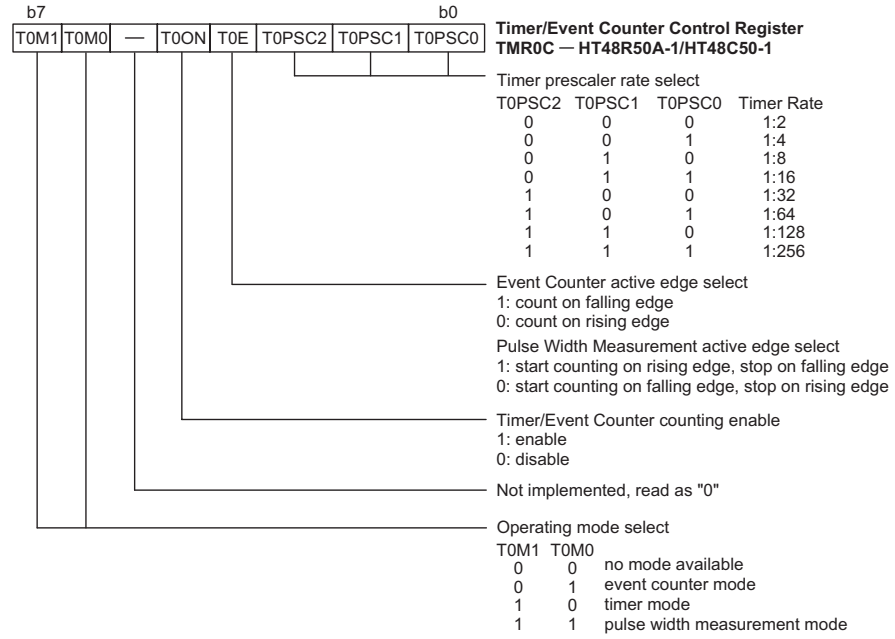
For devices which have an internal 16-bit Timer/Event Counter, and which therefore have both low byte and high byte timer registers, accessing these registers is carried out in a specific way. It must be noted that when using instructions to preload data into the low byte register, namely TMR0L or TMR1L, the data will only be placed in a low byte buffer and not directly into the low byte register. The actual transfer of the data into the low byte register is only carried out when a write to its associated high byte register, namely TMR0H or TMR1H, is executed. On the other hand, using instructions to preload data into the high byte timer register will result in the data being directly written to the high byte register. At the same time the data in the low byte buffer will be transferred into its associated low byte register. For this reason, when preloading data into the 16-bit timer registers, the low byte should be written first. It must also be noted that to read the contents of the low byte register, a read to the high byte register must first be executed to latch the contents of the low byte buffer into its associated low byte register. After this has been done, the low byte register can be read in the normal way. Note that reading the low byte timer register will only result in reading the previously latched contents of the low byte buffer and not the actual contents of the low byte timer register.

Timer Control Registers – TMRC, TMR0C, TMR1C, TMR2C

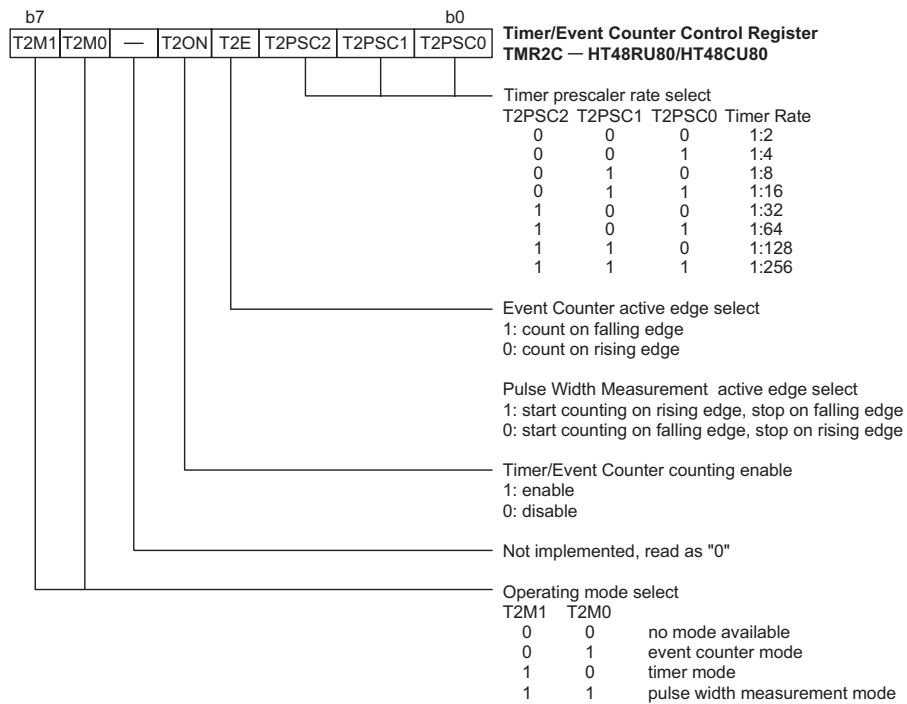
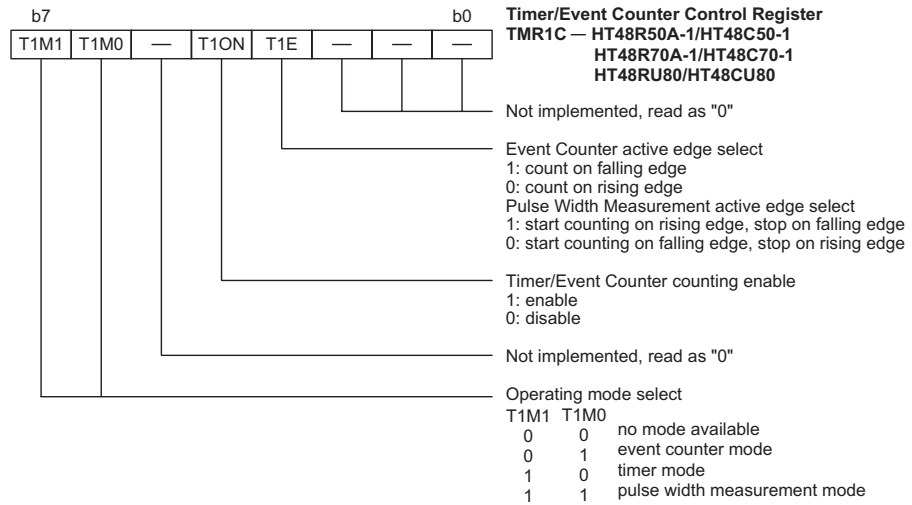
The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of their respective control register. For devices with only one timer, the single timer control register is known as TMRC while for devices with more than one timer, there are two or three timer control registers known as TMR0C, TMR1C and TMR2C. It is the timer control register together with its corresponding timer registers that control the full operation of the Timer/Event Counters. Before the timers can be used, it is essential that the appropriate timer control register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialization.



To choose which of the three modes the timer is to operate in, either in the timer mode, the event counting mode or the pulse width measurement mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair TM1/TM0, T0M1/T0M0, T1M1/T1M0 or T2M1/T2M0 respectively, depending upon which timer is used, must be set to the required logic levels. The timer-on bit, which is bit 4 of the Timer Control Register and known as TON, T0ON, T1ON or T2ON, depending upon which timer is used, provides the basic on/off control of the respective timer. Setting the bit high allows the counter to run, clearing the bit stops the counter. For timers that have prescalers, bits 0~2 of the Timer Control Register determine the division ratio of the input clock prescaler. The prescaler bit settings have no effect if an external clock source is used. If the timer is in the event count or pulse width measurement mode, the active transition edge level type is selected by the logic level of bit 3 of the Timer Control Register which is known as TE, T0E, T1E or T2E, depending upon which timer is used.

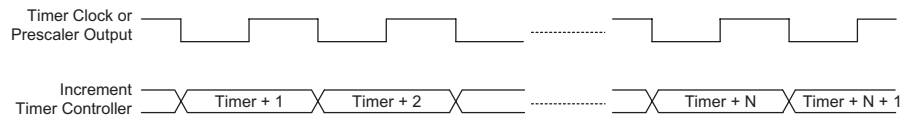


The HT48R50A-1/HT48C50-1 and HT48R70A-1/ HT48C70-1 devices have two internal timers, Timer/Event Counter 0 and Timer/Event Counter 1, and therefore require an additional timer control register TMR1C. The HT48RU80/HT48CU80 devices have an additional 8-bit timer Timer/Event Counter 2 which requires an additional control register TMR2C.



Configuring the Timer Mode

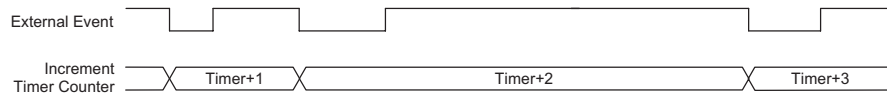
In this mode, the timer can be utilized to measure fixed time intervals, providing an internal interrupt signal each time the counter overflows. To operate in this mode, the bit pair, TM1/TM0, T0M1/T0M0, T1M1/T1M0 or T2M1/T2M0, depending upon which timer is used, must be set to 1 and 0 respectively. In this mode the internal clock is used as the timer clock. Note that for the 8-bit timers which are the single Timer/Event Counters in the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices, Timer/Event Counter 0 in the HT48R50A-1/HT48C50-1, and Timer/Event Counter 2 in the HT48RU80/HT48CU80 devices, the timer input clock source is either f_{SYS} or the f_{RTC} . However, this timer clock source is further divided by a prescaler, the value of which is determined by the bits PSC2~PSC0, T0PSC2~T0PSC0 or T2PSC2~T2PSC0 in the relevant Timer Control Register. For the remaining Timer/Event Counters, which are the 16-bit Timer/Event Counters, the input clock frequency is $f_{SYS}/4$ or the f_{RTC} . There is no prescaler function for the 16-bit timers. The timer-on bit, TON, T0ON, T1ON or T2ON, depending upon which timer is used, must be set high to enable the timer to run. Each time an internal clock high to low transition occurs, the timer increments by one; when the timer is full and overflows, an interrupt signal is generated and the timer will preload the value already loaded into the preload register and continue counting. A timer overflow condition and corresponding internal interrupt is one of the wake-up sources, however, the internal interrupts can be disabled by ensuring that the ETI or ET0I and ET1I or ET2I bits of the INTC, INTC0 and INTC1 registers are reset to zero.



Timer Mode Timing Chart

Configuring the Event Counter Mode

In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the internal timer. For the timer to operate in the event counting mode, the bit pair, TM1/TM0, T0M1/T0M0, T1M1/T1M0 or T2M1/T2M0, depending upon which timer is used, must be set to 0 and 1 respectively. The timer-on bit, TON, T0ON, T1ON or T2ON, depending upon which timer is used, must be set high to enable the timer to count. Depending upon which counter is used, if TE, T0E, T1E or T2E is low, the counter will increment each time the external timer pin receives a low to high transition. If TE, T0E, T1E or T2E is high, the counter will increment each time the external timer pin receives a high to low transition. As in the case of the other two modes, when the counter is full, the timer will overflow and generate an internal interrupt signal. The counter will then preload the value already loaded into the preload register. If the external timer pins are pin-shared with other I/O pins, to ensure that the pin is configured to operate as an event counter input pin, two things have to happen. The first is to ensure that the TM1/TM0, T0M1/T0M0, T1M1/T1M0 or T2M1/T2M0 bits place the Timer/Event Counter in the event counting mode, the second is to ensure that the port control register configures the pin as an input. It should be noted that a timer overflow is one of the interrupt and wake-up sources.

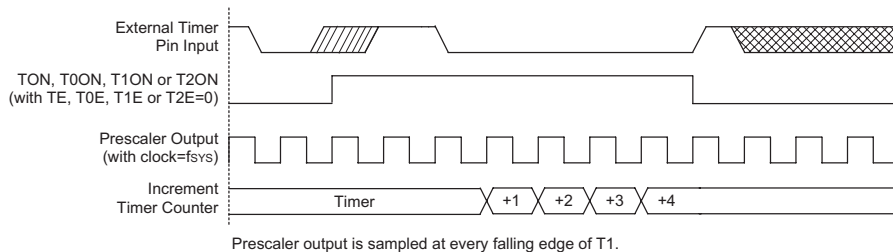


Event Counter Mode Timing Chart

Configuring the Pulse Width Measurement Mode

In this mode, the width of external pulses applied to the external timer pin can be measured. In the Pulse Width Measurement Mode the timer clock source is supplied by the internal clock. For the timer to operate in this mode, the bit pair, TM1/TM0, TOM1/TOM0, T1M1/T1M0 or T2M1/T2M0, depending upon which timer is used, must both be set high. Depending upon which counter is used, if TE, T0E, T1E or T2E is low, once a high to low transition has been received on the external timer pin, the timer will start counting until the external timer pin returns to its original high level. At this point the TON, T0ON, T1ON or T2ON bit, depending upon which counter is used, will be automatically reset to zero and the timer will stop counting. If the TE, T0E, T1E or T2E bit is high, the timer will begin counting once a low to high transition has been received on the external timer pin and stop counting when the external timer pin returns to its original low level. As before, the TON, T0ON, T1ON or T2ON bit will be automatically reset to zero and the timer will stop counting. It is important to note that in the Pulse Width Measurement Mode, the TON, T0ON, T1ON or T2ON bit is automatically reset to zero when the external control signal on the external timer pin returns to its original level, whereas in the other two modes the TON, T0ON, T1ON or T2ON bit can only be reset to zero under program control. The residual value in the timer, which can now be read by the program, therefore represents the length of the pulse received on the external timer pin. As the TON, T0ON, T1ON or T2ON bit has now been reset, any further transitions on the external timer pin, will be ignored. Not until the TON, T0ON, T1ON or T2ON bit is again set high by the program can the timer begin further pulse width measurements. In this way, single shot pulse measurements can be easily made. It should be noted that in this mode the counter is controlled by logical transitions on the external timer pin and not by the logic level.

As in the case of the other two modes, when the counter is full, the timer will overflow and generate an internal interrupt signal. The counter will also be reset to the value already loaded into the preload register. If the external timer pin is pin-shared with other I/O pins, to ensure that the pin is configured to operate as a pulse width measuring input pin, two things have to happen. The first is to ensure that the TM1/TM0, TOM1/TOM0, T1M1/T1M0 or T2M1/T2M0 bits place the Timer/Event Counter in the pulse width measuring mode, the second is to ensure that the port control register configures the pin as an input. It should be noted that a timer overflow is one of the interrupt and wake-up sources.



Pulse Width Measurement Mode Timing Chart

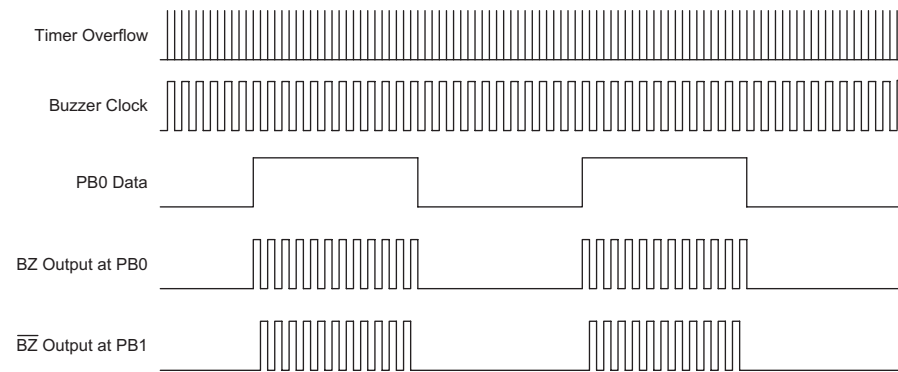
Programmable Frequency Divider (PFD) and Buzzer Application

Operating similar to a programmable frequency divider, the buzzer function within the microcontroller provides a means of producing a variable frequency output suitable for applications such as piezo-buzzer driving or other interfaces requiring a precise frequency generator.

The BZ and $\overline{\text{BZ}}$ are a complimentary pair and pin-shared with I/O pins, PB0 and PB1. The function is selected via configuration option to have single BZ output or both BZ and $\overline{\text{BZ}}$ outputs, however, if not selected, the pins can operate as normal I/O pins. Note that the $\overline{\text{BZ}}$ pin is the inverse of the BZ pin generating a kind of differential output and supplying more power to connected interfaces such as buzzers.

The timer overflow signal is the clock source for the buzzer circuit. The output frequency is controlled by loading the required values into the timer prescaler registers to give the required division ratio. The counter will begin to count-up from this preload register value until full, at which point an overflow signal is generated, causing both the BZ and $\overline{\text{BZ}}$ outputs to change state. The counter will then be automatically reloaded with the preload register value and continue counting-up. Refer to the relevant Timer/Event Counters section for details of its settings and operations. For the HT48R50A-1/HT48C50-1 and HT48RU80/HT48CU80 devices, either Timer/Event Counter 0 or Timer/Event Counter 1 can form the clock source for the Buzzer function. For the HT48R70A-1/HT48C70-1 devices, only Timer/Event Counter 0 can form the Buzzer clock source.

For the buzzer outputs to function, it is essential that the Port B control register PBC bit 0 and PBC bit 1 are setup as outputs. If they are setup as inputs the buzzer output will not function, and the pins can be used as normal input pins. The BZ and $\overline{\text{BZ}}$ outputs will only be activated if bit PB0 is set to "1". This output data bit is used as the on/off control bit for the BZ and $\overline{\text{BZ}}$ outputs. Note that the BZ and $\overline{\text{BZ}}$ outputs will both be low if the PB0 output data bit is cleared to "0". Note that the condition of bit PB1 has no effect on the overall control of the BZ and $\overline{\text{BZ}}$ pins.



PFD Output Control

Using this method of frequency generation, and if a crystal oscillator is used for the system clock, very precise values of frequency can be generated.

Prescaler

The single timer in the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1, Timer/Event Counter 0 in the HT48R50A-1/HT48C50-1 as well as Timer/Event Counter 2 in the HT48RU80/HT48CU80 all possess a prescaler. Bits 0~2 of their associated timer control register, namely bits PSC0~PSC2, T0PSC0~T0PSC2 or T2PSC0~T2PSC2, define the pre-scaling stages of the internal clock source of the Timer/Event Counter. The Timer/Event Counter overflow signal can be used to generate signals for the PFD and as a Timer interrupt.

I/O Interfacing

The Timer/Event Counter when configured to run in the event counter or pulse width measurement mode, require the use of the external timer pin for correct operation. This external timer pin may be pin-shared with other I/O pins, depending upon which device is selected. For pin-shared timer pins, pull-high resistors can be selected for connection to the timer input pins. The timers can also be setup to drive the pin-shared buzzer pins. When the buzzer pins are selected by selecting the correct configuration option, the output of the chosen timer can be made to drive this at a frequency determined by the contents of the timer register and where appropriate the timer.

Programming Considerations

When configured to run in the timer mode, the internal system clock or RTC is used as the timer clock source and is therefore synchronized with the overall operation of the microcontroller. In this mode, when the appropriate timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the pulse width measurement mode, the internal system clock or RTC is also used as the timer clock source but the timer will only run when the correct logic condition appears on the external timer input pin. As this is an external event and not synchronized with the internal timer clock, the microcontroller will only see this external event when the next timer clock pulse arrives. As a result there may be small differences in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to be in the event counting mode which again is an external event and not synchronized with the internal system or timer clock.

Timer Program Example

The following example program section is based on the HT48R50A-1/HT48C50-1 devices, which contain one internal 8-bit Timer/Event Counter and one internal 16-bit Timer/Event Counter. Programming the Timer/Event Counters for other devices is conducted in a very similar way. The program shows how the Timer/Event Counter registers are setup along with how the interrupts are enabled and managed. Points to note in the example are how, for the 16-bit Timer/Event Counters, the low byte must be written first, this is because the 16-bit data will only be written into the actual timer register when the high byte is loaded. Also note how the Timer/Event Counter is turned on, by setting bit 4 of the respective timer control register. The Timer/Event Counter can be turned off in a similar way by clearing the same bit. This example program sets the Timer/Event Counters to be in the timer mode, which uses the internal system clock as their clock source. The Timer/Event Counter clock sources are setup via configuration options.

Interrupts

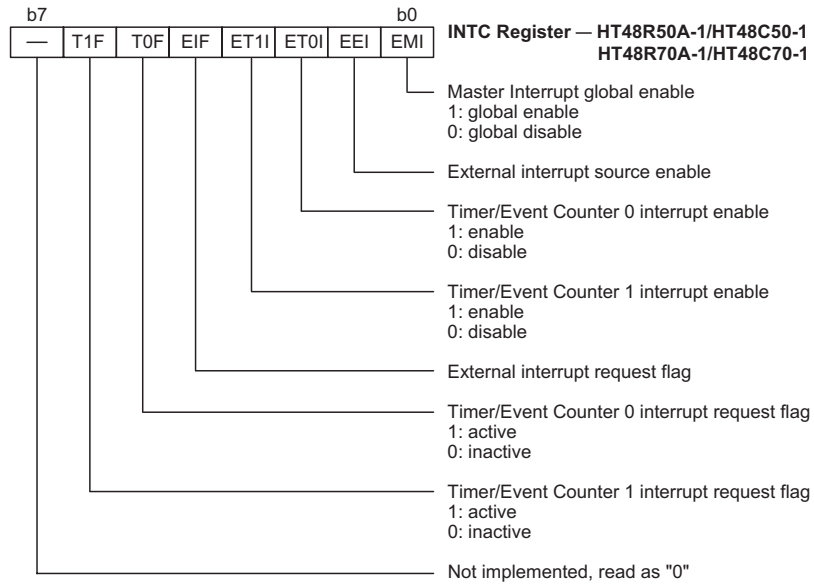
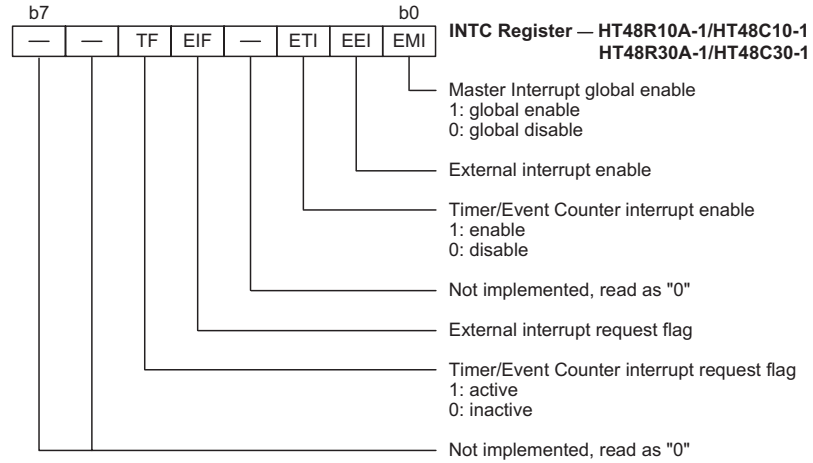
The I/O series of microcontrollers each contains a range of both external and internal interrupt functions. The external interrupts are controlled by the action of one or two external pins, which are present on all devices. The internal interrupts are controlled by the Timer/Event Counters, and in the case of the HT48RU80/HT48CU80 devices by an additional UART function. One or two internal Interrupt Control registers contain the control bits which manage the enable/disable function of the individual interrupts and their corresponding interrupt request flags.

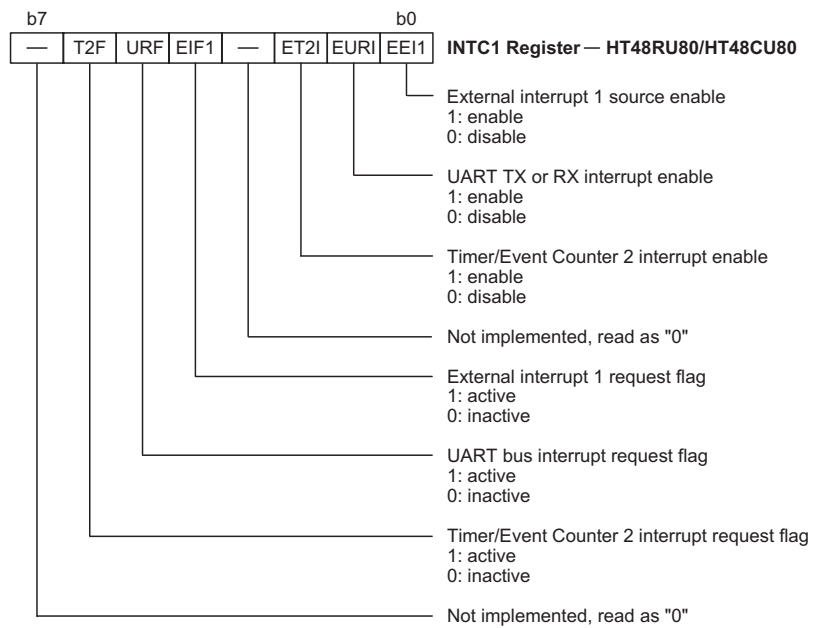
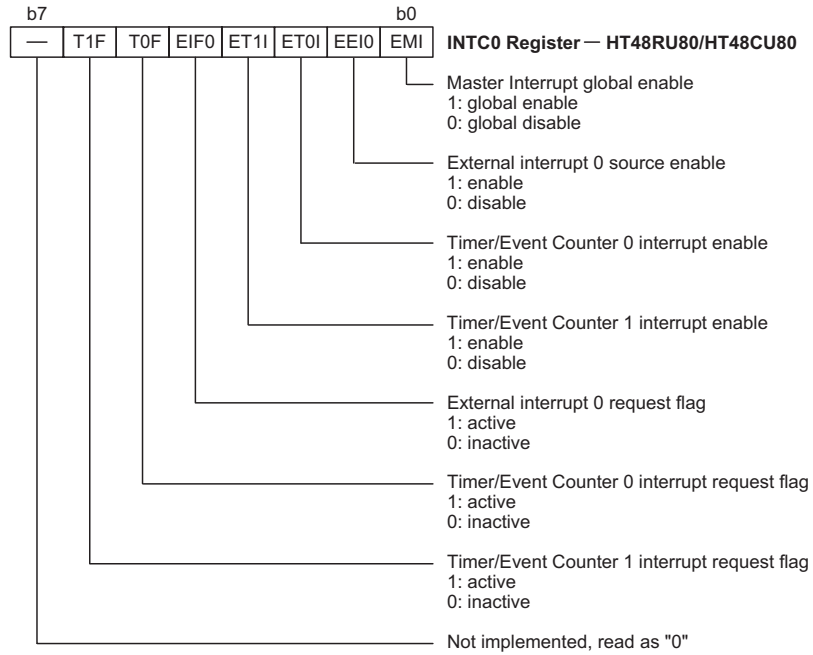
Interrupt Registers

With the exception of the HT48RU80/HT48CU80 devices, a single interrupt control register, known as INTC is provided to control all the interrupt control features. For the HT48RU80/HT48CU80 devices, two interrupt control registers, known as INTC0 and INTC1, are provided.

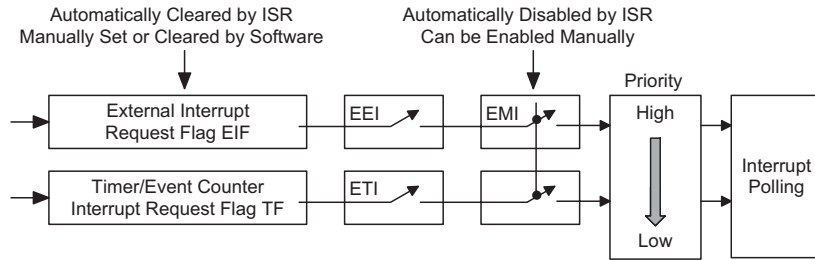
Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately serviced, the request flag will still be recorded. If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full.

