

**HT48RA0-2, HT48RA0-1
HT48RA1, HT48RA3, HT48RA5
Remote Type MCU
Handbook**

March 2005

Copyright © 2005 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	4
Selection Table	5
Block Diagram	6
Pin Assignment	7
Pin Description	7
Absolute Maximum Ratings	9
D.C. Characteristics	9
A.C. Characteristics	11
System Architecture	12
Clocking and Pipelining	12
Program Counter	13
Stack	15
Arithmetic and Logic Unit – ALU	16
Program Memory	17
Organization	17
Special Vectors	18
Managing Multiple Banks	18
Look-up Table	21
Data Memory	25
Organization	25
General Purpose Data Memory	25
Special Purpose Data Memory	26

Special Function Registers	27
Indirect Addressing Registers – IAR, IAR0, IAR1	27
Memory Pointers – MP, MP0, MP1	27
Bank Pointer – BP	28
Accumulator – ACC	29
Program Counter Low Register – PCL	29
Look-up Table Registers – TBLP, TBHP, TBLH	29
Watchdog Timer Register – WDTS	29
Status Register – STATUS	30
Interrupt Control Register – INTC	30
Timer/Event Counter Registers	31
Input/Output Ports and Control Registers	31
Input/Output Ports	32
Pull-high Resistors	32
Port A/Port B Wake-up	32
I/O Port Control Registers	32
Pin-shared Functions	33
Programming Considerations	35
Timer/Event Counters	36
Configuring the Timer/Event Counter Input Clock Source	37
Timer Registers – TMR0, TMR1L/TMR1H	38
Timer Control Registers – TMR0C, TMR1C	38
Configuring the Timer Mode	40
Configuring the Event Counter Mode	40
Configuring the Pulse Width Measurement Mode	41
Programmable Frequency Divider – PFD	41
Prescaler	42
I/O Interfacing	42
Programming Considerations	43
Carrier Generator	43
Interrupts	44
External Interrupt	45
Timer/Event Counter Interrupt	46
Interrupt Priority	46
Programming Considerations	46
Reset and Initialization	47
Reset	47
Oscillator	54
System Clock Configurations	54
System Crystal/Ceramic Oscillator	54
System RC Oscillator	55
Watchdog Timer Oscillator	55
HALT and Wake-up in Power Down Mode	55
Watchdog Timer	56
Configuration Options	59
Application Circuits	60

Part II Programming Language	65
Chapter 2 Instruction Set Introduction	67
Instruction Set	67
Instruction Timing	67
Moving and Transferring Data	68
Arithmetic Operations	68
Logical and Rotate Operations	68
Branches and Control Transfer	68
Bit Operations	69
Table Read Operations	69
Other Operations.....	69
Instruction Set Summary	69
Convention	69
Chapter 3 Instruction Definition	73
Chapter 4 Assembly Language and Cross Assembler	85
Notational Conventions	85
Statement Syntax	86
Name	86
Operation	86
Operand	86
Comment	87
Assembly Directives	87
Conditional Assembly Directives	87
File Control Directives	88
Program Directives	89
Data Definition Directives	92
Macro Directives	94
Assembly Instructions	96
Name	96
Mnemonic	96
Operand, Operator and Expression	96
Miscellaneous	98
Forward References	98
Local Labels	98
Reserved Assembly Language Words	99
Cross Assembler Options	100
Assembly Listing File Format	100
Source Program Listing	100
Summary of Assembly	101
Miscellaneous	101

Part III Development Tools	103
Chapter 5 MCU Programming Tools	105
HT-IDE Development Environment	105
Holtek In-Circuit Emulator – HT-ICE	106
HT-ICE Interface Card	106
OTP Programmer	107
OTP Adapter Card	107
System Configuration	107
HT-ICE Interface Card Settings	108
Installation	109
System Requirement	109
Hardware Installation	109
Software Installation	110
Chapter 6 Quick Start	115
Step 1 – Create a New Project	115
Step 2 – Add Source Program Files to the Project	115
Step 3 – Build the Project	115
Step 4 – Programming the OTP Device	115
Step 5 – Transmit Code to Holtek	116
Appendix	117
Appendix A Device Characteristic Graphics	119
Appendix B Package Information	127
Appendix C General Application Notes	135
System Oscillator	136
Crystal/Ceramic Oscillator	136
One-pin Pull-down RC Oscillator	138
Reset Circuit	139
External RES Line Description	139
Simple RC Reset Circuit	139
RC Circuit for Applications Operating in Noisy Environments	140
Low Voltage Reset Transistor Circuit	140
External Voltage Detector IC Reset Circuit	141
Internal POR Circuit and Internal Low Voltage Reset Circuit	141
Internal Watchdog RC Oscillator	142
Functional Description	142
Process, Working Voltage and Temperature Variations	142

Preface

Since the founding of the company, Holtek Semiconductor Inc. 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 is 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. The HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices within this series, have been designed specifically for remote control applications, and include an integrated carrier generator for development expediency. The HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices, also designed for remote control applications but with higher memory capacities, are particularly suitable for universal remote control applications. Holtek's high quality embedded Remote 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 increase 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 Remote 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 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 Remote 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 Remote 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 Remote Type microcontroller series.

Introduction

The HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1, HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 are 8-bit high performance, RISC architecture microcontroller devices specifically designed for remote control product applications. Device flexibility is enhanced with their internal special features such as HALT and wake-up functions, oscillator options, internal timers, 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 remote control consumer product applications. Many features are common to all devices, however, they differ in areas such as I/O pin count, RAM and ROM capacity, timers, etc.

The HT48RA0-2, HT48RA0-1, HT48RA1, HT48RA3 and HT48RA5 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 HT48CA0-2, HT48CA0-1, HT48CA1, HT48CA3 and HT48CA5 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:
 - 2.0V~3.6V (HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
 - 2.0V~5.5V (HT48RA1/HT48CA1, HT48RA3/HT48CA3, HT48RA5/HT48CA5)
- Power Consumption:
 - 0.7mA Typical at 3V 4MHz (HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
 - 3mA Typical at 3V 4MHz (HT48RA1/HT48CA1, HT48RA3/HT48CA3, HT48RA5/HT48CA5)
 - Maximum of 1 μ A Standby Current at 3V
- Cycle Time:
 - Up to 1 μ s Instruction Cycle with 4MHz System Clock
- Temperature Range:
 - Operating Temperature -40°C to 85°C (Industrial Grade)
 - Storage Temperature -50°C to 125°C

Kernel Features

- Program Memory:
 - 1K \times 14 OTP/Mask ROM (HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
 - 8K \times 16 OTP/Mask ROM (HT48RA1/HT48CA1)
 - 24K \times 16 OTP/Mask ROM (HT48RA3/HT48CA3)
 - 40K \times 16 OTP/Mask ROM (HT48RA5/HT48CA5)
- Data Memory:
 - 32 \times 8 RAM (HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
 - 224 \times 8 RAM (HT48RA1/HT48CA1, HT48RA3/HT48CA3, HT48RA5/HT48CA5)
- Table Read Function
- Hardware Stack:
 - 1-level (HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
 - 8-level (HT48RA1/HT48CA1, HT48RA3/HT48CA3, HT48RA5/HT48CA5)
- Direct and Indirect Data Addressing Mode
- Bit Manipulation Instructions
- 63 Powerful Instructions
- Most Instructions Implemented in 1 Machine Cycle

Peripheral Features

- From 15 to 23 Input, Output or Bidirectional I/O
- Pull-high Resistors on Input lines
- Wake-up Options
- External Interrupt Input (except HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
- Timer Functions (except HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
- Watchdog Timer (WDT)

- HALT and Wake-up Feature for Power Saving Operation
- PFD Output (except HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
- On-chip Crystal and RC Oscillator
- On-chip IR Carrier Generator (except HT48RA1/HT48CA1, HT48RA3/HT48CA3, HT48RA5/HT48CA5)
- 8-bit and 16-bit Programmable Timer/Event Counter with Overflow Interrupts (except HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1)
- 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

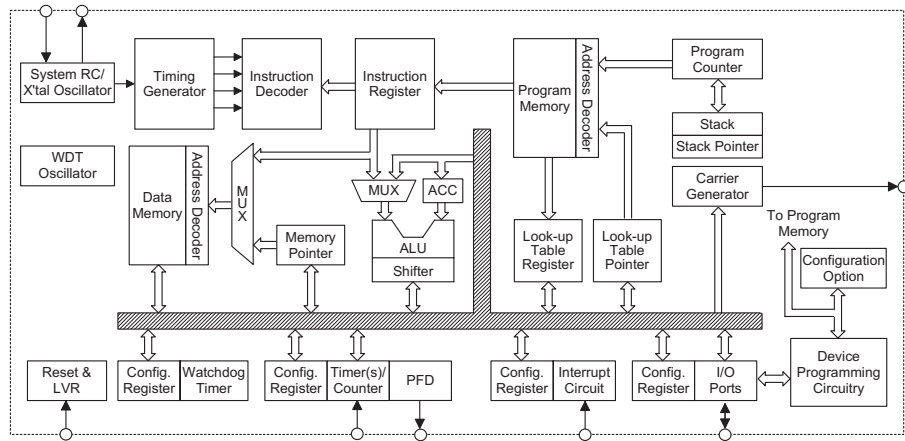
The Remote Type series MCUs include a comprehensive range of features, some of which are standard and some of which are device dependent. Most features are common to all devices, the main features distinguishing them are Program Memory, Data Memory capacity, I/O count and timer functions. To assist users 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.	Program Memory	Data Memory	I/O	Timer	PFD	Carrier Generator	Interrupt	Stack	Package Types
HT48RA0-2 HT48CA0-2	1K×14	32×8	10 I/O+ 4 I/P+ 1 O/P	—	—	√	—	1	20SOP, 20SSOP
HT48RA0-1 HT48CA0-1	1K×14	32×8	10 I/O+ 6 I/P+ 1 O/P	—	—	√	—	1	24SOP, 24SSOP
HT48RA1 HT48CA1	8K×16	224×8	23	8-bit×1 16-bit×1	√	—	3	8	28SOP, 28SSOP
HT48RA3 HT48CA3	24K×16	224×8	23	8-bit×1 16-bit×1	√	—	3	8	28SOP, 28SSOP
HT48RA5 HT48CA5	40K×16	224×8	23	8-bit×1 16-bit×1	√	—	3	8	28SOP, 28SSOP

Note Part numbers including "C" are mask version devices while "R" are OTP devices.

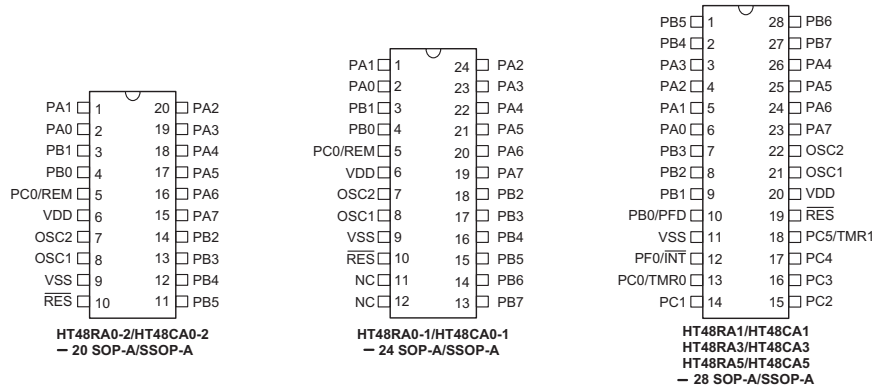
Block Diagram

The following block diagram illustrates the main functional blocks of the Remote Type microcontroller series of devices.



- Note**
1. This block diagram represents the OTP devices, for the mask device there is no Device Programming Circuitry.
 2. The Programmable Timer/Event Counter function does not exist in the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices.
 3. The Carrier Generator only exists in the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices.
 4. The PFD does not exist in the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices.
 5. The interrupt circuit does not exist in the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices.
 6. For the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, there's no dedicated WDT oscillator.

Pin Assignment



Pin Description

HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1

Pin Name	I/O	Configuration Option	Description
PA0~PA7	I/O	—	Bidirectional 8-bit input/output port with pull-high resistors. Software instructions determine if the pin is an NMOS output or Schmitt Trigger input.
PB0, PB1	I/O	Wake-up	Bidirectional 2-bit input/output lines with pull-high resistors. Each individual bit can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is an NMOS output or Schmitt Trigger input.
PB2~PB7	I	Wake-up	6-bit Schmitt Trigger input lines with pull-high resistors. Each individual bit can be configured as a wake-up input by a configuration option.
PC0/REM	O	Carrier Output	Level or carrier output pin. PC0 can be configured as a CMOS output pin or carrier output pin by configuration option.
OSC1 OSC2	I O	Crystal or RC	OSC1, OSC2 are connected to an external resistor 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.
RES	I	—	Schmitt Trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground

- Note**
1. Each pin on PB can be programmed through a configuration option to have a wake-up function.
 2. The input pins PB6 and PB7 are not present in the HT48RA0-2/HT48CA0-2 devices.
 3. A pull-high resistor is permanently connected to each of the PA and PB port pins.

HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5

Pin Name	I/O	Configuration Option	Description
PA0~PA7	I/O	Pull-high Wake-up	Bidirectional 8-bit input/output port. Each individual bit can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins have pull-high resistors. Individual pins cannot be selected to have pull-high resistors.
PB0/PFD PB1~PB7	I/O	Pull-high PFD	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine whether the four pins PB0~PB3 and PB4~PB7 have pull-high resistors. Individual pins cannot be selected to have pull-high resistors. Pin PB0 is pin-shared with PFD, the function of which is selected via a configuration option.
PC0/TMR0 PC1~PC4 PC5/TMR1	I/O	Pull-high	Bidirectional 6-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have a pull-high resistor. PC0 and PC5 are pin-shared with the external timer pins TMR0 and TMR1 respectively.
PF0/ $\overline{\text{INT}}$	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 a pull-high resistor is connected to this pin. PF0 is pin-shared with the external interrupt pin $\overline{\text{INT}}$.
OSC1 OSC2	I O	Crystal or RC	OSC1, OSC2 are connected to an external resistor 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.
$\overline{\text{RES}}$	I	—	Schmitt Trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground

Note Each pin on PA can be programmed through a configuration option to have a wake-up function.

Absolute Maximum Ratings

Supply Voltage (HT48RA0-2/HT48CA0-2, HT48RA0-1/HT48CA0-1) $V_{SS}-0.3V$ to $V_{SS}+4.0V$
 Supply Voltage (HT48RA1/HT48CA1, HT48RA3/HT48CA3, HT48RA5/HT48CA5)
 $V_{SS}-0.3V$ 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

HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V _{DD}	Conditions				
V _{DD}	Operating Voltage	—	—	2.0	—	3.6	V
I _{DD}	Operating Current	3V	No load, f _{SYS} =4MHz	—	0.7	1.5	mA
I _{STB}	Standby Current	3V	No load, system HALT	—	—	1	μA
V _{IL1}	Input Low Voltage for I/O Ports	3V	—	0	—	0.3V _{DD}	V
V _{IH1}	Input High Voltage for I/O Ports	3V	—	0.7V _{DD}	—	V _{DD}	V
V _{IL2}	Input Low Voltage (\overline{RES})	3V	—	0	—	0.4V _{DD}	V
V _{IH2}	Input High Voltage (\overline{RES})	3V	—	0.9V _{DD}	—	V _{DD}	V
V _{LVR}	Low Voltage Reset Voltage	—	—	—	1.9	2.0	V
I _{OL}	I/O Port Sink Current	3V	V _{OL} =0.1V _{DD}	4	8	—	mA
I _{OH}	PC0/REM Port Source Current	3V	V _{OH} =0.9V _{DD}	-2	-4	—	mA
R _{PH}	Pull-high Resistance	3V	—	20	60	100	kΩ

HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V _{DD}	Conditions				
V _{DD}	Operating Voltage	—	—	2.0	—	5.5	V
I _{DD1}	Operating Current	3V	No load, f _{sys} =4MHz	—	0.6	1.5	mA
		5V		—	2	4	
I _{DD2}	Operating Current (Crystal OSC, RC OSC)	5V	No load, f _{sys} =8MHz	—	4	8	mA
I _{STB1}	Standby Current (WDT Enabled, WDT RC OSC On)	3V	No load, system HALT	—	1.1	5	μA
		5V		—	4	10	
I _{STB2}	Standby Current (WDT Disabled)	3V	No load, system HALT	—	0.1	1	μA
		5V		—	0.2	2	
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 ($\overline{\text{RES}}$)	—	—	0	—	0.4V _{DD}	V
V _{IH2}	Input High Voltage ($\overline{\text{RES}}$)	—	—	0.9V _{DD}	—	V _{DD}	V
V _{LVR}	Low Voltage Reset Voltage	—	LVR=2.0V	1.8	1.9	2.0	V
			LVR=3.0V	2.7	3.0	3.3	V
I _{OL}	I/O Port Sink Current	3V	V _{OL} =0.1V _{DD}	4	8	—	mA
		5V		10	20	—	
I _{OH}	I/O Port Source Current	3V	V _{OH} =0.9V _{DD}	-2	-4	—	mA
		5V		-5	-10	—	
R _{PH}	Pull-high Resistance	3V	—	20	60	100	kΩ
		5V		10	30	50	

A.C. Characteristics
HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V _{DD}	Conditions				
f _{SYS}	System Clock	3V	—	400	—	4000	kHz
t _{RES}	External Reset Low Pulse Width	—	—	1	—	—	μs
t _{SST}	System Start-up Timer Period	—	Power-up reset or Wake-up from HALT	—	1024	—	*t _{SYS}
t _{LVR}	Low Voltage Width to Reset	—	—	1	—	—	ms

 *t_{SYS} = 1/f_{SYS}
HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V _{DD}	Conditions				
f _{SYS1}	System Clock (Crystal OSC)	—	2.0V~5.5V	400	—	4000	kHz
		—	3.3V~5.5V	400	—	8000	kHz
f _{SYS2}	System Clock (RC OSC)	—	2.0V~5.5V	400	—	4000	kHz
		—	3.3V~5.5V	400	—	8000	kHz
t _{TIMER}	Timer I/P Frequency (TMR0/TMR1)	3V	50% duty	0	—	4000	kHz
		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 (f _{SYS} /4)	3V	Without WDT prescaler	—	1024	—	*t _{SYS}
t _{RES}	External Reset Low Pulse Width	—	—	1	—	—	μs
t _{SST}	System Start-up Timer Period	—	Power-up reset or Wake-up from HALT	—	1024	—	*t _{SYS}
t _{LVR}	Low Voltage Width to Reset	—	—	1	—	—	ms
t _{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
t _{ACC}	Data ROM Access Time	—	—	1	—	—	μs

 *t_{SYS} = 1/f_{SYS1} or 1/f_{SYS2}

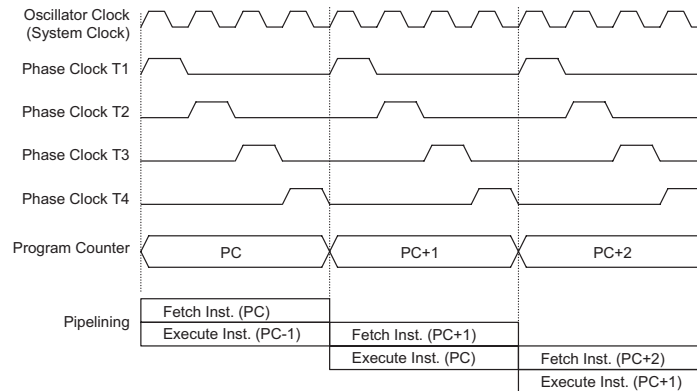
System Architecture

A key factor in the high performance features of the Holtek range of Remote 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 remote control system with maximum reliability and flexibility. This makes these devices suitable for low cost, high-volume production of remote control applications requiring from 1K up to 40K words of program memory and from 32 to 224 bytes of data storage.

