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

September 2003

Contents

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 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. Holtek′s high quality embedded I/O microcontroller solutions provide a means for customers to greatly enhance the functional contents of their products, which when combined with Holtek′s comprehensive range of development tools provide designers with the means to reduce their design to market times and greatly increasing their added value.

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

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

Part I

Microcontroller Profile

Chapter 1

Hardware Structure

1

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

Introduction

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

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

Features

Technology Features

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

Kernel Features

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

Peripheral Features

- From 21 to 56 Bidirectional I/O with Pull-high Options
- Port A Wake-up Options
- External Interrupt Input
- Event Counter Input
- Full Timer Functions with Prescaler and Interrupt
- Watchdog Timer (WDT)
- HALT and Wake-up Feature for Power Saving Operation

- PFD/Buzzer Driver Outputs
- On-chip Crystal and RC Oscillator
- 32768Hz Real Time Clock (RTC) Function
- Low Voltage Reset (LVR) Feature for Brown-out Protection
- In-circuit Programming Interface with Code Protection
- Mask Version Devices Available for High Volume Production
- Full Suite of Supported Hardware and Software Tools Available

Selection Table

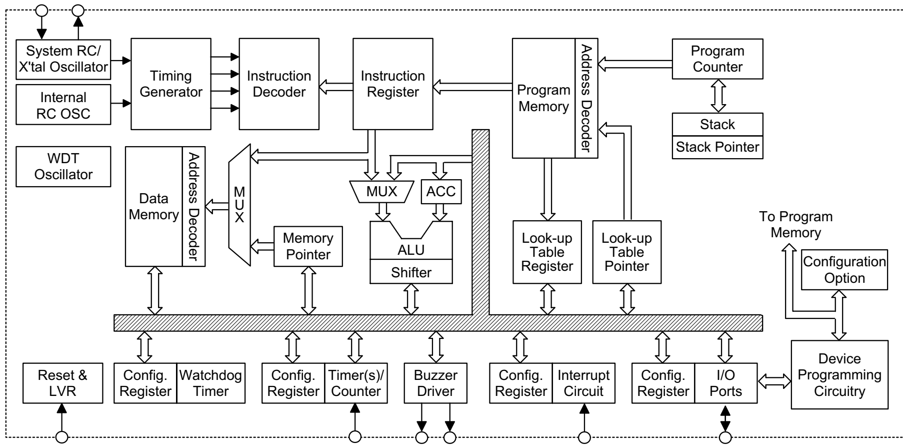
The series of I/O microcontrollers include a comprehensive range of features, some of which are standard and some of which are device dependent. Most features are common to all devices, the main feature distinguishing them are Program Memory, Data Memory capacity, I/O count and timer functions. To assist 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.	VDD	Program Memory	Data Memory	I/O	Timer	Interrupt	Stack	Package Types
HT48R10A-1 HT48C10-1	2.2V~5.5V	1K×14	64×8	21	8-bit×1	2	4	24SKDIP/SOP
HT48R30A-1 HT48C30-1	2.2V~5.5V	2K×14	96×8	25	8-bit×1	2	4	24SKDIP/SOP, 28SKDIP/SOP
HT48R50A-1 HT48C50-1	2.2V~5.5V	4K×15	160×8	35	8-bit×1 16-bit×1	3	6	28SKDIP/SOP, 48SSOP
HT48R70A-1 HT48C70-1	2.2V~5.5V	8K×16	224×8	56	16-bit×2	3	16	48SSOP, 64QFP

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

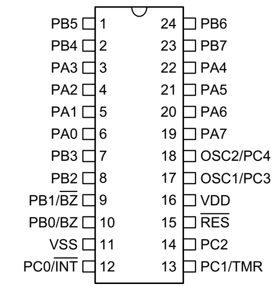
Block Diagram

The following block diagram illustrates the main functional blocks of the I/O Type microcontroller series of devices.

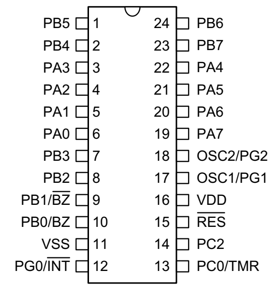


Note This block diagram represents the OTP devices, for the mask device there is no Device Programming Circuitry.