All interrupts have the capability of waking up the processor when in the Power Down Mode. As an interrupt is serviced, a control transfer occurs by pushing the Program Counter onto the stack, followed by a branch to a subroutine at a specified location in the Program Memory. Only the Program Counter is pushed onto the stack. If the contents of the accumulator or status register or other registers are altered by the interrupt service program, which may corrupt the desired control sequence, then the contents should be saved in advance.

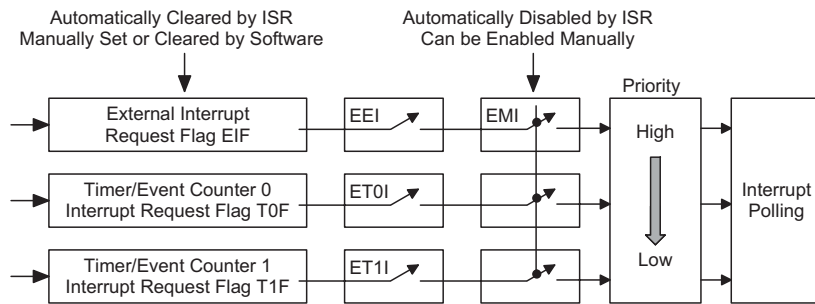




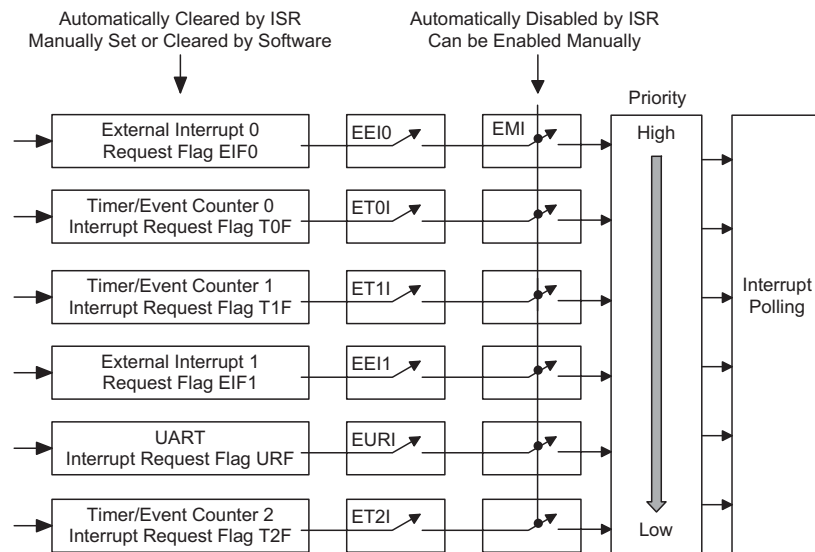
The various interrupt enable bits, together with their associated request flags, are shown in the following diagrams with their order of priority.



Interrupt Scheme – HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1



Interrupt Scheme – HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1



Interrupt Scheme – HT48RU80/HT48CU80

Interrupt Priority

Interrupts, occurring in the interval between the rising edges of two consecutive T2 pulses, will be serviced on the latter of the two T2 pulses, if the corresponding interrupts are enabled. In case of simultaneous requests, the following table shows the priority that is applied.

| Interrupt Source | HT48R10A-1 HT48C10-1 Priority | HT48R30A-1 HT48C30-1 Priority | HT48R50A-1 HT48C50-1 Priority | HT48R70A-1 HT48C70-1 Priority | HT48RU80 HT48CU80 Priority |
|--|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|----------------------------------|
| External Interrupt 0 | 1 | 1 | 1 | 1 | 1 |
| Timer/Event Counter or Timer/Event Counter 0 Overflow | 2 | 2 | 2 | 2 | 2 |
| Timer/Event Counter 1 Overflow | N/A | N/A | 3 | 3 | 3 |
| External Interrupt 1 | N/A | N/A | N/A | N/A | 4 |
| UART Interrupt | N/A | N/A | N/A | N/A | 5 |
| Timer/Event Counter 2 Overflow | N/A | N/A | N/A | N/A | 6 |

- Note**
1. For the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices, there is only one timer. The HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1 devices have two internal timers, and the HT48RU80/HT48CU80 devices have three internal timers.
 2. Only the HT48RU80/HT48CU80 devices have a UART interrupt.

In cases where several interrupts are enabled, and where more than one interrupt occur simultaneously, the interrupt that is serviced first will follow the order shown in the table. Suitable masking of the individual interrupts using the INTC or the INTC0 and INTC1 registers can prevent simultaneous occurrences.

External Interrupt

For an external interrupt to occur, the corresponding external interrupt enable bit must be first set. With the exception of the HT48RU80/HT48CU80 devices, that is for devices with a single external interrupt pin, the enable bit is bit 1 of the INTC register, known as EEI. For the HT48RU80/HT48CU80 devices, which have two external interrupt pins, the enable bit for pin $\overline{\text{INT0}}$ is bit 1 of the INTC0 register, known as EEI0, and bit 0 of the INTC1 register, known as EEI1, for pin $\overline{\text{INT1}}$. An external interrupt is triggered by a high to low transition on the external interrupt pin, after which the related interrupt request flag will be set. With the exception of the HT48RU80/HT48CU80 devices, that is for devices with a single external interrupt pin, this is bit 4 of the INTC register, known as EIF. For the HT48RU80/HT48CU80 devices, there are two corresponding external interrupt request flags, these are bit 4 of the INTC0 register, known as EIF0 and bit 4 of the INTC1 register, known as EIF1. When the master interrupt and external interrupt bits are enabled, the stack is not full and a high to low transition occurs on the external interrupt pin, a subroutine call to the corresponding external interrupt vector will occur. After entering the interrupt execution routine, the corresponding interrupt request flag, either EIF, EIF0 or EIF1 will be reset and the EMI bit will be cleared to disable other interrupts.

For the HT48R10A-1/HT48C10-1, the external interrupt pin $\overline{\text{INT}}$ is pin-shared with PC0 and for the HT48R30A-1/HT48C30-1 and HT48R50A-1/HT48C50-1 devices, the external interrupt pin $\overline{\text{INT}}$ is

pin-shared with PG0. For the HT48RU80/HT48CU80 devices, the external interrupt pin $\overline{\text{INT1}}$ is pin-shared with PB2. Note that the external interrupt pins must be setup as inputs to enable correct operation.

Timer/Event Counter Interrupt

For a timer generated internal interrupt to occur, the corresponding internal interrupt enable bit must be first set. For devices with a single timer, this is bit 2 of the INTC register and is known as ETI. For devices with two timers, the Timer/Event Counter 0 interrupt enable is bit 2 of the INTC register and known as ET0I while the Timer 1 interrupt enable is bit 3 of the INTC register and known as ET1I. In the case of the HT48RU80/HT48CU80 devices, which have three internal Timer/Event Counters, the Timer/Event Counter 0 interrupt enable is bit 2 of the INTC0 register and known as ET0I, the Timer/Event Counter 1 interrupt enable is bit 3 of the INTC0 register and known as ET1I and the Timer/Event Counter 2 interrupt enable is bit 2 of the INTC1 register and is known as ET2I. An actual Timer/Event Counter interrupt will be initialized when the Timer/Event Counter interrupt request flag is set, caused by a timer overflow. For devices which have a single timer, this is bit 5 of the INTC register and is known as TF. For devices which have two timers, the Timer/Event Counter 0 request flag is bit 5 of the INTC register and known as T0F, while the Timer/Event Counter 1 request flag is bit 6 of the INTC register and known as T1F. In the case of the HT48RU80/HT48CU80 devices, which has three timers, the Timer/Event Counter 0 request flag is bit 5 of the INTC0 register and known as T0F, the Timer/Event Counter 1 request flag is bit 6 of the INTC0 register and known as T1F, and the Timer/Event Counter 2 request flag is bit 6 of the INTC1 register and is known as T2F.

When the master interrupt global enable bit is set, the stack is not full and the corresponding timer interrupt enable bit is set, an internal interrupt will be generated when the corresponding timer overflows. This will create a subroutine call to location 08H for devices with a single timer. For devices with two timers, a subroutine call to location 08H will occur for Timer/Event Counter 0 and a subroutine call to location 0CH for Timer/Event Counter 1. For the HT48RU80/HT48CU80 devices, which have three internal Timer/Event Counters, a subroutine call to location 08H will occur for Timer/Event Counter 0, a subroutine call to location 0CH for Timer/Event Counter 1 and a subroutine call to location 018H for Timer/Event Counter 2. After entering the timer interrupt execution routine, the corresponding timer interrupt request flag, either, TF, T0F, T1F or T2F will be reset and the EMI bit will be cleared to disable other interrupts.

UART Interrupt

In the HT48RU80/HT48CU80 devices, which are the only devices which contain an internal UART function, its corresponding UART interrupt is enabled by setting the EURI bit, which is bit 1 of the INTC1 register. An actual UART interrupt will be initialized when the UART interrupt request flag URF is set, which is bit 5 of the INTC1 register. When the master interrupt global bit is set, the stack is not full and the corresponding EURI interrupt enable bit is set, a UART internal interrupt will be generated when a UART interrupt request occurs. This will create a subroutine call to its corresponding vector location 014H. When a UART internal interrupt occurs, the interrupt request flag URF will be reset and the EMI bit cleared to disable other interrupts.

There are various UART conditions, which can generate a UART interrupt, such as certain data transmission and reception conditions, overrun errors as well as an address detect condition. These conditions are reflected by various flags within the UART's status register, known as the

USR register. Various bits in the UART's setup register, UCR2, determine if these flags can generate a UART interrupt signal. More details on these two registers and how they influence the operation of the UART interrupt can be found in the UART section of the handbook.

Programming Considerations

The interrupt request flags, TF, T0F, T1F, T2F, URF, EIF, EIF0 and EIF1, together with the interrupt enable bits ETI, ET0I, ET1I, ET2I, EURI, EEI, EEI0 and EEI1, form the interrupt control registers INTC, INTC0 and INTC1, which are located in the Data Memory. By disabling the interrupt enable bits, a requested interrupt can be prevented from being serviced, however, once an interrupt request flag is set, it will remain in this condition in the INTC, INTC0 or INTC1 register until the corresponding interrupt is serviced or until the request flag is cleared by a software instruction.

It is recommended that programs do not use the "CALL subroutine" instruction within the interrupt subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately in some applications. If only one stack is left and the interrupt is not well controlled, the original control sequence will be damaged once a "CALL subroutine" is executed in the interrupt subroutine.

Reset and Initialization

A reset function is a fundamental part of any microcontroller ensuring that the device can be set to some predetermined condition irrespective of outside parameters. The most important reset condition is after power is first applied to the microcontroller. In this case, internal circuitry will ensure that the microcontroller, after a short delay, will be in a well defined state and ready to execute the first program instruction. After this power-on reset, certain important internal registers will be set to defined states before the program commences. One of these registers is the Program Counter, which will be reset to zero forcing the microcontroller to begin program execution from the lowest Program Memory address.

In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is running. One example of this is where after power has been applied and the microcontroller is already running, the RES line is forcefully pulled low. In such case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to proceed with normal operation after the reset line is allowed to return high. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{\text{RES}}$ reset is implemented in situations where the power supply voltage falls below a certain threshold.

Reset

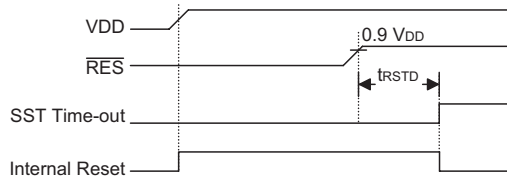
There are five ways in which a microcontroller reset can occur, through events occurring both internally and externally:

Power-on Reset

The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first

memory address, a power-on reset also ensures that certain other registers are preset to known conditions. All the I/O port and port control registers will power-up in a high condition ensuring that all pins will be first set to inputs.

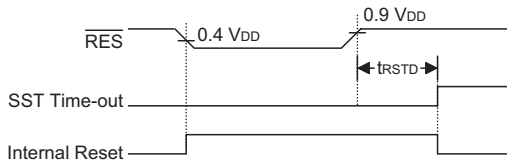
Although the microcontroller has an internal RC reset function, due to unstable power-on conditions, an external RC network connected to the $\overline{\text{RES}}$ pin is generally recommended. This time delay created by the RC network ensures that the $\overline{\text{RES}}$ pin remains low for an extended period while the power supply stabilizes. During this time, normal operation of the microcontroller is inhibited. After the RES line reaches a certain voltage value, the reset delay time t_{RSTD} is invoked to provide an extra delay time after which the microcontroller can begin normal operation. The abbreviation SST in the figures stands for System Start-up Timer.



Power-On Reset Timing Chart

$\overline{\text{RES}}$ Pin Reset

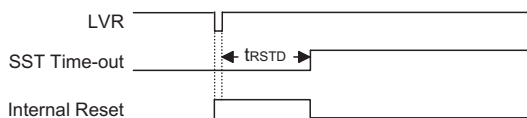
This type of reset occurs when the microcontroller is already running and the $\overline{\text{RES}}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point.



RES Reset Timing Chart

Low Voltage Reset – LVR

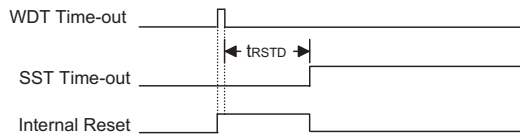
The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. If the supply voltage of the device drops to within a range of $0.9V \sim V_{\text{LVR}}$ such as might occur when changing the battery, the LVR will automatically reset the device internally. The LVR includes the following specifications: For a valid LVR signal, a low voltage, i.e. a voltage in the range between $0.9V \sim V_{\text{LVR}}$ must exist for greater than 1ms. If the low voltage state does not exceed 1ms, the LVR will ignore it and will not perform a reset function.



Low Voltage Reset Timing Chart

Watchdog Time-out Reset during Normal Operation

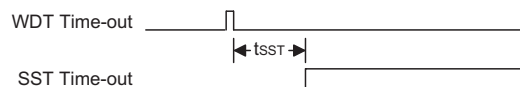
The Watchdog time-out Reset during normal operation is the same as $\overline{\text{RES}}$ reset except that the Watchdog time-out flag TO will be set to "1".



WDT Time-out Reset during Normal Operation Timing Chart

Watchdog Time-out Reset during HALT

The Watchdog time-out Reset during HALT is a little different from other kinds of reset. Most of the conditions remain unchanged except that the Program Counter and the Stack Pointer will be cleared to "0" and the TO flag will be set to "1". Refer to the A.C. Characteristics for t_{SST} details.



WDT Time-out Reset during HALT Timing Chart

The different types of reset described affect the reset flags in different ways. These flags known as PDF and TO are located in the status register and are controlled by various microcontroller operations such as the HALT function or Watchdog Timer. The reset flags are shown in the table:

| TO | PDF | RESET Conditions |
|----|-----|--|
| 0 | 0 | $\overline{\text{RES}}$ reset during power-on |
| u | u | $\overline{\text{RES}}$ or LVR reset during normal operation |
| 1 | u | WDT time-out reset during normal operation |
| 1 | 1 | WDT time-out reset during HALT |

"u" stands for unchanged

The following table indicates the way in which the various components of the microcontroller are affected after a power-on reset occurs.

| Item | Condition After RESET |
|---------------------|--|
| Program Counter | Reset to zero |
| Interrupts | All interrupts will be disabled |
| WDT | Clear after reset, WDT begins counting |
| Timer/Event Counter | All Timer Counters will be turned off |
| Prescaler | The Timer Counter Prescaler will be cleared |
| Input/Output Ports | All I/O ports will be setup as inputs |
| Stack Pointer | Stack Pointer will point to the top of the stack |
| UART | UART disabled, TX and RX pins are setup as PC0/PC1 |

The different kinds of resets all affect the internal registers of the microcontroller in different ways. To ensure reliable continuation of normal program execution after a reset occurs, it is important to know what condition the microcontroller is in after a particular reset occurs. The following table describes how each type of reset affects each of the microcontroller internal registers.

HT48R10A-1/HT48C10-1

| Register | Reset (Power-on) | RES or LVR Reset | WDT Time-out (Normal Operation) | WDT Time-out (HALT) |
|----------|------------------|------------------|---------------------------------|---------------------|
| MP | -xxx xxxx | -uuu uuuu | -uuu uuuu | -uuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| PCL | 0000 0000 | 0000 0000 | 0000 0000 | 0000 0000 |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | --xx xxxx | --uu uuuu | --uu uuuu | --uu uuuu |
| STATUS | --00 xxxx | --uu uuuu | --1u uuuu | --11 uuuu |
| INTC | --00 -000 | --00 -000 | --00 -000 | --uu -uuu |
| WDTS | 0000 0111 | 0000 0111 | 0000 0111 | uuuu uuuu |
| TMR | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRC | 00-0 1000 | 00-0 1000 | 00-0 1000 | uu-u uuuu |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PC | ---1 1111 | ---1 1111 | ---1 1111 | ---u uuuu |
| PCC | ---1 1111 | ---1 1111 | ---1 1111 | ---u uuuu |

"u" stands for unchanged

"x" stands for unknown

"-" stands for unimplemented

HT48R30A-1/HT48C30-1

| Register | Reset (Power-on) | RES or LVR Reset | WDT Time-out (Normal Operation) | WDT Time-out (HALT) |
|----------|---------------------|---------------------|------------------------------------|------------------------|
| MP | -xxx xxxx | -uuu uuuu | -uuu uuuu | -uuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| PCL | 0000 0000 | 0000 0000 | 0000 0000 | 0000 0000 |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | --xx xxxx | --uu uuuu | --uu uuuu | --uu uuuu |
| STATUS | --00 xxxx | --uu uuuu | --1u uuuu | --11 uuuu |
| INTC | --00 -000 | --00 -000 | --00 -000 | --uu -uuu |
| WDTS | 0000 0111 | 0000 0111 | 0000 0111 | uuuu uuuu |
| TMR | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRC | 00-0 1000 | 00-0 1000 | 00-0 1000 | uu-u uuuu |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PC | --11 1111 | --11 1111 | --11 1111 | --uu uuuu |
| PCC | --11 1111 | --11 1111 | --11 1111 | --uu uuuu |
| PG | ---- -111 | ---- -111 | ---- -111 | ---- -uuu |
| PGC | ---- -111 | ---- -111 | ---- -111 | ---- -uuu |

"u" stands for unchanged

"x" stands for unknown

"-" stands for unimplemented

HT48R50A-1/HT48C50-1

| Register | Reset (Power-on) | RES or LVR Reset | WDT Time-out (Normal Operation) | WDT Time-out (HALT) |
|----------|-------------------|-------------------|---------------------------------|---------------------|
| MP0 | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| MP1 | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| ACC | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| PCL | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| TBLP | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| TBLH | - x x x x x x x | - u u u u u u u | - u u u u u u u | - u u u u u u u |
| STATUS | -- 0 0 x x x x | -- u u u u u u | -- 1 u u u u u | -- 1 1 u u u u |
| INTC | - 0 0 0 0 0 0 0 0 | - 0 0 0 0 0 0 0 0 | - 0 0 0 0 0 0 0 0 | - u u u u u u u u |
| WDTS | 0 0 0 0 0 1 1 1 | 0 0 0 0 0 1 1 1 | 0 0 0 0 0 1 1 1 | u u u u u u u u |
| TMR0 | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR0C | 0 0 - 0 1 0 0 0 | 0 0 - 0 1 0 0 0 | 0 0 - 0 1 0 0 0 | u u - u u u u u |
| TMR1H | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR1L | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR1C | 0 0 - 0 1 - - - | 0 0 - 0 1 - - - | 0 0 - 0 1 - - - | u u - u u - - - |
| PA | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PAC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PB | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PBC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PCC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PD | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PDC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PG | - - - - - 1 1 1 | - - - - - 1 1 1 | - - - - - 1 1 1 | - - - - - u u u |
| PGC | - - - - - 1 1 1 | - - - - - 1 1 1 | - - - - - 1 1 1 | - - - - - u u u |

"u" stands for unchanged
"x" stands for unknown
"- " stands for unimplemented

HT48R70A-1/HT48C70-1

| Register | Reset (Power-on) | RES or LVR Reset | WDT Time-out (Normal Operation) | WDT Time-out (HALT) |
|----------|---------------------|---------------------|------------------------------------|------------------------|
| MP0 | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| MP1 | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| ACC | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| PCL | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| TBLP | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| TBLH | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| STATUS | --00 x x x x | --uu u u u u | --1u u u u u | --11 u u u u |
| INTC | -000 0000 | -000 0000 | -000 0000 | -uuu u u u u |
| WDTS | 0000 0111 | 0000 0111 | 0000 0111 | u u u u u u u u |
| TMR0H | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR0L | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR0C | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- |
| TMR1H | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR1L | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR1C | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PBC | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PC | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PCC | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PD | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PDC | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PE | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PEC | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PF | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PFC | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PG | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |
| PGC | 1111 1111 | 1111 1111 | 1111 1111 | u u u u u u u u |

"u" stands for unchanged

"x" stands for unknown

"-" stands for unimplemented

HT48RU80/HT48CU80

| Register | Reset (Power-on) | RES or LVR Reset | WDT Time-out (Normal Operation) | WDT Time-out (HALT) |
|----------|------------------|------------------|---------------------------------|---------------------|
| MP0 | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| MP1 | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| ACC | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| PCL | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| BP | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | u u u u u u u u |
| TBLP | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| TBLH | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| TBHP | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| STATUS | --00 x x x x | --uu u u u u | --1u u u u u | --11 u u u u |
| INTC0 | -000 0000 | -000 0000 | -000 0000 | -uuu u u u u |
| INTC1 | -000 -000 | -000 -000 | -000 -000 | -uuu -uuu |
| WDTS | 0000 0111 | 0000 0111 | 0000 0111 | u u u u u u u u |
| TMR0H | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR0L | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR0C | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- |
| TMR1H | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR1L | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR1C | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- |
| TMR2 | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR2C | 00-0 1000 | 00-0 1000 | 00-0 1000 | uu-u u u u u |
| PA | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PAC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PB | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PBC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PCC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PD | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PDC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PE | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PEC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PF | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PFC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PG | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PGC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| USR | 0000 1011 | 0000 1011 | 0000 1011 | u u u u u u u u |
| UCR1 | 0000 00x0 | 0000 00x0 | 0000 00x0 | u u u u u u u u |
| UCR2 | 0000 0000 | 0000 0000 | 0000 0000 | u u u u u u u u |
| TXR/RXR | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| BRG | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |

"u" stands for unchanged
"x" stands for unknown
"--" stands for unimplemented

Universal Asynchronous Receiver/Transmitter – UART

This section applies only to the HT48RU80/HT48CU80 which are the only devices in the series that have an internal UART function. The HT48RU80/HT48CU80 devices contain an integrated full-duplex asynchronous serial communications UART interface that enables communication with external devices that contain a serial interface. The UART function has many features and can transmit and receive data serially by transferring a frame of data with eight or nine data bits per transmission as well as being able to detect errors when the data is overwritten or incorrectly framed. The UART function possesses its own internal interrupt which can be used to indicate when a reception occurs or when a transmission terminates.

UART Features

The integrated UART function contains the following features:

- Full-duplex, Asynchronous Communication
- 8 or 9 Bits Character Length
- Even, Odd or No Parity Options
- One or Two Stop Bits
- Baud Rate Generator with 8-bit Prescaler
- Parity, Framing, Noise and Overrun Error Detection
- Support for Interrupt on Address Detect (Last Character bit=1)
- Separately Enabled Transmitter and Receiver
- 2-byte Deep FIFO Receive Data Buffer
- Transmit and Receive Interrupts
- Interrupts Can be Initialized by the Following Conditions:
 - Transmitter Empty
 - Transmitter Idle
 - Receiver Full
 - Receiver Overrun
 - Address Mode Detect

UART External Pin Interfacing

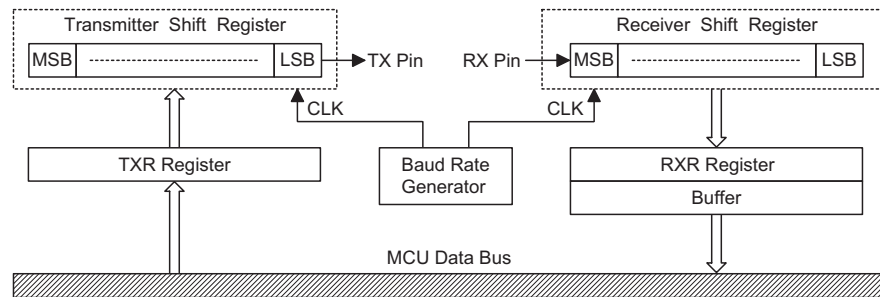
To communicate with an external serial interface, the internal UART has two external pins known as TX and RX. The TX pin is the UART transmitter pin, which can be used as a general purpose I/O pin if the pin is not configured as a UART transmitter, which occurs when the TXEN bit in the UCR2 control register is equal to zero. Similarly, the RX pin is the UART receiver pin, which can also be used as a general purpose I/O pin, if the pin is not configured as a receiver, which occurs if the RXEN bit in the UCR2 register is equal to zero. Along with the UARTEN bit, the TXEN and RXEN bits, if set, will automatically setup these I/O pins to their respective TX output and RX input conditions and disable any pull-high resistor option which may exist on the RX pin.

UART Data Transfer Scheme

The block diagram shows the overall data transfer structure arrangement for the UART. The actual data to be transmitted from the MCU is first transferred to the TXR register by the application program. The data will then be transferred to the Transmit Shift Register from where it will be shifted out, LSB first, onto the TX pin at a rate controlled by the Baud Rate Generator. Only the TXR register is mapped onto the MCU Data Memory, the Transmit Shift Register is not mapped and is therefore inaccessible to the application program.

Data to be received by the UART is accepted on the external RX pin, from where it is shifted in, LSB first, to the Receiver Shift Register at a rate controlled by the Baud Rate Generator. When the shift register is full, the data will then be transferred from the shift register to the internal RXR register, where it is buffered and can be manipulated by the application program. Only the RXR register is mapped onto the MCU Data Memory, the Receiver Shift Register is not mapped and is therefore inaccessible to the application program.

It should be noted that the actual register for data transmission and reception, although referred to in the text, and in application programs, as separate TXR and RXR registers, only exists as a single shared register in the Data Memory. This shared register known as the TXR/RXR register is used for both data transmission and data reception.