Clocking and Pipelining

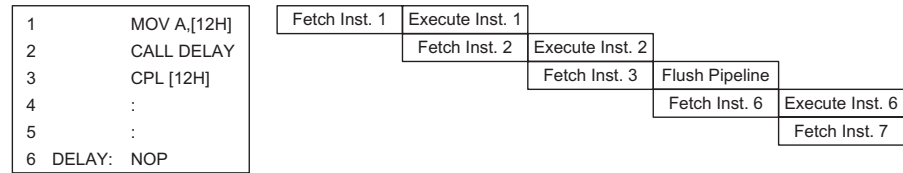
The main system clock, derived from either a Crystal/Resonator or RC oscillator is subdivided into four internally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry 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 Remote Type 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 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.

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.

HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1

Mode	Program Counter Bits									
	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Initial Reset	0	0	0	0	0	0	0	0	0	0
Skip	Program Counter + 2									
Loading PCL	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

HT48RA1/HT48CA1

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	

HT48RA3/HT48CA3

Mode	Program Counter Bits														
	b14	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	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	0	1	0	0	0
Timer/Event Counter 1 Overflow	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Skip	PC14~PC13, (PC12~PC0 + 2): (within current bank)														
Loading PCL	PC 14	PC 13	PC 12	PC 11	PC 10	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#14	#13	#12	#11	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

HT48RA5/HT48CA5

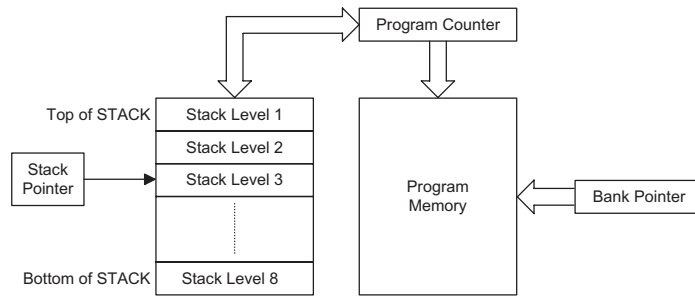
Mode	Program Counter Bits															
	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Initial Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
External Interrupt	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Timer/Event Counter 0 Overflow	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Timer/Event Counter 1 Overflow	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Skip	PC15~PC13, (PC12~PC0 + 2): (within current bank)															
Loading PCL	PC 15	PC 14	PC 13	PC 12	PC 11	PC 10	PC 9	PC 8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#15	#14	#13	#12	#11	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

- Note**
1. PC15~PC8: Current Program Counter bits
 2. @7~@0: PCL bits
 3. #15~#0: Instruction code bits
 4. S15~S0: Stack register bits
 5. Program memory in excess of 8K, that is memory area controlled by PC13, PC14 and PC15 is only accessible via the bank pointer.

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 either 1 or 8 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 SP 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 instruction 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 HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1, only 1 stack level is available.
 2. For the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5, 8 levels of stack are available.
 3. Only the HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices have Bank Pointers.
-

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

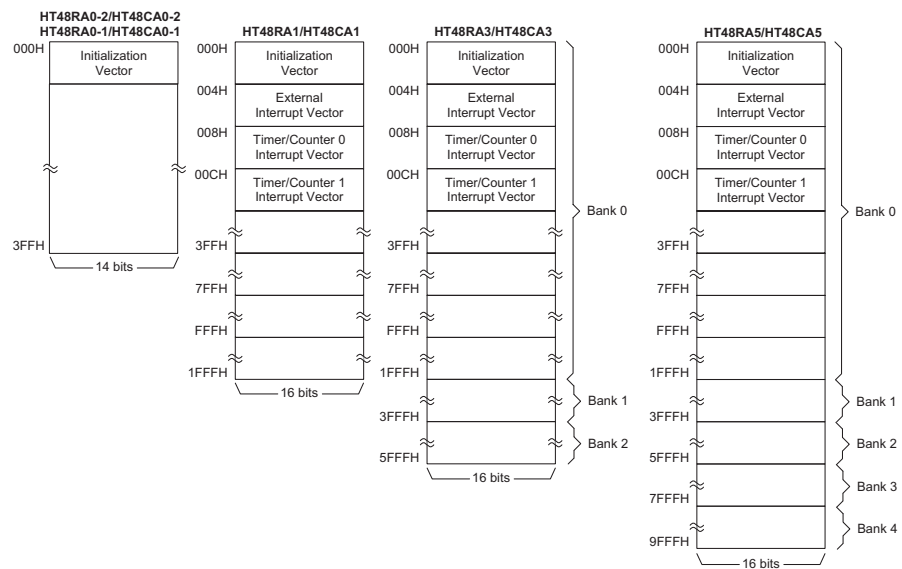
Program Memory

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 40K 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 single or pair of separate table pointer register depending upon which device is selected. For the HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices, which have memory capacities of 24K and 40K respectively, the Program Memory area is divided into several banks, each of which has a capacity of 8K.

The following diagram shows the Program Memory Organization for the Remote Type microcontroller 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
With the exception of HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, this vector is used by the external interrupt. 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.
- Location 008H
With the exception of HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, this internal interrupt vector is used by the Timer/Event Counter 0. If a counter overflow occurs, the program will jump to this location and begin execution if the internal interrupt is enabled and the stack is not full.
- Location 00CH
With the exception of HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, this internal interrupt vector is used by the Timer/Event Counter 1. If a counter overflow occurs, the program will jump to this location and begin execution if the internal interrupt is enabled and the stack is not full.

Managing Multiple Banks

For the HT48RA3/HT48CA3 and HT48RA5/HT48CA5 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 is best achieved using the BANK directive as shown. 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 HT48RA3/HT48CA3 devices illustrates how to use the "CALL" and "JMP" instructions between different banks:

```
include HT48RA3.inc
:
rombank 0 codesec0           ; define rombank 0
rombank 1 codesec1           ; define rombank 1
rombank 2 codesec2           ; define rombank 2
:
codesec0 .section at 000h 'code' ; locates following program section
; into Bank 0
```

```

clr bp                                ; re-initializing the BP
jmp start
:
:
start:
:
:
lab0:
:
:
mov a, BANK routb2                    ; routine "routb2" is located in Bank 2
mov bp, a                              ; load bank number for routb2 into BP
call routb2                             ; call subroutine located in Bank 2
clr bp                                  ; program will return to this location
:                                       ; after RET in Bank 2
:                                       ; but BP will retain Bank 2 value
:                                       ; so clear the BP
mov a, BANK lab1                       ; lab1 is located in Bank 1
mov bp, a                              ; load bank number for lab1 into BP
jmp lab1                                ; as bank pointer has been setup
:                                       ; program will jump to lab1
:
:
codesecl .section at 000h 'code' ; locates following program section
:                                       ; into Bank 1
:
lab1:
:
:
mov a, BANK lab3                       ; lab3 is located in Bank 3
mov bp, a                              ; load bank number for lab3 into BP
jmp lab3                                ; as bank pointer has been setup
:                                       ; program will jump to lab3
codesecl2 .section at 000h 'code' ; locates following program section
:                                       ; into Bank 2
:
:
routb2 proc
:
:
ret                                     ; return program to Bank 0 but BP will keep
:                                       ; Bank 2 value
routb2 endp
:
:
codesecl3 .section at 000h 'code' ; locates following program section into
:                                       ; Bank 3
:
:
lab3:
mov a, BANK lab0                       ; lab0 is located in Bank 0
mov bp, a                              ; as bank pointer has been setup
jmp lab0                                ; program will jump to lab0

```

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 a timer 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 also clear the Bank Pointer and backup its original value immediately 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 correct Bank and point from where the

subroutine was called. The following example illustrates how interrupts can be managed:

```

include HT48RA3.inc
:
:
rombank 0 codesec0           ; define rombank 0
rombank 1 codesec1           ; define rombank 1
rombank 2 codesec2           ; define rombank 2
:
:
codesec0 .section at 000h 'code' ; locates 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
mov a,bp_exti                  ; restore bank pointer
mov bp,a
mov a,accbuf0                  ; 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
:
:
mov a,statusbuf1               ; restore status register
mov status,a
mov a,bp_tmr0                  ; restore bank pointer
mov bp,a
mov a,accbuf1                  ; restore accumulator
reti                          ; return to main program and original
:                               ; calling bank
:
:

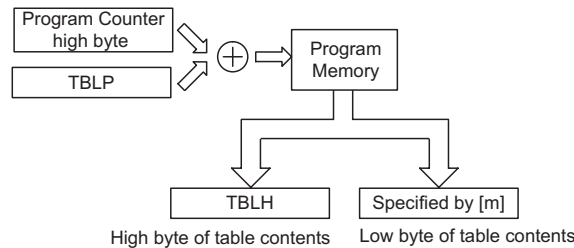
```

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.

For the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, there is 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".

The following diagram illustrates the addressing/data flow of the look-up table for the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices:



The following example shows how the table pointer is defined and table data retrieved from the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices. 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 "300" hex which refers to the start address of the last page within the 1K Program Memory of the HT48RA0-2/HT48CA0-2 or HT48RA0-1/HT48CA0-1 microcontrollers. The table pointer is setup here to have an initial value of 06 hex. This will ensure that the first data read from the data table will be at the Program Memory address 306 hex 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 low byte of the table data will be transferred to the specified register while the high byte of the table data, which in this case is equal to zero, will be transferred to the TBLH register.

```

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 "1A" is transferred to
                  ; tempreg1
                  ; and data "0F" to register tempreg2
                  ; the value "0" will be transferred to the high byte
                  ; register TBLH
:
:

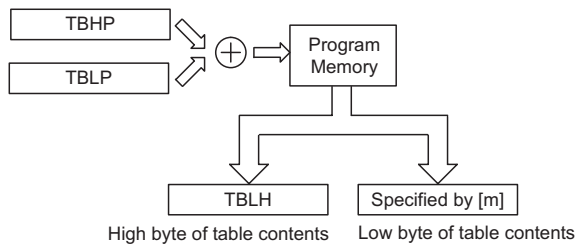
org 300h          ; sets initial address of last page

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

```

For the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 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 Program Memory 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. Any unused bits in this transferred higher order byte will be read as "0".

The following diagram illustrates the addressing/data flow of the look-up table for the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices:



The following example shows how the table pointer is defined and table data retrieved from the HT48RA3/HT48CA3 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 2 which in this case is "4000H". The table pointer high byte is setup to have a value of "40H" 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 "4005H", or 5 locations after the first address defined by the ORG statement. When the "TABRDC" 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 HT48RA3.inc
:
:
data      .section   'data'
temp     db ?
:
rombank 0 codesec0      ; Bank 0 definition
rombank 1 codesec1      ; Bank 1 definition
rombank 2 codesec2      ; Bank 2 definition
:
codesec0 .section at 0 'code'
jmp start
:
org 010h

start:
:
:
mov      a,040h          ; setup table high byte address
mov      tbhp,a
:
:
mov      a,005h          ; setup table low byte address
mov      tblp,a          ; table pointer address is now 4005H
tabrdc  temp             ; read table data from PC address 4005
nop      ; "FF" will be placed in temp register
:                               ; and 55 will be placed in TBLH register

codesec1 .section at 000h 'code' ; Bank 1 code located here
codesec2 .section at 000h 'code' ; Bank 2 code located here
:
:
org 0000h                ; this defines the offset from the start
:                               ; address of Bank 2 which is 4000H

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.

HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1

Instruction	Table Location Bits									
	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
TABRDC [m]	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL [m]	1	1	@7	@6	@5	@4	@3	@2	@1	@0

HT48RA1/HT48CA1

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

HT48RA3/HT48CA3

Instruction	Table Location Bits														
	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
TABRDC [m]	TBHP	TBHP	TBHP	TBHP	TBHP	TBHP	TBHP	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL [m]	1	0	1	1	1	1	1	@7	@6	@5	@4	@3	@2	@1	@0

HT48RA5/HT48CA5

Instruction	Table Location Bits															
	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
TABRDC [m]	TBHP	TBHP	TBHP	TBHP	TBHP	TBHP	TBHP	TBHP	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL [m]	1	0	0	1	1	1	1	1	@7	@6	@5	@4	@3	@2	@1	@0

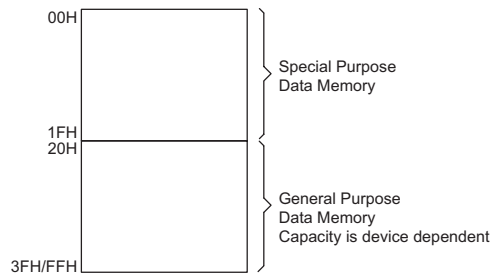
- Note**
1. PC15~PC8: Current Program Counter bits
 2. @7~@0: Table Pointer TBLP bits
 3. For the HT48RA5/HT48CA5, the Table address location is 16 bits wide, i.e. from b15~b0.
 4. For the HT48RA3/HT48CA3, the Table address location is 15 bits wide, i.e. from b14~b0.
 5. For the HT48RA1/HT48CA1, the Table address location is 13 bits wide, i.e. from b12~b0.
 6. For the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1, the Table address location is 10 bits wide, 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.

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 3FH for the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, and FFH for the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 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 registers MP, 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.

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 table pointers, status, 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 HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, one Indirect Addressing Register, IAR, and one Memory Pointer, MP, is provided. For the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 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 itself using an indirect method will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointers – MP, MP0, MP1

For the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, one memory pointer known as MP is provided, whereas for the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 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 HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 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 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 adres1     ; 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