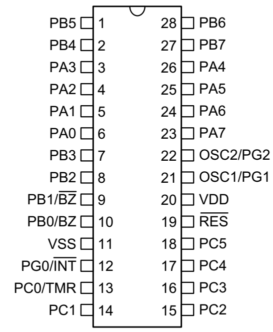
Pin Assignment



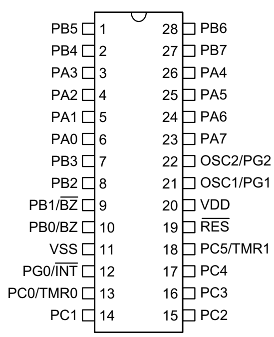
HT48R10A-1/HT48C10-1
24 SKDIP-A/SOP-A



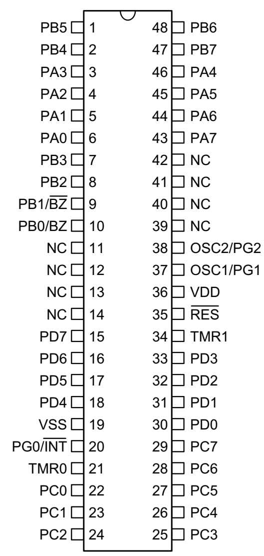
HT48R30A-1/HT48C30-1
24 SKDIP-A/SOP-A



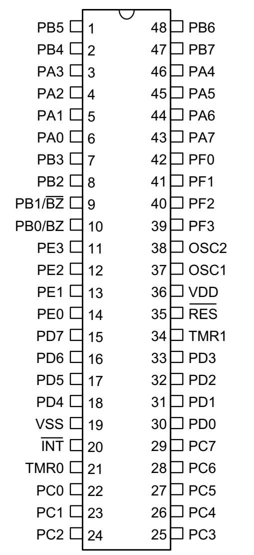
HT48R30A-1/HT48C30-1
28 SKDIP-A/SOP-A



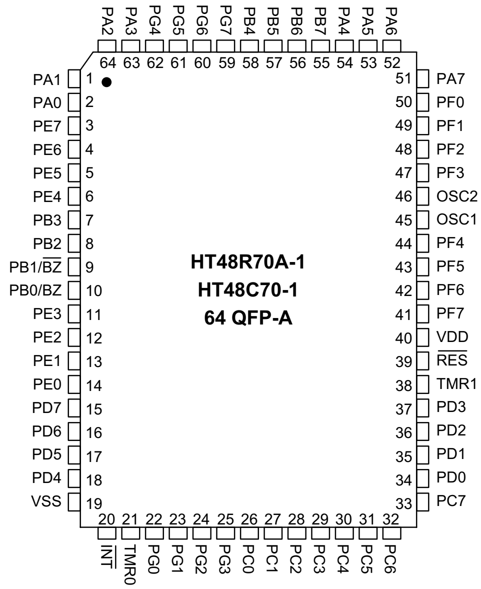
HT48R50A-1/HT48C50-1
28 SKDIP-A/SOP-A



HT48R50A-1/HT48C50-1
48 SSOP-A



HT48R70A-1/HT48C70-1
48 SSOP-A



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

Pin Description

HT48R10A-1/HT48C10-1

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

Pin Name	I/O	Configuration Option	Description
PC0/$\overline{INT}$ PC1/TMR PC2	I/O	Pull-high	Bidirectional 3-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. Pin PC0 is pin-shared with external interrupt pin $\overline{INT}$ and PC1 shared with external timer pin TMR. The external interrupt is activated on a high to low transition.
OSC1/PC3 OSC2/PC4	I O	Pull-high Crystal or RC or Int. RC+I/O or Int. RC+RTC	OSC1, OSC2 are connected to an external RC network or external Crystal (determined by configuration option) for the internal system clock. For external RC system clock operation, OSC2 is an output pin for 1/4 system clock. These two pins can also be optioned as an RTC oscillator (32768Hz) or I/O lines. In these two cases, the system clock comes from an internal RC oscillator whose nominal frequency at 5V has 4 options, 3.2MHz, 1.6MHz, 800kHz, 400kHz. If the pins are used as normal I/O pins, then pull-high options are available. If used as oscillator pins, bits PC3 and PC4 will be free for use by the application program. In this case the pull-high options are disabled.
$\overline{RES}$	I	—	Schmitt Trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground