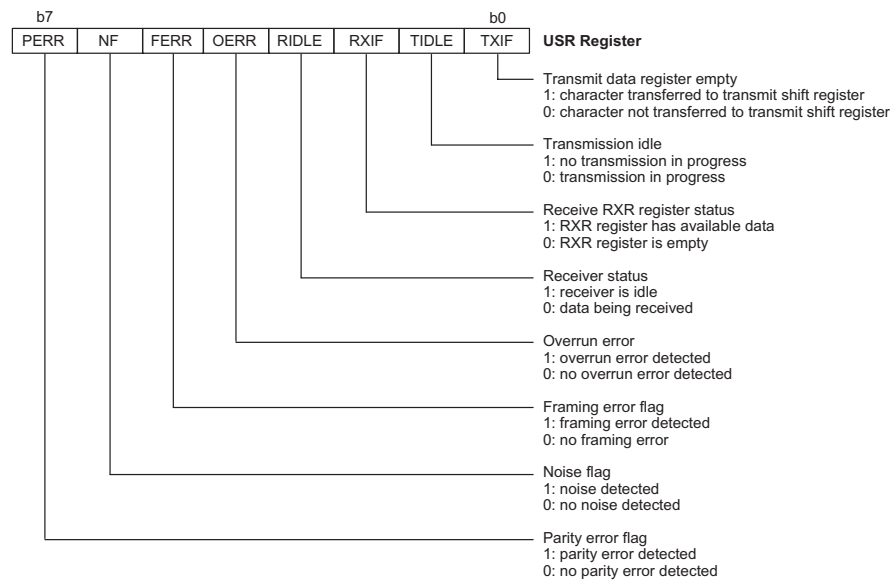
UART Data Transfer Scheme

UART Status and Control Registers

There are five control registers associated with the UART function. The USR, UCR1 and UCR2 registers control the overall function of the UART, while the BRG register controls the Baud rate. The actual data to be transmitted and received on the serial interface is managed through the TXR/RXR data registers.

USR Register

The USR register is the status register for the UART, which can be read by the program to determine the present status of the UART. All flags within the USR register are read only.



Further explanation on each of the flags is given below:

- TXIF**
 The TXIF flag is the transmit data register empty flag. When this read only flag is "0", it indicates that the character is not transferred to the transmit shift registers. When the flag is "1", it indicates that the transmit shift register has received a character from the TXR data register. The TXIF flag is cleared by reading the UART status register (USR) with TXIF set and then writing to the TXR data register. Note that when the TXEN bit is set, the TXIF flag bit will also be set since the transmit buffer is not yet full.
- TIDLE**
 The TIDLE flag is known as the transmission complete flag. When this read only flag is "0", it indicates that a transmission is in progress. This flag will be set to "1" when the TXIF flag is "1" and when there is no transmit data, or break character being transmitted. When TIDLE is "1", the TX pin becomes idle. The TIDLE flag is cleared by reading the USR register with TIDLE set and then writing to the TXR register. The flag is not generated when a data character, or a break is queued and ready to be sent.
- RXIF**
 The RXIF flag is the receive register status flag. When this read only flag is "0", it indicates that the RXR read data register is empty. When the flag is "1", it indicates that the RXR read data register contains new data. When the contents of the shift register are transferred to the RXR register, an interrupt is generated if RIE=1 in the UCR2 register. If one or more errors are detected in the received word, the appropriate receive-related flag(s) NF, FERR, and/or PERR are set within the

same clock cycle. The RXIF flag is cleared when the USR register is read with RXIF set, followed by a read from the RXR register, and if the RXR register has no data available.

- **RIDLE**

The RIDLE flag is the receiver status flag. When this read only flag is "0", it indicates that the receiver is between the initial detection of the start bit and the completion of the stop bit. When the flag is "1", it indicates that the receiver is idle. Between the completion of the stop bit and the detection of the next start bit, the RIDLE bit is "1", indicating that the UART is idle.

- **OERR**

The OERR flag is the overrun error flag, which indicates when the receiver buffer has overflowed. When this read only flag is "0" there is no overrun error. When the flag is "1", an overrun error occurs which will inhibit further transfers to the RXR receive data register. The flag is cleared by a software sequence, which is a read to the status register USR followed by an access to the RXR data register.

- **FERR**

The FERR flag is the framing error flag. When this read only flag is "0", it indicates that there's no framing error. When the flag is "1", it indicates that a framing error has been detected for the current character. The flag can also be cleared by a software sequence which will involve a read to the USR status register followed by an access to the RXR data register.

- **NF**

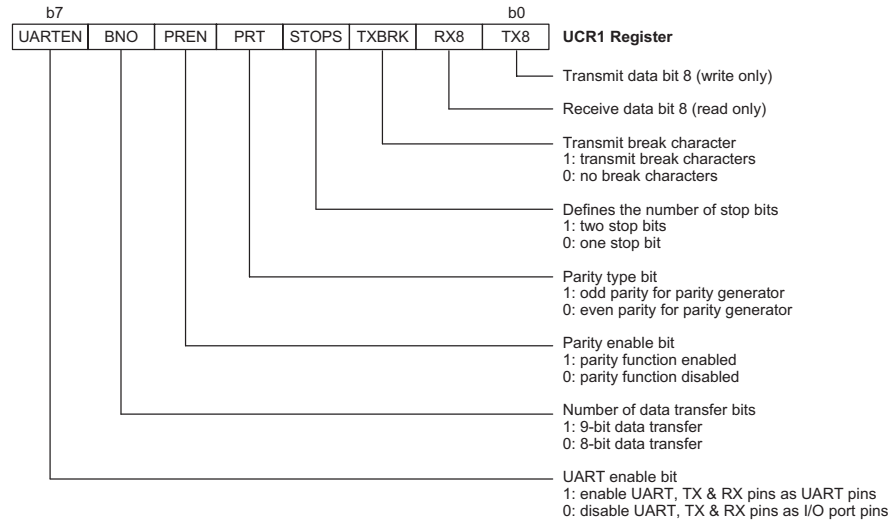
The NF flag is the noise flag. When this read only flag is "0", it indicates a no noise condition. When the flag is "1", it indicates that the UART has detected noise on the receiver input. The NF flag is set during the same cycle as the RXIF flag but will not be set in the case of an overrun. The NF flag can be cleared by a software sequence which will involve a read to the USR status register, followed by an access to the RXR data register.

- **PERR**

The PERR flag is the parity error flag. When this read only flag is "0", it indicates that a parity error has not been detected. When the flag is "1", it indicates that the parity of the received word is incorrect. This error flag is applicable only if Parity mode (odd or even) is selected. The flag can also be cleared by a software sequence which involves a read to the USR status register, followed by an access to the RXR data register.

UCR1 Register

The UCR1 register together with the UCR2 register are the two UART control registers that are used to set the various options for the UART function, such as overall on/off control, parity control, data transfer bit length etc.



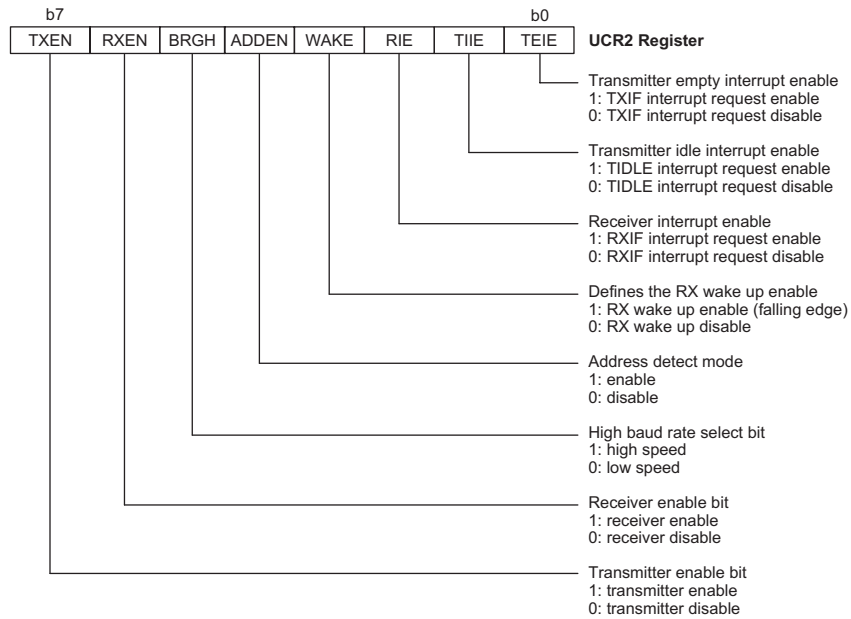
Further explanation on each of the bits is given below:

- **TX8**
This bit is only used if 9-bit data transfers are used, in which case this bit location will store the 9th bit of the transmitted data, known as TX8. The BNO bit is used to determine whether data transfers are in 8-bit or 9-bit format.
- **RX8**
This bit is only used if 9-bit data transfers are used, in which case this bit location will store the 9th bit of the received data, known as RX8. The BNO bit is used to determine whether data transfers are in 8-bit or 9-bit format.
- **TXBRK**
The TXBRK bit is the Transmit Break Character bit. When this bit is "0" there are no break characters and the TX pin operates normally. When the bit is "1" there are transmit break characters and the transmitter will send logic zeros. When equal to "1", after the buffered data has been transmitted, the transmitter output is held low for a minimum of a 13-bit length and until the TXBRK bit is reset.
- **STOPS**
This bit determines if one or two stop bits are to be used. When this bit is equal to "1" two stop bits are used, if the bit is equal to "0" then only one stop bit is used.
- **PRT**
This is the parity type selection bit. When this bit is equal to "1" odd parity will be selected, if the bit is equal to "0" then even parity will be selected.

- **PREN**
This is parity enable bit. When this bit is equal to "1" the parity function will be enabled, if the bit is equal to "0" then the parity function will be disabled.
- **BNO**
This bit is used to select the data length format, which can have a choice of either 8-bits or 9-bits. If this bit is equal to "1" then a 9-bit data length will be selected, if the bit is equal to "0" then an 8-bit data length will be selected. If 9-bit data length is selected then bits RX8 and TX8 will be used to store the 9th bit of the received and transmitted data respectively.
- **UARTEN**
The UARTEN bit is the UART enable bit. When the bit is "0", the UART will be disabled and the RX and TX pins will function as General Purpose I/O pins. When the bit is "1", the UART will be enabled and the TX and RX pins will function as defined by the TXEN and RXEN control bits. When the UART is disabled it will empty the buffer so any character remaining in the buffer will be discarded. In addition, the baud rate counter value will be reset. When the UART is disabled, all error and status flags will be reset. The TXEN, RXEN, TXBRK, RXIF, OERR, FERR, PERR, and NF bits will be cleared, while the TIDLE, TXIF and RIDLE bits will be set. Other control bits in UCR1, UCR2, and BRG registers will remain unaffected. If the UART is active and the UARTEN bit is cleared, all pending transmissions and receptions will be terminated and the module will be reset as defined above. When the UART is re-enabled it will restart in the same configuration.

UCR2 Register

The UCR2 register is the second of the two UART control registers and serves several purposes. One of its main functions is to control the basic enable/disable operation of the UART Transmitter and Receiver as well as enabling the various UART interrupt sources. The register also serves to control the baud rate speed, receiver wake-up enable and the address detect enable.



Further explanation on each of the bits is given below:

- **TEIE**

This bit enables or disables the transmitter empty interrupt. If this bit is equal to "1", when the transmitter empty TXIF flag is set, due to a transmitter empty condition, the UART interrupt request flag will be set. If this bit is equal to "0", the UART interrupt request flag will not be influenced by the condition of the TXIF flag.
- **TIIE**

This bit enables or disables the transmitter idle interrupt. If this bit is equal to "1", when the transmitter idle TIDLE flag is set, the UART interrupt request flag will be set. If this bit is equal to "0", the UART interrupt request flag will not be influenced by the condition of the TIDLE flag.
- **RIE**

This bit enables or disables the receiver interrupt. If this bit is equal to "1", when the receiver overrun OERR flag or receive data available RXIF flag is set, the UART interrupt request flag will be set. If this bit is equal to "0", the UART interrupt will not be influenced by the condition of the OERR or RXIF flags.
- **WAKE**

This bit enables or disables the receiver wake-up function. If this bit is equal to "1", and if the MCU is in the Power Down Mode, a low going edge on the RX input pin will wake-up the device. If this bit is equal to "0", and if the MCU is in the Power Down Mode, any edge transitions on the RX pin will not wake-up the device.
- **ADDEN**

The ADDEN bit is the address detect mode bit. When this bit is "1" the address detect mode is enabled. When this occurs, if the 8th bit, which corresponds to RX7 if BNO=0, or the 9th bit, which corresponds to RX8 if BNO=1, has a value of "1" then the received word will be identified as an address, rather than data. If the corresponding interrupt is enabled, an interrupt request will be generated each time the received word has the address bit set, which is the 8th or 9th bit depending on the value of BNO. If the address bit is "0", an interrupt will not be generated, and the received data will be discarded.
- **BRGH**

The BRGH bit selects the high or low speed mode of the Baud Rate Generator. This bit, together with the value placed in the BRG register, controls the Baud Rate of the UART. If this bit is equal to "1", the high speed mode is selected. If the bit is equal to "0" the low speed mode is selected.
- **RXEN**

The RXEN bit is the Receiver Enable Bit. When this bit is equal to "0", the receiver will be disabled with any pending data receptions being aborted. In addition the buffer will be reset. In this situation the RX pin can be used as a general purpose I/O pin. If the RXEN bit is equal to "1", the receiver will be enabled and if the UARTEN bit is equal to "1", the RX pin will be controlled by the UART. Clearing the RXEN bit during a transmission will cause the data reception to be aborted and will reset the receiver. If this occurs, the RX pin can be used as a general purpose I/O pin.
- **TXEN**

The TXEN bit is the Transmitter Enable Bit. When this bit is equal to "0", the transmitter will be disabled with any pending transmissions being aborted. In addition the buffer will be reset. In this situation the TX pin can be used as a general purpose I/O pin. If the TXEN bit is equal to "1", the transmitter will be enabled and if the UARTEN bit is equal to "1", the TX pin will be controlled

by the UART. Clearing the TXEN bit during a transmission will cause the transmission to be aborted and will reset the transmitter. If this occurs, the TX pin can be used as a general purpose I/O pin.

Baud Rate Generator

To setup the speed of the serial data communication, the UART function contains its own dedicated baud rate generator. The baud rate is controlled by its own internal free running 8-bit timer, the period of which is determined by two factors. The first of these is the value placed in the BRG register and the second is the value of the BRGH bit within the UCR2 control register. The BRGH bit decides, if the baud rate generator is to be used in a high speed mode or low speed mode, which in turn determines the formula that is used to calculate the baud rate. The value in the BRG register determines the division factor, N, which is used in the following baud rate calculation formula. Note that N is the decimal value placed in the BRG register and has a range of between 0 and 255.

| UCR2 BRGH Bit | 0 | 1 |
|---------------|-------------------------------------|-------------------------------------|
| Baud Rate | $\frac{f_{\text{SYS}}}{[64 (N+1)]}$ | $\frac{f_{\text{SYS}}}{[16 (N+1)]}$ |

By programming the BRGH bit which allows selection of the related formula and programming the required value in the BRG register, the required baud rate can be setup. Note that because the actual baud rate is determined using a discrete value, N, placed in the BRG register, there will be an error associated between the actual and requested value. The following example shows how the BRG register value N and the error value can be calculated.

Calculating the Register and Error Values

For a clock frequency of 8MHz, and with BRGH set to "0", determine the BRG register value N, the actual baud rate and the error value for a desired baud rate of 9600.

From the above table the desired baud rate $BR = \frac{f_{\text{SYS}}}{[64 (N+1)]}$

Re-arranging this equation gives $N = \frac{f_{\text{SYS}}}{(BR \times 64)} - 1$

Giving a value for N = $\frac{8000000}{(9600 \times 64)} - 1 = 12.0208$

To obtain the closest value, a decimal value of 12 should be placed into the BRG register. This gives an actual or calculated baud rate value of

$$BR = \frac{8000000}{[64(12+1)]} = 9615$$

Therefore the error is equal to $\frac{9615 - 9600}{9600} = 0.16\%$

The following tables show actual values of baud rate and error values for the two values of BRGH.

| Baud Rate Kbps | Baud Rates for BRGH = 0 | | | | | | | | | | | |
|----------------|-------------------------|--------|-------|-----------------------------|--------|-------|-------------------------|-------|-------|--------------------------------|--------|-------|
| | f _{sys} = 8MHz | | | f _{sys} = 7.159MHz | | | f _{sys} = 4MHz | | | f _{sys} = 3.579545MHz | | |
| | BRG | Kbaud | Error | BRG | Kbaud | Error | BRG | Kbaud | Error | BRG | Kbaud | Error |
| 0.3 | — | — | — | — | — | — | 207 | 0.300 | 0.00 | 185 | 0.300 | 0.00 |
| 1.2 | 103 | 1.202 | 0.16 | 92 | 1.203 | 0.23 | 51 | 1.202 | 0.16 | 46 | 1.19 | -0.83 |
| 2.4 | 51 | 2.404 | 0.16 | 46 | 2.38 | -0.83 | 25 | 2.404 | 0.16 | 22 | 2.432 | 1.32 |
| 4.8 | 25 | 4.807 | 0.16 | 22 | 4.863 | 1.32 | 12 | 4.808 | 0.16 | 11 | 4.661 | -2.9 |
| 9.6 | 12 | 9.615 | 0.16 | 11 | 9.322 | -2.9 | 6 | 8.929 | -6.99 | 5 | 9.321 | -2.9 |
| 19.2 | 6 | 17.857 | -6.99 | 5 | 16.64 | -2.9 | 2 | 20.83 | 8.51 | 2 | 18.643 | -2.9 |
| 38.4 | 2 | 41.667 | 8.51 | 2 | 37.29 | -2.9 | 1 | — | — | 1 | — | — |
| 57.6 | 1 | 62.5 | 8.51 | 1 | 55.93 | -2.9 | 0 | 62.5 | 8.51 | 0 | 55.93 | -2.9 |
| 115.2 | 0 | 125 | 8.51 | 0 | 111.86 | -2.9 | — | — | — | — | — | — |

Baud Rates and Error Values for BRGH = 0

| Baud Rate Kbps | Baud Rates for BRGH = 1 | | | | | | | | | | | |
|----------------|-------------------------|--------|-------|-----------------------------|--------|--------|-------------------------|--------|-------|--------------------------------|--------|-------|
| | f _{sys} = 8MHz | | | f _{sys} = 7.159MHz | | | f _{sys} = 4MHz | | | f _{sys} = 3.579545MHz | | |
| | BRG | Kbaud | Error | BRG | Kbaud | Error | BRG | Kbaud | Error | BRG | Kbaud | Error |
| 0.3 | — | — | — | — | — | — | — | — | — | — | — | — |
| 1.2 | — | — | — | — | — | — | 207 | 1.202 | 0.16 | 185 | 1.203 | 0.23 |
| 2.4 | 207 | 2.404 | 0.16 | 185 | 2.405 | 0.23 | 103 | 2.404 | 0.16 | 92 | 2.406 | 0.23 |
| 4.8 | 103 | 4.808 | 0.16 | 92 | 4.811 | 0.23 | 51 | 4.808 | 0.16 | 46 | 4.76 | -0.83 |
| 9.6 | 51 | 9.615 | 0.16 | 46 | 9.520 | -0.832 | 25 | 9.615 | 0.16 | 22 | 9.727 | 1.32 |
| 19.2 | 25 | 19.231 | 0.16 | 22 | 19.454 | 1.32 | 12 | 19.231 | 0.16 | 11 | 18.643 | -2.9 |
| 38.4 | 12 | 38.462 | 0.16 | 11 | 37.287 | -2.9 | 6 | 35.714 | -6.99 | 5 | 37.286 | -2.9 |
| 57.6 | 8 | 55.556 | -3.55 | 7 | 55.93 | -2.9 | 3 | 62.5 | 8.51 | 3 | 55.930 | -2.9 |
| 115.2 | 3 | 125 | 8.51 | 3 | 111.86 | -2.9 | 1 | 125 | 8.51 | 1 | 111.86 | -2.9 |
| 250 | 1 | 250 | 0 | — | — | — | 0 | 250 | 0 | — | — | — |

Baud Rates and Error Values for BRGH = 1

Setting Up and Controlling the UART

Introduction

For data transfer, the UART function utilizes a non-return-to-zero, more commonly known as NRZ, format. This is composed of one start bit, eight or nine data bits, and one or two stop bits. Parity is supported by the UART hardware, and can be setup to be even, odd or no parity. For the most common data format, 8 data bits along with no parity and one stop bit, denoted as 8, N, 1, is used as the default setting, which is the setting at power-on. The number of data bits and stop bits, along with the parity, are setup by programming the corresponding BNO, PRT, PREN, and STOPS bits in the UCR1 register. The baud rate used to transmit and receive data is setup using the internal 8-bit baud rate generator, while the data is transmitted and received LSB first. Although the UART's transmitter and receiver are functionally independent, they both use the same data format and baud rate. In all cases stop bits will be used for data transmission.

Enabling/Disabling the UART

The basic on/off function of the internal UART function is controlled using the UARTEN bit in the UCR1 register. As the UART transmit and receive pins, TX and RX respectively, are pin-shared with normal I/O pins, one of the basic functions of the UARTEN control bit is to control the UART function of these two pins. If the UARTEN, TXEN and RXEN bits are set, then these two I/O pins will be setup as a TX output pin and an RX input pin respectively, in effect disabling the normal I/O pin function. If no data is being transmitted on the TX pin then it will default to a logic high value.

Clearing the UARTEN bit will disable the TX and RX pins and allow these two pins to be used as normal I/O pins. When the UART function is disabled the buffer will be reset to an empty condition, at the same time discarding any remaining residual data. Disabling the UART will also reset the error and status flags with bits TXEN, RXEN, TXBRK, RXIF, OERR, FERR, PERR and NF being cleared while bits TIDLE, TXIF and RIDLE will be set. The remaining control bits in the UCR1, UCR2 and BRG registers will remain unaffected. If the UARTEN bit in the UCR1 register is cleared while the UART is active, then all pending transmissions and receptions will be immediately suspended and the UART will be reset to a condition as defined above. If the UART is then subsequently re-enabled, it will restart again in the same configuration.

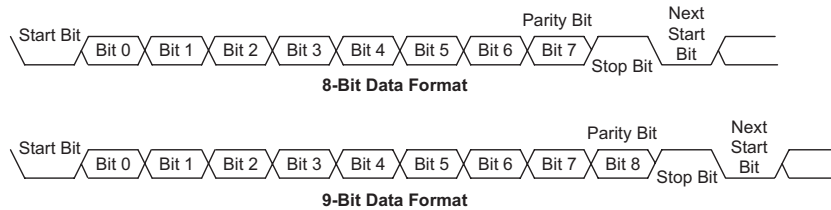
Data, Parity and Stop Bit Selection

The format of the data to be transferred, is composed of various factors such as data bit length, parity on/off, parity type, address bits and the number of stop bits. These factors are determined by the setup of various bits within the UCR1 register. The BNO bit controls the number of data bits which can be set to either 8 or 9, the PRT bit controls the choice of odd or even parity, the PREN bit controls the parity on/off function and the STOPS bit decides whether one or two stop bits are to be used. The following table shows various formats for data transmission. The address bit identifies the frame as an address character. The number of stop bits, which can be either one or two, is independent of the data length.

| Start Bit | Data Bits | Address Bits | Parity Bits | Stops Bit |
|--------------------------------------|-----------|--------------|-------------|-----------|
| Example of 8-bit Data Formats | | | | |
| 1 | 8 | 0 | 0 | 1 |
| 1 | 7 | 0 | 1 | 1 |
| 1 | 7 | 1 | 0 | 1 |
| Example of 9-bit Data Formats | | | | |
| 1 | 9 | 0 | 0 | 1 |
| 1 | 8 | 0 | 1 | 1 |
| 1 | 8 | 1 | 0 | 1 |

Transmitter Receiver Data Format

The following diagram shows the transmit and receive waveforms for both 8-bit and 9-bit data formats.



UART Transmitter

Data word lengths of either 8 or 9 bits, can be selected by programming the BNO bit in the UCR1 register. When BNO bit is set, the word length will be set to 9 bits. In this case the 9th bit, which is the MSB, needs to be stored in the TX8 bit in the UCR1 register. At the transmitter core lies the Transmitter Shift Register, more commonly known as the TSR, whose data is obtained from the transmit data register, which is known as the TXR register. The data to be transmitted is loaded into this TXR register by the application program. The TSR register is not written to with new data until the stop bit from the previous transmission has been sent out. As soon as this stop bit has been transmitted, the TSR can then be loaded with new data from the TXR register, if it is available. It should be noted that the TSR register, unlike many other registers, is not directly mapped into the Data Memory area and as such is not available to the application program for direct read/write operations. An actual transmission of data will normally be enabled when the TXEN bit is set, but the data will not be transmitted until the TXR register has been loaded with data and the baud rate generator has defined a shift clock source. However, the transmission can also be initiated by first loading data into the TXR register, after which the TXEN bit can be set. When a transmission of data begins, the TSR is normally empty, in which case a transfer to the TXR register will

result in an immediate transfer to the TSR. If during a transmission the TXEN bit is cleared, the transmission will immediately cease and the transmitter will be reset. The TX output pin will then return to having a normal general purpose I/O pin function.

Transmitting Data

When the UART is transmitting data, the data is shifted on the TX pin from the shift register, with the least significant bit first. In the transmit mode, the TXR register forms a buffer between the internal bus and the transmitter shift register. It should be noted that if 9-bit data format has been selected, then the MSB will be taken from the TX8 bit in the UCR1 register. The steps to initiate a data transfer can be summarized as follows:

- Make the correct selection of the BNO, PRT, PREN and STOPS bits to define the required word length, parity type and number of stop bits.
- Setup the BRG register to select the desired baud rate.
- Set the TXEN bit to ensure that the TX pin is used as a UART transmitter pin and not as an I/O pin.
- Access the USR register and write the data that is to be transmitted into the TXR register. Note that this step will clear the TXIF bit.
- This sequence of events can now be repeated to send additional data.

It should be noted that when TXIF=0, data will be inhibited from being written to the TXR register. Clearing the TXIF flag is always achieved using the following software sequence:

1. A USR register access
2. A TXR register write execution

The read-only TXIF flag is set by the UART hardware and if set indicates that the TXR register is empty and that other data can now be written into the TXR register without overwriting the previous data. If the TEIE bit is set then the TXIF flag will generate an interrupt.

During a data transmission, a write instruction to the TXR register will place the data into the TXR register, which will be copied to the shift register at the end of the present transmission. When there is no data transmission in progress, a write instruction to the TXR register will place the data directly into the shift register, resulting in the commencement of data transmission, and the TXIF bit being immediately set. When a frame transmission is complete, which happens after stop bits are sent or after the break frame, the TIDLE bit will be set. To clear the TIDLE bit the following software sequence is used:

1. A USR register access
2. A TXR register write execution

Note that both the TXIF and TIDLE bits are cleared by the same software sequence.

Transmit Break

If the TXBRK bit is set then break characters will be sent on the next transmission. Break character transmission consists of a start bit, followed by $13 \times N$ '0' bits and stop bits, where $N=1, 2$, etc. If a break character is to be transmitted then the TXBRK bit must be first set by the application program, then cleared to generate the stop bits. Transmitting a break character will not generate a transmit interrupt. Note that a break condition length is at least 13 bits long. If the TXBRK bit is continually kept at a logic high level then the transmitter circuitry will transmit continuous break characters. After the application program has cleared the TXBRK bit, the transmitter will finish transmitting the last break character and subsequently send out one or two stop bits. The automatic logic highs at the end of the last break character will ensure that the start bit of the next frame is recognized.

UART Receiver

Introduction

The UART is capable of receiving word lengths of either 8 or 9 bits. If the BNO bit is set, the word length will be set to 9 bits with the MSB being stored in the RX8 bit of the UCR1 register. At the receiver core lies the Receive Serial Shift Register, commonly known as the RSR. The data which is received on the RX external input pin, is sent to the data recovery block. The data recovery block operating speed is 16 times that of the baud rate, while the main receive serial shifter operates at the baud rate. After the RX pin is sampled for the stop bit, the received data in RSR is transferred to the receive data register, if the register is empty. The data which is received on the external RX input pin is sampled three times by a majority detect circuit to determine the logic level that has been placed onto the RX pin. It should be noted that the RSR register, unlike many other registers, is not directly mapped into the Data Memory area and as such is not available to the application program for direct read/write operations.