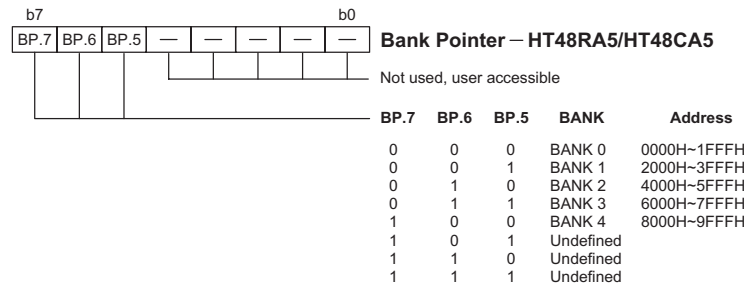
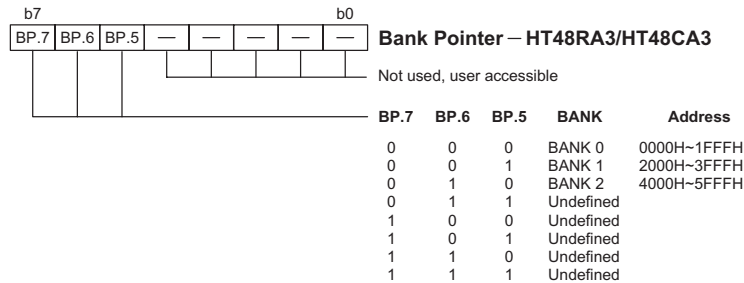
continue:

```

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

Bank Pointer – BP

The Bank Pointer only exists in the HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices. The extended program memory capacity of 24K and 40K for the HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices respectively, require the use of a Bank Pointer to obtain access to memory capacity in excess of 8K. The Program Memory is therefore subdivided into several memory banks each of which contains 8K×16 address space. It is therefore important to ensure that the correct program memory bank is selected by loading the Bank Pointer with the correct value before accessing memory in excess of 8K when instructions such as "JMP" or "CALL" are executed. The Bank Pointer will be cleared to "00H" when any type of reset situation occurs except for a WDT time-out reset during HALT, in which case it will remain unchanged.



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 and TBHP are the table low-byte and high-byte pointers respectively which indicate the address where the look-up table data is located. Note that the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices do not contain a TBHP table high-byte pointer. Their value must be setup before any table read commands are executed. The address of the data to be accessed in the Program Memory table must be first setup by loading the appropriate table address into the TBLP and if applicable the TBHP registers before any table read commands are executed. Their value can be changed, for example using the "INC" or "DEC" instructions, allowing for easy table data pointing and reading. 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 WDTS register only exists in the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices. 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.

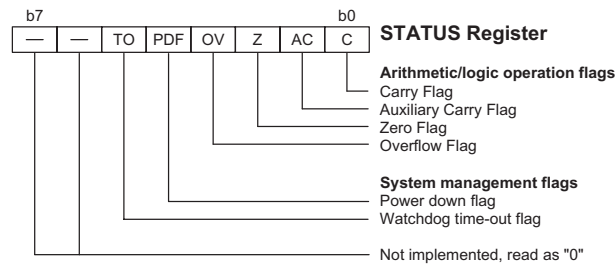
Status Register – STATUS

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 Register – INTC

The INTC register only exist in the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices. This 8-bit register controls the operation of both external and internal interrupts. By setting various bits within this register using standard bit manipulation instructions, the enable/disable function of the external interrupt and each of the internal interrupts can be independently controlled. A master interrupt bit within this register, the EMI bit, acts like a global en-

able/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

The HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices have no internal programmable timers, while the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices contain two integrated Timer/Event Counters, one 8-bit and one 16-bit counter. The 8-bit timer stores its timer value in the TMR0 register and also has an associated control register known as TMR0C. The 16-bit timer stores its timer value in the TMR1L/TMR1H register pair and also has an associated control register known as TMR1C.

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 the exception of the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, each I/O port of the other devices have 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.

Input/Output Ports

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

Depending upon which device or package is chosen, the Remote MCU devices provide a range of both input and bidirectional input/output lines labeled with port names PA, PB, PC, etc. These input and 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 dedicated input ports or for I/O ports that are setup as inputs, note that 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 of the input and I/O pins on the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices are permanently connected to pull high resistors. On the other devices, the connection of pull-high resistors to the I/O pins is determined by configuration options. All pull-high resistors are implemented using a weak PMOS transistor.

Port A/Port B 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 or Port B pins, depending upon which device is used, 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 or Port B changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Depending upon which device is chosen, note that each pin on Port A or Port B can be selected individually via configuration options to have this wake-up feature.

I/O Port Control Registers

Within the area of Special Function Registers, the input and I/O registers, and if applicable, their associated control registers, play a prominent role. All input and I/O ports have a designated register correspondingly labeled as PA, PB, PC, etc. These 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 the exception of the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, which have no CMOS I/O pins, each I/O port of the other devices have an associated control register labeled PAC, PBC, PCC, etc., also mapped to specific addresses within 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.

For the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, there are no port control registers. Setting up these pins as inputs is achieved by first setting its output high which effectively places its transistor in a high impedance state allowing the pin to be now used as an input. Note that as bits PB.6 and PB.7 on the HT48RA0-2/HT48CA0-2 devices do not exist, any value read from these bits will return a "0" value.

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.

→ PFD Output

With the exception of the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, which have no PFD function, the PFD pin for the other devices is pin-shared with I/O pin PB0. The output function of this pin is chosen via a configuration option and remains fixed after the device is programmed. Note that the corresponding bit of the port control register, PBC.0, must setup the pin as an output to enable the PFD output. If the PBC port control register has setup the pin as an input, then the pin will function as a normal logic input, even if the PFD configuration option has been selected.

→ External Interrupt Input

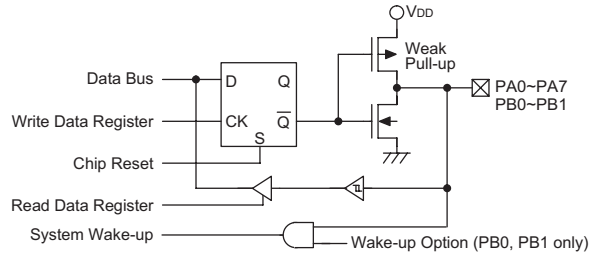
The HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices have no external interrupt function and therefore no external interrupt pin. The HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices, have an external interrupt pin \overline{INT} which is pin-shared with the I/O pin PF0.

→ External Timer Clock Input

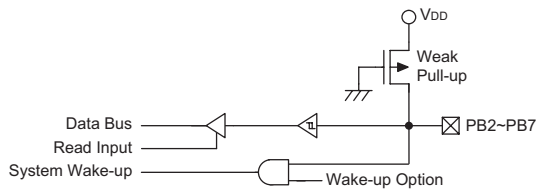
The HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices each contain one 8-bit timer and one 16-bit timer. Both the 8-bit and 16-bit timers have external timer input control pins known as TMR0 and TMR1 respectively. These external timer input control pins, TMR0 and TMR1, are pin-shared with pins PC0 and PC5 respectively. If these pins are to be configured as timer inputs, the corresponding control bits in the timer control register must be correctly set. These external timer 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 (internal clock source) to prevent the I/O from interfering with the timer operation.

→ **REM Output**

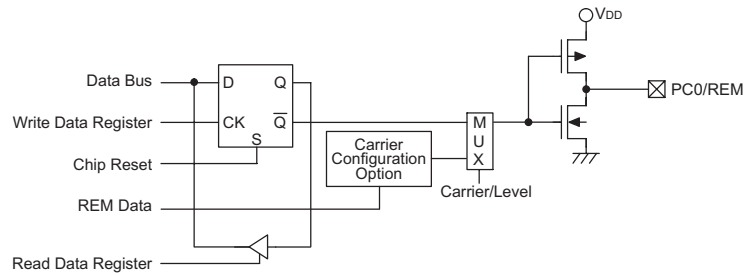
The HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices each contain a carrier function whose output is known as REM and which is pin-shared with output pin PC0. The output function of this pin is chosen via a configuration option and remains fixed after the device is programmed.



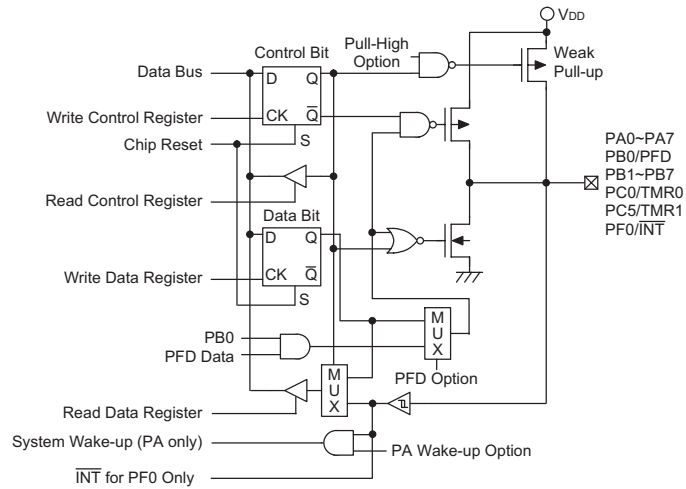
PA, PB0, PB1 Input/Output Lines – HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1



PB2~PB7 Input Lines – HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1



PC0/REM Output Lines – HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1

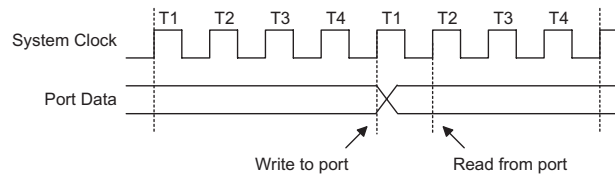


Input/Output Ports – HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5

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 registers and, if applicable, 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, if applicable, whether pull-high options have been selected.

If the port control registers, PAC, PBC, etc., in the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices 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. For the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices, 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. In the case of the NMOS type pins in the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, there are some special considerations that must be noted. For an NMOS pin that is set high by the microcontroller, i.e. placed into a high impedance condition, but driven low by externally connected circuitry, this pin would be read as being in a low condition during the read phase of the "SET [m].i" and "CLR [m].i" instructions. When the ensuing write phase occurs, this pin, having been read as being in a low condition during the read phase, would then be consequently erroneously set low. For this reason great care must be taken when using these bit control instructions.



Depending upon which device is used, either Port A or Port B 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 these applicable Port A or Port B pins which have wake-up function.

Timer/Event Counters

As the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 do not contain internal timers, note that this section does not apply to these devices.

The provision of internal timers form an important part of any microcontroller giving the designer a means of carrying out time related functions. Although the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices do not possess any internal Timer/Event Counters, the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices contain two internal Timer/Event Counters, namely one 8-bit timer and one 16-bit timer. 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. In the case of 8-bit timers, the provision of an internal 8-stage prescaler in the timer clock circuitry gives added range to the 8-bit timers.

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. The timer clock source for the 8-bit timer can be configured to come from the internal system clock source f_{SYS} or alternately, from the external timer input pin TMR0. The timer clock source for the 16-bit timer can be configured to come from the internal system clock source divided by 4, namely $f_{SYS}/4$ or alternately from the external timer input pin TMR1.

	HT48RA1/HT48CA1, HT48RA3/HT48CA3, HT48RA5/HT48CA5
8-bit Timer	1
Timer Register Name	TMR0
Timer Control Register	TMR0C
16-bit Timer	1
Timer Pair Register Name	TMR1L/TMR1H
Timer Control Register	TMR1C

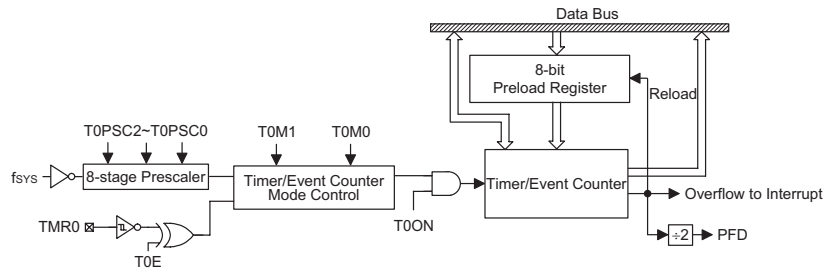
Note The HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices have no timer function.

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 TMR0 or TMR1, which are pin-shared with the PC0 and PC5 I/O pins respectively. Depending upon the condition of the T0E/T1E 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.

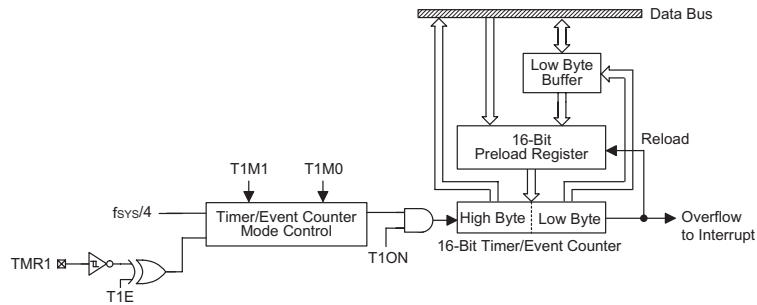
Configuring the Timer/Event Counter Input Clock Source

The internal timer’s clock source can originate from either the system clock or from an external clock source. The system clock input timer source is used when the timer is in the timer mode or in the pulse width measurement mode. For the 8-bit timers, the internal timer clock also passes through a prescaler, the value of which is conditioned by the TMR0C Timer Control Register bits, TOPSC2, TOPSC1 and TOPSC0.

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, TMR0 or TMR1 depending upon which timer is used. Depending upon the condition of the T0E/T1E 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 – HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 TMR0



16-bit Timer/Event Counter Structure – HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 TMR1

Timer Registers – TMR0, TMR1L/TMR1H

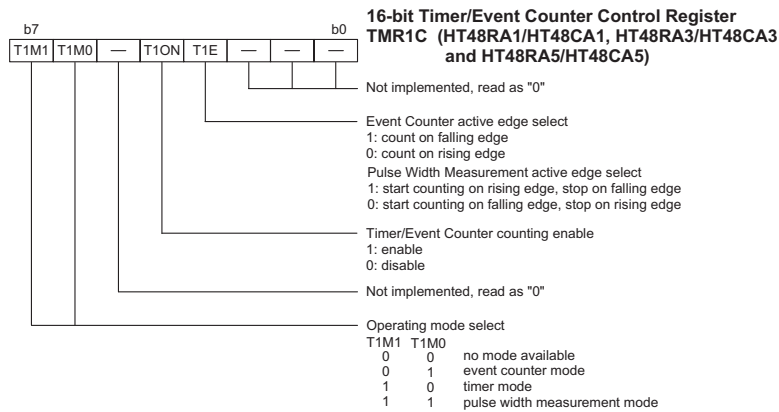
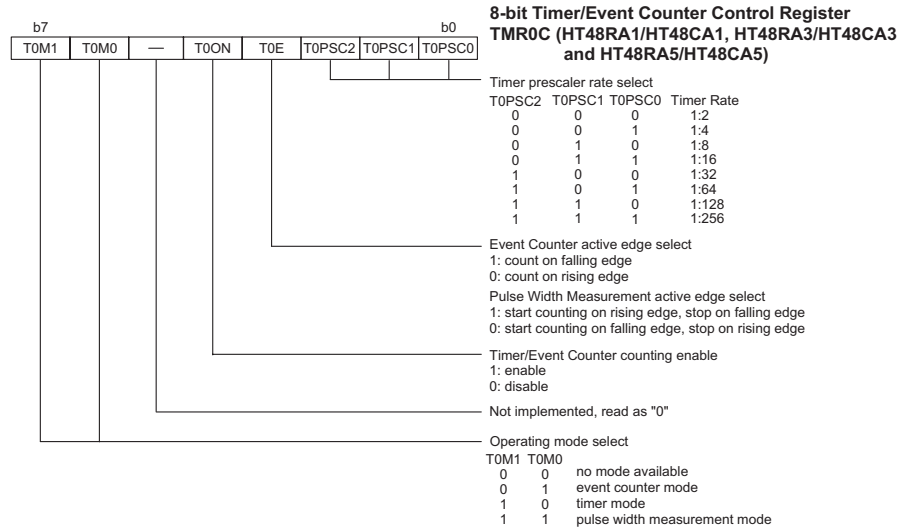
The timer registers are special function registers located in the special purpose Data Memory and is the place where the actual timer value is stored. For the 8-bit timer, this register is known as TMR0. In the case of the 16-bit timer, a pair of 8-bit registers, known as TMR1L and TMR1H, are required to store the 16-bit timer value. 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 timer, at which point the timer overflows and an internal timer 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 the 16-bit timer, which has both a low byte and a high byte timer register, 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 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 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 – TMR0C, TMR1C

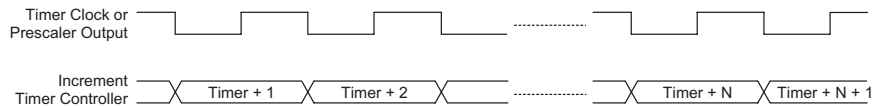
The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of their respective control register. The HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices contain two internal timers, one 8-bit timer, whose control register is known as TMR0C and one 16-bit timer whose control register is known as TMR1C. 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 their appropriate timer control register is fully programmed with the right data to ensure their 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 TOM1/TOM0 and T1M1/T1M0 must be set to the required logic levels. The timer-on bit T0ON/T1ON or bit 4 of the Timer Control Register provides the basic on/off control of the timer, setting the bit high allows the counter to run, clearing the bit stops the counter. For the 8-bit Timer/Event Counter, bits 0~2 of the TMR0C 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 the T0E/T1E or bit 3 of the TMR0C or TMR1C register.

Configuring the Timer Mode

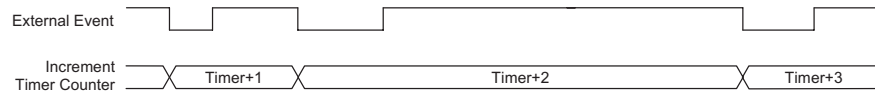
In this mode, the timer can be utilized to measure fixed time intervals, providing an internal interrupt signal each time the counter overflows. To operate in this mode, bits T0M1/T1M1 (bit7) and T0M0/T1M0 (bit6) of the TMRC register must be set to "1" and "0" respectively. In this mode, the internal clock is used as the timer clock. For the 8-bit Timer/Event Counter, the input clock frequency to the timer is f_{SYS} divided by the value programmed into the timer prescaler, the value of which is determined by bits T0PSC2~T0PSC0 of the TMR0C register. For the 16-bit Timer/Event Counter, the input clock frequency to the timer is $f_{SYS}/4$. There is no prescaler function for the 16-bit timer. The timer-on bit, T0ON/T1ON 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. It should be noted that a timer overflow is one of the interrupt and wake-up sources.



Timer Mode Timing Chart

Configuring the Event Counter Mode

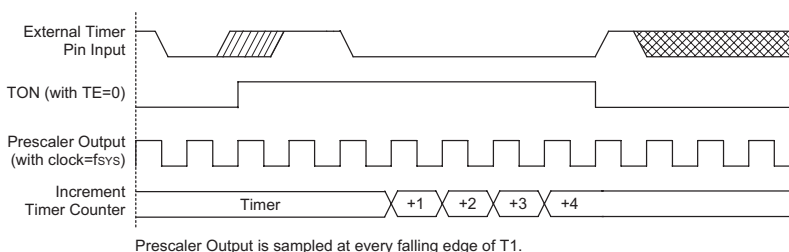
In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the internal timer. For the timer to operate in the event counting mode, bits T0M1/T1M1 and T0M0/T1M0 of the TMRC register must be set to "0" and "1" respectively. The timer-on bit, T0ON/T1ON must be set high to enable the timer to count. With T0E/T1E low, the counter will increment each time the external timer pin receives a low to high transition. If T0E/T1E 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 preload 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 an event counter input pin, two things have to happen. The first is to ensure that the T0M0/T1M0 and T0M1/T1M1 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, bits T0M0/T1M0 and T0M1/T1M1 must both be set high. If the T0E/T1E bit 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 T0ON/T1ON bit will be automatically reset to zero and the timer will stop counting. If the T0E/T1E 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 T0ON/T1ON 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 T0ON/T1ON 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 T0ON/T1ON 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 pin external timer pin. As the T0ON/T1ON bit has now been reset any further transitions on the external timer pin will be ignored. Not until the T0ON/T1ON 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 T0M0/T1M0 and T0M1/T1M1 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.

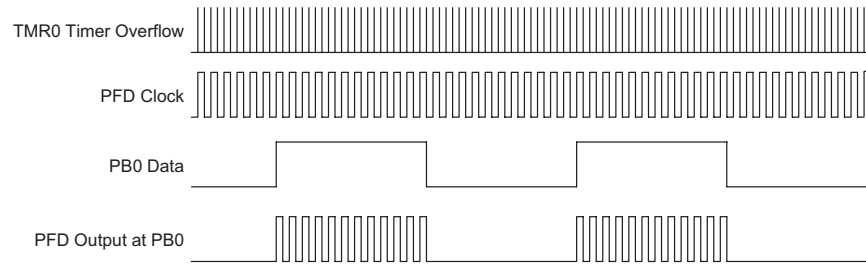


Pulse Width Measurement Mode Timing Chart

Programmable Frequency Divider – PFD

As the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices do not contain a PFD function, note that this section does not apply to these devices. For the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices, each contains a PFD function which is driven by the 8-bit counter TMR0. By controlling the frequency of the overflow signal from TMR0, the required remote control signal can be generated on the PFD pin. Note that a Carrier Generator and associated REM pin is provided on the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices for the generation of the remote control signal.

The PFD output is pin-shared with the I/O pin PB0. The PFD function is selected via a configuration option, however, if not selected the pin can operate as a normal I/O pin. The timer overflow signal from the TMR0 8-bit counter is the clock source for the PFD circuit. The counter is driven by the f_{SYS} clock source or the external timer pin and has an initial value controlled by the value written into the registers. The counter will begin to count-up from this preload register value until full, at which point an overflow signal is generated, which is the clock source for the divide-by-two circuit which drives the PFD output. The counter will then be automatically reloaded with the preload register value and continue counting-up. The PFD frequency will therefore be half the frequency of the timer overflow signal. Refer to the relevant Timer/Event Counters section for details of its settings and operations. The PFD output 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 PFD output. Note that the PFD output will be low if the PB0 output data bit is cleared to "0". For the PFD output to function, it is also necessary to ensure that the PBC.0 bit in the PBC port control register is cleared to "0" to configure the pin as an output. If the PBC.0 bit is set to "1" the PB0 pin will function as an input even if the configuration options have selected the pin to be a PFD output.



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

For the 8-bit Timer/Event Counter, bit 0~2 of the TMR0C Timer Control Register can be used to define the pre-scaling stages of the internal clock sources of the Timer/Event Counter. The Timer/Event Counter overflow signal can be used to generate PFD signals and as a Timer Interrupt.

I/O Interfacing

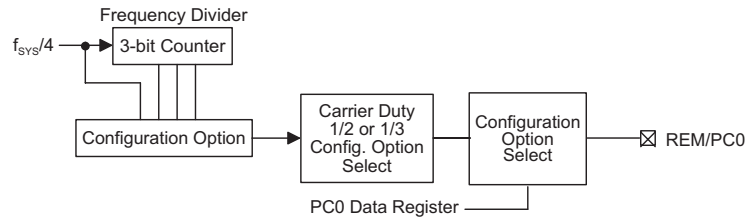
The Timer/Event Counters when configured to run in the event counter or pulse width measurement mode, require the use of the external timer pin for correct operation. These external timer pins are pin-shared with other Port C input pins, which have the option of being connected to pull-high resistors via a configuration option. The 8-bit timer can also be setup to drive the pin-shared PFD pin. When the PFD pin is selected by choosing the correct configuration option, the output of the 8-bit timer can be made to drive this pin at a frequency determined by the contents of the timer register and the source clock frequency.

Programming Considerations

When configured to run in the timer mode, the internal system clock source 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, one of the internal system clock sources 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.

Carrier Generator

As only the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices contain an internal carrier generator, note that this section only applies to these devices. All remote control transmitter applications require a carrier frequency generator to transmit the remote control signal at the appropriate frequency to the receiving device. Within the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, an internal carrier frequency generator is supplied for this purpose, the frequency of which is specified by selecting the correct configuration options. For the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices, an internal PFD generator is provided which can be used to provide the remote control carrier signal.



Carrier/Level Output – HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1

This carrier signal is supplied on the REM pin, which is also pin-shared with PC0. The selection of the required function, whether remote output or CMOS output, is implemented by selecting the required configuration option. If the remote output REM is selected by configuration option, the REM output will be activated if the PC0 data bit in the PC register is set to "0". This output data bit is used as the on/off control bit for the REM output. Note that the REM output will be low if the PC0 output data bit is set to "1". Note that for the OTP devices, namely the HT48RA0-2 and HT48RA0-1, during the power-on reset time, the REM/PC0 pin will remain at a "0" level, no matter how the pin is configured. After the reset time has ended, if the line is configured as a REM output, the line will remain low, however, if the line is configured as a PC0 output pin it will switch to a "1" level and remain so until the application program resets the pin to a "0". It is therefore important to note that, for OTP devices, if the pin is configured as a PC0 output pin, and a PNP transistor is connected to this output to drive an infrared LED, the LED will be turned on during this power-on reset

period. For general purpose remote controller applications, it is therefore recommended that the REM configuration option is selected together with an external NPN transistor to drive the infrared LED.

The clock source for the Carrier Generator is supplied by the system clock divided by 4. By selecting values for "m" and "n" using configuration options in association with the following equation the required carrier frequency can be generated.

$$\text{Carrier Frequency} = \frac{\text{Clock Source}}{m \times 2^n}$$

The value of "m" can be either 2 or 3 while the value of "n" can range from 0 to 3, both values are chosen by selecting the required configuration options. If "m" is equal to "2" the duty cycle of the output waveform will always be equal to 1/2. If "m" is equal to "3", with the exception of "n" being equal to "0", the duty cycle can be either 1/2 or 1/3, the actual value of which is determined by configuration options.