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

HT48R30A-1/HT48C30-1

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

Pin Name	I/O	Configuration Option	Description
PC0/TMR PC1~PC5	I/O	Pull-high	Bidirectional 6-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. PC0 is pin-shared with external timer pin TMR.
PG0/$\overline{\text{INT}}$	I/O	Pull-high	Bidirectional 1-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if the pin has a pull-high resistor. PG0 is pin-shared with external interrupt pin $\overline{\text{INT}}$. The external interrupt input is activated on a high to low transition.
OSC1/PG1 OSC2/PG2	I O	Pull-high Crystal or RC or Int. RC+I/O or Int. RC+RTC	OSC1, OSC2 are connected to an external RC network or external Crystal (determined by configurations option) for the internal system clock. For external RC system clock operation, OSC2 is an output pin for 1/4 system clock. These two pins can also be optioned as an RTC oscillator (32768Hz) or I/O lines. In these two cases, the system clock comes from an internal RC oscillator whose nominal frequency at 5V has 4 options, 3.2MHz, 1.6MHz, 800kHz, 400kHz. If the pins are used as normal I/O pins, then pull-high options are available. If used as oscillator pins, bits PG1 and PG2 will be free for use by the application program. In this case the pull-high options are disabled.
$\overline{\text{RES}}$	I	—	Schmitt Trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground

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

HT48R50A-1/HT48C50-1

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

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

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

HT48R70A-1/HT48C70-1

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

Note

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

Absolute Maximum Ratings

Supply Voltage .... V_{SS}–0.3V to V_{SS}+6.0V
Input Voltage .... V_{SS}–0.3V to V_{DD}+0.3V
Storage Temperature .... –50°C to 125°C
Operating Temperature .... –40°C to 85°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

Ta=25°C

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

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
R_{PH}	Pull-high Resistance	3V	—	40	60	80	kΩ
		5V	—	10	30	50	kΩ

A.C. Characteristics

Ta=25°C

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

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

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

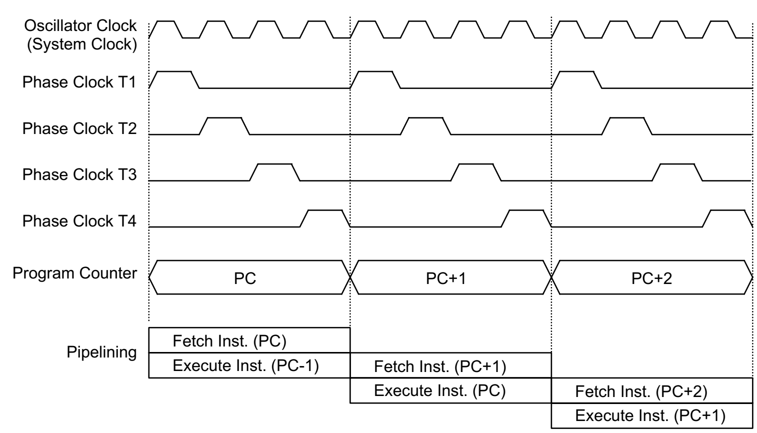
System Architecture

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

```
1            MOV A,[12H]
2            CALL DELAY
3            CPL [12H]
4            :
5            :
6  DELAY:    NOP
```

Fetch Inst. 1	Execute Inst. 1			
	Fetch Inst. 2	Execute Inst. 2		
		Fetch Inst. 3	Flush Pipeline	
			Fetch Inst. 6	Execute Inst. 6
				Fetch Inst. 7