Receiving Data

When the UART receiver is receiving data, the data is serially shifted in on the external RX input pin, LSB first. In the read mode, the RXR register forms a buffer between the internal bus and the receiver shift register. The RXR register is a two byte deep FIFO data buffer, where two bytes can be held in the FIFO while a third byte can continue to be received. Note that the application program must ensure that the data is read from RXR before the third byte has been completely shifted in, otherwise this third byte will be discarded and an overrun error OERR will be subsequently indicated. The steps to initiate a data transfer can be summarized as follows:

- Make the correct selection of BNO, PRT, PREN and STOPS bits to define the word length, parity type and number of stop bits.
- Setup the BRG register to select the desired baud rate.
- Set the RXEN bit to ensure that the RX pin is used as a UART receiver pin and not as an I/O pin.

At this point the receiver will be enabled which will begin to look for a start bit.

When a character is received the following sequence of events will occur:

- The RXIF bit in the USR register will be set when RXR register has data available, at least one more character can be read.
- When the contents of the shift register have been transferred to the RXR register, then if the RIE bit is set, an interrupt will be generated.
- If during reception, a frame error, noise error, parity error, or an overrun error has been detected, then the error flags can be set.

The RXIF bit can be cleared using the following software sequence:

1. A USR register access
2. An RXR register read execution

Receive Break

Any break character received by the UART will be managed as a framing error. The receiver will count and expect a certain number of bit times as specified by the values programmed into the BNO and STOPS bits. If the break is much longer than 13 bit times, the reception will be considered as complete after the number of bit times specified by BNO and STOPS. The RXIF bit is set, FERR is set, zeros are loaded into the receive data register, interrupts are generated if appropriate and the RIDLE bit is set. If a long break signal has been detected and the receiver has received a start bit, the data bits and the invalid stop bit, which sets the FERR flag, the receiver must wait for a valid stop bit before looking for the next start bit. The receiver will not make the assumption that the break condition on the line is the next start bit. A break is regarded as a character that contains only zeros with the FERR flag set. The break character will be loaded into the buffer and no further data will be received until stop bits are received. It should be noted that the RIDLE read only flag will go high when the stop bits have not yet been received. The reception of a break character on the UART registers will result in the following:

- The framing error flag, FERR, will be set.
- The receive data register, RXR, will be cleared.
- The OERR, NF, PERR, RIDLE or RXIF flags will possibly be set.

Idle Status

When the receiver is reading data, which means it will be in between the detection of a start bit and the reading of a stop bit, the receiver status flag in the USR register, otherwise known as the RIDLE flag, will have a zero value. In between the reception of a stop bit and the detection of the next start bit, the RIDLE flag will have a high value, which indicates the receiver is in an idle condition.

Receiver Interrupt

The read only receive interrupt flag RXIF in the USR register is set by an edge generated by the receiver. An interrupt is generated if RIE=1, when a word is transferred from the Receive Shift Register, RSR, to the Receive Data Register, RXR. An overrun error can also generate an interrupt if RIE=1.

Managing Receiver Errors

Several types of reception errors can occur within the UART module, the following section describes the various types and how they are managed by the UART.

Overrun Error – OERR Flag

The RXR register is composed of a two byte deep FIFO data buffer, where two bytes can be held in the FIFO register, while a third byte can continue to be received. Before this third byte has been entirely shifted in, the data should be read from the RXR register. If this is not done, the overrun error flag OERR will be consequently indicated.

In the event of an overrun error occurring, the following will happen:

- The OERR flag in the USR register will be set.
- The RXR contents will not be lost.
- The shift register will be overwritten.
- An interrupt will be generated if the RIE bit is set.

The OERR flag can be cleared by an access to the USR register followed by a read to the RXR register.

Noise Error – NF Flag

Over-sampling is used for data recovery to identify valid incoming data and noise. If noise is detected within a frame the following will occur:

- The read only noise flag, NF, in the USR register will be set on the rising edge of the RXIF bit.
- Data will be transferred from the shift register to the RXR register.
- No interrupt will be generated. However this bit rises at the same time as the RXIF bit which itself generates an interrupt.

Note that the NF flag is reset by a USR register read operation followed by an RXR register read operation.

Framing Error – FERR Flag

The read only framing error flag, FERR, in the USR register, is set if a zero is detected instead of stop bits. If two stop bits are selected, both stop bits must be high, otherwise the FERR flag will be set. The FERR flag is buffered along with the received data and is cleared on any reset.

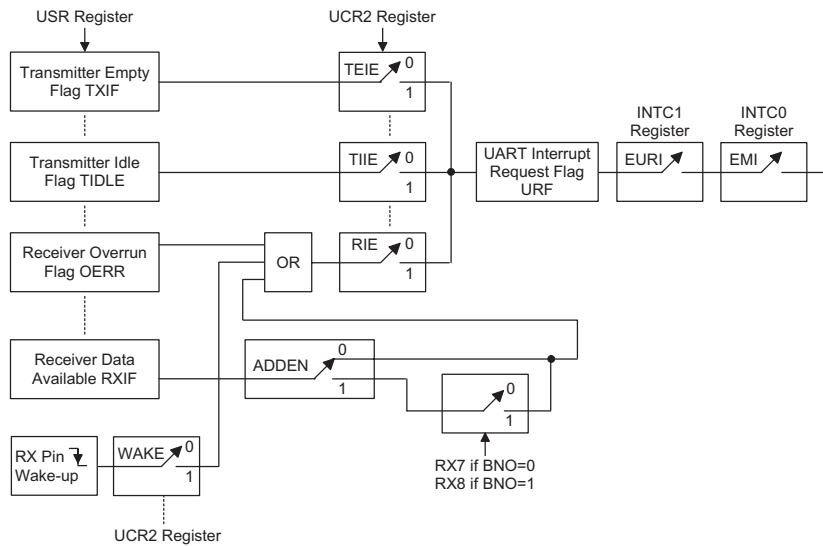
Parity Error – PERR Flag

The read only parity error flag, PERR, in the USR register, is set if the parity of the received word is incorrect. This error flag is only applicable if the parity is enabled, PREN = 1, and if the parity type, odd or even is selected. The read only PERR flag is buffered along with the received data bytes. It is cleared on any reset. It should be noted that the FERR and PERR flags are buffered along with the corresponding word and should be read before reading the data word.

UART Interrupt Scheme

The UART internal function possesses its own internal interrupt and independent interrupt vector. Several individual UART conditions can generate an internal UART interrupt. These conditions are, a transmitter data register empty, transmitter idle, receiver data available, receiver overrun, address detect and an RX pin wake-up. When any of these conditions are created, if the UART interrupt is enabled and the stack is not full, the program will jump to the UART interrupt vector where it can be serviced before returning to the main program. Four of these conditions, have a corresponding USR register flag, which will generate a UART interrupt if its associated interrupt enable flag in the UCR2 register is set. The two transmitter interrupt conditions have their own corresponding enable bits, while the two receiver interrupt conditions have a shared enable bit. These enable bits can be used to mask out individual UART interrupt sources.

The address detect condition, which is also a UART interrupt source, does not have an associated flag, but will generate a UART interrupt when an address detect condition occurs if its function is enabled by setting the ADDEN bit in the UCR2 register. An RX pin wake-up, which is also a UART interrupt source, does not have an associated flag, but will generate a UART interrupt if the microcontroller is woken up by a low going edge on the RX pin, if the WAKE and RIE bits in the UCR2 register are set. Note that in the event of an RX wake-up interrupt occurring, there will be a delay of 1024 system clock cycles before the system resumes normal operation.



UART Interrupt Scheme

Note that the USR register flags are read only and cannot be cleared or set by the application program, neither will they be cleared when the program jumps to the corresponding interrupt servicing routine, as is the case for some of the other interrupts. The flags will be cleared automatically when certain actions are taken by the UART, the details of which are given in the UART register section. The overall UART interrupt can be disabled or enabled by the EURI bit in the INTC1 interrupt control register to prevent a UART interrupt from occurring.

Address Detect Mode

Setting the Address Detect Mode bit, ADDEN, in the UCR2 register, enables this special mode. If this bit is enabled then an additional qualifier will be placed on the generation of a Receiver Data Available Interrupt, which is requested by the RXIF flag. If the ADDEN bit is enabled, then when data is available, an interrupt will only be generated, if the highest received bit has a high value. Note that the EURI and EMI interrupt enable bits must also be enabled for correct interrupt generation. This highest address bit is the 9th bit if BNO=1 or the 8th bit if BNO=0. If this bit is high, then the received word will be defined as an address rather than data. A Data Available Interrupt will be generated every time the last bit of the received word is set. If the ADDEN bit is not enabled, then a Receiver Data Available Interrupt will be generated each time the RXIF flag is set, irrespective of the data last bit status. The address detect mode and parity enable are mutually exclusive functions. Therefore if the address detect mode is enabled, then to ensure correct operation, the parity function should be disabled by resetting the parity enable bit to zero.

| ADDEN | Bit 9 if BNO=1, Bit 8 if BNO=0 | UART Interrupt Generated |
|-------|--------------------------------|--------------------------|
| 0 | 0 | √ |
| | 1 | √ |
| 1 | 0 | X |
| | 1 | √ |

ADDEN Bit Function

UART Operation in Power Down Mode

When the MCU is in the Power Down Mode the UART will cease to function. When the device enters the Power Down Mode, all clock sources to the module are shutdown. If the MCU enters the Power Down Mode while a transmission is still in progress, then the transmission will be terminated and the external TX transmit pin will be forced to a logic high level. In a similar way, if the MCU enters the Power Down Mode while receiving data, then the reception of data will likewise be terminated. When the MCU enters the Power Down Mode, note that the USR, UCR1, UCR2, transmit and receive registers, as well as the BRG register will not be affected.

The UART function contains a receiver RX pin wake-up function, which is enabled or disabled by the WAKE bit in the UCR2 register. If this bit, along with the UART enable bit, UARTEN, the receiver enable bit, RXEN and the receiver interrupt bit, RIE, are all set before the MCU enters the Power Down Mode, then a falling edge on the RX pin will wake-up the MCU from the Power Down Mode. Note that as it takes 1024 system clock cycles after a wake-up, before normal microcontroller operation resumes, any data received during this time on the RX pin will be ignored.

For a UART wake-up interrupt to occur, in addition to the bits for the wake-up being set, the global interrupt enable bit, EMI, and the UART interrupt enable bit, EURI must also be set. If these two bits are not set then only a wake up event will occur and no interrupt will be generated. Note also that as it takes 1024 system clock cycles after a wake-up before normal microcontroller resumes, the UART interrupt will not be generated until after this time has elapsed.

UART Sample Program

The following application program shows how the UART can be used for the transmission and reception of external data:

```

t_uart_TX:
  clr  intc0                ; disable intc0
  clr  intc1                ; disable intc1
  mov  a,80h
  mov  ucrl,a               ; enable uarten
  mov  brg,a                ; set brg=80H
  mov  ucr2,a               ; enable txen
  mov  a,055h
  mov  txr,a                ; set txr=55H
  :
  :
  jmp  t_uart_TX

t_uart_RX:
  clr  intc0                ; disable intc0
  clr  intc1                ; disable intc1
  mov  a,80h
  mov  ucrl,a               ; enable uarten
  mov  brg,a                ; set brg=80H
  mov  a,40h
  mov  ucr2,a               ; enable rxen
  mov  a,rxr
  mov  pa,a                 ; pa=rxr
  :
  :
  jmp  t_uart_RX

```

Oscillator

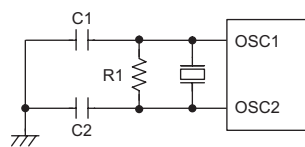
Various oscillator options offer the user a wide range of functions according to their various application requirements. Three types of system clocks can be selected while various clock source options for the Watchdog Timer as well as a real time clock function are provided for maximum flexibility. All oscillator options are selected through the configuration options.

System Clock Configurations

There are three methods of generating the system clock, using an external crystal/ceramic oscillator, an external RC network or using the internal RC clock source. The chosen method is selected through the configuration options.

System Crystal/Ceramic Oscillator

For most crystal oscillator configurations, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation. However, to ensure oscillation for certain lower crystal frequencies and for all ceramic resonator applications, it is recommended that two small value capacitors and a resistor, the values of which are shown in the table, should be connected as shown in the diagram.



Crystal/Ceramic Oscillator

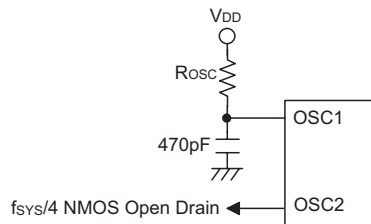
The table below shows the C1, C2 and R1 values for various crystal/ceramic oscillating frequencies.

| Crystal or Resonator | C1, C2 | R1 |
|----------------------------|--------|-------|
| 4MHz Crystal | 0pF | 10kΩ |
| 4MHz Resonator | 10pF | 12kΩ |
| 3.58MHz Crystal | 0pF | 10kΩ |
| 3.58MHz Resonator | 25pF | 10kΩ |
| 2MHz Crystal and Resonator | 25pF | 10kΩ |
| 1MHz Crystal | 35pF | 27kΩ |
| 480kHz Resonator | 300pF | 9.1kΩ |
| 455kHz Resonator | 300pF | 10kΩ |
| 429kHz Resonator | 300pF | 10kΩ |

The function of the resistor R1 is to ensure that the oscillator will switch off should low voltage conditions occur. Such a low voltage, as mentioned here, is one which is less than the lowest value of the MCU operating voltage. Note however that if the LVR is enabled then R1 can be removed.

System RC Oscillator

Using the external RC network as an oscillator requires that a resistor, with a value between 24kΩ and 1MΩ, is connected between OSC1 and VDD, and a 470pF capacitor is connected to ground. The generated system clock divided by 4 will be provided on OSC2 as an output which can be used for external synchronization purposes. Although this is a cost effective oscillator configuration, the oscillation frequency can vary with VDD, temperature and process variations on the chip itself and is therefore not suitable for applications where timing is critical or where accurate oscillator frequencies are required. For the value of the external resistor R_{OSC} please refer to the Appendix section for typical RC Oscillator vs. Temperature and V_{DD} characteristics graphics.

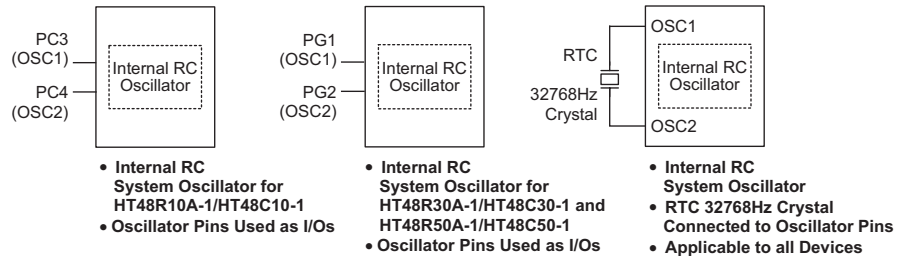


RC Oscillator

Internal System RC Oscillator

In addition to the external crystal/resonator or external RC system clock configurations, the devices also contain an internal RC system clock. This oscillator is integral to the microcontroller and requires no external components. The frequency of this internal RC oscillator can be selected by configuration option to have a typical value at 5V of either 3.2MHz, 1.6MHz, 800kHz or 400kHz. The 1.6MHz, 800kHz and 400kHz frequencies are internally generated by sequentially dividing the base 3.2MHz frequency by two. Note that if this internal system clock option is selected then with the exception of the HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80 devices, the

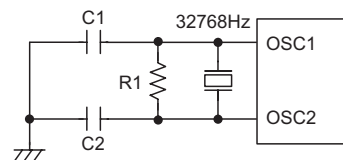
OSC1 and OSC2 pins are free for use as normal I/O pins. The OSC1 and OSC2 pins can also be connected to a 32768Hz crystal for use as an RTC oscillator on all devices. If the RTC oscillator is to be used in user applications, the internal RC oscillator must be used as the system clock. Note that the oscillation frequency of this internal system RC oscillator can vary with VDD, temperature and process variations.



RTC Oscillator

When microcontrollers enter a power down or HALT condition, the system clock is switched off to stop microcontroller activity and to conserve power. However, in many microcontroller applications it may be necessary to keep the internal timers operational even when the microcontroller is in a HALT state. However, to do this, another clock, independent of the system clock, must be provided. To provide this feature, all of Holtek's I/O range of microcontrollers incorporate a Real Time Clock or RTC. This clock source has a fixed frequency of 32768Hz and requires a 32768Hz crystal to be connected between pins OSC1 and OSC2. For applications using the RTC oscillator, the internal RC Oscillator must be used as the system clock. Oscillator configuration options determine if the RTC is to be used. If the RTC oscillator configuration option is selected, then the timer(s) have the configuration option of selecting either the internal RC system clock or RTC as their clock source.

The RTC, if selected as the clock source for the timers, allows the timer functions to remain active even if the microcontroller is in the Power Down Mode and as such will issue the usual internal interrupt signal when the counter is full. This signal will cause the microcontroller to wake-up from its HALT state and continue with normal operation until the next "HALT" instruction is executed.



RTC Oscillator

Note The external resistor and capacitor components connected to the 32768Hz crystal are not necessary to provide oscillation. For applications where precise RTC frequencies are essential, these components may be required to provide frequency compensation due to different crystal manufacturing tolerances. In some applications only capacitor C1 is required.

Watchdog Timer Oscillator

The WDT oscillator is a fully self-contained free running on-chip RC oscillator with a typical period of 65 μ s at 5V requiring no external components. When the device enters the Power Down Mode, the system clock will stop running but the WDT oscillator continues to free-run and to keep the watchdog active. However, to preserve power in certain applications the WDT oscillator can be disabled via a configuration option.

Power Down Mode and Wake-up

Power Down Mode

All of the Holtek microcontrollers have the ability to enter a Power Down Mode, also known as the HALT Mode or Sleep Mode. When the device enters this mode, the normal operating current, will be reduced to an extremely low standby current level. This occurs because when the device enters the Power Down Mode, the system oscillator is stopped which reduces the power consumption to extremely low levels, however, as the device maintains its present internal condition, it can be woken up at a later stage and continue running, without requiring a full reset. This feature is extremely important in application areas where the MCU must have its power supply constantly maintained to keep the device in a known condition but where the power supply capacity is limited such as in battery applications.

Entering the Power Down Mode

There is only one way for the device to enter the Power Down Mode and that is to execute the "HALT" instruction in the application program. When this instruction is executed, the following will occur:

- The system oscillator will stop running and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The WDT will be cleared and resume counting if the WDT clock source is selected to come from the WDT oscillator. The WDT will stop if its clock source originates from the system clock.
- The I/O ports will maintain their present condition.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.

Standby Current Considerations

As the main reason for entering the Power Down Mode is to keep the current consumption of the MCU to as low a value as possible, perhaps only in the order of several micro-amps, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimized. Special attention must be made to the I/O pins on the device. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. Care must also be taken with the loads which are connected to I/Os which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current such as other CMOS inputs. Also note that additional standby current will also be required if the configuration options are setup to keep the various Watchdog, RTC oscillator functions active during the Power Down Mode.

Wake-up

After the system enters the Power Down Mode, it can be woken up from one of various sources listed as follows:

- An external reset
- An external falling edge on Port A
- A system interrupt
- A WDT overflow
- An external falling edge on the UART RX pin

If the system is woken up by an external reset, the device will experience a full system reset, however, if the device is woken up by a WDT overflow, a Watchdog Timer reset will be initiated. Although both of these wake-up methods will initiate a reset operation, the actual source of the wake-up can be determined by examining the TO and PDF flags. The PDF flag is cleared by a system power-up or executing the clear Watchdog Timer instructions and is set when executing the HALT instruction. The TO flag is set if a WDT time-out occurs, and causes a wake-up that only resets the Program Counter and Stack Pointer, the other flags remain in their original status

If the system is woken up due to a low going edge on any wake-up configured Port A input lines or in the case of the HT48RU80/HT48CU80 devices, a low going edge on the UART RX pin, then the program will continue execution from the statement following the "HALT" instruction. The contents of the Data Memory and all register values will remain unchanged from the value at the time of the "HALT" instruction execution and the program will continue running normally. Note that each line on Port A can be independently selected to wake-up the device by a configuration option. For the UART RX pin wake-up to be operational, the UART receiver and its corresponding wake-up function must be enabled by setting the appropriate bits in the UART control registers.

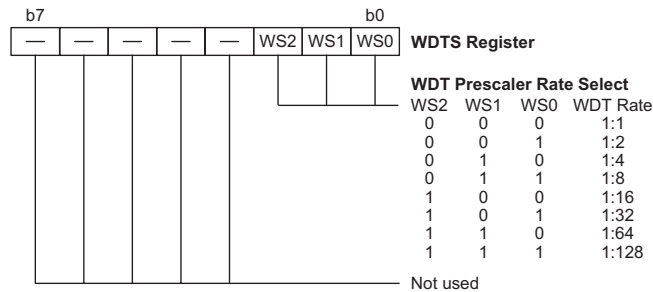
If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt which woke-up the device will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set to 1 before entering the Power Down Mode, the wake-up function of the related interrupt will be disabled.

No matter what the source of the wake-up event is, once a wake-up situation occurs, a time period equal to 1024 system clock periods will be required before normal system operation resumes. However, if the wake-up has originated due to an interrupt, the actual interrupt subroutine execution will be delayed by an additional one or more cycles. If the wake-up results in the execution of the next instruction following the "HALT" instruction, this will be executed immediately after the 1024 system clock period delay has ended.

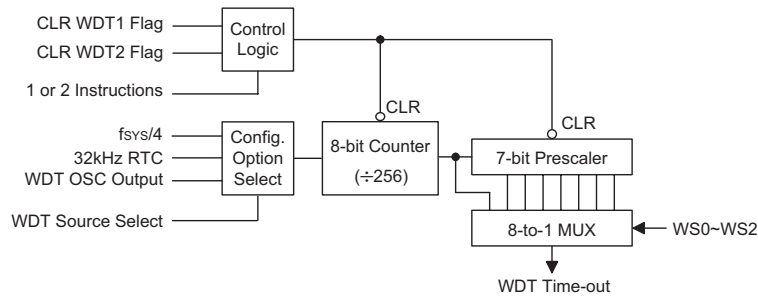
Watchdog Timer

The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a "chip reset" when the WDT counter overflows. The WDT clock is supplied by one of three sources selected by configuration option: its own self contained dedicated internal WDT oscillator, the instruction clock (system clock divided by 4) or the 32kHz RTC oscillator. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

The internal WDT oscillator has an approximate period of 65µs at a supply voltage of 5V. If selected, it is first divided by 256 via an 8-stage counter to give a nominal period of 18ms. Note that this period can vary with VDD, temperature and process variations. For longer WDT time-out periods the WDT prescaler can be utilized. By writing the required value to bits 0, 1 and 2 of the WDTS register, known as WS0, WS1 and WS2, longer time-out periods can be achieved. With WS0, WS1 and WS2 all equal to 1, the division ratio is 1:128 which gives a maximum time-out period of about 2.1s. The high nibble and bit 3 of the WDTS are reserved for user defined flags, which can be used to indicate some specified status.



The WDT oscillator can be disabled and the WDT clock source can be supplied from the instruction clock (system clock divided by 4). If the instruction clock is used as the clock source it should be noted that when the system enters the Power Down Mode, then the instruction clock is stopped and the WDT will lose its protecting purposes. In such cases the system can only be restarted via external logic. For systems that operate in noisy environments, using the internal WDT oscillator or the 32kHz RTC oscillator is strongly recommended.



Watchdog Timer

Under normal program operation, the WDT time-out will initialize a "chip reset" and set the status bit "TO". However, if the system is in the Power Down Mode, only a WDT time-out reset from "HALT" will be initialized which will only reset the Program Counter and Stack Pointer. Three methods can be adopted to clear the contents of the WDT including the WDT prescaler. The first is an external hardware reset (a low level on the RES pin), the second is via software instructions and the third is via a "HALT" instruction. There are two methods of using software instructions to clear the Watchdog Timer, one of which must be chosen by configuration option. The first option is to use the single "CLR WDT" instruction while the second is to use the two commands "CLR WDT1" and "CLR WDT2". For the first option, a simple execution of "CLR WDT" will clear the WDT while for the second option, both "CLR WDT1" and "CLR WDT2" must both be executed to successfully clear the WDT. Note that for this second option, if "CLR WDT1" is used to clear the WDT, successive executions of this instruction will have no effect, only the execution of a "CLR WDT2" instruction will clear the WDT. Similarly, after the "CLR WDT2" instruction has been executed, only a successive "CLR WDT1" instruction can clear the Watchdog Timer.

Configuration Options

Configuration options refer to certain options within the MCU that are programmed into the device during the programming process. During the development process, these options are selected using the HT-IDE software development tools. As these options are programmed into the device using the hardware programming tools, once they are selected they cannot be changed later as the application software has no control over the configuration options. Although functionally similar, the OTP and Mask type devices differ in the way that their configuration options are setup. For OTP devices these options can be selected using the HT-IDE development tools, and can therefore be re-configured before the device is programmed, however for Mask level devices the options are programmed into the device during the manufacturing stage.

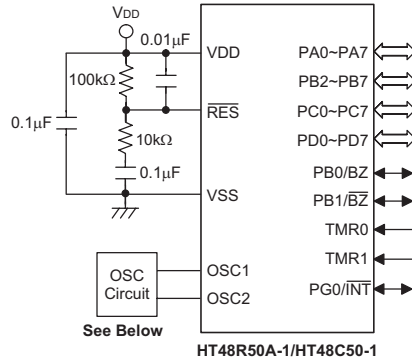
All options must be defined for proper system function, the details of which are shown in the table.

| No. | Option |
|-----|--|
| 1 | WDT clock source: WDT oscillator or $f_{SYS}/4$ or RTC oscillator or disable |
| 2 | CLRWDT instructions: 1 or 2 instructions |
| 3 | Timer/Event Counter clock source: f_{SYS} or RTC oscillator (HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1) |
| 4 | Timer/Event Counter 0 clock source: f_{SYS} or RTC oscillator (HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1) |
| 5 | Timer/Event Counter 0 clock source: $f_{SYS}/4$ or RTC oscillator (HT48RU80/HT48CU80) |
| 6 | Timer/Event Counter 1 clock source: $f_{SYS}/4$ or RTC oscillator (HT48R50A-1/HT48C50-1, HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80) |
| 7 | Timer/Event Counter 2 clock source: f_{SYS} or RTC oscillator (HT48RU80/HT48CU80) |
| 8 | PA0~PA7 wake-up: enable or disable |
| 9 | PA Schmitt Trigger or non Schmitt Trigger |
| 10 | PA, PB, PC, PD, PE, PF and PG pull-high enable or disable (port numbers are device dependent) |
| 11 | Buzzer function: single BZ enable or both BZ and \overline{BZ} enable or both disable Buzzer clock source: timer 0 or timer 1 (HT48R50A-1/HT48C50-1, HT48RU80/HT48CU80) |

| No. | Option |
|-----|---|
| 12 | LVR function: enable or disable |
| 13 | External RC system clock/External Crystal System Clock/Internal RC system clock plus RTC clock/Internal RC system plus I/O (last option not applicable to HT48R70A-1/HT48C70-1 and HT48RU80/HT48CU80) |
| 14 | Internal RC frequency selection 3.2MHz, 1.6MHz, 800kHz or 400kHz. |

Application Circuits

The following application circuits although based around the HT48R50A-1 device equally apply to all devices in the I/O type range.



| | |
|--|---|
| | <p>RC System Oscillator $24k\Omega < R_{osc} < 1M\Omega$</p> |
| | <p>Crystal System Oscillator For component values, consult Oscillator section</p> |
| | <p>Internal RC Oscillator OSC1 and OSC2 left unconnected</p> |
| | <p>Internal RC Oscillator with RTC</p> |

OSC Circuit

Part II

Programming Language

Chapter 2**Instruction Set Introduction****2****Instruction Set**

Central to the successful operation of any microcontroller is its instruction set, which is a set of program instruction codes that directs the microcontroller to perform certain operations. In the case of Holtek microcontrollers, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

For easier understanding of the various instruction codes, they have been subdivided into several functional groupings.