$m \times 2^n$	Duty Cycle
2, 4, 8, 16	1/2
3	1/3
6, 12, 24	1/2 or 1/3

The following table shows examples of different carrier frequencies:

f_{SYS}	f_{CARRIER}	Duty	$m \times 2^n$
455kHz	37.92kHz	1/3 only	3
	56.9kHz	1/2 only	2

Interrupts

As the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices do not contain interrupt functions, note that this section does not apply to these devices.

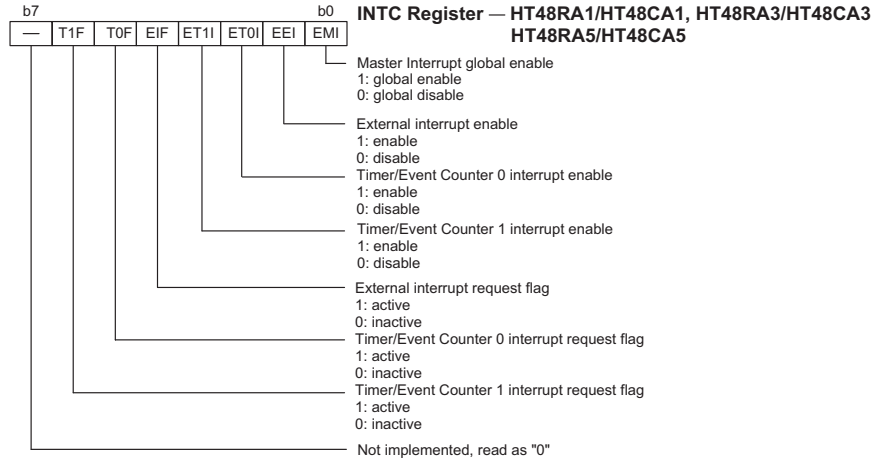
Interrupt Register

For the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices in the Remote series, both external interrupt and internal Timer/Event Counter interrupt functions are provided. The external interrupt is controlled by the action of the external pin $\overline{\text{INT}}$ which is pin-shared with PF0. The internal Timer/Event Counters also utilize an internal interrupt function for their operation. A single Interrupt Control Register, known as INTC, is provided to control all the interrupt control features.

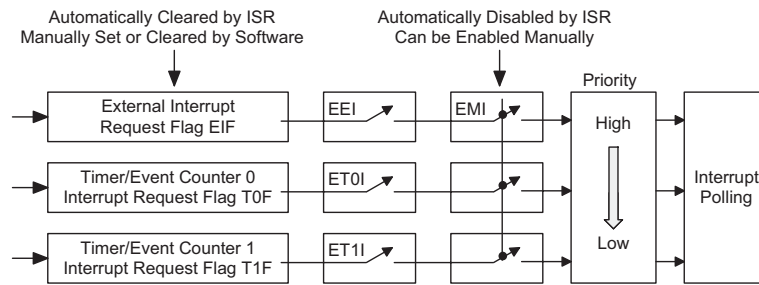
Once an interrupt subroutine is serviced, all the other interrupts will be blocked, by clearing the EMI bit. This scheme may prevent any further interrupt nesting. Other interrupt requests may occur during this interval but only the interrupt request flag is recorded. If another interrupt requires servicing while the program is in the 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 acknowl-

edged, 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 HALT 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, status register or other registers are altered by the interrupt service routine, 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 diagram with their order of priority.



Interrupt Scheme

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. These can be masked by resetting the EMI bit.

Interrupt Source	Priority	Vector
External Interrupt	1	04H
Timer/Event Counter 0 Overflow	2	08H
Timer/Event Counter 1 Overflow	3	0CH

In cases where both external and internal interrupts are enabled and where an external and internal interrupt occurs simultaneously, the external interrupt will always have priority and will therefore be serviced first. Suitable masking of the individual interrupts using the INTC register can prevent simultaneous occurrences.

External Interrupt

For an external interrupt to occur, the corresponding external interrupt enable bit must be first set. This is bit 1 of the INTC register and shown as EEI. An external interrupt is triggered by a high to low transition of the $\overline{\text{INT}}$ line, after which the related interrupt request flag (EIF; bit 4 of the INTC) will be set. When the interrupt is enabled, the stack is not full and the external interrupt is active, a subroutine call to location 04H will occur. The interrupt request flag EIF will be reset and the EMI bit will be cleared to disable other interrupts.

Timer/Event Counter Interrupt

For a timer generated internal interrupt to occur, the corresponding internal interrupt enable bit must be first properly set. For the 8-bit timer, the interrupt enable is bit 2 of the INTC register and known as ET0I while for the 16-bit timer, the interrupt enable is bit 3 of the INTC register and known as ET1I. An actual Timer/Event Counter interrupt will be initialized when the Timer/Event Counter interrupt request flag is set, caused by a timer overflow. In the case of the 8-bit timer, this is bit 5 of the INTC register and is known as T0F. In the case of the 16-bit timer, this is bit 6 of the INTC register and known as T1F. When the master interrupt global enable bit is set, the stack is not full and the corresponding internal interrupt enable bit is set, an internal timer interrupt will be generated when the timer overflows. This will create a subroutine call to location 08H for the 8-bit timer and a subroutine call to location 0CH for the 16-bit timer. When an internal interrupt occurs, the interrupt request flag, T0F or T1F will be reset and the EMI bit will be cleared to disable other interrupts.

Programming Considerations

The Timer/Event Counter interrupt request flags, T0F and T1F, external interrupt request flag EIF, enable Timer/Event Counter interrupt bits ET0I and ET1I, external interrupt enable bit EEI and master interrupt bit EMI form the interrupt control register INTC which is located at 0BH 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 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 $\overline{\text{RES}}$ 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.

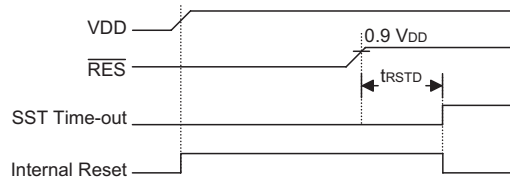
Reset

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.

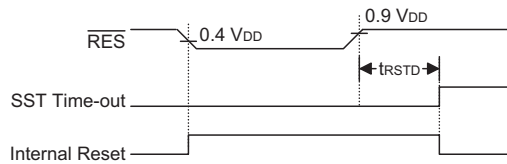
Although the microcontroller has an internal RC reset function, due to unstable power on conditions, an external RC network connected to the $\overline{\text{RES}}$ pin is generally recommended. This time delay created by the RC network ensures that the $\overline{\text{RES}}$ pin remains low for an extended period while the power supply stabilizes. During this time, normal operation of the microcontroller is inhibited. After the $\overline{\text{RES}}$ 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

→ **RES Pin Reset**

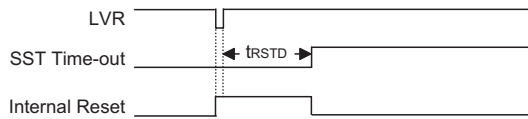
This type of reset occurs when the microcontroller is already running and the $\overline{\text{RES}}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this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



RES Reset Timing Chart

→ **Low Voltage Reset – LVR**

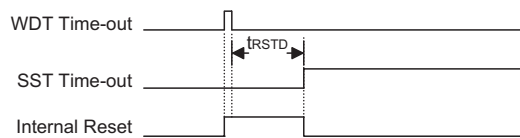
All the Remote Type microcontrollers contain a low voltage reset circuit in order to monitor the supply voltage of the device. If the supply voltage of the device drops to within a range of $0.9V \sim V_{LVR}$ such as might occur when changing the battery, the LVR will automatically reset the device internally. For a valid LVR signal, a low voltage, i.e. a voltage in the range between $0.9V \sim V_{LVR}$ 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. For all devices, the LVR enable/disable function is selected via a configuration option. For the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, the LVR voltage is fixed at a nominal value of 2.0V, while for the HT48RA1/HT48CA1, HT48RA3/HT48CA3, HT48RA5/HT48CA5 devices, an additional configuration option allows the LVR voltage to have a nominal value of either 2.0V or 3.0V.



Low Voltage Reset Timing Chart

→ **Watchdog Time-out Reset during Normal Operation**

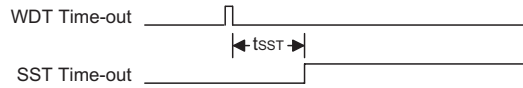
The Watchdog Time-out Reset during normal operation is the same as $\overline{\text{RES}}$ reset except that the Watchdog Time-out flag TO will be set to "1".



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 A.C. Characteristics for t_{SST} details.



WDT Time-out Reset during HALT Timing Chart

The different types of resets 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	RES reset during power on
u	u	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
Bank Pointer	Reset to zero

The different kinds of reset 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.

HT48RA0-2/HT48CA0-2

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
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	0011 1111	0011 1111	0011 1111	uuuu uuuu
PC	---- --1	---- --1	---- --1	---- --u

"u" stands for unchanged

"x" stands for unknown

"-" stands for unimplemented

HT48RA0-1/HT48CA0-1

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
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	---- --1	---- --1	---- --1	---- --u

"u" stands for unchanged

"x" stands for unknown

"-" stands for unimplemented

HT48RA1/HT48CA1

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
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
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
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	00-0 1000	00-0 1000	00-0 1000	uu-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	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	--11 1111	--11 1111	--11 1111	--uu u u u u
PCC	--11 1111	--11 1111	--11 1111	--uu u u u u
PF	---- --1	---- --1	---- --1	---- --u
PFC	---- --1	---- --1	---- --1	---- --u

"u" stands for unchanged

"x" stands for unknown

"-" stands for unimplemented

HT48RA3/HT48CA3

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
TBHP	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 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
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	00-0 1000	00-0 1000	00-0 1000	uu-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	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	--11 1111	--11 1111	--11 1111	--uu u u u u
PCC	--11 1111	--11 1111	--11 1111	--uu u u u u
PF	---- --1	---- --1	---- --1	---- --u
PFC	---- --1	---- --1	---- --1	---- --u

"u" stands for unchanged

"x" stands for unknown

"-" stands for unimplemented

HT48RA5/HT48CA5

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
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
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
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	00-0 1000	00-0 1000	00-0 1000	uu-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	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	--11 1111	--11 1111	--11 1111	--uu u u u u
PCC	--11 1111	--11 1111	--11 1111	--uu u u u u
PF	---- --1	---- --1	---- --1	---- --u
PFC	---- --1	---- --1	---- --1	---- --u

"u" stands for unchanged

"x" stands for unknown

"-" stands for unimplemented

Oscillator

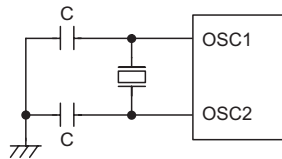
Various oscillator options offer the user a range of functions according to their various application requirements. Two types of system oscillators can be selected while with the exception of the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1, an internal free running oscillator is also provided for the Watchdog Timer. All oscillator options are selected through the configuration options.

System Clock Configurations

There are two methods of generating the system clock, using an external crystal/ceramic oscillator or an external RC network. 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 for some devices a resistor, the values of which are shown in the table, should be connected as shown in the diagram.



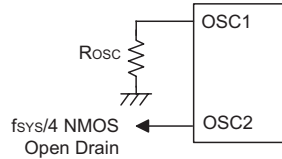
Crystal/Ceramic Oscillator

The table below shows the capacitor values for various crystal/ceramic resonator oscillating frequencies.

Crystal or Resonator	C
4MHz Crystal	0pF
4MHz Resonator	10pF
3.58MHz Crystal	0pF
3.58MHz Resonator	25pF
2MHz Crystal & Resonator	25pF
1MHz Crystal	35pF
480kHz Resonator	300pF
455kHz Resonator	300pF
429kHz Resonator	300pF

System RC Oscillator

All devices in the Remote MCU series can use an RC oscillator as their system clock. To achieve this, the simple connection of an external resistor of the correct value between OSC1 and GND is all that is required. For the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, the value of this resistor ranges from 51kΩ to 1MΩ, while for the other devices the value of this resistor ranges from 100kΩ to 820kΩ. 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 ROSC, refer to the Appendix section for typical RC Oscillator vs. Temperature and VDD characteristics graphics.



RC Oscillator

Watchdog Timer Oscillator

With the exception of the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, the other devices all contain a WDT oscillator. The WDT oscillator is a fully self-contained free running on-chip RC oscillator with a typical period of 90μs at 3V 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 function active. However, to preserve power in certain applications the WDT oscillator can be disabled via a configuration option.

HALT and Wake-up in Power Down Mode

The HALT mode is initialized by the "HALT" instruction and results in the following:

- The system oscillator will be turned off
- The contents of the on chip RAM and registers remain unchanged
- The WDT and WDT prescaler will be cleared and resume counting if the WDT clock source is selected to come from the WDT oscillator – HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 only
- All of the I/O ports remain unchanged
- The PDF flag is set and the TO flag is cleared

The system can leave the HALT mode by means of a reset, an interrupt or a WDT overflow. Additionally, the system can leave the HALT mode if an external falling edge signal appears on one of the configuration option enabled wake-up pins, either on Port A or Port B, depending upon which device is chosen. A reset will initialize a chip reset and a WDT overflow will initialize a WDT time-out reset from HALT, but by examining the TO and PDF flags, the source of the reset can be

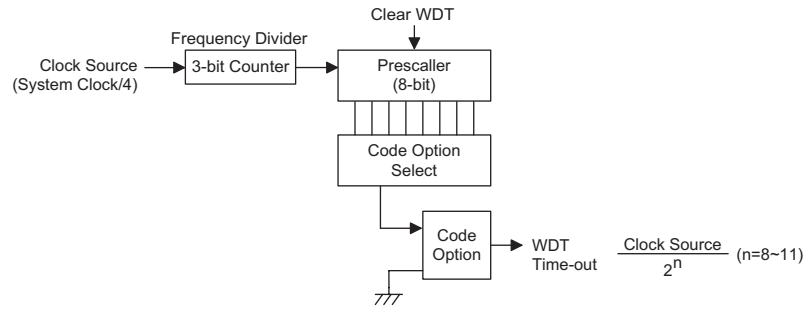
determined. The PDF flag is cleared by a system power-up or executing the "CLR WDT" instruction and is set when executing the "HALT" instruction. The TO flag is set if a WDT time-out occurs, and causes a wake-up that only resets the Program Counter and SP, the other flags remain in their original status.

An external pin wake-up, either on Port A or Port B, depending upon which device is used, and interrupt methods can be considered as a continuation of normal execution. Each bit, either on Port A or Port B, depending upon which device is used, can be independently selected to wake-up the device by configuration option. Awakening from an I/O port stimulus, the program will resume execution at the next instruction. If the system is woken up via an interrupt, two possibilities may occur. If the related interrupt is disabled or the interrupt is enabled but the stack is full, the program will resume execution at the next instruction. If the interrupt is enabled and the stack is not full, the regular interrupt response takes place. If an interrupt request flag is set to "1" before entering the HALT mode, the wake-up function of the related interrupt will be disabled. Once a wake-up event occurs, it takes 1024 system clock periods to resume normal operation. In other words, a dummy period will be inserted after a wake-up. If the wake-up results from an interrupt acknowledge signal, the actual interrupt subroutine execution will be delayed by one or more cycles. If the wake-up results in the next instruction execution, this will be executed immediately after the dummy period is finished.

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. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

For the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices, the watchdog timer clock source originates from the system clock divided by four, namely $f_{SYS}/4$. This $f_{SYS}/4$ internal clock passes through a prescaler, the division ratio of which can be varied by a configuration option, to give a 2^8 to 2^{11} division ratio range, which results in a $2^9/f_S$ to $2^{12}/f_S$ actual watchdog time-out period. The actual watchdog timer time-out value depends therefore on the fixed clock source, $f_{SYS}/4$, and on the division ratio of the prescaler, which is set by a configuration option. There are no internal registers associated with the Watchdog Timer for the HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 devices. Note that as the $f_{SYS}/4$ is used as the WDT 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.



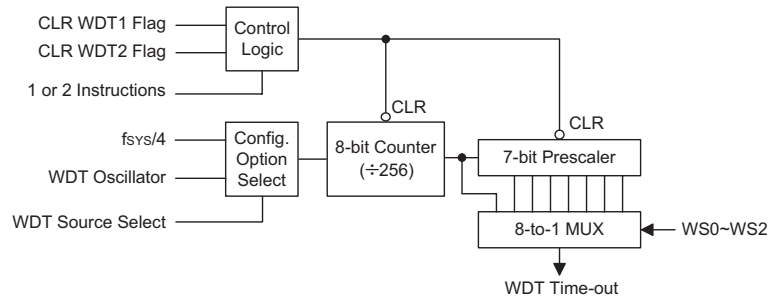
Watchdog Timer – HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1

For the HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 devices, the WDT clock source is supplied by one of two sources, the choice of which is selected by a configuration option. The two clock source choices are its own self-contained internal WDT oscillator or the system clock divided by four, namely $f_{SYS}/4$.

This WDT clock source, whether it is the WDT oscillator or the $f_{SYS}/4$ internal clock, is first divided by 256 via an 8-bit counter to give longer time-out periods. For even longer WDT time out periods, the WDT prescaler can be utilized for which a WDTS register is provided. 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, the value of which is shown in the table.

b7					b0			WDTS Register
—	—	—	—	—	WS2	WS1	WS0	
					WDT prescaler rate select			
					WS2	WS1	WS0	WDT Rate
					0	0	0	1:1
					0	0	1	1:2
					0	1	0	1:4
					0	1	1	1:8
					1	0	0	1:16
					1	0	1	1:32
					1	1	0	1:64
					1	1	1	1:128
					Not used, user accessible			

Note that if the WDT oscillator is selected by configuration option as the WDT clock source, the time out periods can vary with VDD, temperature and process variations. If the $f_{SYS}/4$ is selected by configuration option as the WDT clock source, it should be noted that when the system enters the power-down mode, then the system 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 is strongly recommended.



Watchdog Timer – HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5

For all devices, 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 SP. Three methods can be adopted to clear the contents of the WDT including the WDTs register prescaler value. The first is an external hardware reset (a low level on the RES pin), the second is via 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

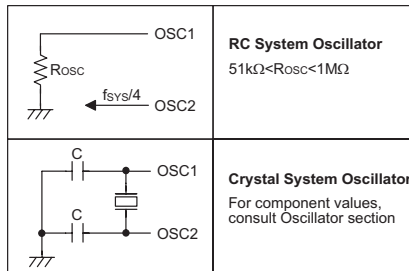
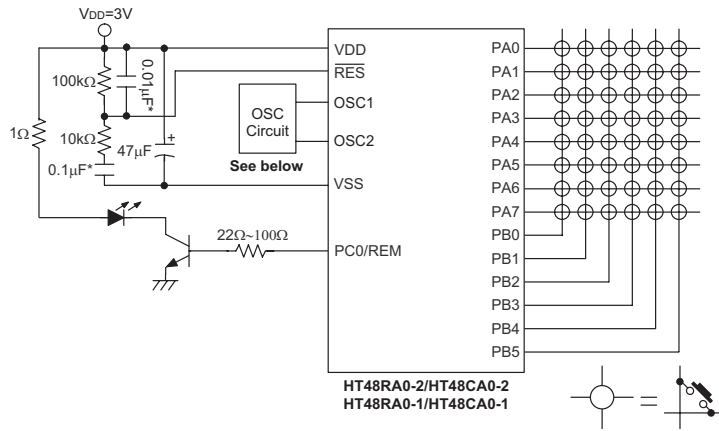
The various microcontroller configuration options selected using the HT-IDE are stored in the option memory. All bits must be defined for proper system function, the details of which are shown in the table. After the configuration options have been programmed into the microcontroller by the user, it is important to note that they cannot be altered later by the application program. For the mask version devices, these configuration options, once defined, are implemented into the microcontroller during the manufacturing process and therefore cannot be reconfigured by the user.

HT48RA0-2/HT48CA0-2 and HT48RA0-1/HT48CA0-1 Options	
I/O Options	
1	PB0~PB1: wake-up enable or disable (bit option)
2	PB2~PB7: wake-up enable or disable (bit option)
3	PC0: CMOS output or carrier output (bit option)
Oscillator Options	
4	OSC type selection: RC or crystal
Watchdog Options	
5	WDT: enable or disable
6	CLRWDT instructions: 1 or 2 instructions
7	WDT time-out period: $2^{10}/f_{SYS}$, $2^{11}/f_{SYS}$, $2^{12}/f_{SYS}$, $2^{13}/f_{SYS}$
Carrier Options	
8	Carrier duty: 1/2 duty or 1/3 duty
9	Carrier frequency: $f_{SYS}/8$, $f_{SYS}/16$, $f_{SYS}/32$, $f_{SYS}/64$, for 1/2 duty cycle
10	Carrier frequency: $f_{SYS}/12$, 1/3 duty cycle
11	Carrier frequency: $f_{SYS}/24$, $f_{SYS}/48$, $f_{SYS}/96$, for 1/2 duty or 1/3 duty cycle
LVR Options	
12	LVR function: enable or disable

HT48RA1/HT48CA1, HT48RA3/HT48CA3 and HT48RA5/HT48CA5 Options	
I/O Options	
1	PA0~PA7: wake-up enable or disable (bit option)
2	PA0~PA7: pull-high enable or disable (byte option)
3	PB0~PB3: pull-high enable or disable (nibble option)
4	PB4~PB7: pull-high enable or disable (nibble option)
5	PC0~PC5: pull-high enable or disable (bit option)
6	PF0: pull-high enable or disable
Oscillator Options	
7	OSC type selection: RC or crystal
PFD Options	
8	PB0: normal I/O or PFD output
Watchdog Options	
9	WDT: enable or disable
10	CLRWDT instructions: 1 or 2 instructions
11	WDT clock source: WDT oscillator or $f_{SYS}/4$
LVR Options	
12	LVR function: enable or disable
13	LVR voltage: 2.0V or 3.0V

Application Circuits

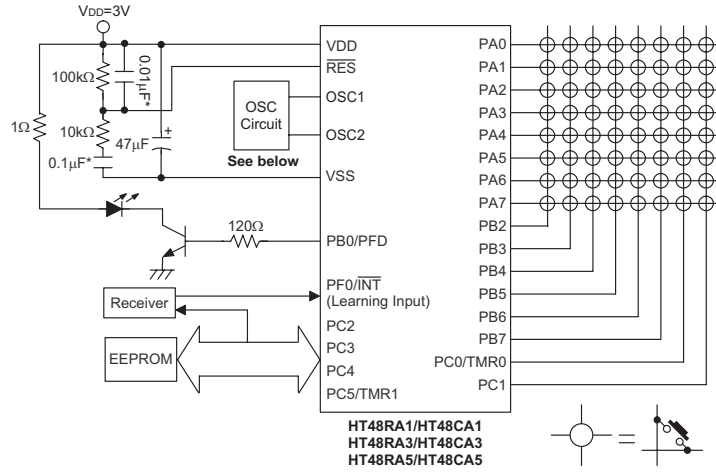
RC or Crystal Oscillator


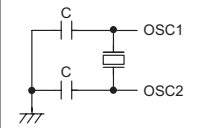


OSC Circuit

- Note**
1. It is recommended that a 100μF decoupling capacitor is placed between VSS and VDD.
 2. The resistance and capacitance for reset circuit should be designed to ensure that the VDD is stable and remains in a valid range of the operating voltage before bringing $\overline{\text{RES}}$ to high.

RC or Crystal System Oscillator

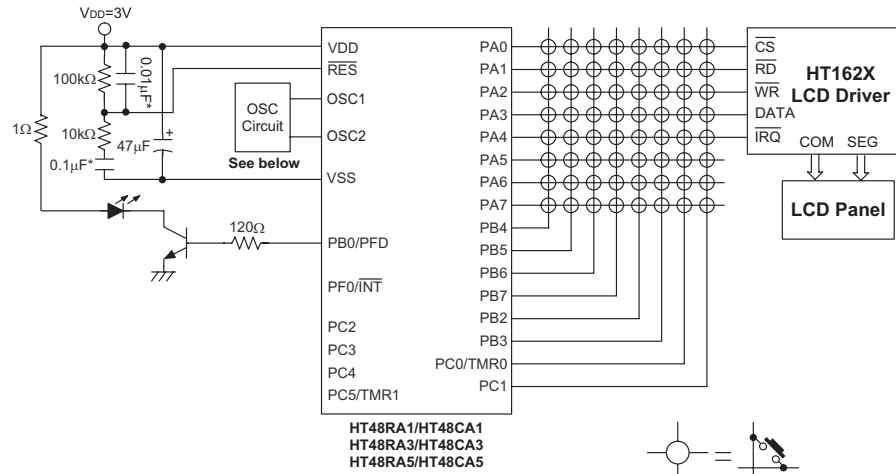


	<p>RC System Oscillator $100k\Omega < R_{osc} < 820k\Omega$</p>
	<p>Crystal System Oscillator For component values, consult Oscillator section</p>

OSC Circuit

- Note**
1. It is recommended that a 100μF decoupling capacitor is placed between VSS and VDD.
 2. The resistance and capacitance for reset circuit should be designed to ensure that the VDD is stable and remains in a valid range of the operating voltage before bringing RES to high.
 3. Make the length of the wiring, which is connected to the RES pin as short as possible, to avoid noise interference.

RC or Crystal System Oscillator with LCD



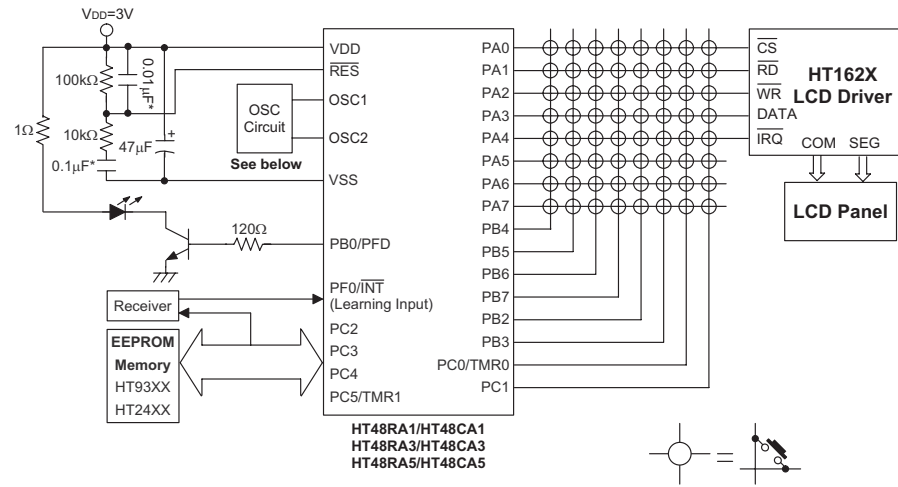
HT48RA1/HT48CA1
HT48RA3/HT48CA3
HT48RA5/HT48CA5

	<p>RC System Oscillator 100kΩ < R_{osc} < 820kΩ</p>
	<p>Crystal System Oscillator For component values, consult Oscillator section</p>

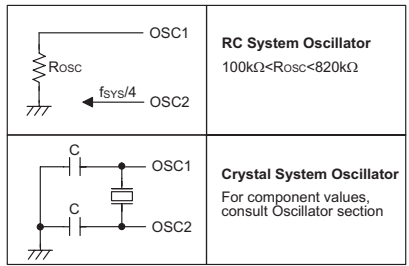
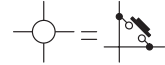
OSC Circuit

- Note**
1. It is recommended that a 100μF decoupling capacitor is placed between VSS and VDD.
 2. The resistance and capacitance for reset circuit should be designed to ensure that the VDD is stable and remains in a valid range of the operating voltage before bringing RES to high.
 3. Make the length of the wiring, which is connected to the RES pin as short as possible, to avoid noise interference.
 4. Refer to the Holtek website for detailed information on the HT162X LCD Driver Series.

RC or Crystal System Oscillator with EEPROM Interface (Learning Remote Controller)



HT48RA1/HT48CA1
HT48RA3/HT48CA3
HT48RA5/HT48CA5



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	$[m].i \leftarrow [m].(i+1); (i = 0\sim 6)$ $[m].7 \leftarrow C$ $C \leftarrow [m].0$
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	$ACC.i \leftarrow [m].(i+1); (i = 0\sim 6)$ $ACC.7 \leftarrow C$ $C \leftarrow [m].0$
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>]	Syntax elements that are enclosed by a pair of brackets are optional. For example, the syntax of the command line is as follows: HASM [<i>options</i>] <i>filename</i> [;]
{ <i>choice1</i> <i>choice2</i> }	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.
{ <i>choice1</i> <i>choice2</i> }	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.

Example of Convention	Description of Convention
Repeating elements...	<p>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:</p> <p>PUBLIC <i>name1</i> [<i>,name2</i> [...]]</p> <p>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.</p>

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
  ENDF