Program Counter

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

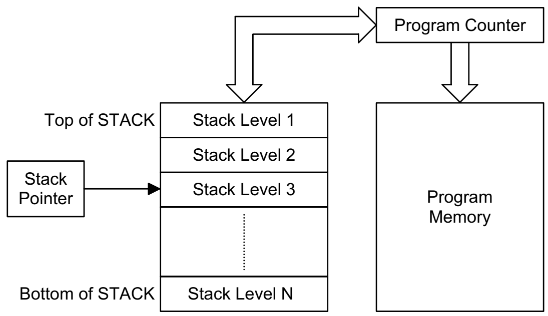
Mode	Program Counter Bits												
	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Initial Reset	0	0	0	0	0	0	0	0	0	0	0	0	0
External Interrupt	0	0	0	0	0	0	0	0	0	0	1	0	0
Timer/Event Counter 0 Overflow	0	0	0	0	0	0	0	0	0	1	0	0	0
Timer/Event Counter 1 Overflow	0	0	0	0	0	0	0	0	0	1	1	0	0
Skip	Program Counter + 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

Note
1. PC12~PC8: Current Program Counter bits
2. @7~@0: PCL bits
3. #12~#0: Instruction code bits
4. S12~S0: Stack register bits
5. For the HT48R70A-1/HT48C70-1, the Program Counter is 13 bits wide, i.e. from b12~b0.
6. For HT48R50A-1/HT48C50-1, since its Program Counter is 12 bits wide, the b12 column in the table is not applicable.
7. For the HT48R30A-1/HT48C30-1, since its Program Counter is 11 bits wide, the b11 and b12 columns in the table are not applicable.
8. For the HT48R10A-1/HT48C10-1, since its Program Counter is 10 bits wide, the b10, b11 and b12 columns in the table are not applicable.
9. The Timer/Event Counter 1 Overflow row is available only for HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1.
10. For the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 the Timer/Event Counter 0 represents the single timer, known as TMR.

Stack

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

If the stack is full and a non-masked 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 HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1, N=4, i.e. 4 levels of stack available.
2. For the HT48R50A-1/HT48C50-1, N=6, i.e. 6 levels of stack available.
3. For the HT48R70A-1/HT48C70-1, N=16, i.e. 16 levels of stack available.

Arithmetic and Logic Unit – ALU

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

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

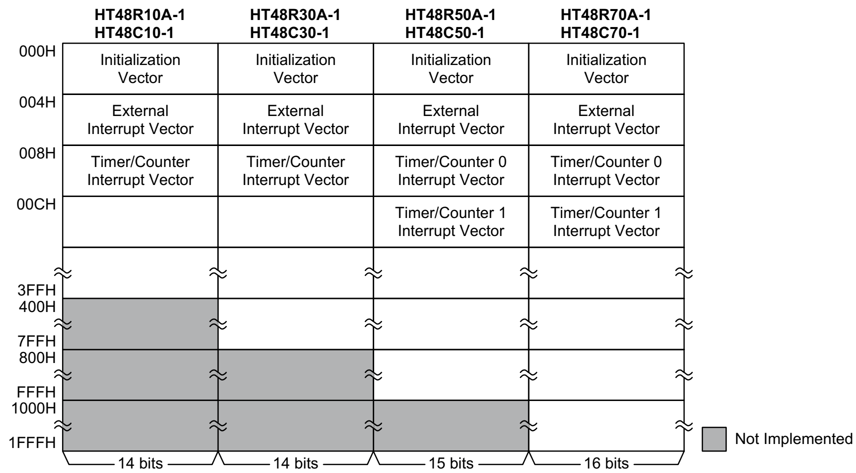
Program Memory

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

Organization

The Program Memory has a capacity of 1K by 14 to 8K 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 separate table pointer registers.

The following diagram shows the Program Memory for the I/O 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
  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
  This internal interrupt vector is used by the Timer/Event Counter. If a counter overflow occurs, the program will jump to this location and begin execution if the internal interrupt is enabled and the stack is not full. For the HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1, the Timer/Event Counter is known as Timer/Event Counter 0.
- Location 00CH
  This internal interrupt vector is used by the Timer/Event Counter. If a counter overflow occurs, the program will jump to this location and begin execution if the internal interrupt is enabled and the stack is not full. This vector is available for the HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1 only. The Timer/Event Counter is known as Timer/Event Counter 1.