Instruction Timing

Most instructions are implemented within one instruction cycle. The exceptions to this are branch, call, or table read instructions where two instruction cycles are required. One instruction cycle is equal to 4 system clock cycles, therefore in the case of an 8MHz system oscillator, most instructions would be implemented within 0.5 μ s and branch or call instructions would be implemented within 1 μ s. Although instructions which require one more cycle to implement are generally limited to the JMP, CALL, RET, RETI and table read instructions, it is important to realize that any other instructions which involve manipulation of the Program Counter Low register or PCL will also take one more cycle to implement. As instructions which change the contents of the PCL will imply a direct jump to that new address, one more cycle will be required. Examples of such instructions would be "CLR PCL" or "MOV PCL, A". For the case of skip instructions, it must be noted that if the result of the comparison involves a skip operation then this will also take one more cycle, if no skip is involved then only one cycle is required.

Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of three kinds of MOV instructions, data can be transferred from registers to the Accumulator and vice-versa as well as being able to move specific immediate data directly into the Accumulator. One of the most important data transfer applications is to receive data from the input ports and transfer data to the output ports.

Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of Carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

Logical and Rotate Operations

The standard logical operations such as AND, OR, XOR and CPL all have their own instruction within the Holtek microcontroller instruction set. As with the case of most instructions involving data manipulation, data must pass through the Accumulator which may involve additional programming steps. In all logical data operations, the zero flag may be set if the result of the operation is zero. Another form of logical data manipulation comes from the rotate instructions such as RR, RL, RRC and RLC which provide a simple means of rotating one bit right or left. Different rotate instructions exist depending on program requirements. Rotate instructions are useful for serial port programming applications where data can be rotated from an internal register into the Carry bit from where it can be examined and the necessary serial bit set high or low. Another application where rotate data operations are used is to implement multiplication and division calculations.

Branches and Control Transfer

Program branching takes the form of either jumps to specified locations using the JMP instruction or to a subroutine using the CALL instruction. They differ in the sense that in the case of a subroutine call, the program must return to the instruction immediately when the subroutine has been carried out. This is done by placing a return instruction RET in the subroutine which will cause the program to jump back to the address right after the CALL instruction. In the case of a JMP instruction, the program simply jumps to the desired location. There is no requirement to jump back to the original jumping off point as in the case of the CALL. One special and extremely useful set of branch instructions are the conditional branches. Here a decision is first made regarding the condition of a certain Data Memory or individual bits. Depending upon the conditions, the program will continue with the next instruction or skip over it and jump to the following instruction. These instructions are the key to decision making and branching within the program perhaps determined by the condition of certain input switches or by the condition of internal data bits.

Bit Operations

The ability to provide single bit operations on Data Memory is an extremely flexible feature of all Holtek microcontrollers. This feature is especially useful for output port bit programming where individual bits or port pins can be directly set high or low using either the "SET [m].i" or "CLR [m].i" instructions respectively. The feature removes the need for programmers to first read the 8-bit output port, manipulate the input data to ensure that other bits are not changed and then output the port with the correct new data. This read-modify-write process is taken care of automatically when these bit operation instructions are used.

Table Read Operations

Data storage is normally implemented by using registers. However, when working with large amounts of fixed data, the volume involved often makes it inconvenient to store the fixed data in individual memory. To overcome this problem, Holtek microcontrollers allow an area of Program Memory to be setup as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

Other Operations

In addition to the above functional instructions, a range of other instructions also exist such as "HALT" instruction for power-down operation and instructions to control the operation of the Watchdog Timer for reliable program operations under extreme electric or electromagnetic environment. For their relevant operations, refer to the functional related sections.

Instruction Set Summary

Convention

x: Bits immediate data
 m: Data Memory address
 A: Accumulator
 i: 0~7 number of bits
 addr: Program Memory address

| Mnemonic | Description | Cycles | Flag Affected |
|-------------------|---|-------------------|---------------|
| Arithmetic | | | |
| ADD A,[m] | Add Data Memory to ACC | 1 | Z, C, AC, OV |
| ADDM A,[m] | Add ACC to Data Memory | 1 ^{Note} | Z, C, AC, OV |
| ADD A,x | Add immediate data to ACC | 1 | Z, C, AC, OV |
| ADC A,[m] | Add Data Memory to ACC with Carry | 1 | Z, C, AC, OV |
| ADCM A,[m] | Add ACC to Data Memory with Carry | 1 ^{Note} | Z, C, AC, OV |
| SUB A,x | Subtract immediate data from the ACC | 1 | Z, C, AC, OV |
| SUB A,[m] | Subtract Data Memory from ACC | 1 | Z, C, AC, OV |
| SUBM A,[m] | Subtract Data Memory from ACC with result in Data Memory | 1 ^{Note} | Z, C, AC, OV |
| SBC A,[m] | Subtract Data Memory from ACC with Carry | 1 | Z, C, AC, OV |
| SBCM A,[m] | Subtract Data Memory from ACC with Carry, result in Data Memory | 1 ^{Note} | Z, C, AC, OV |
| DAA [m] | Decimal adjust ACC for Addition with result in Data Memory | 1 ^{Note} | C |

| Mnemonic | Description | Cycles | Flag Affected |
|----------------------------------|---|-------------------|---------------|
| Logic Operation | | | |
| AND A,[m] | Logical AND Data Memory to ACC | 1 | Z |
| OR A,[m] | Logical OR Data Memory to ACC | 1 | Z |
| XOR A,[m] | Logical XOR Data Memory to ACC | 1 | Z |
| ANDM A,[m] | Logical AND ACC to Data Memory | 1 ^{Note} | Z |
| ORM A,[m] | Logical OR ACC to Data Memory | 1 ^{Note} | Z |
| XORM A,[m] | Logical XOR ACC to Data Memory | 1 ^{Note} | Z |
| AND A,x | Logical AND immediate Data to ACC | 1 | Z |
| OR A,x | Logical OR immediate Data to ACC | 1 | Z |
| XOR A,x | Logical XOR immediate Data to ACC | 1 | Z |
| CPL [m] | Complement Data Memory | 1 ^{Note} | Z |
| CPLA [m] | Complement Data Memory with result in ACC | 1 | Z |
| Increment & Decrement | | | |
| INCA [m] | Increment Data Memory with result in ACC | 1 | Z |
| INC [m] | Increment Data Memory | 1 ^{Note} | Z |
| DECA [m] | Decrement Data Memory with result in ACC | 1 | Z |
| DEC [m] | Decrement Data Memory | 1 ^{Note} | Z |
| Rotate | | | |
| RRA [m] | Rotate Data Memory right with result in ACC | 1 | None |
| RR [m] | Rotate Data Memory right | 1 ^{Note} | None |
| RRCA [m] | Rotate Data Memory right through Carry with result in ACC | 1 | C |
| RRC [m] | Rotate Data Memory right through Carry | 1 ^{Note} | C |
| RLA [m] | Rotate Data Memory left with result in ACC | 1 | None |
| RL [m] | Rotate Data Memory left | 1 ^{Note} | None |
| RLCA [m] | Rotate Data Memory left through Carry with result in ACC | 1 | C |
| RLC [m] | Rotate Data Memory left through Carry | 1 ^{Note} | C |
| Data Move | | | |
| MOV A,[m] | Move Data Memory to ACC | 1 | None |
| MOV [m],A | Move ACC to Data Memory | 1 ^{Note} | None |
| MOV A,x | Move immediate data to ACC | 1 | None |
| Bit Operation | | | |
| CLR [m].i | Clear bit of Data Memory | 1 ^{Note} | None |
| SET [m].i | Set bit of Data Memory | 1 ^{Note} | None |

| Mnemonic | Description | Cycles | Flag Affected |
|----------------------|--|-------------------|---------------|
| Branch | | | |
| JMP addr | Jump unconditionally | 2 | None |
| SZ [m] | Skip if Data Memory is zero | 1 ^{Note} | None |
| SZA [m] | Skip if Data Memory is zero with data movement to ACC | 1 ^{Note} | None |
| SZ [m].i | Skip if bit i of Data Memory is zero | 1 ^{Note} | None |
| SNZ [m].i | Skip if bit i of Data Memory is not zero | 1 ^{Note} | None |
| SIZ [m] | Skip if increment Data Memory is zero | 1 ^{Note} | None |
| SDZ [m] | Skip if decrement Data Memory is zero | 1 ^{Note} | None |
| SIZA [m] | Skip if increment Data Memory is zero with result in ACC | 1 ^{Note} | None |
| SDZA [m] | Skip if decrement Data Memory is zero with result in ACC | 1 ^{Note} | None |
| CALL addr | Subroutine call | 2 | None |
| RET | Return from subroutine | 2 | None |
| RET A,x | Return from subroutine and load immediate data to ACC | 2 | None |
| RETI | Return from interrupt | 2 | None |
| Table Read | | | |
| TABRDC [m] | Read table (current page) to TBLH and Data Memory | 2 ^{Note} | None |
| TABRDL [m] | Read table (last page) to TBLH and Data Memory | 2 ^{Note} | None |
| Miscellaneous | | | |
| NOP | No operation | 1 | None |
| CLR [m] | Clear Data Memory | 1 ^{Note} | None |
| SET [m] | Set Data Memory | 1 ^{Note} | None |
| CLR WDT | Clear Watchdog Timer | 1 | TO, PDF |
| CLR WDT1 | Pre-clear Watchdog Timer | 1 | TO, PDF |
| CLR WDT2 | Pre-clear Watchdog Timer | 1 | TO, PDF |
| SWAP [m] | Swap nibbles of Data Memory | 1 ^{Note} | None |
| SWAPA [m] | Swap nibbles of Data Memory with result in ACC | 1 | None |
| HALT | Enter Power Down Mode | 1 | TO, PDF |

- Note**
1. For skip instructions, if the result of the comparison involves a skip then two cycles are required if no skip takes place only one cycle is required.
 2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.
 3. For the "CLR WDT1" and "CLR WDT2" instructions, the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after both "CLR WDT1" and "CLR WDT2" instructions are consecutively executed. Otherwise the TO and PDF flags remain unchanged.

Chapter 3

Instruction Definition

3

| | |
|-------------------|---|
| ADC A,[m] | Add Data Memory to ACC with Carry |
| Description | The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC + [m] + C$ |
| Affected flag(s) | OV, Z, AC, C |
| ADCM A,[m] | Add ACC to Data Memory with Carry |
| Description | The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory. |
| Operation | $[m] \leftarrow ACC + [m] + C$ |
| Affected flag(s) | OV, Z, AC, C |
| ADD A,[m] | Add Data Memory to ACC |
| Description | The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC + [m]$ |
| Affected flag(s) | OV, Z, AC, C |
| ADD A,x | Add immediate data to ACC |
| Description | The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC + x$ |
| Affected flag(s) | OV, Z, AC, C |
| ADDM A,[m] | Add ACC to Data Memory |
| Description | The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory. |
| Operation | $[m] \leftarrow ACC + [m]$ |
| Affected flag(s) | OV, Z, AC, C |

| | |
|-------------------|---|
| AND A,[m] | Logical AND Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC "AND" [m] |
| Affected flag(s) | Z |
| AND A,x | Logical AND immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bitwise logical AND operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC "AND" x |
| Affected flag(s) | Z |
| ANDM A,[m] | Logical AND ACC to Data Memory |
| Description | Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory. |
| Operation | [m] ← ACC "AND" [m] |
| Affected flag(s) | Z |
| CALL addr | Subroutine call |
| Description | Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction. |
| Operation | Stack ← Program Counter + 1 Program Counter ← addr |
| Affected flag(s) | None |
| CLR [m] | Clear Data Memory |
| Description | Each bit of the specified Data Memory is cleared to 0. |
| Operation | [m] ← 00H |
| Affected flag(s) | None |
| CLR [m].i | Clear bit of Data Memory |
| Description | Bit i of the specified Data Memory is cleared to 0. |
| Operation | [m].i ← 0 |
| Affected flag(s) | None |

| | |
|------------------|---|
| CLR WDT | Clear Watchdog Timer |
| Description | The TO, PDF flags and the WDT are all cleared. |
| Operation | WDT cleared TO ← 0 PDF ← 0 |
| Affected flag(s) | TO, PDF |
| CLR WDT1 | Pre-clear Watchdog Timer |
| Description | The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repeatedly executing this instruction without alternately executing CLR WDT2 will have no effect. |
| Operation | WDT cleared TO ← 0 PDF ← 0 |
| Affected flag(s) | TO, PDF |
| CLR WDT2 | Pre-clear Watchdog Timer |
| Description | The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repeatedly executing this instruction without alternately executing CLR WDT1 will have no effect. |
| Operation | WDT cleared TO ← 0 PDF ← 0 |
| Affected flag(s) | TO, PDF |
| CPL [m] | Complement Data Memory |
| Description | Each bit of the specified Data Memory is logically complemented (1's complement). Bits which previously contained a 1 are changed to 0 and vice versa. |
| Operation | $[m] \leftarrow \overline{[m]}$ |
| Affected flag(s) | Z |
| CPLA [m] | Complement Data Memory with result in ACC |
| Description | Each bit of the specified Data Memory is logically complemented (1's complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC \leftarrow \overline{[m]}$ |
| Affected flag(s) | Z |

| | |
|------------------|---|
| DAA [m] | Decimal-Adjust ACC for addition with result in Data Memory |
| Description | Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition. |
| Operation | [m] ← ACC + 00H or [m] ← ACC + 06H or [m] ← ACC + 60H or [m] ← ACC + 66H |
| Affected flag(s) | C |
| | |
| DEC [m] | Decrement Data Memory |
| Description | Data in the specified Data Memory is decremented by 1. |
| Operation | [m] ← [m] – 1 |
| Affected flag(s) | Z |
| | |
| DECA [m] | Decrement Data Memory with result in ACC |
| Description | Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | ACC ← [m] – 1 |
| Affected flag(s) | Z |
| | |
| HALT | Enter Power Down Mode |
| Description | This instruction stops the program execution and turns off the system clock. The contents of the Data Memory and registers are retained. The WDT and prescaler are cleared. The power down flag PDF is set and the WDT time-out flag TO is cleared. |
| Operation | TO ← 0 PDF ← 1 |
| Affected flag(s) | TO, PDF |
| | |
| INC [m] | Increment Data Memory |
| Description | Data in the specified Data Memory is incremented by 1. |
| Operation | [m] ← [m] + 1 |
| Affected flag(s) | Z |

| | |
|------------------|--|
| INCA [m] | Increment Data Memory with result in ACC |
| Description | Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | $ACC \leftarrow [m] + 1$ |
| Affected flag(s) | Z |
| JMP addr | Jump unconditionally |
| Description | The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction. |
| Operation | Program Counter \leftarrow addr |
| Affected flag(s) | None |
| MOV A,[m] | Move Data Memory to ACC |
| Description | The contents of the specified Data Memory are copied to the Accumulator. |
| Operation | $ACC \leftarrow [m]$ |
| Affected flag(s) | None |
| MOV A,x | Move immediate data to ACC |
| Description | The immediate data specified is loaded into the Accumulator. |
| Operation | $ACC \leftarrow x$ |
| Affected flag(s) | None |
| MOV [m],A | Move ACC to Data Memory |
| Description | The contents of the Accumulator are copied to the specified Data Memory. |
| Operation | $[m] \leftarrow ACC$ |
| Affected flag(s) | None |
| NOP | No operation |
| Description | No operation is performed. Execution continues with the next instruction. |
| Operation | No operation |
| Affected flag(s) | None |
| OR A,[m] | Logical OR Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC \text{ "OR" } [m]$ |
| Affected flag(s) | Z |

| | |
|------------------|---|
| OR A,x | Logical OR immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC "OR" x |
| Affected flag(s) | Z |
| ORM A,[m] | Logical OR ACC to Data Memory |
| Description | Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory. |
| Operation | [m] ← ACC "OR" [m] |
| Affected flag(s) | Z |
| RET | Return from subroutine |
| Description | The Program Counter is restored from the stack. Program execution continues at the restored address. |
| Operation | Program Counter ← Stack |
| Affected flag(s) | None |
| RET A,x | Return from subroutine and load immediate data to ACC |
| Description | The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address. |
| Operation | Program Counter ← Stack ACC ← x |
| Affected flag(s) | None |
| RETI | Return from interrupt |
| Description | The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the enable master (global) interrupt bit (bit 0; register INTC). If an interrupt was pending when the RETI instruction is executed, the pending interrupt routine will be processed before returning to the main program. |
| Operation | Program Counter ← Stack EMI ← 1 |
| Affected flag(s) | None |
| RL [m] | Rotate Data Memory left |
| Description | The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. |
| Operation | [m].(i+1) ← [m].i; (i = 0~6) [m].0 ← [m].7 |
| Affected flag(s) | None |

| | |
|------------------|---|
| RLA [m] | Rotate Data Memory left with result in ACC |
| Description | The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | ACC.(i+1) ← [m].i; (i = 0~6) ACC.0 ← [m].7 |
| Affected flag(s) | None |
| RLC [m] | Rotate Data Memory left through Carry |
| Description | The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0. |
| Operation | [m].(i+1) ← [m].i; (i = 0~6) [m].0 ← C C ← [m].7 |
| Affected flag(s) | C |
| RLCA [m] | Rotate Data Memory left through Carry with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | ACC.(i+1) ← [m].i; (i = 0~6) ACC.0 ← C C ← [m].7 |
| Affected flag(s) | C |
| RR [m] | Rotate Data Memory right |
| Description | The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7. |
| Operation | [m].i ← [m].(i+1); (i = 0~6) [m].7 ← [m].0 |
| Affected flag(s) | None |
| RRA [m] | Rotate Data Memory right with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | ACC.i ← [m].(i+1); (i = 0~6) ACC.7 ← [m].0 |
| Affected flag(s) | None |

| | |
|-------------------|--|
| RRC [m] | Rotate Data Memory right through Carry |
| Description | The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. |
| Operation | $[m].i \leftarrow [m].(i+1); (i = 0-6)$ $[m].7 \leftarrow C$ $C \leftarrow [m].0$ |
| Affected flag(s) | C |
| RRCA [m] | Rotate Data Memory right through Carry with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC.i \leftarrow [m].(i+1); (i = 0-6)$ $ACC.7 \leftarrow C$ $C \leftarrow [m].0$ |
| Affected flag(s) | C |
| SBC A,[m] | Subtract Data Memory from ACC with Carry |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - [m] - \bar{C}$ |
| Affected flag(s) | OV, Z, AC, C |
| SBCM A,[m] | Subtract Data Memory from ACC with Carry and result in Data Memory |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $[m] \leftarrow ACC - [m] - \bar{C}$ |
| Affected flag(s) | OV, Z, AC, C |
| SDZ [m] | Skip if decrement Data Memory is 0 |
| Description | The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $[m] \leftarrow [m] - 1$ Skip if $[m] = 0$ |
| Affected flag(s) | None |

| | |
|------------------|---|
| SDZA [m] | Skip if decrement Data Memory is zero with result in ACC |
| Description | The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction. |
| Operation | ACC ← [m] – 1 Skip if ACC = 0 |
| Affected flag(s) | None |
| SET [m] | Set Data Memory |
| Description | Each bit of the specified Data Memory is set to 1. |
| Operation | [m] ← FFH |
| Affected flag(s) | None |
| SET [m].i | Set bit of Data Memory |
| Description | Bit i of the specified Data Memory is set to 1. |
| Operation | [m].i ← 1 |
| Affected flag(s) | None |
| SIZ [m] | Skip if increment Data Memory is 0 |
| Description | The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | [m] ← [m] + 1 Skip if [m] = 0 |
| Affected flag(s) | None |
| SIZA [m] | Skip if increment Data Memory is zero with result in ACC |
| Description | The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | ACC ← [m] + 1 Skip if ACC = 0 |
| Affected flag(s) | None |

| | |
|-------------------|--|
| SNZ [m].i | Skip if bit i of Data Memory is not 0 |
| Description | If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction. |
| Operation | Skip if [m].i \neq 0 |
| Affected flag(s) | None |
| SUB A,[m] | Subtract Data Memory from ACC |
| Description | The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - [m]$ |
| Affected flag(s) | OV, Z, AC, C |
| SUBM A,[m] | Subtract Data Memory from ACC with result in Data Memory |
| Description | The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $[m] \leftarrow ACC - [m]$ |
| Affected flag(s) | OV, Z, AC, C |
| SUB A,x | Subtract immediate data from ACC |
| Description | The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - x$ |
| Affected flag(s) | OV, Z, AC, C |
| SWAP [m] | Swap nibbles of Data Memory |
| Description | The low-order and high-order nibbles of the specified Data Memory are interchanged. |
| Operation | $[m].3 \sim [m].0 \leftrightarrow [m].7 \sim [m].4$ |
| Affected flag(s) | None |
| SWAPA [m] | Swap nibbles of Data Memory with result in ACC |
| Description | The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | $ACC.3 \sim ACC.0 \leftarrow [m].7 \sim [m].4$ $ACC.7 \sim ACC.4 \leftarrow [m].3 \sim [m].0$ |
| Affected flag(s) | None |

| | |
|-------------------|--|
| SZ [m] | Skip if Data Memory is 0 |
| Description | If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | Skip if [m] = 0 |
| Affected flag(s) | None |
| | |
| SZA [m] | Skip if Data Memory is 0 with data movement to ACC |
| Description | The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | ACC ← [m] Skip if [m] = 0 |
| Affected flag(s) | None |
| | |
| SZ [m].i | Skip if bit i of Data Memory is 0 |
| Description | If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction. |
| Operation | Skip if [m].i = 0 |
| Affected flag(s) | None |
| | |
| TABRDC [m] | Read table (current page) to TBLH and Data Memory |
| Description | The low byte of the program code (current page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |
| | |
| TABRDL [m] | Read table (last page) to TBLH and Data Memory |
| Description | The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |

| | |
|-------------------|--|
| XOR A,[m] | Logical XOR Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC "XOR" [m] |
| Affected flag(s) | Z |
| XORM A,[m] | Logical XOR ACC to Data Memory |
| Description | Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory. |
| Operation | [m] ← ACC "XOR" [m] |
| Affected flag(s) | Z |
| XOR A,x | Logical XOR immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bitwise logical XOR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC "XOR" x |
| Affected flag(s) | Z |

Chapter 4

Assembly Language and Cross Assembler

4

Assembly-Language programs are written as source files. They can be assembled into object files by the Holtek Cross Assembler. Object files are combined by the Cross Linker to generate a task file.

A source program is made up of statements and look up tables, giving directions to the Cross Assembler at assembly time or to the processor at run time. Statements are constituted by mnemonics (operations), operands and comments.

Notational Conventions

The following list describes the notations used by this document.

| Example of Convention | Description of Convention |
|-------------------------------------|--|
| [<i>optional items</i>] | <p>Syntax elements that are enclosed by a pair of brackets are optional. For example, the syntax of the command line is as follows:</p> <p style="text-align: center;">HASM [<i>options</i>] <i>filename</i> [;]</p> <p>In the above command line, <i>options</i> and semicolon; are both optional, but <i>filename</i> is required, except for the following case: Brackets in the instruction operands. In this case, the brackets refer to memory address.</p> |
| { <i>choice1</i> <i>choice2</i> } | <p>Braces and vertical bars stand for a choice between two or more items. Braces enclose the choices whereas vertical bars separate the choices. Only one item can be chosen.</p> |

| Example of Convention | Description of Convention |
|-----------------------|--|
| Repeating elements... | Three dots following an item signify that more items with the same form may be entered. For example, the directive PUBLIC has the following form: PUBLIC <i>name1</i> [, <i>name2</i> [...]] |
| | In the above form, the three dots following <i>name2</i> indicate that many names can be entered as long as each is preceded by a comma. |

Statement Syntax

The construction of each statement is as follows:

[*name*] [*operation*] [*operands*] [;*comment*]

- All fields are optional.
- Each field (except the comment field) must be separated from other fields by at least one space or one tab character.
- Fields are not case-sensitive, i.e., lower-case characters are changed to upper-case characters before processing.

Name

Statements can be assigned labels to enable easy access by other statements. A name consists of the following characters:

A-Z a-z 0-9 ? _ @

with the following restrictions :

- 0-9 cannot be the first character of a name
- ? cannot stand alone as a name
- Only the first 31 characters are recognized

Operation

The operation defines the statement action of which two types exist, directives and instructions. Directives give directions to the Cross Assembler, specifying the manner in which the Cross Assembler is to generate the object code at assembly time. Instructions, on the other hand, give directions to the processor. They are translated to object code at assembly time, the object code in turn controls the behavior of the processor at run time.

Operand

Operands define the data used by directives and instructions. They can be made up of symbols, constants, expressions and registers.

Comment

Comments are the descriptions of codes. They are used for documentation only and are ignored by the Cross Assembler. Any text following a semicolon is considered a comment.

Assembly Directives

Directives give direction to the Cross Assembler, specifying the manner in which the Cross Assembler generates object code at assembly time. Directives can be further classified according to their behavior as described below.

Conditional Assembly Directives

The conditional block has the following form:

```

IF
  statements
[ELSE
  statements]
ENDIF

```

Syntax

```

IF expression
IFE expression

```

- Description

The directives **IF** and **IFE** test the *expression* following them.

The **IF** directive grants assembly if the value of the *expression* is true, i.e. non-zero.

The **IFE** directive grants assembly if the value of the *expression* is false, i.e. zero.

- Example

```

IF debugcase
      ACC1 equ 5
      extern username: byte
ENDIF

```

In this example, the value of the variable `ACC1` is set to 5 and the `username` is declared as an external variable if the symbol `debugcase` is evaluated as true, i.e. nonzero.

Syntax

```

IFDEF name
IFNDEF name

```

- Description

The directives **IFDEF** and **IFNDEF** test whether or not the given *name* has been defined. The **IFDEF** directive grants assembly only if the *name* is a label, a variable or a symbol. The **IFNDEF** directive grants assembly only if the *name* has not yet been defined. The conditional assembly directives support a nesting structure, with a maximum nesting level of 7.

- Example

```

IFDEF      buf_flag
      buffer DB 20 dup(?)
ENDIF

```

In this example, the `buffer` is allocated only if the `buf_flag` has been previously defined.

File Control Directives

Syntax

INCLUDE *file-name*

or

INCLUDE "*file-name*"

- Description

This directive inserts source codes from the source file given by *file-name* into the current source file during assembly. Cross Assembler supports at most 7 nesting levels.

- Example

```
INCLUDE macro.def
```

In this example, the Cross Assembler inserts the source codes from the file `macro.def` into the current source file.