```

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(?)
  ENDF

```

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:                                     ; label
    $1:                                     ;; local label
        mov a, 1
        jmp $3
    $2:                                     ;; local label
        mov a, 2
        jmp $1
    $3:                                     ;; local label
        jmp $2
Label2:                                     ; label
    $0:                                     ;; local label
        jmp $1
    $1:                                     ;; local label
        jmp Label1
        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...
IIII  oooo hhhh hhhh EC source-program-statement
                        Rn
```

IIII → 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      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)
 - 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 CN1 and CN2 connectors of the HT-ICE.

OTP Programmer

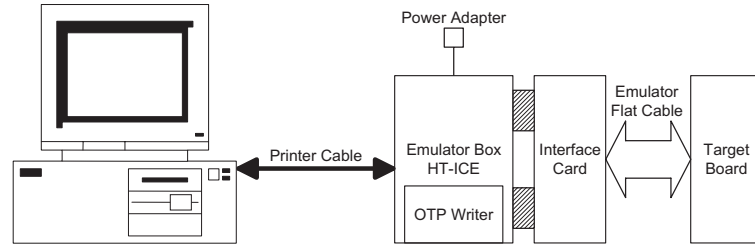
Holtek's OTP devices are fully supported by a range of programmers. For engineering level OTP device programming, Holtek supplies its stand alone programming tool which provides a quick and efficient means for low volume OTP programming. The HT-ICE In-Circuit Emulators has integrated a writer as part 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.

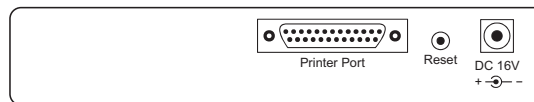
System Configuration

The HT-IDE3000 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.

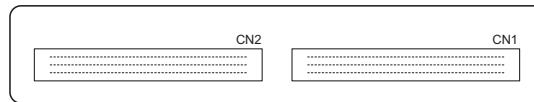


The HT-IDE system contains the following hardware components:

- 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

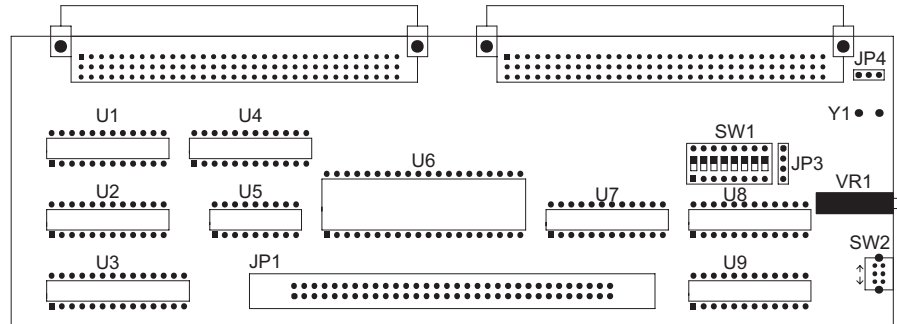


HT-ICE Front View

HT-ICE Interface Card Settings

The HT-ICE interface card (CPCB48MIO001A) as shown below, is a PCB used to connect the HT-ICE emulator to user's target board. It has the following functions:

- External clock source
- External signal trace input
- MCU socket pin assignment



The external clock source has two modes, RC and Crystal. If a crystal clock is used, short positions 2 and 3 on Jumper JP4 and insert a suitable crystal into location Y1. Otherwise if an RC clock is used, short positions 1 and 2, then adjust the system frequency 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 four external signal trace inputs, marked as ET0 to ET3 at jumper location JP3, provide a means for external signals to trigger the internal breakpoint and trace functions. For more information, refer to the chapters on Breakpoint and Trace the Application Program within the Holtek HT-IDE3000 User's Guide.

DIP switch SW1 should be set according to which device is selected and in accordance with the following table:

Part No.	Socket	SW1								
		1	2	3	4	5	6	7	8	
HT48RA0-2	U8	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	—
HT48RA0-1	U8	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF	—
HT48RA1	U3	ON	OFF	OFF	ON	OFF	OFF	ON	—	—
HT48RA3	U3	ON	OFF	OFF	ON	OFF	OFF	ON	—	—
HT48RA5	U3	ON	OFF	OFF	ON	OFF	OFF	ON	—	—

Note that the 20-pin package HT48RA0-2 device uses the U8 socket which has 24 pins. Pin 1 of the HT48RA0-2 device should be aligned with pin 1 of the U8 socket. Pins 13~16 of the U8 socket will be unused for this device. As the Interface Card for the Remote series MCUs is a universal one that is also used by devices from other series, the sockets provided on the interface card do not indicate that that actual package size exists for the Remote series devices.