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, the table pointer must first be setup by placing the lower-order address of the look-up data to be retrieved in the Table Pointer Register TBLP. This register defines the lower 8-bit address of the look-up table. 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:

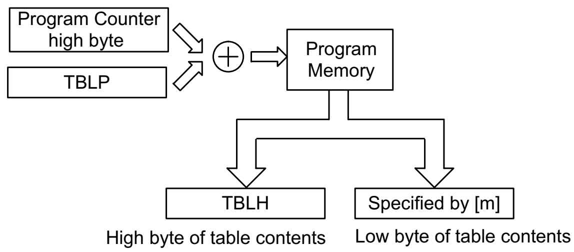


Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the HT48R10A-1 I/O microcontroller. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is ″300″ hex which refers to the start address of the last page within the 1K Program Memory of the HT48R10A-1 microcontroller. 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 high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the ″TABRDL [m]″ instruction is executed.

```
tempreg1  db  ?    ; temporary register #1
tempreg2  db  ?    ; temporary register #2
    :
    :

mov     a,06h     ; initialize table pointer - note that this address is
                  ; referenced

mov     tblp,a    ; to the last page or present page
    :
    :

tabrdl  tempreg1  ; transfers value in table referenced by table pointer
                  ; to tempregl
                  ; 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 (for HT48R10A-1)

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

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 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.

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

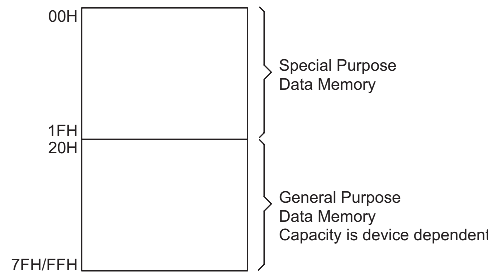
Note
1. PC12~PC8: Current Program Counter bits
2. @7~@0: Table Pointer TBLP bits
3. For HT48R70A-1/HT48C70-1, the Table address location is 13 bits, i.e. from b12~b0.
4. For HT48R50A-1/HT48C50-1, the Table address location is 12 bits, i.e. from b11~b0.
5. For HT48R30A-1/HT48C30-1, the Table address location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

Data Memory

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

Organization

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

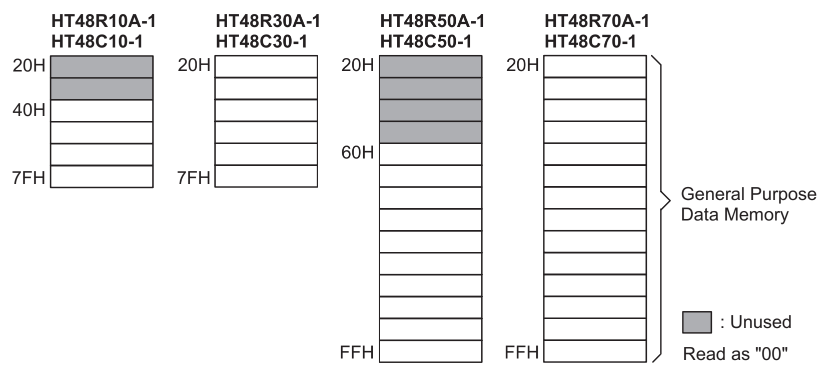


Note Most of the Data Memory bits can be directly manipulated using the ″SET [m].i″ and ″CLR [m].i″ with the exception of a few dedicated bits. The Data Memory can also be accessed through the memory pointer register MP.

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.

The following diagram shows a General Purpose Data Memory Organization Map of the I/O Type microcontroller:



Special Purpose Data Memory

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

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

	HT48R10A-1 HT48C10-1	HT48R30A-1 HT48C30-1	HT48R50A-1 HT48C50-1	HT48R70A-1 HT48C70-1
00H	IAR	IAR	IAR0	IAR0
01H	MP	MP	MP0	MP0
02H			IAR1	IAR1
03H			MP1	MP1
04H				
05H	ACC	ACC	ACC	ACC
06H	PCL	PCL	PCL	PCL
07H	TBLP	TBLP	TBLP	TBLP
08H	TBLH	TBLH	TBLH	TBLH
09H	WDTS	WDTS	WDTS	WDTS
0AH	STATUS	STATUS	STATUS	STATUS
0BH	INTC	INTC	INTC	INTC
0CH				TMR0H
0DH	TMR	TMR	TMR0	TMR0L
0EH	TMRC	TMRC	TMR0C	TMR0C
0FH			TMR1H	TMR1H
10H			TMR1L	TMR1L
11H			TMR1C	TMR1C
12H	PA	PA	PA	PA
13H	PAC	PAC	PAC	PAC
14H	PB	PB	PB	PB
15H	PBC	PBC	PBC	PBC
16H	PC	PC	PC	PC
17H	PCC	PCC	PCC	PCC
18H			PD	PD
19H			PDC	PDC
1AH				PE
1BH				PEC
1CH				PF
1DH				PFC
1EH		PG	PG	PG
1FH		PGC	PGC	PGC

Special Purpose Data Memory

: Unused

Read as "00"

Special Function Registers

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

Indirect Addressing Registers – IAR, IAR0, IAR1

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

Memory Pointers – MP, MP0, MP1

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

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

The following example 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 [00h]           ; 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.