Syntax

PAGE *size*

- Description

This directive specifies the number of the lines in a page of the program listing file. The page size must be within the range from 10 to 255, the default page size is 60.

- Example

```
PAGE 57
```

This example sets the maximum page size of the listing file to 57 lines.

Syntax

.LIST

.NOLIST

- Description

The directives **.LIST** and **.NOLIST** decide whether or not the source program lines are to be copied to the program listing file. **.NOLIST** suppresses copying of subsequent source lines to the program listing file. **.LIST** restores the copying of subsequent source lines to the program listing file. The default is **.LIST**.

- Example

```
.NOLIST
mov a, 1
mov b1, a
.LIST
```

In this example, the two instructions in the block enclosed by **.NOLIST** and **.LIST** are suppressed from copying to the source listing file.

Syntax

.LISTMACRO

.NOLISTMACRO

- Description

The directive **.LISTMACRO** causes the Cross Assembler to list all the source statements, including comments, in a macro. The directive **.NOLISTMACRO** suppresses the listing of all macro expansions. The default is **.NOLISTMACRO**.

Syntax

```
.LISTINCLUDE
.NOLISTINCLUDE
```

- Description
The directive **.LISTINCLUDE** inserts the contents of all included files into the program listing. The directive **.NOLISTINCLUDE** suppresses the addition of included files. The default is **.NOLISTINCLUDE**.

Syntax

```
MESSAGE 'text-string'
```

- Description
The directive **MESSAGE** directs the Cross Assembler to display the *text-string* on the screen. The characters in the *text-string* must be enclosed by a pair of single quotation marks.

Syntax

```
ERRMESSAGE 'error-string'
```

- Description
The directive **ERRMESSAGE** directs the Cross Assembler to issue an error. The characters in the *error-string* must be enclosed by a pair of single quotation marks.

Program Directives

Syntax (comment)

```
; text
```

- Description
A comment consists of characters preceded by a semicolon (;) and terminated by an embedded carriage-return/line-feed.

Syntax

```
name .SECTION [align] [combine] 'class'
```

- Description
The **.SECTION** directive marks the beginning of a program section. A program section is a collection of instructions and/or data whose addresses are relative to the section beginning with the name which defines that section. The *name* of a section can be unique or be the same as the name given to other sections in the program. Sections with the same complete names are treated as the same section.
The optional *align* type defines the alignment of the given section. It can be one of the following:

| | |
|-------------|--|
| BYTE | uses any byte address (the default align type) |
| WORD | uses any word address |
| PARA | uses a paragraph address |
| PAGE | uses a page address |

For the **CODE** section, the byte address is in a single instruction unit. **BYTE** aligns the section at any instruction address, **WORD** aligns the section at any even instruction address, **PARA** aligns the section at any instruction address which is a multiple of 16, and **PAGE** aligns the section at any instruction address with a multiple of 256.

For DATA sections, the byte address is in one byte units (8 bits/byte). **BYTE** aligns the section at any byte address, **WORD** aligns the section at any even address, **PARA** aligns the section at any address which is a multiple of 16, and **PAGE** aligns the section at any address which is a multiple of 256.

The optional *combine* type defines the way of combining sections having the same complete name (section and class name). It can be any one of the following:

– **COMMON**

Creates overlapping sections by placing the start of all sections with the same complete name at the same address. The length of the resulting area is the length of the longest section.

– **AT address**

Causes all label and variable addresses defined in a section to be relative to the given address. The *address* can be any valid expression except a forward reference. It is an absolute address in a specified ROM/RAM bank and must be within the ROM/RAM range.

If no *combine* type is given, the section is combinative, i.e., this section can be concatenated with all sections having the same complete name to form a single, contiguous section.

The *class* type defines the sections that are to be loaded in the contiguous memory. Sections with the same class name are loaded into the memory one after another. The class name **CODE** is used for sections stored in ROM, and the class name **DATA** is used for sections stored in RAM. The complete name of a section consists of a section name and a class name. The named section includes all codes and data below (after) it until the next section is defined.

Syntax

ROMBANK *banknum section-name [,section-name,...]*

- Description

This directive declares which sections are allocated to the specified ROM bank. The *banknum* specifies the ROM bank, ranging from 0 to the maximum bank number of the destination MCU. The *section-name* is the name of the section defined previously in the program. More than one section can be declared in a bank as long as the total size of the sections does not exceed the bank size of 8K words. If this directive is not declared, bank 0 is assumed and all CODE sections defined in this program will be in bank 0. If a CODE section is not declared in any ROM bank, then bank 0 is assumed.

Syntax

RAMBANK *banknum section-name [,section-name,...]*

- Description

This directive is similar to **ROMBANK** except that it specifies the RAM bank, the size of RAM bank is 256 bytes.

Syntax

END

- Description

This directive marks the end of a program. Adding this directive to any included file should be avoided.

Syntax

ORG *expression*

- Description

This directive sets the location counter to *expression*. The subsequent code and data offsets begin at the new offset specified by *expression*. The code or data offset is relative to the beginning of the section where the directive **ORG** is defined. The attribute of a section determines the actual value of offset, absolute or relative.

- Example

```
ORG 8
mov A, 1
```

In this example, the statement `mov A, 1` begins at location 8 in the current section.

Syntax

PUBLIC *name1* [, *name2* [, ...]]

EXTERN *name1:type* [, *name2:type* [, ...]]

- Description

The **PUBLIC** directive marks the variable or label specified by a name that is available to other modules in the program. The **EXTERN** directive, on the other hand, declares an external variable, label or symbol of the specified name and type. The type can be one of the four types: **BYTE**, **WORD** and **BIT** (these three types are for data variables), and **NEAR** (a label type and used by `call` or `jmp`).

- Example

```
PUBLIC start, setflag
EXTERN tmpbuf:byte
CODE .SECTION 'CODE'
start:
    mov a, 55h
    call setflag
    ....
setflag proc
    mov tmpbuf, a
    ret
setflag endp
end
```

In this example, both the label `start` and the procedure `setflag` are declared as public variables. Programs in other sources may refer to these variables. The variable `tmpbuf` is also declared as external. There should be a source file defining a byte that is named `tmpbuf` and is declared as a public variable.

Syntax

name **PROC**

name **ENDP**

- Description

The **PROC** and **ENDP** directives mark a block of code which can be called or jumped to from other modules. The **PROC** creates a label *name* which stands for the address of the first instruction of a procedure. The Cross Assembler will set the value of the label to the current value of the location counter.

- Example

```
toggle      PROC
mov         tmpbuf, a
mov         a, 1
xorm       a, flag
mov         a, tmpbuf
ret
toggle      ENDP
```

Syntax

[*label*:] **DC** *expression1* [,*expression2* [,...]]

- Description

The **DC** directive stores the value of *expression1*, *expression2* etc. in consecutive memory locations. This directive is used for the CODE section only. The bit size of the result value is dependent on the ROM size of the MCU. The Cross Assembler will clear any redundant bits; *expression1* has to be a value or a label. This directive may also be employed to setup the table in the code section.

- Example

```
table1: DC 0128h, 025CH
```

In this example, the Cross Assembler reserves two units of ROM space and also stores 0128H and 025CH into these two ROM units.

Data Definition Directives

An assembly language program consists of one or more statements and comments. A statement or comment is a composition of characters, numbers, and names. The assembly language supports integer numbers. An integer number is a collection of binary, octal, decimal, or hexadecimal digits along with an optional radix. If no radix is given, the Cross Assembler uses the default radix (decimal). The table lists the digits that can be used with each radix.

| Radix | Type | Digits |
|-------|-------------|------------------|
| B | Binary | 01 |
| O | Octal | 01234567 |
| D | Decimal | 0123456789 |
| H | Hexadecimal | 0123456789ABCDEF |

Syntax

```
[name] DB value1 [,value2 [, ...]]
[name] DW value1 [,value2 [, ...]]
[name] DBIT
[name] DB repeated-count DUP(?)
[name] DW repeated-count DUP(?)
```

- Description

These directives reserve the number of bytes/words specified by the repeated-count or reserve bytes/words only. *value1* and *value2* should be ? due to the microcontroller RAM. The Cross Assembler will not initialize the RAM data. **DBIT** reserves a bit. The content ? denotes uninitialized data, i.e., reserves the space of the data. The Cross Assembler will gather every 8 **DBIT** together and reserve a byte for these 8 **DBIT** variables.

- Example

```
DATA          .SECTION  'DATA'
tbuf          DB  ?
chksum       DW  ?
flag1        DBIT
sbuf         DB  ?
cflag        DBIT
```

In this example, the Cross Assembler reserves byte location 0 for *tbuf*, location 1 and 2 for *chksum*, bit 0 of location 3 for *flag1*, location 4 for *sbuf* and bit 1 of location 3 for *cflag*.

Syntax

```
name LABEL {BIT|BYTE|WORD}
```

- Description

The *name* with the data type has the same address as the following data variable

- Example

```
lab1 LABEL WORD
d1 DB ?
d2 DB ?
```

In this example, *d1* is the low byte of *lab1* and *d2* is the high byte of *lab1*.

Syntax

```
name EQU expression
```

- Description

The **EQU** directive creates absolute symbols, aliases, or text symbols by assigning an *expression* to *name*. An absolute symbol is a name standing for a 16-bit value; an alias is a name representing another symbol; a text symbol is a name for another combination of characters. The *name* must be unique, i.e. not having been defined previously. The *expression* can be an integer, a string constant, an instruction mnemonic, a constant expression, or an address expression.

- Example

```
accreg EQU 5
bmove EQU mov
```

In this example, the variable *accreg* is equal to 5, and *bmove* is equal to the instruction *mov*.

Macro Directives

Macro directives enable a block of source statements to be named, and then that name to be re-used in the source file to represent the statements. During assembly, the Cross Assembler automatically replaces each occurrence of the macro name with the statements in the macro definition.

A macro can be defined at any place in the source file as long as the definition precedes the first source line that calls this macro. In the macro definition, the macro to be defined may refer to other macros which have been previously defined. The Cross Assembler supports a maximum of 7 nesting levels.

Syntax

```
name  MACRO [dummy-parameter [, ...]]
      statements
      ENDM
```

The Cross Assembler supports a directive **LOCAL** for the macro definition.

Syntax

```
name  LOCAL dummy-name [, ...]
```

- Description

The **LOCAL** directive defines symbols available only in the defined macro. It must be the first line following the **MACRO** directive, if it is present. The *dummy-name* is a temporary name that is replaced by a unique name when the macro is expanded. The Cross Assembler creates a new actual name for *dummy-name* each time the macro is expanded. The actual name has the form ??digit, where digit is a hexadecimal number within the range from 0000 to FFFF. A label should be added to the **LOCAL** directive when labels are used within the **MACRO/ENDM** block. Otherwise, the Cross Assembler will issue an error if this **MACRO** is referred to more than once in the source file.

In the following example, tmp1 and tmp2 are both dummy parameters, and are replaced by actual parameters when calling this macro. label1 and label2 are both declared **LOCAL**, and are replaced by ??0000 and ??0001 respectively at the first reference, if no other **MACRO** is referred. If no **LOCAL** declaration takes place, label1 and label2 will be referred to labels, similar to the declaration in the source program. At the second reference of this macro, a multiple define error message is displayed.

```
Delay  MACRO  tmp1, tmp2
      LOCAL   label1, label2
      mov     a, 70h
      mov     tmp1, a
label1:
      mov     tmp2, a
label2:
      clr     wdt1
      clr     wdt2
      sdz     tmp2
      jmp     label2
      sdz     tmp1
      jmp     label1
      ENDM
```

The following source program refers to the macro Delay:

```

; T.ASM
; Sample program for MACRO.
.ListMacro
Delay MACRO tmp1, tmp2
    LOCAL label1, label2
    mov a, 70h
    mov tmp1, a
label1:
    mov tmp2, a
label2:
    clr wdt1
    clr wdt2
    sdz tmp2
    jmp label2
    sdz tmp1
    jmp label1
ENDM

data .section 'data'
BCnt db ?
SCnt db ?

code .section at 0 'code'
Delay BCnt, SCnt
end

```

The Cross Assembler will expand the macro Delay as shown in the following listing file. Note that the offset of each line in the macro body, from line 4 to line 17, is 0000. Line 24 is expanded to 11 lines and forms the macro body. In addition the formal parameters, `tmp1` and `tmp2`, are replaced with the actual parameters, `BCnt` and `SCnt`, respectively.

```

File: T.asm           Holtek Cross-Assembler Version 2.80           Page 1

1 0000                ; T.ASM
2 0000                ; Sample program for MACRO.
3 0000                .ListMacro
4 0000                Delay MACRO tmp1, tmp2
5 0000                  LOCAL label1, label2
6 0000                  mov a, 70h
7 0000                  mov tmp1, a
8 0000                label1:
9 0000                  mov tmp2, a
10 0000               label2:
11 0000                  clr wdt1
12 0000                  clr wdt2
13 0000                  sdz tmp2
14 0000                  jmp label2
15 0000                  sdz tmp1
16 0000                  jmp label1
17 0000                  ENDM
18 0000
19 0000                data .section 'data'
20 0000 00            BCnt db ?
21 0001 00            SCnt db ?
22 0002
23 0000                code .section at 0 'code'
24 0000                Delay BCnt, SCnt
24 0000 0F70          1      mov a, 70h
24 0001 0080          R1     mov BCnt, a
24 0002                1      ??0000:
24 0002 0080          R1     mov SCnt, a
24 0003                1      ??0001:
24 0003 0001          1      clr wdt1
24 0004 0005          1      clr wdt2
24 0005 1780          R1     sdz SCnt
24 0006 2803          1      jmp ??0001
24 0007 1780          R1     sdz BCnt
24 0008 2802          1      jmp ??0000
25 0009                end

0 Errors

```

Assembly Instructions

The syntax of an instruction has the following form:

```
[name:] mnemonic [operand1[, operand2]] [;comment]
```

where

| | |
|-----------------|--|
| <i>name:</i> | → label name |
| <i>mnemonic</i> | → instruction name (keywords) |
| <i>operand1</i> | → registers memory address |
| <i>operand2</i> | → registers memory address immediate value |

Name

A name is made up of letters, digits, and special characters, and is used as a label.

Mnemonic

Mnemonic is an instruction name dependent upon the type of the MCU used in the source program.

Operand, Operator and Expression

Operands (source or destination) are the argument defining values that are to be acted on by instructions. They can be constants, variables, registers, expressions or keywords. When using the instruction statements, care must be taken to select the correct operand type, i.e. source operand or destination operand. The dollar sign \$ is a special operand, namely the current location operand.

An expression consists of many operands that are combined to describe a value or a memory location. The combined operators are evaluated at assembly time. They can contain constants, symbols, or any combination of constants and symbols that are separated by arithmetic operators.

Operators specify the operations to be performed while combining the operands of an expression. The Cross Assembler provides many operators to combine and evaluate operands. Some operators work with integer constants, some with memory values, and some with both. Operators handle the calculation of constant values that are known at the assembly time. The following are some operators provided by the Cross Assembler.

- Arithmetic operators + - * / % (MOD)
- SHL and SHR operators

– Syntax

```
expression SHR count  
expression SHL count
```

The values of these shift bit operators are all constant values. The *expression* is shifted right **SHR** or left **SHL** by the number of bits specified by *count*. If bits are shifted out of position, the corresponding bits that are shifted in are zero-filled. The following are such examples:

```
mov A, 01110111b SHR 3 ; result ACC=00001110b
mov A, 01110111b SHL 4 ; result ACC=01110000b
```

- Bitwise operators NOT, AND, OR, XOR

– Syntax

```
NOT expression
expression1 AND expression2
expression1 OR expression2
expression1 XOR expression2
```

NOT is a bitwise complement.
AND is a bitwise AND.
OR is a bitwise inclusive OR.
XOR is a bitwise exclusive OR.

- OFFSET operator

– Syntax

```
OFFSET expression
```

The **OFFSET** operator returns the offset address of an *expression*. The *expression* can be a label, a variable, or other direct memory operand. The value returned by the **OFFSET** operator is an immediate operand.

- LOW, MID and HIGH operator

– Syntax

```
LOW expression
MID expression
HIGH expression
```

The **LOW/MID/HIGH** operator returns the value of an *expression* if the result of the *expression* is an immediate value. The **LOW/MID/HIGH** operators will then take the low/middle/high byte of this value. But if the *expression* is a label, the **LOW/MID/HIGH** operator will take the values of the low/middle/high byte of the program count of this label.

- BANK operator

– Syntax

```
BANK name
```

The **BANK** operator returns the bank number allocated to the section of the *name* declared. If the *name* is a label then it returns the rom bank number. If the *name* is a data variable then it returns the ram bank number. The format of the bank number is the same as the BP defined. For more information of the format please refer to the data sheets of the corresponding MCUs. (Note: The format of the BP might be different between MCUs.)

Example 1:

```
mov A, BANK start
mov BP, A
jmp start
```

Example 2:

```

mov A, BANK var
mov BP, A
mov A, OFFSET var
mov MP1, A
mov A, IAR1

```

- Operator precedence

| Precedence | Operators |
|-------------|---|
| 1 (Highest) | (), [] |
| 2 | +, - (unary), LOW, MID, HIGH, OFFSET, BANK |
| 3 | *, /, %, SHL, SHR |
| 4 | +, - (binary) |
| 5 | > (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to) |
| 6 | == (equal to), != (not equal to) |
| 7 | ! (bitwise NOT) |
| 8 | & (bitwise AND) |
| 9 (Lowest) | (bitwise OR), ^ (bitwise XOR) |

Miscellaneous

Forward References

The Cross Assembler allows reference to labels, variable names, and other symbols before they are declared in the source code (forward named references). But symbols to the right of **EQU** are not allowed to be forward referenced.

Local Labels

A local label is a label with a fixed form such as \$number. The number can be 0~29. The function of a local label is the same as a label except that the local label can be used repeatedly. The local label should be used between any two consecutive labels and the same local label name may used between other two consecutive labels. The Cross Assembler will transfer every local label into a unique label before assembling the source file. At most 30 local labels can be defined between two consecutive labels.

Example.

```

Label1:
    $1:                ; label
           mov a, 1    ;; local label
           jmp $3
    $2:                ; ; local label
           mov a, 2
           jmp $1
    $3:                ; ; local label
           jmp $2
Label2:
           jmp $1      ; label
    $0:                ; ; local label
           jmp Label1
    $1:                jmp $0
Label3:

```

Reserved Assembly Language Words

The following tables list all reserved words used by the assembly language.

- Reserved Names (directives, operators)

| | | | |
|------|------------|----------------|----------|
| \$ | DUP | INCLUDE | NOT |
| * | DW | LABEL | OFFSET |
| + | ELSE | .LIST | OR |
| - | END | .LISTINCLUDE | ORG |
| . | ENDIF | .LISTMACRO | PAGE |
| / | ENDM | LOCAL | PARA |
| = | ENDP | LOW | PROC |
| ? | EQU | MACRO | PUBLIC |
| [] | ERRMESSAGE | MESSAGE | RAMBANK |
| AND | EXTERN | MID | ROMBANK |
| BANK | HIGH | MOD | .SECTION |
| BYTE | IF | NEAR | SHL |
| DB | IFDEF | .NOLIST | SHR |
| DBIT | IFE | .NOLISTINCLUDE | WORD |
| DC | IFNDEF | .NOLISTMACRO | XOR |

- Reserved Names (instruction mnemonics)

| | | | |
|------|------|------|--------|
| ADC | HALT | RLCA | SUB |
| ADCM | INC | RR | SUBM |
| ADD | INCA | RRA | SWAP |
| ADDM | JMP | RRC | SWAPA |
| AND | MOV | RRCA | SZ |
| ANDM | NOP | SBC | SZA |
| CALL | OR | SBCM | TABRDC |
| CLR | ORM | SDZ | TABRDL |
| CPL | RET | SDZA | XOR |
| CPLA | RETI | SET | XORM |
| DAA | RL | SIZ | |
| DEC | RLA | SIZA | |
| DECA | RLC | SNZ | |

- Reserved Names (registers names)

| | | | |
|---|-----|------|------|
| A | WDT | WDT1 | WDT2 |
|---|-----|------|------|

Cross Assembler Options

The Cross Assembler options can be set via the Options menu Project command in HT-IDE3000. The Cross Assembler Options is located on the center part of the Project Option dialog box.

The symbols could be defined in the *Define Symbol* edit box.

Syntax

```
symbol1[=value1] [, symbol2[=value2] [, ...]]
```

- Example,

```
debugflag=1, newver=3
```

The check box of the *Generate listing file* is used to decide whether the listing file should be generated or not. If the check box is checked, the listing file will be generated. Otherwise, it won't be generated.

Assembly Listing File Format

The Assembly Listing File contains the source program listing and summary information. The first line of each page is a title line which include company name, the Cross Assembler version number, source file name, date/time of assembly and page number.

Source Program Listing

Each line in the source program has the following syntax:

```
line-number offset [code] statement
```

- *Line-number* is the number of the line starting from the first statement in the assembly source file (4 decimal digits).
- The 2nd field – *offset* – is the offset from the beginning of the current section to the code (4 hexadecimal digits)
- The 3rd field – *code* – is present only if the statement generates code or data (two hexadecimal 4-digit data)

The *code* shows the numeric value in hexadecimal if the value is known at assembly time. Otherwise, a proper flag will indicate the action required to compute the value. The following two flags may appear behind the code field.

R → relocatable address (Cross Linker must resolve)

E → external symbol (Cross Linker must resolve)

The following flag may appear before the code field

= → **EQU** or equal-sign directive

The following 2 flags may appear in the code field

---- → section address (Cross Linker must resolve)

nn[xx] → **DUP** expression: nn **DUP**(?)

- The 4th field – *statement* – is the source statement shown exactly as it appears in the source file, or as expanded by a macro. The following flags may appear before a statement.

n → Macro-expansion nesting level

C → line from **INCLUDE** file

- Summary

```
0          1          2          3          4          5          6
123456789012345678901234567890123456789012345678901234567890...
| | | |  oooo  hhhh  hhhh  EC  source-program-statement
                        Rn
```

| | | | → line number (4 digits, right alignment)

oooo → offset of code (4 digits)

hhhh → two 4-digits for opcode

E → external reference

C → statement from included file

R → relocatable name

n → Macro-expansion nesting level

Summary of Assembly

The total warning number and total error number is the information provided at the end of the Cross Assembler listing file.

Miscellaneous

If any errors occur during assembly, each error message and error number will appear directly below the statement where the error occurred.

Example of assembly listing file

File: SAMPLE.ASM Holtek Cross-Assembler Version 2.86 Page 1

```

1 0000          page 60
2 0000          message      'Sample Program 1'
3 0000
4 0000          .listinclude
5 0000          .listmacro
6 0000
7 0000          #include "sample.inc"

1 0000          C pa      equ      [12h]
2 0000          C pac     equ      [13h]
3 0000          C pb      equ      [14h]
4 0000          C pbc     equ      [15h]
5 0000          C pc      equ      [16h]
6 0000          C pcc     equ      [17h]
7 0000          C

8 0000
9 0000          extern extlab : near
10 0000         extern extbl : byte
11 0000
12 0000         clrpb macro
13 0000         clr pb
14 0000         endm
15 0000
16 0000         clrpa macro
17 0000         mov a, 00h
18 0000         mov pa, a
19 0000         clrpb
20 0000         endm
21 0000
22 0000         data .section 'data'
23 0000 00      b1      db ?
24 0001 00      b2      db ?
25 0002 00      bit1    dbit
26 0003
27 0000         code .section 'code'
28 0000 0F55     mov a, 055h
29 0001 0080     R mov bl, a
30 0002 0080     E mov extbl, a
31 0003 0FAA     mov a, 0aah
32 0004 0093     mov pac, a
33 0005         clrpa
33 0005 0F00     1 mov a, 00h
33 0006 0092     1 mov [12h], a
33 0007         1 clrpb
33 0007 1F14     2 clr [14h]
34 0008 0700     R mov a, bl
35 0009 0F00     E mov a, bank extlab
36 000A 0F00     E mov a, offset extbl
37 000B 2800     E jmp  extlab
38 000C
39 000C 1234 5678 dw 1234h, 5678h, 0abcdh, 0ef12h
   ABCD EF12
40 0010         end

```

0 Errors

Part III

Development Tools

Chapter 5**MCU Programming Tools****5**

To ease the process of application development, the importance and availability of supporting tools for microcontrollers cannot be underestimated. To support its range of MCUs, Holtek is fully committed to the development and release of easy to use and fully functional tools for its full range of devices. The overall development environment is known as the HT-IDE, while the operating software is known as the HT-IDE3000. The software provides an extremely user friendly Windows based approach for program editing and debugging while the HT-ICE emulator hardware provides full real time emulation with multi functional trace, stepping and breakpoint functions. With a complete set of interface cards for its full device range and regular software Service Pack updates, the HT-IDE development environment ensures that designers have the best tools to maximize efficiency in the design and release of their microcontroller applications.

HT-IDE Development Environment

The Holtek Integrated Development Environment, otherwise known as the HT-IDE, is a high performance integrated development environment designed around Holtek's series of 8-bit MCU devices. Incorporated within the system is the hardware and software tools necessary for rapid and easy development of applications based on the Holtek range of 8-bit MCUs. The key component within the HT-IDE system is the HT-ICE In-Circuit Emulator, capable of emulating the Holtek 8-bit MCU in real time, in addition to providing powerful debugging and trace features. The latest version of the HT-ICE In-Circuit Emulator also incorporates a complete OTP writer which provides the user with all the tools required to design, debug and program their OTP devices.

As for the software, the HT-IDE3000 provides a friendly workbench to ease the process of application program development, by integrating all of the software tools, such as editor, Cross Assembler, Cross Linker, library and symbolic debugger into a user friendly Windows based environment. In addition, the HT-IDE3000 provides a software simulator which is capable of simulating the behavior of Holtek's 8-bit MCU range without connection to the HT-ICE. All fundamental functions of the HT-ICE hardware are valid for the simulator.

More detailed information on the HT-IDE3000 development system is contained within the HT-IDE3000 User's Guide. Installed in conjunction with the HT-IDE3000 and to ensure that the development system contains information on new microcontrollers and the latest software updates, Holtek provides regular HT-IDE3000 Service Packs. These Service Packs, which can be downloaded from the Holtek website, do not replace the HT-IDE3000 but are installed after the HT-IDE3000 system software has been installed.

Some of the special features provided by the HT-IDE3000 include:

- **Emulation**
 - Real-time program instruction emulation
- **Hardware**
 - Easy installation and usage
 - Either internal or external oscillator
 - Breakpoint mechanism
 - Trace functions and trigger qualification supported by trace emulation chip
 - Printer port for connecting the HT-ICE to a host computer
 - I/O interface card for connecting the user's application board to the HT-ICE
 - OTP writer hardware integrated within the HT-ICE
- **Software**
 - Windows based software utilities
 - Source program level debugger (symbolic debugger)
 - Workbench for multiple source program files (more than one source program file in one application project)
 - All tools are included for the development, debug, evaluation and generation of the final application program code (mask ROM file and OTP file)
 - Library for the setting up of common procedures which can be linked at a later date to other projects.
 - Simulator can simulate and debug programs without connection to the HT-ICE hardware
 - Virtual Peripheral Manager (VPM) simulates the behavior of the peripheral devices.
 - LCD simulator simulates the behavior of the LCD panel.