Switch SW2 should also be set in accordance with the following table:

Part No.	SW2
HT48RA0-1 HT48RA0-2	↑ – carrier output
HT48RA1 HT48RA3 HT48RA5	↓ – no carrier output

The pin assignments in locations U3 and U8 are defined so as to match the datasheet pin assignments for the Remote MCU series. The interface card VME connectors directly interface to the CN1 and CN2 connectors on the HT-ICE.

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.

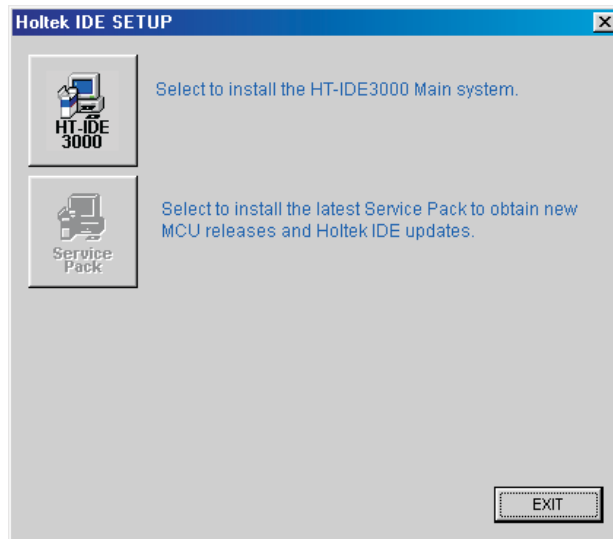
Software Installation

- Step1

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



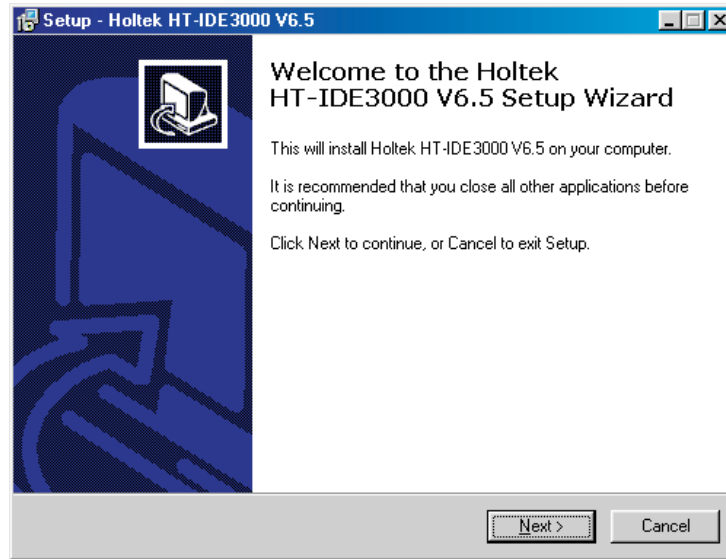
Click <HT-IDE3000> button and the following dialog will be shown.



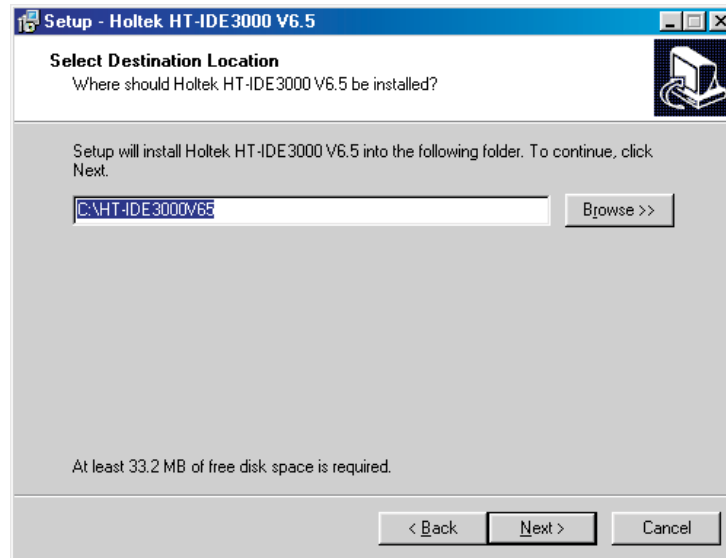
Click <HT-IDE3000> or <Service Pack> as you want.

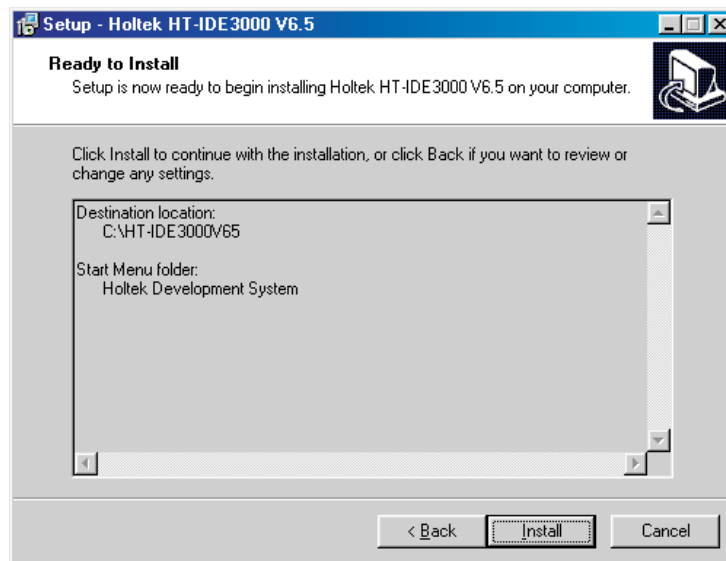
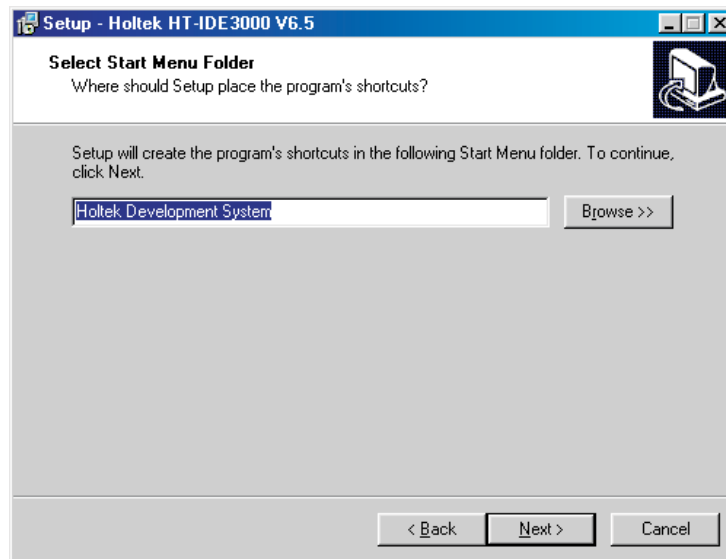
Here's an Example of installing HT-IDE3000
Click <HT-IDE3000> button.

- Step 2
Press the <Next> button to continue setup or press <Cancel> button to abort.



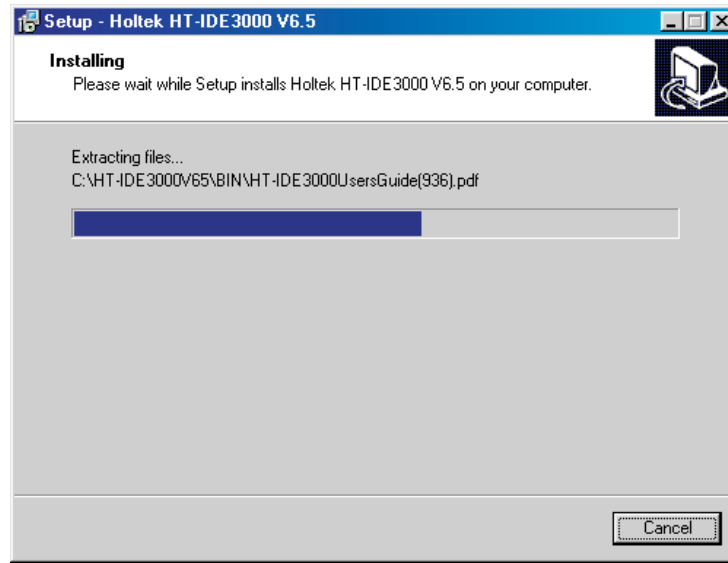
- Step 3
The following dialog will be shown to ask the user to enter a directory name.



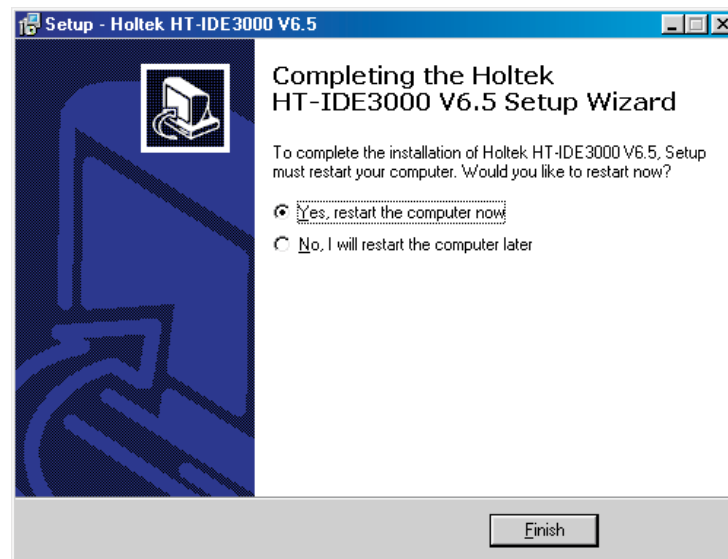


- Step 4
Specify the path you want to install the HT-IDE3000 and click <Next> button.

- Step 5
SETUP will copy all files to the specified directory.



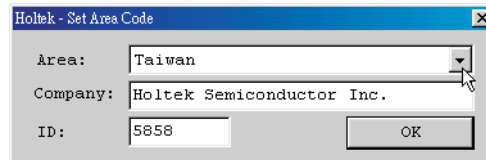
- Step 6
If the process is successful a dialog will be shown.



- 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 mask options
- Setup all mask 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

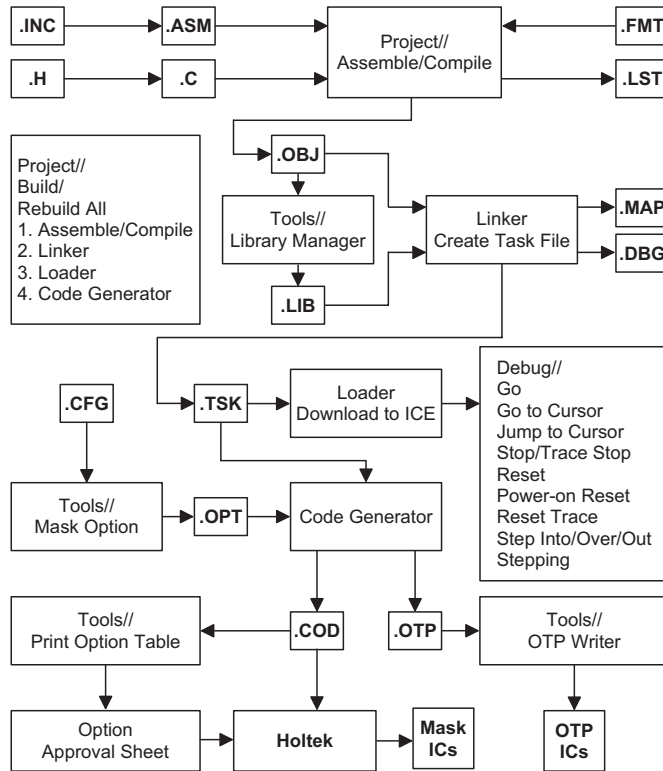
Step 4 – Programming the OTP Device

- Build the project for creating the .OTP file
- Click on Tools menu and select the OTP 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:



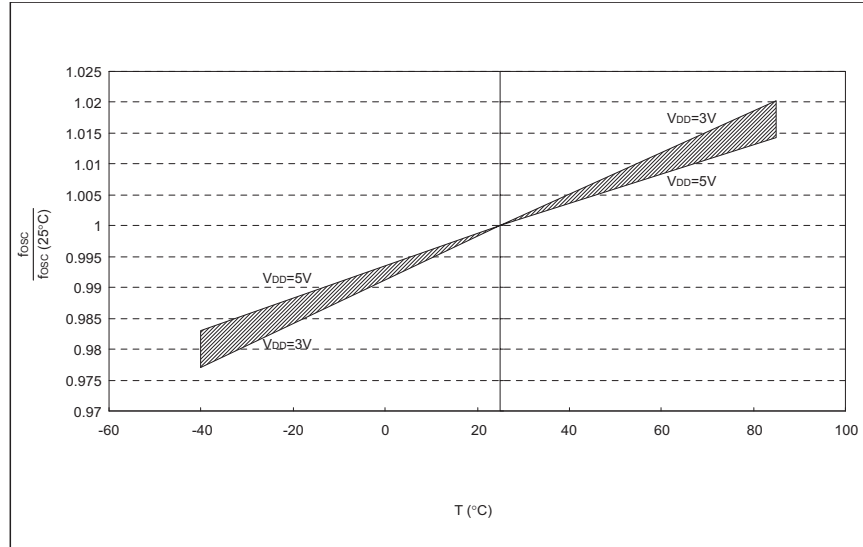
Appendix

Appendix A**Device Characteristic Graphics**

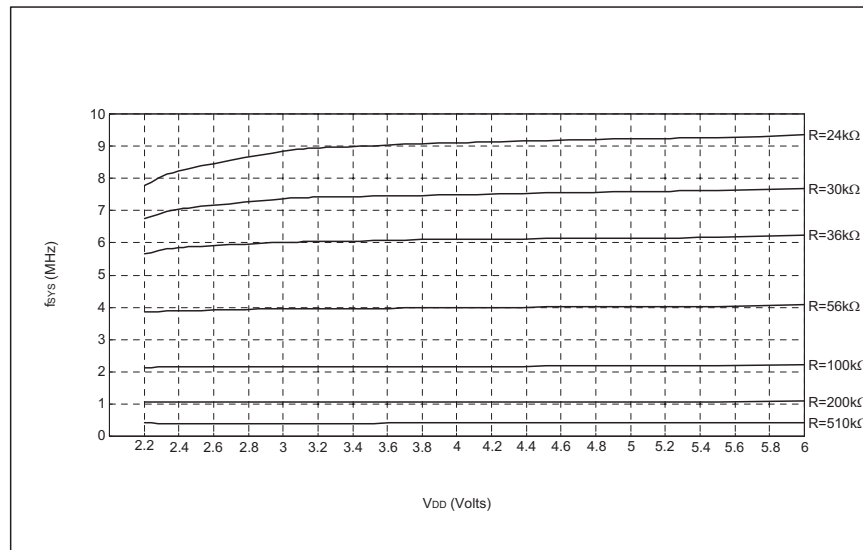
The following characteristic graphics depicts typical device behavior. The data presented here is a statistical summary of data gathered on units from different lots over a period of time. This is for information only and the figures were not tested during manufacturing.

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.

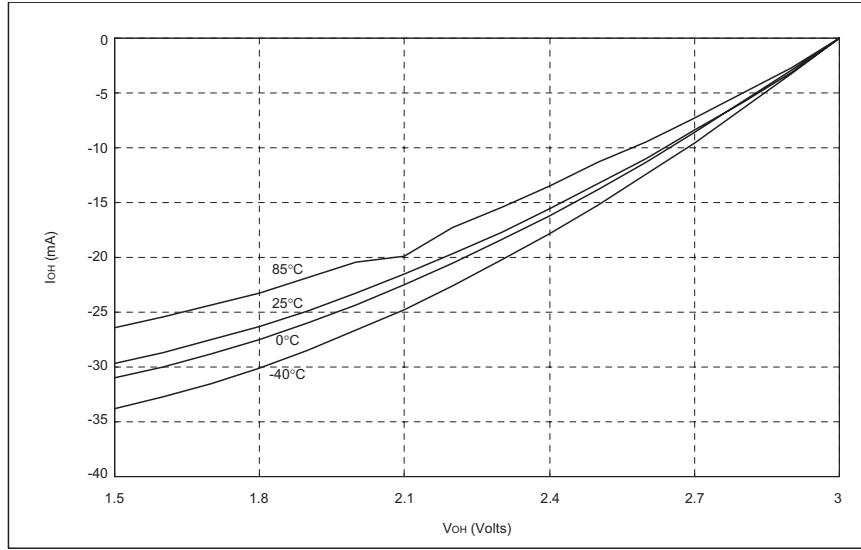
Typical RC OSC vs. Temperature



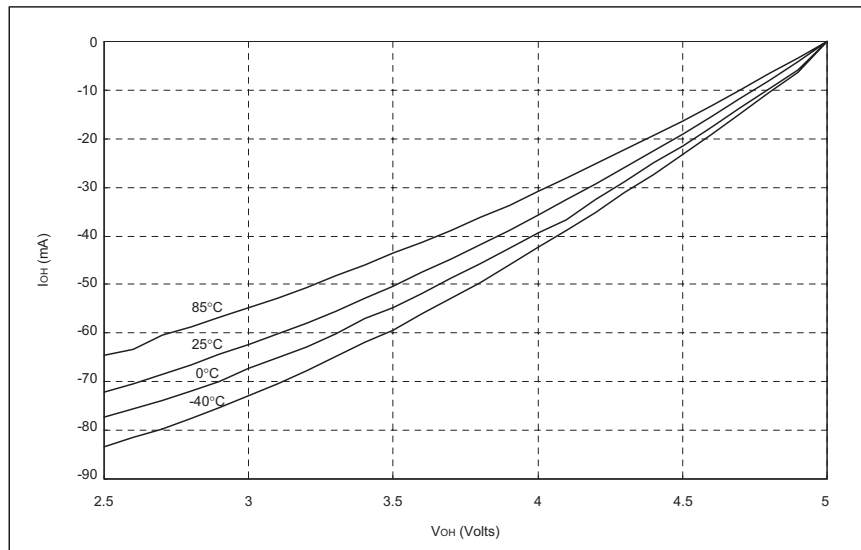
Typical RC Oscillator Frequency vs. V_{DD}