Holtek In-Circuit Emulator – HT-ICE

Developed alongside the Holtek 8-bit microcontroller device range, the Holtek ICE is a fully functional in-circuit emulator for Holtek's 8-bit microcontroller devices. Incorporated within the system are a comprehensive set of hardware and software tools for rapid and easy development of user applications. Central to the system is the in-circuit hardware emulator, capable of emulating all of Holtek's 8-bit devices in real-time, while also providing a range of powerful debugging and trace facilities. Regarding software functions, the system incorporates a user-friendly Windows based workbench which integrates together functions such as program editor, Cross Assembler, Cross Linker and library manager. In addition, the system is capable of running in software simulation mode without connection to the HT-ICE hardware.

HT-ICE Interface Card

The interface cards supplied with the HT-ICE can be used for most applications, however, it is possible for the user to omit the supplied interface card and design their own interface card. By including the necessary interface circuitry on their own interface card, the user has a means of directly connecting their target boards to the CON1 and CON2 connectors of the HT-ICE.

OTP Programmer

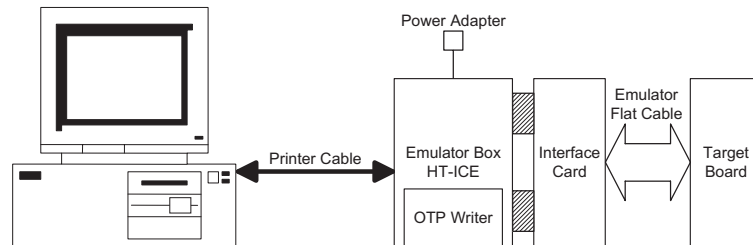
Holtek's OTP devices are fully supported by a range of programmers. For engineering level OTP device programming, Holtek supplies its stand alone programming tool which provides a quick and efficient means for low volume OTP programming. The HT-ICE In-Circuit Emulators has integrated a writer as part of the hardware package, facilitating complete design, debug and OTP device programming all within the HT-ICE. More programmers from other suppliers are available which provide more efficient and higher volume production capability. Please refer to our website for further suppliers information.

OTP Adapter Card

The Holtek OTP programmers are supplied with a standard Textool chip socket. The OTP Adapter Card is used to connect the Holtek OTP programmers to the various sizes of available OTP chip packages that are unable to use this supplied socket.

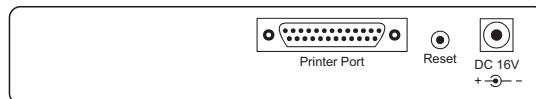
System Configuration

The HT-IDE system configuration is shown below, in which the host computer is a Pentium compatible machine with Windows 95/98/NT/2000/XP or later. Note that if Windows NT/2000/XP or later systems are used, then the HT-IDE3000 software must be installed in Supervisor Privilege mode.

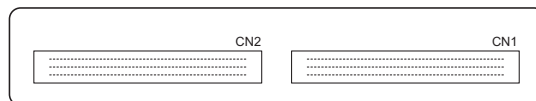


The HT-IDE system contains the following hardware components:

- The HT-ICE box contains the emulator box with 1 printer port connector for connecting to the host machine, I/O signal connector and one power-on LED
- I/O interface card for connecting the target board to the HT-ICE box
- Power Adapter, output 16V
- 25-pin D-type printer cable
- Integrated OTP writer



HT-ICE Rear View

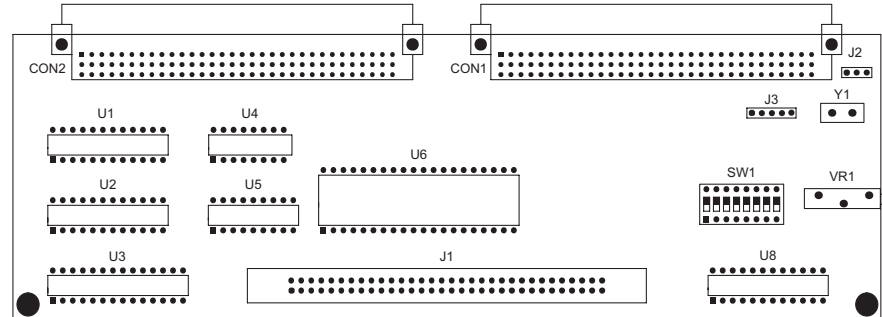


HT-ICE Front View

HT-ICE Interface Card Settings

The HT-ICE interface card (CPCB48E000004A) as shown below, is a PCB used to connect the HT-ICE emulator to the user's target board. It has the following functions:

- External clock source
- MCU socket pin assignment



The external clock source has two modes, RC and Crystal. If a crystal clock is to be used, positions 2 and 3 should be shorted on J2 and a suitable crystal inserted into location Y1. Otherwise, if an RC clock is to be used, positions 1 and 2 should be shorted and the system frequency adjusted using VR1. Refer to the Tools/Mask Option menu of the HT-IDE3000 User's Guide for the clock source and system frequency selection.

The J1 connector provides the I/O port connections as well as other pins. The DIP switch, SW1, should be set according to which device is selected and in accordance with the following table:

| Part No. | Package | Socket | SW1 | | | | | | | |
|------------|----------------|--------|-----|-----|-----|-----|-----|-----|-----|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| HT48R10A-1 | 24SKDIP/SOP | U1 | OFF | OFF | ON | OFF | ON | OFF | OFF | — |
| HT48R30A-1 | 24/28SKDIP/SOP | U2, U3 | ON | OFF | OFF | ON | OFF | OFF | OFF | — |
| HT48R50A-1 | 28SKDIP/SOP | U3 | ON | OFF | OFF | ON | OFF | OFF | ON | — |
| HT48R50A-1 | 48SSOP | U6 | ON | OFF | OFF | OFF | OFF | OFF | OFF | — |
| HT48R70A-1 | 48SSOP | J1 | OFF | OFF | OFF | OFF | OFF | OFF | OFF | — |
| HT48R70A-1 | 64QFP | J1 | OFF | OFF | OFF | OFF | OFF | OFF | OFF | — |
| HT48RU80 | 48SSOP | J1 | OFF | ON | OFF | OFF | OFF | ON | OFF | — |
| HT48RU80 | 64QFP | J1 | OFF | ON | OFF | OFF | OFF | ON | OFF | — |

The pin assignments in locations U1 to U8 are defined so as to match the datasheet pin assignments. The interface card VME connectors directly interface to the CON1 and CON2 connectors on the HT-ICE. Note that although the U6 socket has only 40-pin, the 48-pin HT48R50A-1 SSOP device can still interface to it, as 8 pins on this device are unused.

Installation

System Requirement

The hardware and software requirements for installing HT-IDE3000 system are as follows:

- PC/AT compatible machine with Pentium or higher CPU
- SVGA color monitor
- At least 32M RAM for best performance
- CD ROM drive (for CD installation)
- At least 20M free disk space
- Parallel port to connect PC and HT-ICE
- Windows 95/98/NT/2000/XP

Windows 95/98/NT/2000/XP are trademarks of Microsoft Corporation.

Hardware Installation

- Step 1
Plug the power adapter into the power connector of the HT-ICE
- Step 2
Connect the target board to the HT-ICE by using the I/O interface card or flat cable
- Step 3
Connect the HT-ICE to the host machine using the printer cable

The LED on the HT-ICE should now be lit, if not, there is an error and your dealer should be contacted.

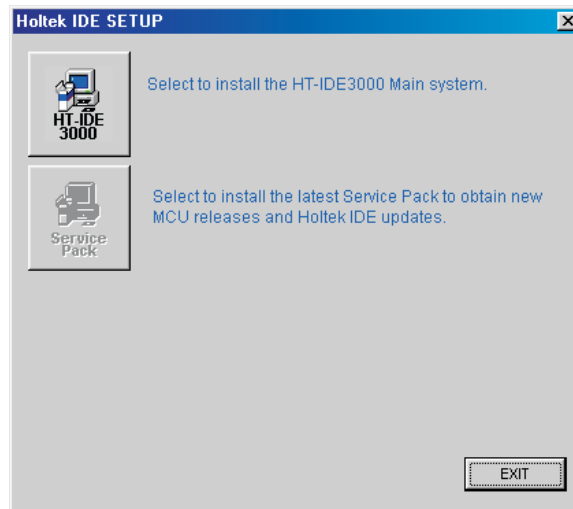
Caution Exercise care when using the power adapter. Do not use a power adapter whose output voltage is not 16V, otherwise the HT-ICE may be damaged. It is strongly recommended that only the power adapter supplied by Holtek be used. First plug the power adapter to the power connector of the HT-ICE.

Software Installation

- Step1
Insert the HT-IDE3000 CD into the CD ROM drive, the following dialog will be shown.



Click <HT-IDE3000> button and the following dialog will be shown.



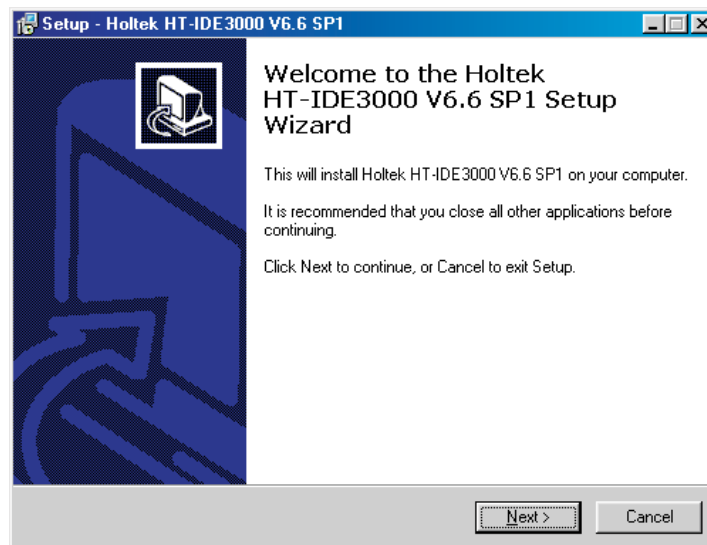
Click <HT-IDE3000> or <Service Pack> as you want.

Here's an Example of installing HT-IDE3000

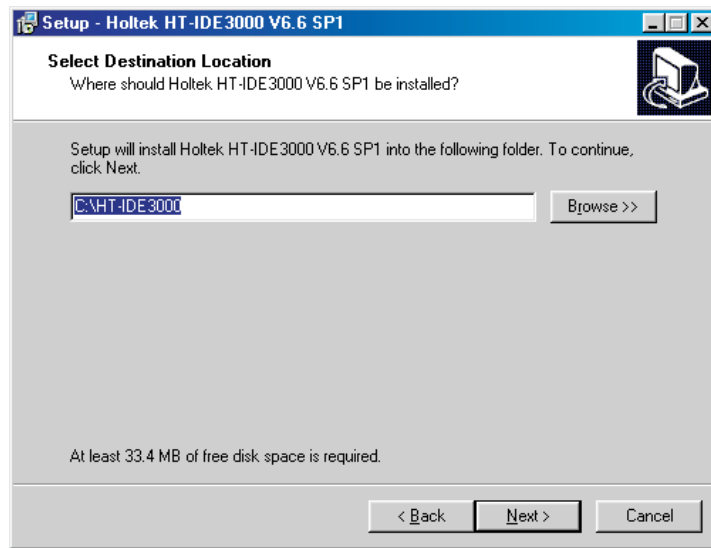
Click <HT-IDE3000> button.

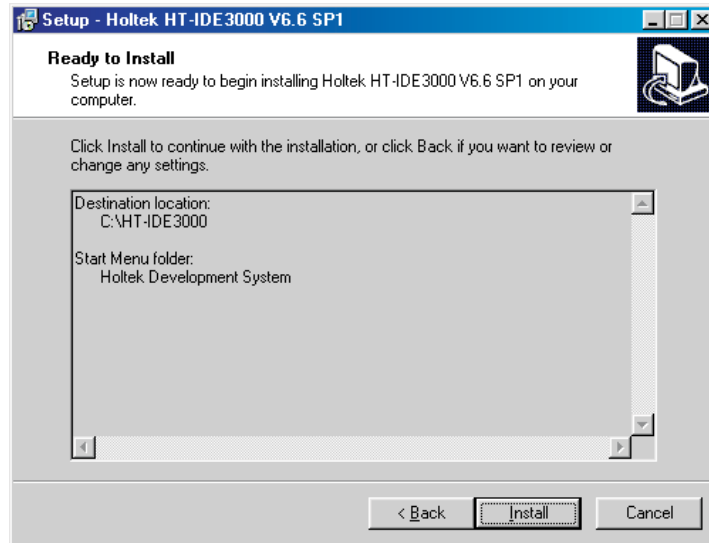
- Step 2

Press the <Next> button to continue setup or press <Cancel> button to abort.

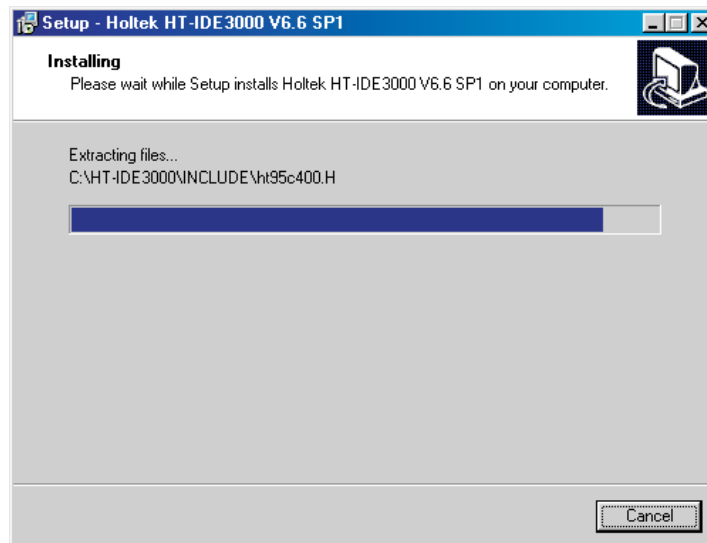


- Step 3
The following dialog will be shown to ask the user to enter a directory name.

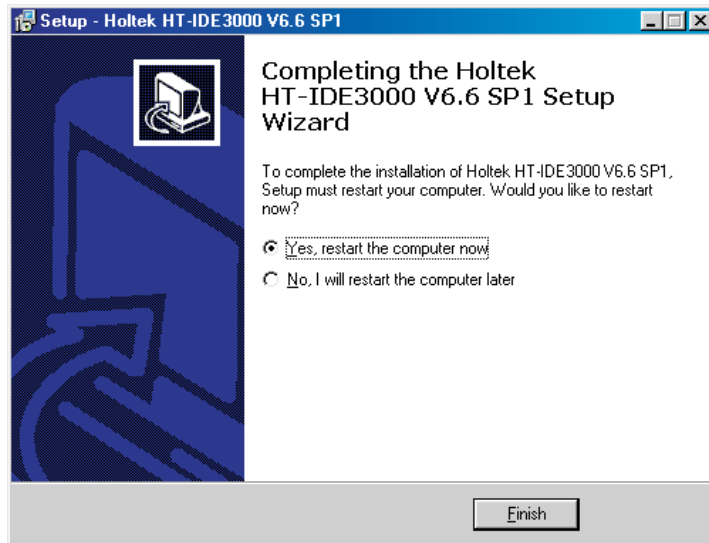




- Step 4
Specify the path you want to install the HT-IDE3000 and click <Next> button.
- Step 5
Setup will copy all files to the specified directory.

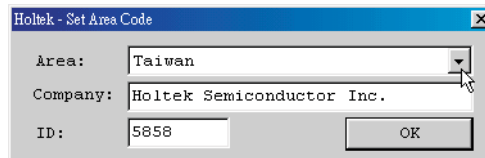


- Step 6
If the process is successful a dialog will be shown.



- Step 7
Press the Finish button and restart the computer system. Then you can run HT-IDE3000 now. SETUP will create four subdirectories, BIN, INCLUDE, LIB, SAMPLE, under the destination directory you specified in Step 4. The BIN subdirectory contains all the system executables (EXE), dynamic link libraries (DLL) and configuration files (CFG, FMT) for all supported MCU. The INCLUDE subdirectory contains all the include files (.H, .INC) provided by Holtek. The LIB subdirectory contains the library files (.LIB) provided by Holtek. The SAMPLE subdirectory contains some sample programs.

Note that before running the HT-IDE3000 for the first time, the system will ask for company information as shown in the figure below. Select appropriate area and fill in the company name and ID. The HT-IDE3000 provider can be requested to supply an ID number.



Chapter 6

Quick Start

6

This chapter gives a brief description of using HT-IDE3000 to develop an application project.

Step 1 – Create a New Project

- Click on Project menu and select New command
- Enter your project name and select an MCU from the combo box
- Click OK button and the system will ask you to setup the configuration options
- Setup all configuration options and click Save button

Step 2 – Add Source Program Files to the Project

- Create your source files by using File/New command
- Write your program and save them with a file name, say TEST.ASM
- Click on Project menu and select Edit command
- An Edit Project dialog will ask you to add/delete files to/from the project
- Select a source file name, say TEST.ASM, and click Add button
- Click OK button after you setup all files in the project

Step 3 – Build the Project

- Click on Project menu and select Build command
- The system will assemble/compile all source files in the project
 - If there are some errors in the programs, double click on the error message line and the system will prompt you the position where the error happened.
 - If all the program files are error free, the system will create a Task file and download to the HT-ICE for debug.
- You may repeat this step before you finish debugging your programs

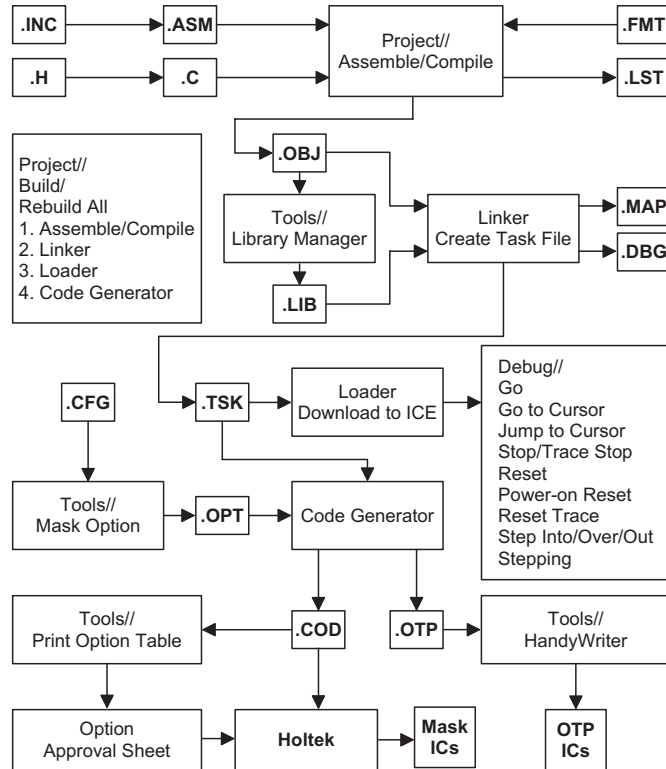
Step 4 – Programming the OTP Device

- Build the project for creating the .OTP file
- Click on Tools menu and select the Writer command to program the OTP devices

Step 5 – Transmit Code to Holtek

- Click on Project menu and select Print Option Table command
- Send the .COD file and the Option Approval Sheet to Holtek

The Programming and data flow is illustrated by the following diagram:



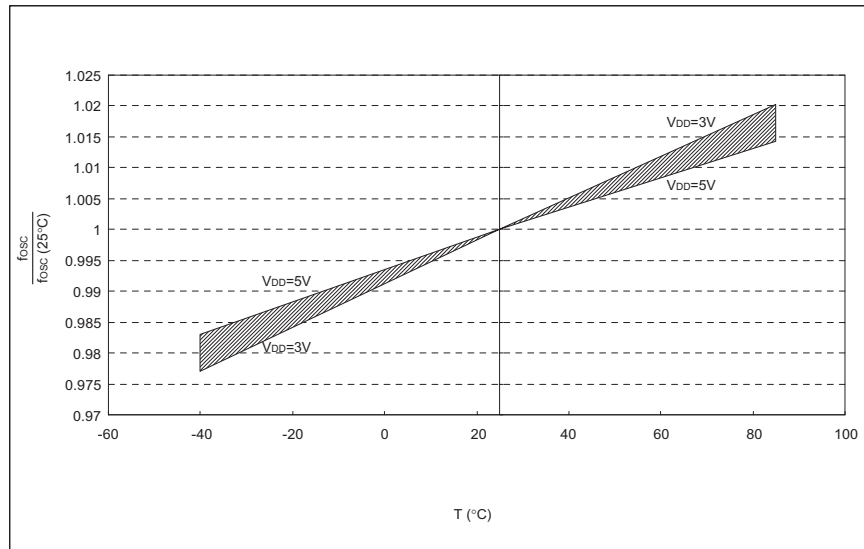
Appendix

Appendix A**Device Characteristic Graphics**

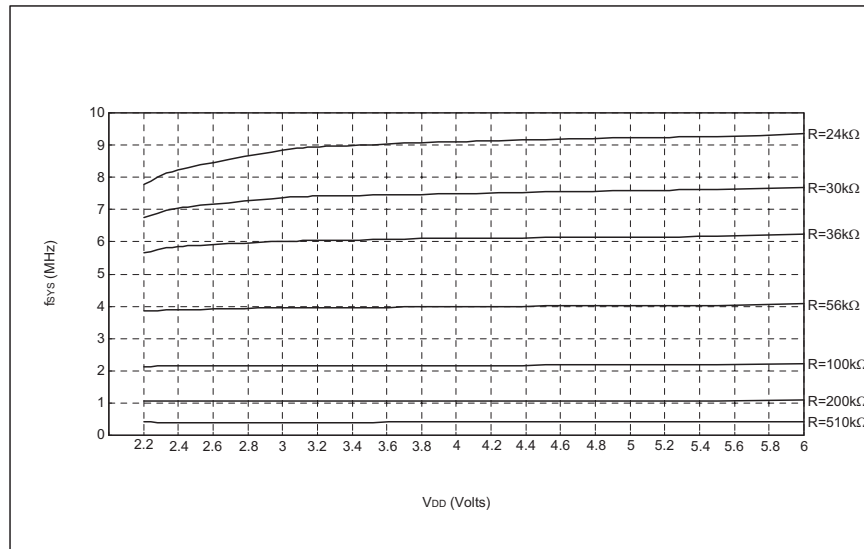
The following characteristic graphics depicts typical device behavior. The data presented here is a statistical summary of data gathered on units from different lots over a period of time. This is for information only and the figures were not tested during manufacturing.

In some of the graphs, the data exceeding the specified operating range are shown for information purposes only. The device will operate properly only within the specified range.

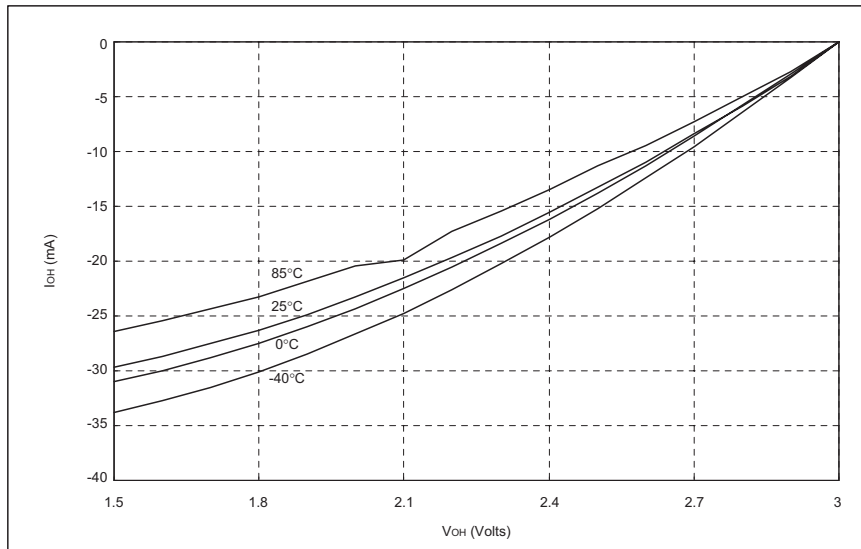
Typical RC OSC vs. Temperature



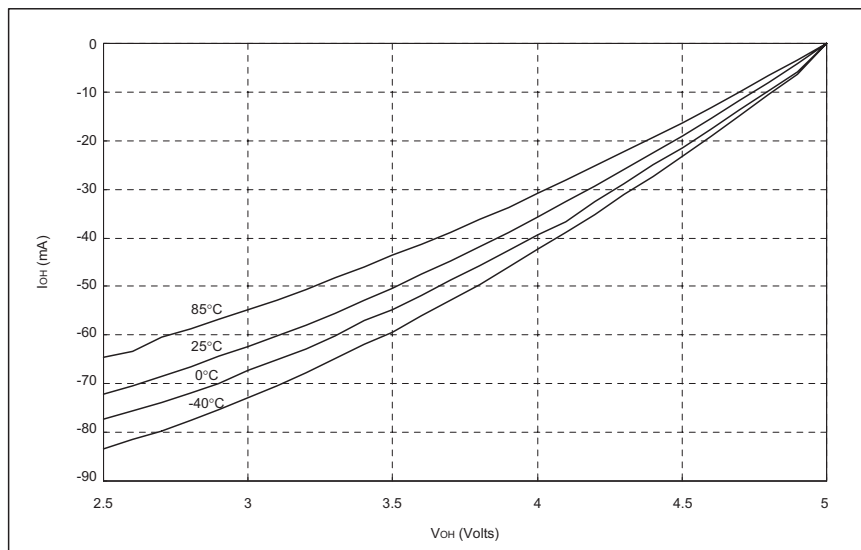
Typical RC Oscillator Frequency vs. V_{DD}



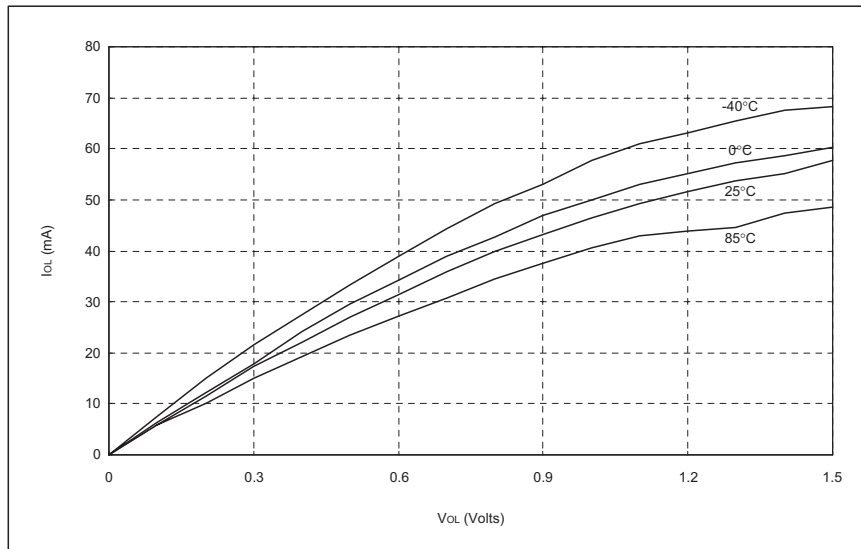
I_{OH} vs. V_{OH} , $V_{DD}=3V$



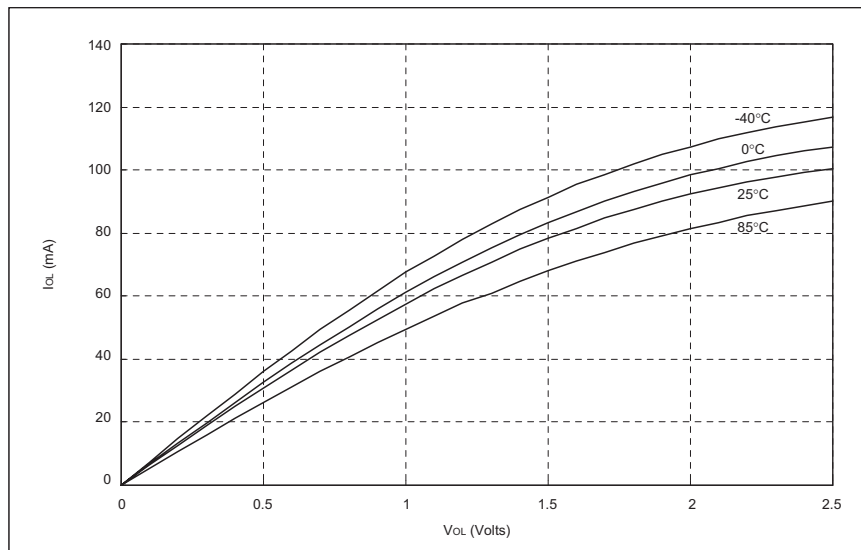
I_{OH} vs. V_{OH} , $V_{DD}=5V$



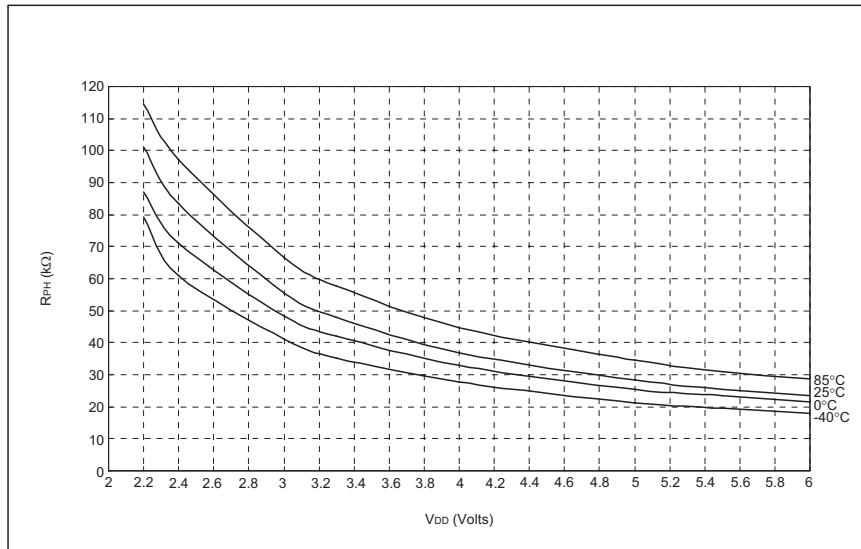
I_{OL} vs. V_{OL} , $V_{DD}=3V$



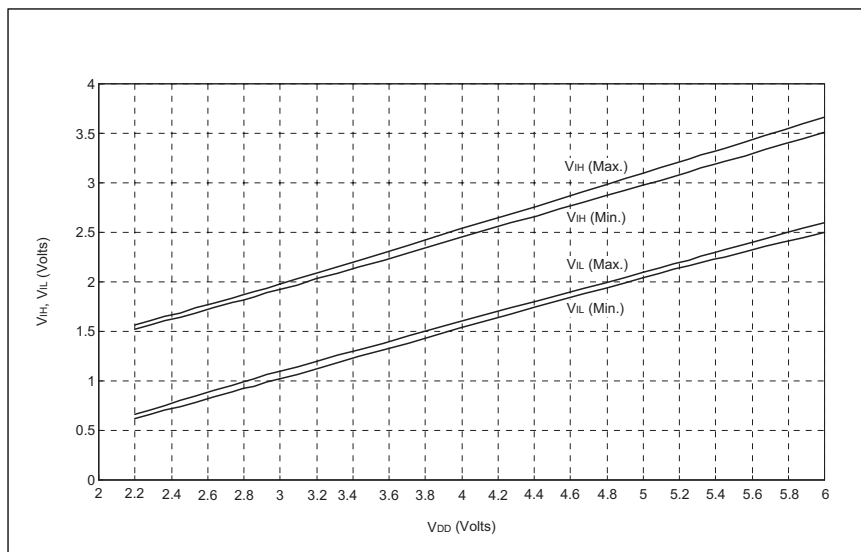
I_{OL} vs. V_{OL} , $V_{DD}=5V$



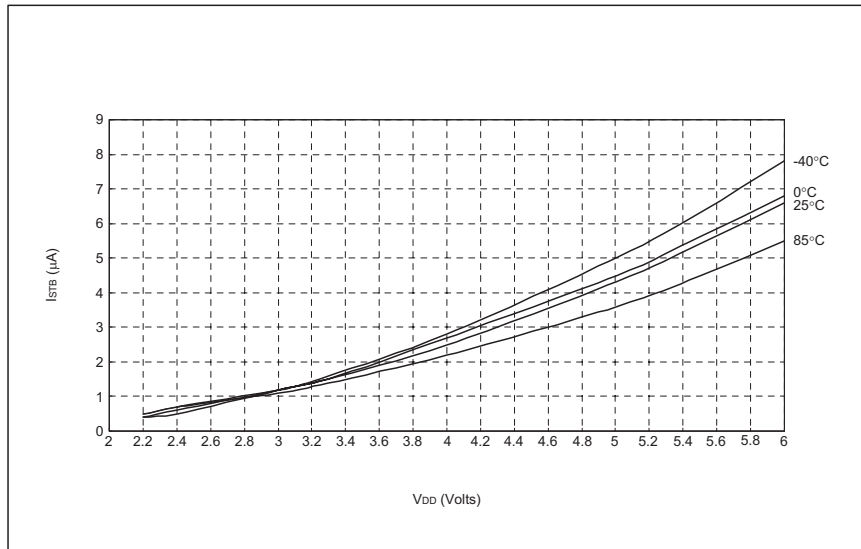
Typical R_{PH} vs. V_{DD}



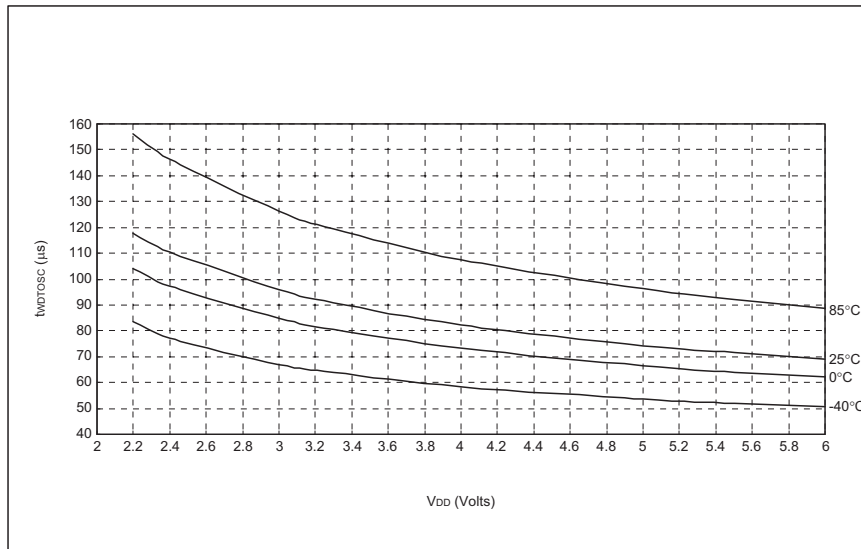
Typical V_{IH} , V_{IL} vs. V_{DD} in -40°C to $+85^{\circ}\text{C}$



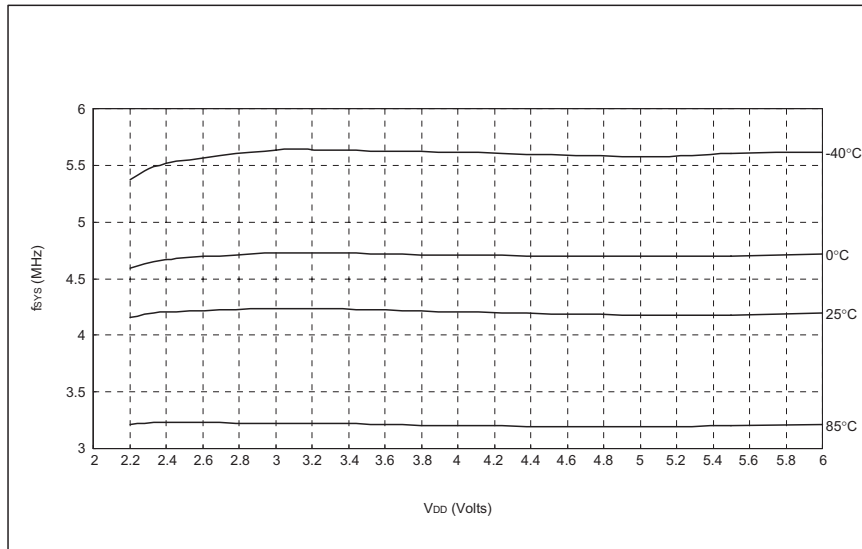
Typical I_{STB} vs. V_{DD} Watchdog Enable



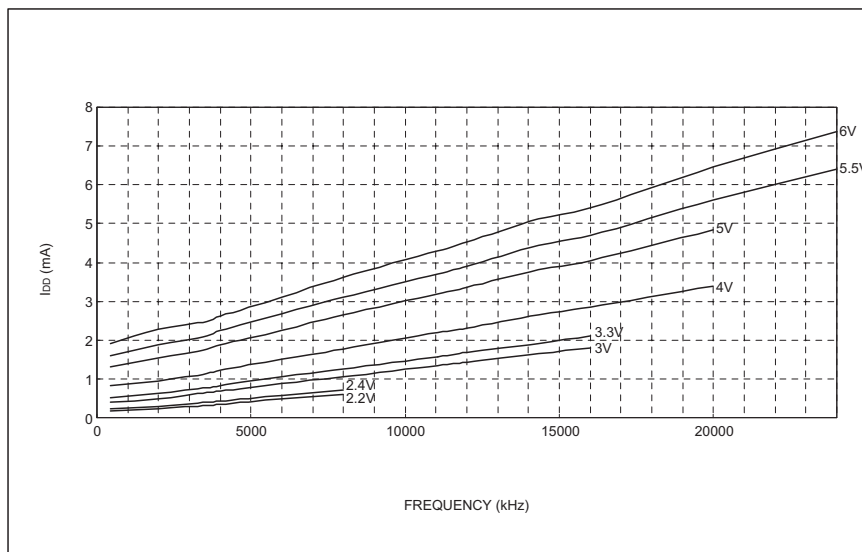
Typical t_{WDOSC} vs. V_{DD}



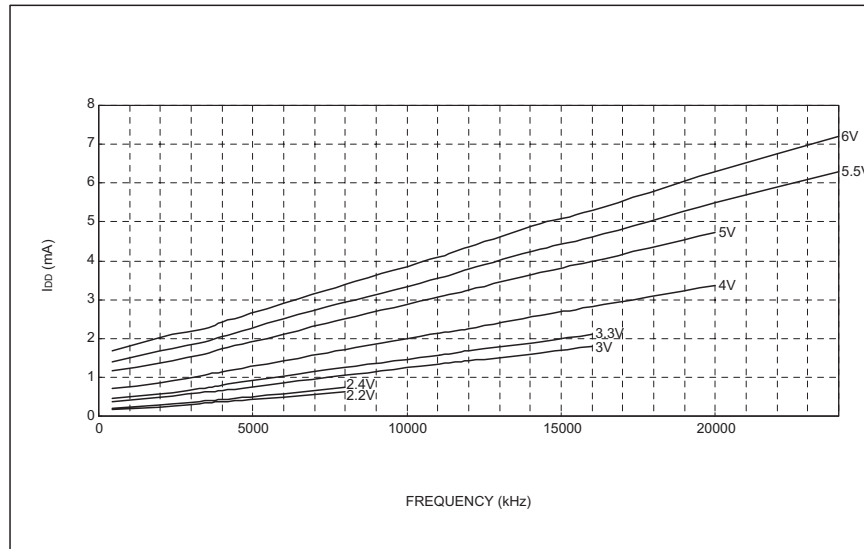
Typical Internal RC OSC vs. V_{DD}



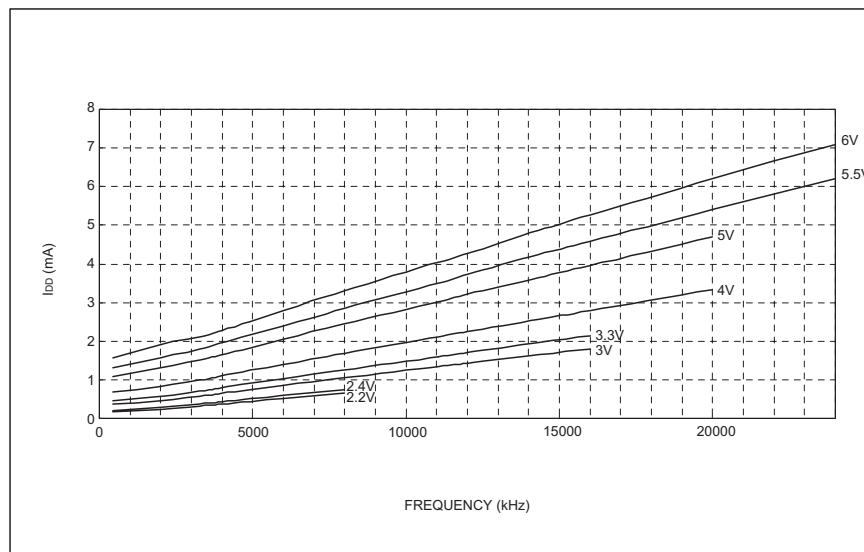
Typical I_{DD} vs. Frequency (External Clock, $T_a = -40^\circ\text{C}$)



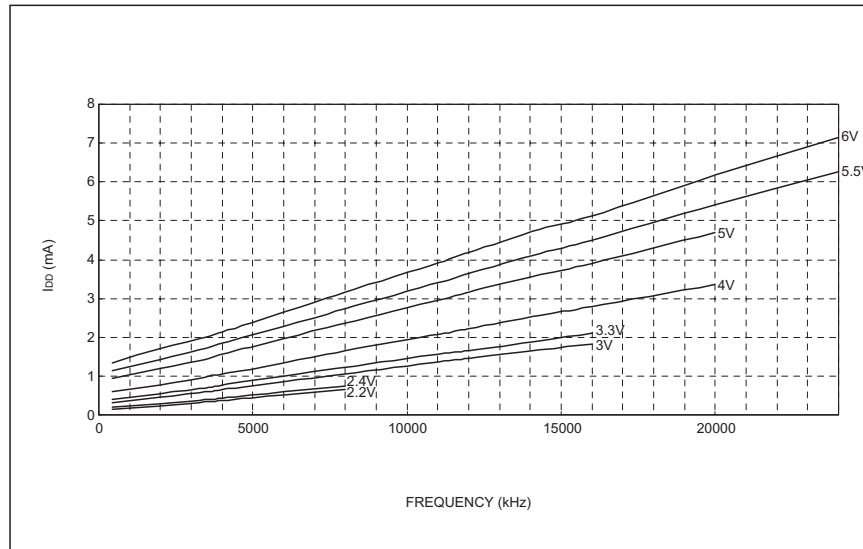
Typical I_{DD} vs. Frequency (External Clock, $T_a=0^\circ\text{C}$)



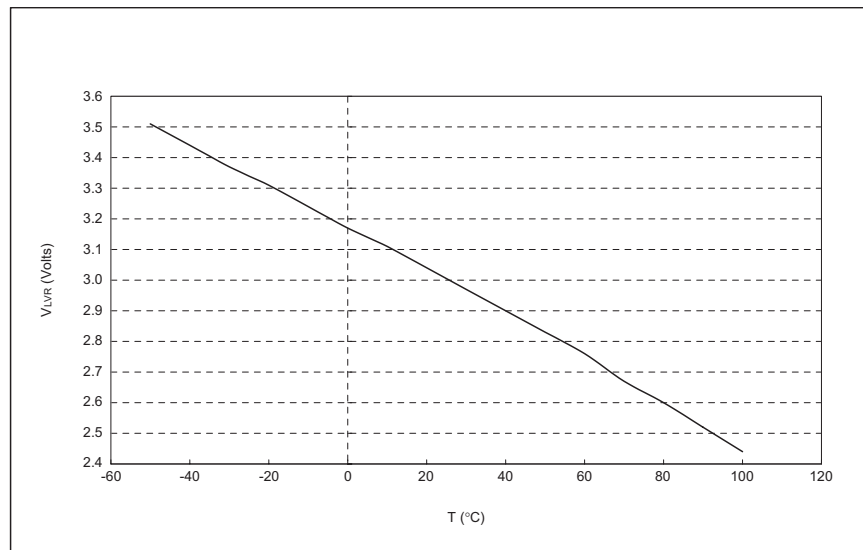
Typical I_{DD} vs. Frequency (External Clock, $T_a=+25^\circ\text{C}$)



Typical I_{DD} vs. Frequency (External Clock, $T_a=+85^\circ\text{C}$)



Typical V_{LVR} vs. Temperature

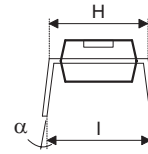
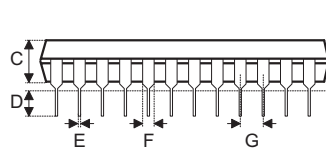
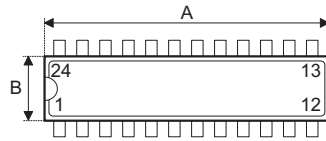


Appendix B

Package Information

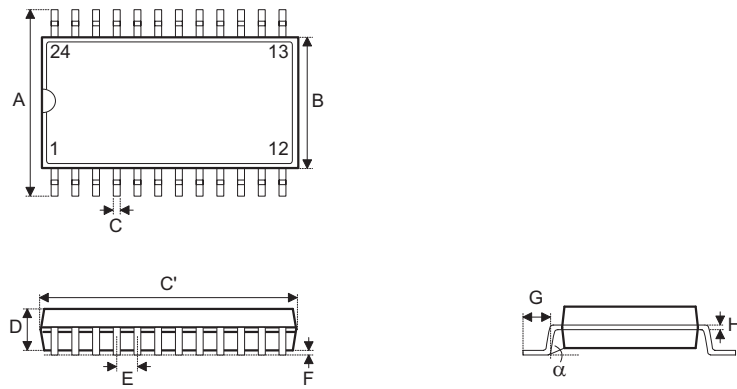
B

24-pin SKDIP (300mil) Outline Dimensions



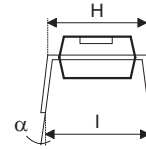
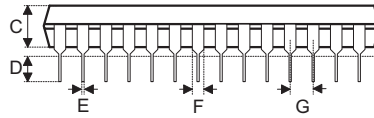
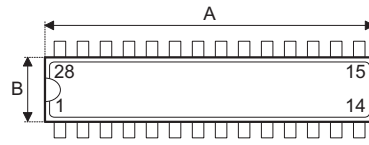
| Symbol | Dimensions in mil | | |
|----------|-------------------|------|------|
| | Min. | Nom. | Max. |
| A | 1235 | — | 1265 |
| B | 255 | — | 265 |
| C | 125 | — | 135 |
| D | 125 | — | 145 |
| E | 16 | — | 20 |
| F | 50 | — | 70 |
| G | — | 100 | — |
| H | 295 | — | 315 |
| I | 345 | — | 360 |
| α | 0° | — | 15° |

24-pin SOP (300mil) Outline Dimensions



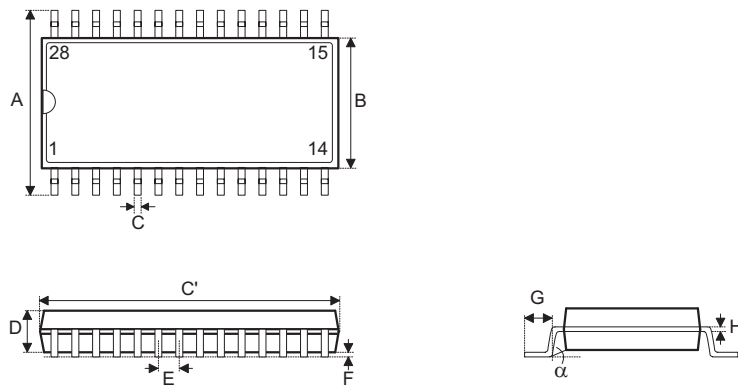
| Symbol | Dimensions in mil | | |
|----------|-------------------|------|------|
| | Min. | Nom. | Max. |
| A | 394 | — | 419 |
| B | 290 | — | 300 |
| C | 14 | — | 20 |
| C' | 590 | — | 614 |
| D | 92 | — | 104 |
| E | — | 50 | — |
| F | 4 | — | — |
| G | 32 | — | 38 |
| H | 4 | — | 12 |
| α | 0° | — | 10° |

28-pin SKDIP (300mil) Outline Dimensions



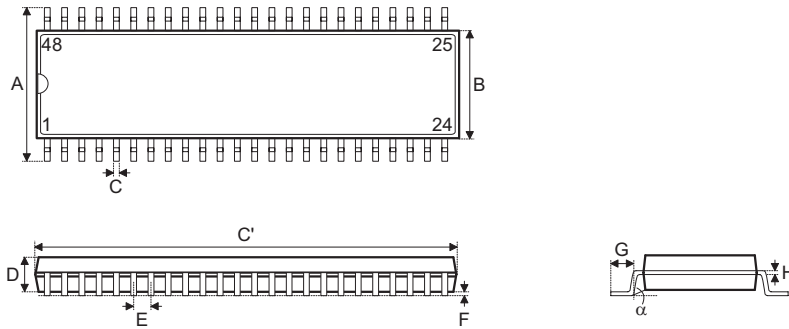
| Symbol | Dimensions in mil | | |
|----------|-------------------|------|------|
| | Min. | Nom. | Max. |
| A | 1375 | — | 1395 |
| B | 278 | — | 298 |
| C | 125 | — | 135 |
| D | 125 | — | 145 |
| E | 16 | — | 20 |
| F | 50 | — | 70 |
| G | — | 100 | — |
| H | 295 | — | 315 |
| I | 330 | — | 375 |
| α | 0° | — | 15° |

28-pin SOP (300mil) Outline Dimensions



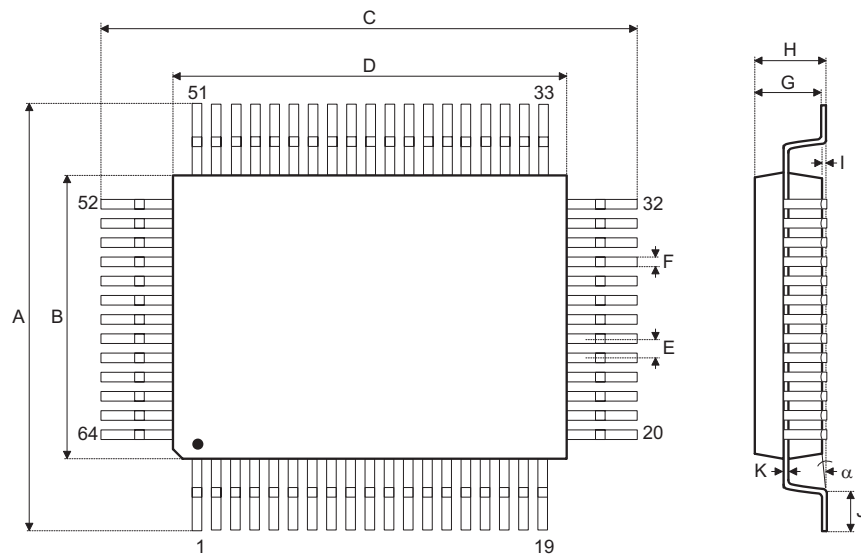
| Symbol | Dimensions in mil | | |
|----------|-------------------|------|------|
| | Min. | Nom. | Max. |
| A | 394 | — | 419 |
| B | 290 | — | 300 |
| C | 14 | — | 20 |
| C' | 697 | — | 713 |
| D | 92 | — | 104 |
| E | — | 50 | — |
| F | 4 | — | — |
| G | 32 | — | 38 |
| H | 4 | — | 12 |
| α | 0° | — | 10° |

48-pin SSOP (300mil) Outline Dimensions



| Symbol | Dimensions in mil | | |
|----------|-------------------|------|------|
| | Min. | Nom. | Max. |
| A | 395 | — | 420 |
| B | 291 | — | 299 |
| C | 8 | — | 12 |
| C' | 613 | — | 637 |
| D | 85 | — | 99 |
| E | — | 25 | — |
| F | 4 | — | 10 |
| G | 25 | — | 35 |
| H | 4 | — | 12 |
| α | 0° | — | 8° |

64-pin QFP (14x20) Outline Dimensions



| Symbol | Dimensions in mm | | |
|----------|------------------|------|-------|
| | Min. | Nom. | Max. |
| A | 18.80 | — | 19.20 |
| B | 13.90 | — | 14.10 |
| C | 24.80 | — | 25.20 |
| D | 19.90 | — | 20.10 |
| E | — | 1 | — |
| F | — | 0.40 | — |
| G | 2.50 | — | 3.10 |
| H | — | — | 3.40 |
| I | — | 0.10 | — |
| J | 1.15 | — | 1.45 |
| K | 0.10 | — | 0.20 |
| α | 0° | — | 7° |

Holtek Semiconductor Inc. (Headquarters)

No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
<http://www.holtek.com.tw>

Holtek Semiconductor Inc. (Taipei Sales Office)

4F-2, No. 3-2, YuanQu St., Nankang Software Park, Taipei 115, Taiwan
Tel: 886-2-2655-7070
Fax: 886-2-2655-7373
Fax: 886-2-2655-7383 (International sales hotline)

Holtek Semiconductor Inc. (Shanghai Sales Office)

7th Floor, Building 2, No.889, Yi Shan Rd., Shanghai, China 200233
Tel: 021-6485-5560
Fax: 021-6485-0313
<http://www.holtek.com.cn>

Holtek Semiconductor Inc. (Shenzhen Sales Office)

5/F, Unit A, Productivity Building, Cross of Science M 3rd Road and Gaoxin M 2nd Road, Science Park, Nanshan District, Shenzhen, China 518057
Tel: 0755-8616-9908, 8616-9308
Fax: 0755-8616-9533

Holtek Semiconductor Inc. (Beijing Sales Office)

Suite 1721, Jinyu Tower, A129 West Xuan Wu Men Street, Xicheng District, Beijing, China 100031
Tel: 010-6641-0030, 6641-7751, 6641-7752
Fax: 010-6641-0125

Holtek Semiconductor Inc. (Chengdu Sales Office)

709, Building 3, Champagne Plaza, No.97 Dongda Street, Chengdu, Sichuan, China 610016
Tel: 028-6653-6590
Fax: 028-6653-6591

Holmate Semiconductor, Inc. (North America Sales Office)

46729 Fremont Blvd., Fremont, CA 94538
Tel: 510-252-9880
Fax: 510-252-9885
<http://www.holmate.com>

Copyright © 2006 by HOLTEK SEMICONDUCTOR INC.

The information appearing in this handbook is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek's products are not authorized for use as critical components in life support devices or systems. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at <http://www.holtek.com.tw>.