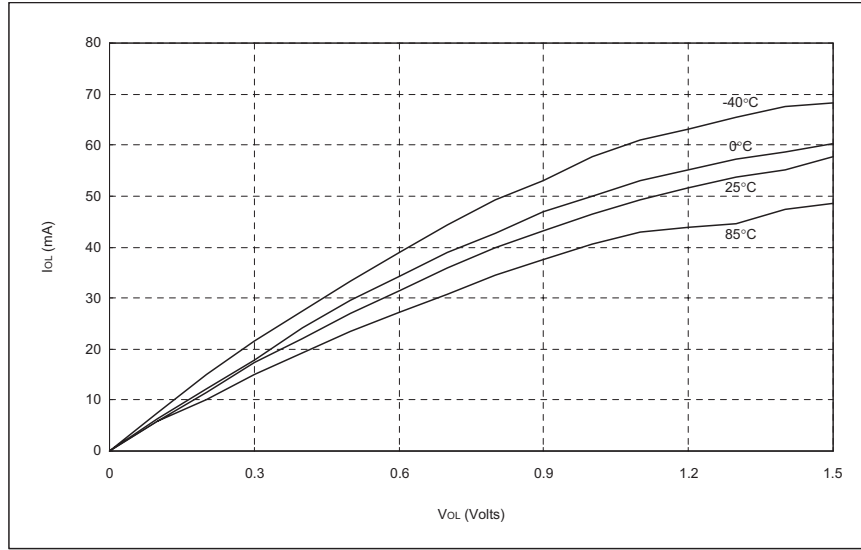
I_{OH} vs. V_{OH} , $V_{DD}=3V$



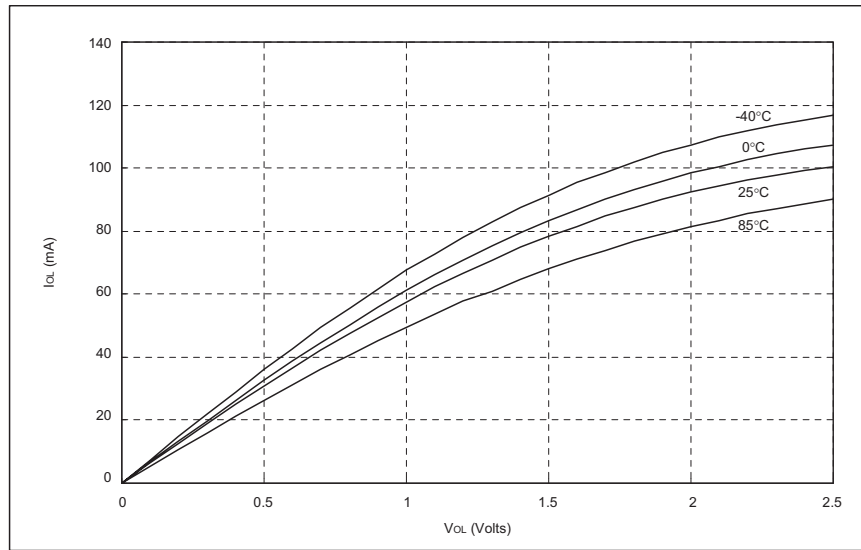
I_{OH} vs. V_{OH} , $V_{DD}=5V$



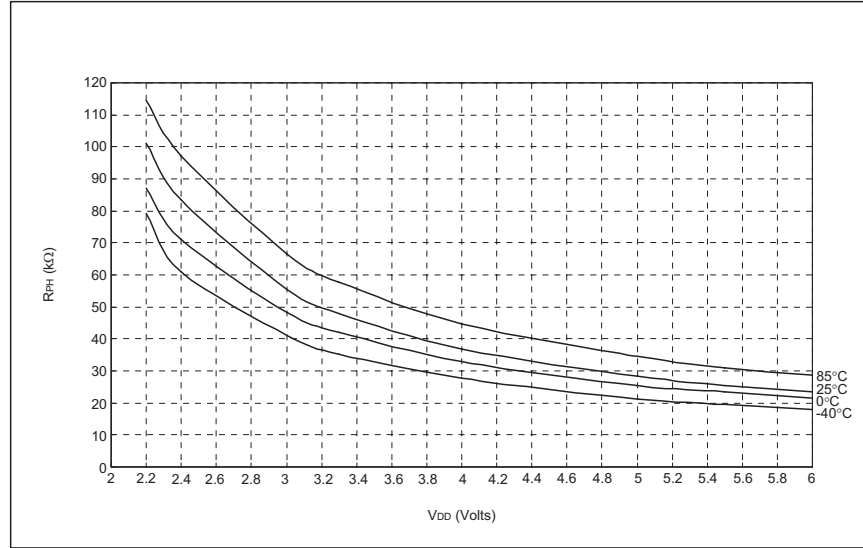
I_{OL} vs. V_{OL} , $V_{DD}=3V$



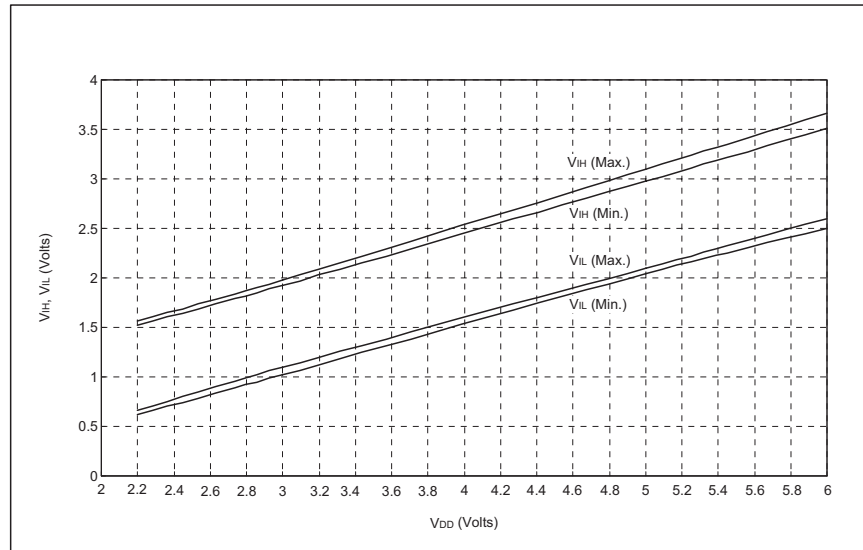
I_{OL} vs. V_{OL} , $V_{DD}=5V$



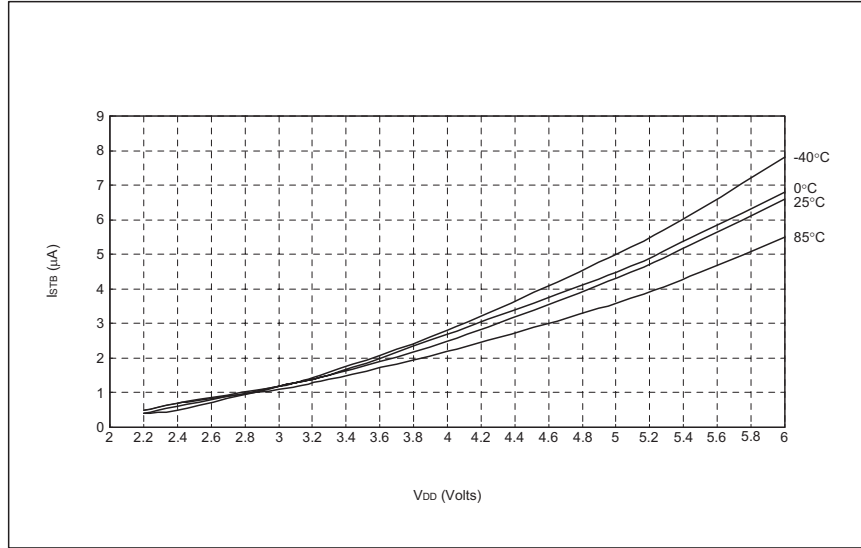
Typical R_{PH} vs. V_{DD}



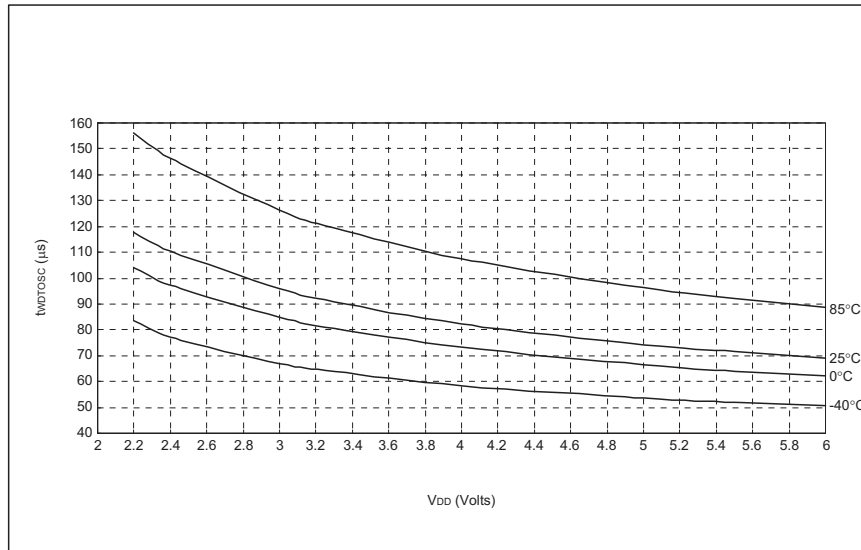
Typical V_{IH} , V_{IL} vs. V_{DD} in -40°C to $+85^{\circ}\text{C}$



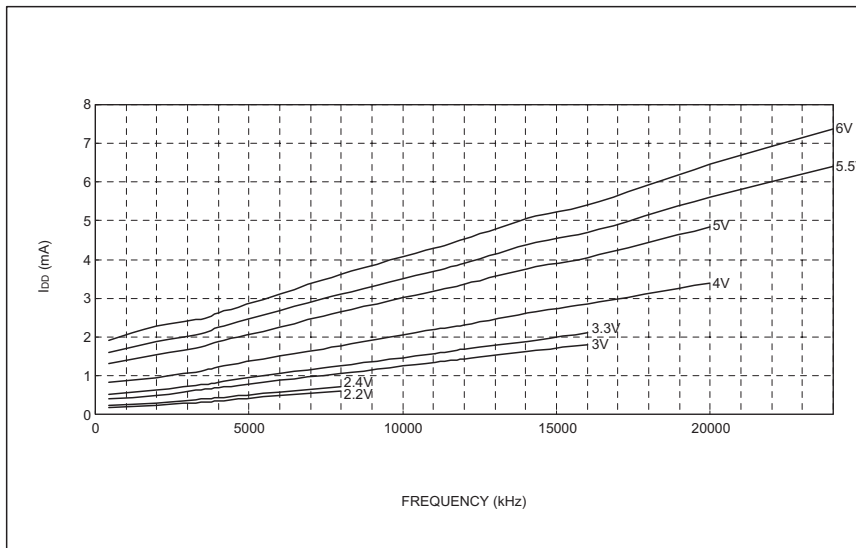
Typical I_{STB} vs. V_{DD} Watchdog Enable



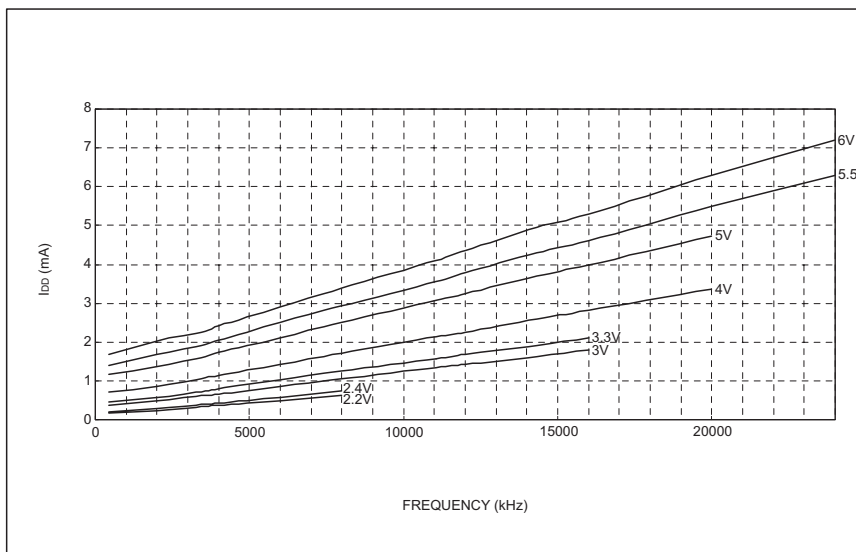
Typical t_{WDOSC} vs. V_{DD}



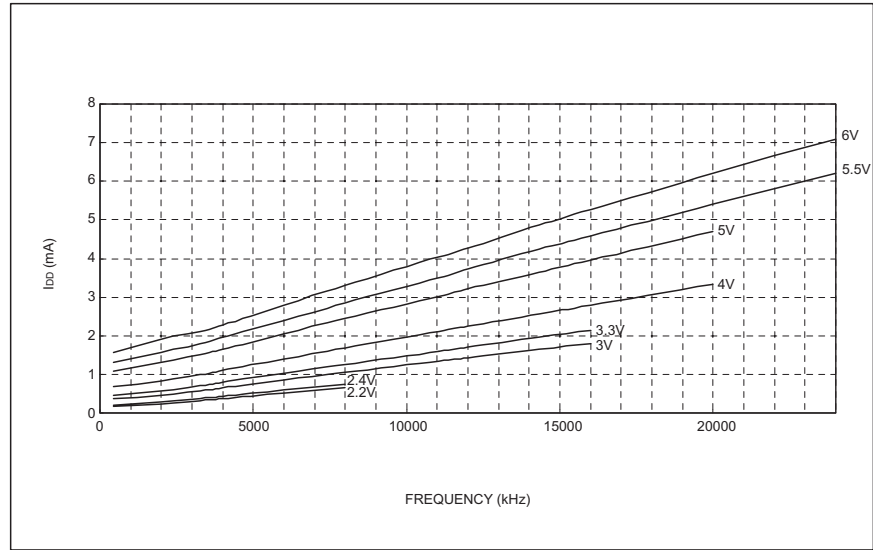
Typical I_{DD} vs. Frequency (External Clock, $T_a = -40^\circ\text{C}$)



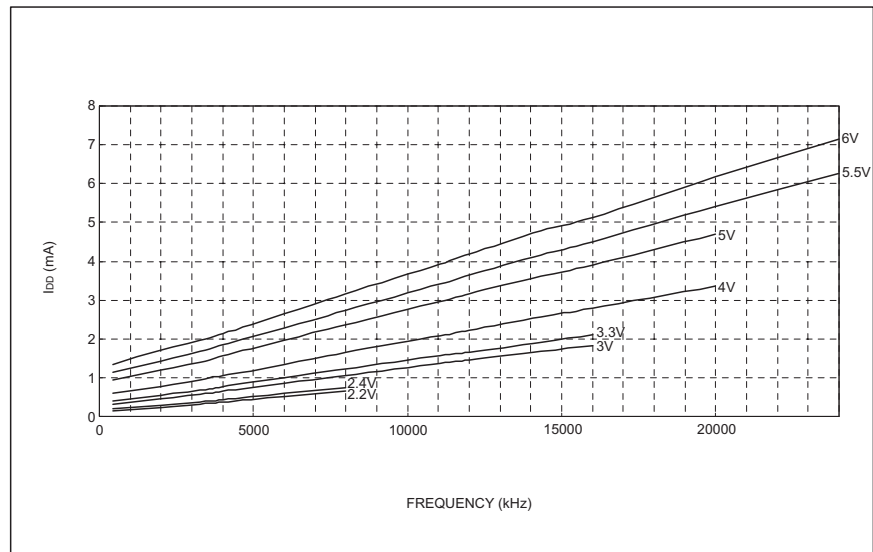
Typical I_{DD} vs. Frequency (External Clock, $T_a = 0^\circ\text{C}$)



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



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

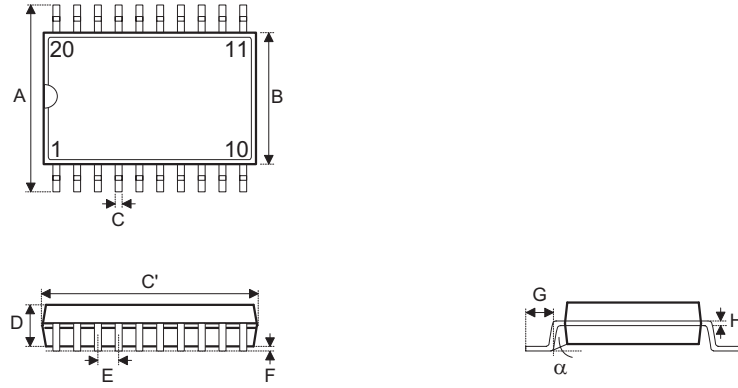


Appendix B

Package Information

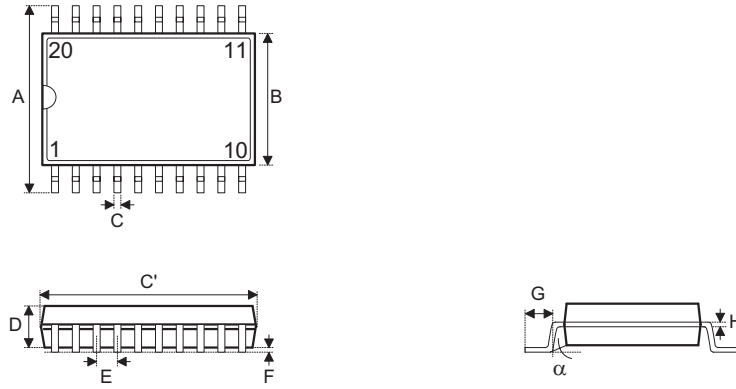
B

20-pin SOP (300mil) Outline Dimensions



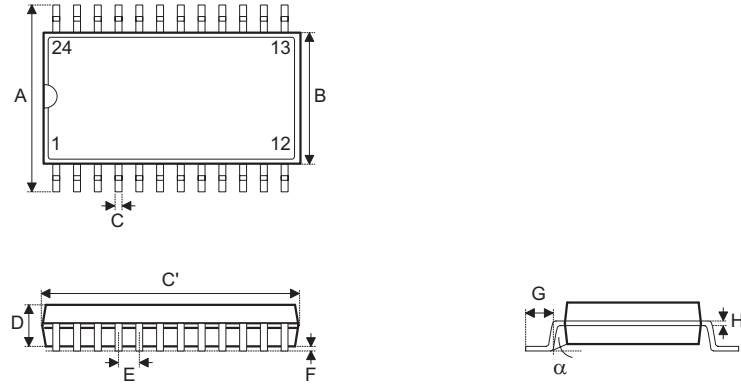
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	394	—	419
B	290	—	300
C	14	—	20
C'	490	—	510
D	92	—	104
E	—	50	—
F	4	—	—
G	32	—	38
H	4	—	12
α	0°	—	10°

20-pin SSOP (150mil) Outline Dimensions



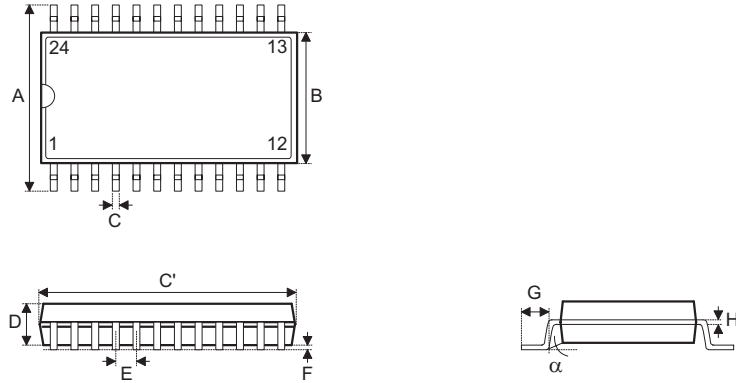
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	228	—	244
B	150	—	158
C	8	—	12
C'	335	—	347
D	49	—	65
E	—	25	—
F	4	—	10
G	15	—	50
H	7	—	10
α	0°	—	8°

24-pin SOP (300mil) Outline Dimensions



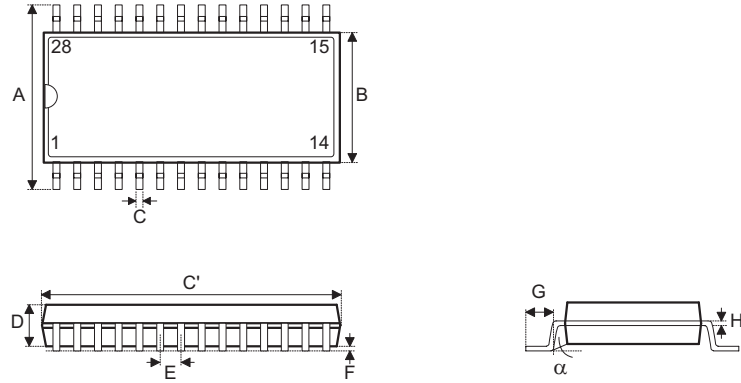
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	394	—	419
B	290	—	300
C	14	—	20
C'	590	—	614
D	92	—	104
E	—	50	—
F	4	—	—
G	32	—	38
H	4	—	12
α	0°	—	10°

24-pin SSOP (150mil) Outline Dimensions



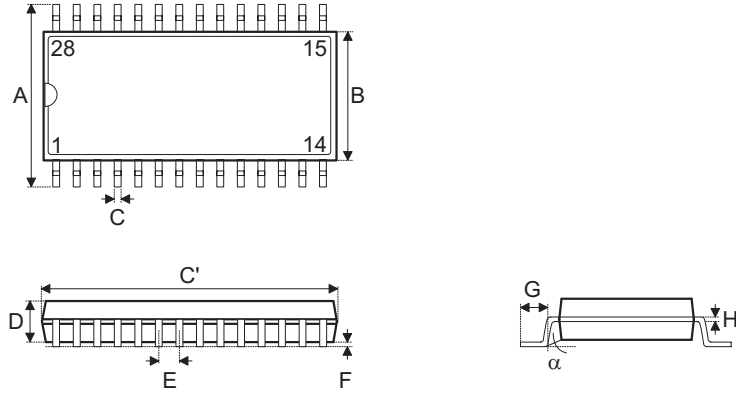
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	228	—	244
B	150	—	157
C	8	—	12
C'	335	—	346
D	54	—	60
E	—	25	—
F	4	—	10
G	22	—	28
H	7	—	10
α	0°	—	8°

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°

28-pin SSOP (209mil) Outline Dimensions



Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	291	—	323
B	196	—	220
C	9	—	15
C'	396	—	407
D	65	—	73
E	—	25.59	—
F	4	—	10
G	26	—	34
H	4	—	8
α	0°	—	8°

Appendix C

General Application Notes

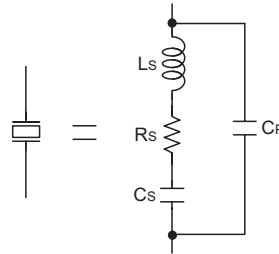


System Oscillator

Crystal/Ceramic Oscillator

→ **Crystal/Ceramic Oscillator Equivalent Circuit**

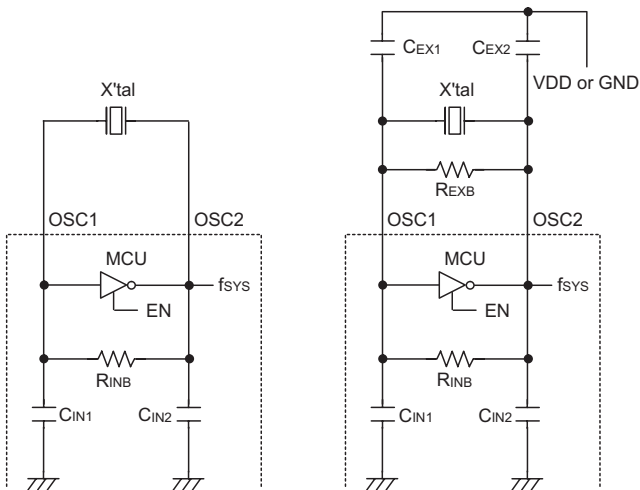
The following circuit combination of resistors, capacitors and inductors depicts an equivalent circuit for a crystal or ceramic oscillator.



- Note**
1. L_s is a series inductor, R_s is a series resistor, C_s is a series capacitor and C_p is a parallel connected capacitor.
 2. The resonance frequency is given by a series connected LC pair where $L = L_s$ and $C = C_s \times C_p / (C_s + C_p)$
 3. The main difference between the crystal and the ceramic resonator equivalent circuits is, when oscillating at the same frequency, the equivalent inductor for the crystal will be greater than that of the ceramic resonator.

→ **Crystal/Ceramic Oscillator Basic Circuits**

The following diagrams show two application circuits for the crystal/ceramic system oscillators.



-
- Note**
1. The internal bias resistor R_{INB} is one of the components required to produce oscillation.
 2. The C_{IN1} and C_{IN2} internal capacitors together with the external crystal/ceramic oscillator form a Pierce oscillator. When oscillating, the crystal/ceramic oscillator can be seen as an inductive element. This circuit is effective in reducing EMI.
 3. The R_{EXB} external bias resistor is used to ensure the oscillator stops should a low voltage condition occur. This resistor works in conjunction with C_{EX1} where the time constant $R_{EXB} \times C_{EX1}$ is greater than $2\pi f_{SYS}$. The main principle here is to increase the loading on the oscillator circuit to ensure that the MCU will not exhibit erroneous operation under low voltage conditions. However, for applications that will not undergo low voltage conditions, this component is not required.
 4. The two external oscillator capacitors C_{EX1} and C_{EX2} are used to provide small adjustments to the oscillation frequency or for crystal/ceramic oscillator matching or for adjustments to oscillator start-up time. These components are not necessary for normal applications.
 5. The Holtek MCU datasheets and handbooks provide application circuits where suggested capacitor and resistor values are shown for consultation.
-

→ **Crystal/Ceramic Oscillator Warm-up Time**

- The crystal/ceramic oscillators all require a short warm-up time before they start oscillating.
- The length of this warm-up time is related to the characteristics of the warm-up time and power-down time. In most cases, if the MCU is powered down, a warm-up time of 3–5ms is required.

→ **System Start-up Timer**

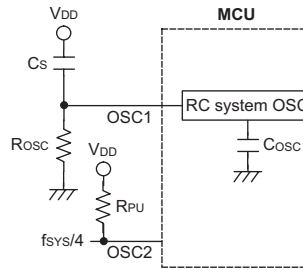
- This is a period of time added to allow the oscillator to reach stability.
- This time is fixed at 1024 clock cycles.

→ **EMI/EMS (EMC) Considerations**

- The crystal/ceramic oscillator should be located as close to the MCU oscillator pins as possible. In addition, the interconnections to the crystal/ceramic oscillator should be kept as short as possible.
- To reduce EMI, the crystal/ceramic oscillator should have a VDD or GND (VSS) guard ring for shielding.
- The interconnections between C_{EX1} and C_{EX2} and VDD or GND (VSS) should be kept as short as possible.

One-pin Pull-down RC Oscillator

The following drawing shows the circuit for RC oscillators which require an external pull-down resistor.



- Note**
1. The R_{osc} oscillating resistor works in combination with the internal capacitor C_{osc} to form an RC oscillator. The value of this resistor is dependent upon the oscillation frequency required. The value of this resistor is inversely proportional to the oscillation frequency, hence, the larger the resistor, the lower the frequency.
 2. The C_{osc} oscillating capacitor is internal to the MCU which together with the external R_{osc} oscillator forms the RC system oscillator.
 3. The capacitor C_s is provided for reasons of frequency stability and its recommended value is 470pF.
 4. The resistor R_{PU} should be added if the OSC2 (system clock/4) test output is used. Its recommended value is 2k Ω .

→ **System Start-up Timer**

- This is a period of time added to allow the oscillator to reach stability.
- This time is fixed at 1024 clock cycles.

→ **Manufacturing and Temperature Variations**

- Because the frequency of the RC oscillator is dependent upon the value of an internal capacitor, the value of which is dependent upon manufacturing parameters, different MCUs will exhibit different characteristics. Under conditions of similar voltage and temperature, this will account for an oscillator frequency variation of approximately $\pm 25\%$.
- Within the same MCU, there is no manufacturing variations, the oscillator frequency will be affected by variations in the operating voltage and operating temperature. The effect of temperature and voltage on the frequency can be found by visiting our Holtek web site.

→ **EMI/EMS (EMC) Considerations**

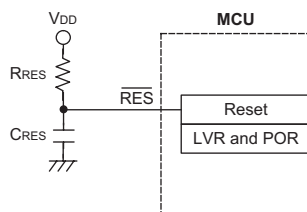
- The resistor R_{osc} should be located as close to the OSC1 pin as possible with the interconnecting line as short as possible.
- The capacitor C_s will improve the noise performance of the oscillator. The two lines connecting this capacitor to the MCU OSC1 and VDD pins should be kept as short as possible.
- After the resistor R_{PU} has been added to verify the system frequency, for high volume production it is recommended that this resistor is not added. This is because this pin may have an adverse effect on the system frequency produced on pin OSC1.

Reset Circuit

External $\overline{\text{RES}}$ Line Description

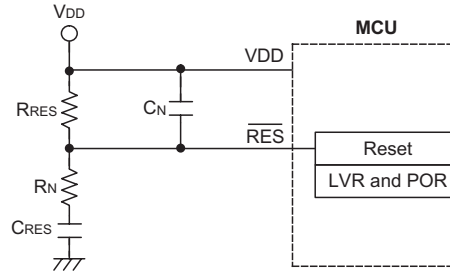
- Used to start up the MCU from a known condition, the effect of which will be to reset the internal special registers (with the exception of the TO and PDF flags) and reset the I/O ports to a known condition. The program counter will also be reset to 0000H where the program will begin execution.
- If the WDT is enabled the WDT will be cleared and begin counting anew
- The RAM contents will be unchanged
- The stack pointer will be reset

Simple RC Reset Circuit



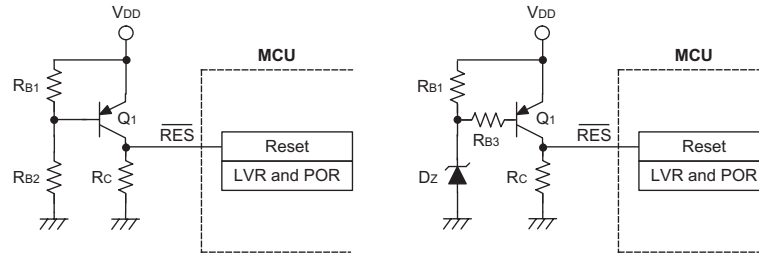
- For applications with only low levels of noise, this simple RC reset circuit is applicable.
- The reset time is governed by the values of R_{RES} and C_{RES} .
- Regarding the length of the reset, the main consideration is stability and the length of time required for the power supply to reach the operating voltage of the MCU. The reset time should always be greater than this time. When the power is turned off, the charge in the capacitor must be discharged as quickly as possible.
- The recommended values for R_{RES} and C_{RES} would be $100\text{k}\Omega$ and $0.1\mu\text{F}$.
- The layout of the external reset components is important, the lines connecting the C_{RES} capacitor to the MCU $\overline{\text{RES}}$ pin and VSS should be kept as short as possible.

RC Circuit for Applications Operating in Noisy Environments



- For circuits with higher levels of noise this circuit is applicable.
- The reset time is governed by the values of RRES and CRES.
- Regarding the length of the reset, the main consideration is stability and the length of time required for the power supply to reach the operating voltage of the MCU. The reset time should always be greater than this time. When the power is turned off the pull-high resistor connected to the capacitor must be capable of discharging the capacitor in as quick a time as possible.
- Recommended values for RRES and CRES would be 100kΩ and 0.1μF.
- The RN matching resistor and CN matching capacitor are matched to the internal design of the MCU. Recommended values for these two components would be 10kΩ and 0.01μF, about 1/10 of the values of RRES and CRES.
- As this circuit is used in noisy environments, the lines connecting the CN capacitor to the MCU RES pin and VDD should be kept as short as possible.

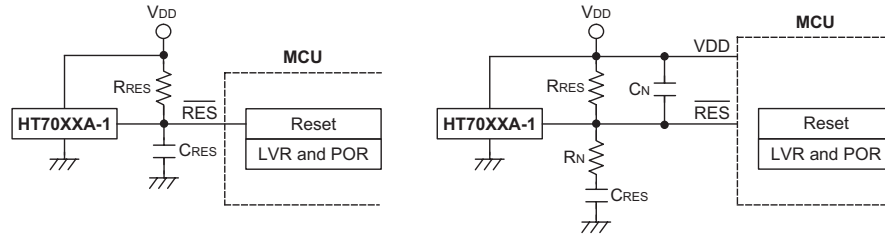
Low Voltage Reset Transistor Circuit



- When the internal low voltage reset circuit's voltage is not the same as the application's specification, an external transistor circuit can be used to supply the low voltage function.
- This circuit can provide a low voltage reset function for use in noisy environments.
- The function of the low voltage reset is determined by the resistor voltage divider RB1 and RB2 or by the Zener diode voltage.
- When using the resistor voltage divider, the low voltage reset activation point is given by the ratio $(R_{B1} + R_{B2}) / (2 \times R_{B1})$. The value of the RC should be greater than $R_{B2}/30$.

- When using the Zener diode, the low voltage reset activation point is given by $V_Z + 0.5V$, the R_{B1} resistor is provided to set the working point V_Z . It is recommended that the R_C resistor should have a value greater than $100k\Omega$ and R_{B3} has a value of about $10k\Omega$.
- The placement of transistor Q1 is important, the connections between the emitter and collector and the V_{DD} and MCU \overline{RES} pins should be as short as possible.

External Voltage Detector IC Reset Circuit



- When the internal low voltage reset circuit's voltage is not the same as the application's specification, an external voltage detector IC circuit can be used to provide the low voltage reset function.
- This circuit can provide a low voltage reset function. It requires the connection of a simple RC network or the RC network that is used for noisy environments to provide the reset function.
- The recommended values for R_{RES} , C_{RES} , R_N and C_N are the same as those for the simple RC network or the RC network that operates in noisy environments.
- The component C_{RES} which is used in the simple RC circuit, or C_N which is used in noisy environments circuit have the same requirements as the reset circuits for the simple RC network or the RC network that operates in noisy environments.

Internal POR Circuit and Internal Low Voltage Reset Circuit

- In order to increase protection for the MCU and to simplify the need for external circuitry and to reduce costs, the MCU includes both internal power on reset (POR) and low voltage reset (LVR) circuits.
- The internal POR is an internal RC circuit that will provide a short reset time during the power on time, which will enable the MCU to power up in a known initial condition. Apart from the TO and PDF flags, which are reset to "0", the function of this reset is the same as that of the \overline{RES} line. If this reset is to be used to reset the MCU, because its reset time is short, the power supply voltage must rise to its operating value in as short a time as possible.
- The LVR internal reset main function is to provide a reset to the MCU should the V_{DD} voltage fall below a specified value for a time greater than 1ms. Its effect is the same as that of the \overline{RES} line.

Internal Watchdog RC Oscillator

Functional Description

- The main function of the Watchdog timer (WDT) is to monitor the normal internal operation of the MCU hardware and software. By correct use of the clear "WDT" instructions ("CLR WDT", "CLR WDT1" and "CLR WDT2") within the application program, if the MCU is running correctly, the Watchdog timer will be prevented from overflowing. However, should the MCU malfunction the WDT will overflow and a WDT reset will be activated.
- The internal watchdog oscillator is formed by a free running fully integrated RC oscillator. Irrespective of how the program is running, even if the MCU is in the halt power down mode, the internal watchdog oscillator will always run. This oscillator will provide the timing for the WDT, and enable the WDT to continually check on the correct operation of the MCU.
- The watchdog internal timer is a free running internal RC oscillator, which if selected by the configuration options, will continually run. When the MCU enters the halt mode, as the WDT oscillator will still continue to run, it will consume some power, in the region of several mA. If in the halt mode, when the WDT overflows, a WDT reset during halt will be generated. When this happens the TO and PDF flags will be set to a known condition. By reading these two flags, the application program can determine if a WDT reset has indeed occurred and appropriate action then taken.

Process, Working Voltage and Temperature Variations

- Because the internal WDT oscillator is a fully integrated RC oscillator, the values of the Resistor and Capacitor, which are fabricated within the device, will have a high interdependency on process and temperature, which will create variations in the oscillator frequency.
- Because the oscillator is an RC oscillator, it will be affected by the operating voltage, which will in turn create variations in the oscillator frequency.
- Because of these three combined factors, process, operating voltage and temperature variations, during the design phase, the designer must take special care to ensure that an erroneous WDT reset does not occur. With regard to the characteristics of the internal RC WDT oscillator and the effects of voltage and temperature with frequency, user can consult the relevant up-to-date MCU datasheet or handbook on the Holtek web site.

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 © 2005 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>.