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, TBLH

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

Watchdog Timer Register – WDTS

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

Status Register – STATUS

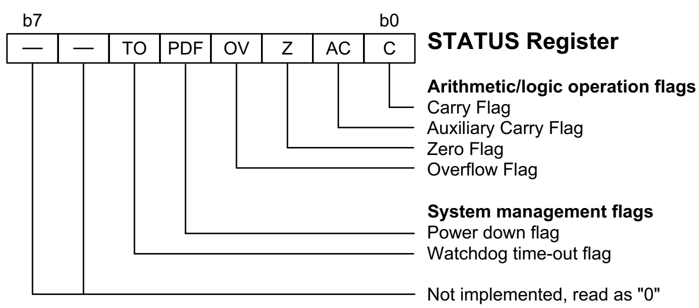
This 8-bit register (0AH) contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), power down flag (PDF), and watchdog time-out flag (TO). It also records the status information and controls the operation sequence.

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

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

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

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



Interrupt Control Register – INTC

This 8-bit register, known as the INTC 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 enable/disable and is used to set all of the interrupt enable bits on or off. This bit is cleared when an interrupt routine is entered to disable further interrupt and is set by executing the ″RETI″ instruction.

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

Timer/Event Counter Registers

Depending upon which device is selected, all devices contain one or two integrated Timer/Event Counters of either 8-bit or 16-bit size. For devices with a single 8-bit timer counter, an associated register, known as TMR, is the location where the timer value is located. An associated control register, known as TMRC, contains the setup information for the TMR register. For devices with two timers, the individual timers are known as TMR0 and TMR1 with their respective control registers known as TMR0C and TMR1C. In the case of 16-bit timers, the actual value stored in the timer requires two bytes, a high byte and a low byte. These register pairs are known as TMR0L/TMR0H and TMR1L/TMR1H. Note that the timer registers can be directly written to in order to preload their contents with fixed data to allow different time intervals to be setup.

Input/Output Ports and Control Registers

Within the area of Special Function Registers, the I/O registers and their associated control registers play a prominent role. All I/O ports have a designated register correspondingly labeled as PA, PB, PC, etc. These labeled I/O registers are mapped to specific addresses within the Data Memory as shown in the Data Memory table which are used to transfer the appropriate output or input data on that port. With each I/O port there is an associated control register labeled PAC, PBC, PCC, etc. also mapped to specific addresses with the Data Memory. The control register specifies which pins of that port are set as inputs and which are set as outputs. To setup a pin as an input, the corresponding bit of the control register must be set high, for an output it must be set low. During program initialization, it is important to first setup the control registers to specify which pins are outputs and which are inputs before reading data from or writing data to the I/O ports. One flexible feature of these registers is the ability to directly program single bits using the ″SET [m].i″ and ″CLR [m].i″ instructions. The ability to change I/O pins from output to input and vice-versa by manipulating specific bits of the I/O control registers during normal program operation is a useful feature of these devices.

Input/Output Ports

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

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

Pull-high Resistors

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

Port A Wake-up

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

I/O Port Control Registers

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

Pin-shared Functions

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

→ **Buzzer**

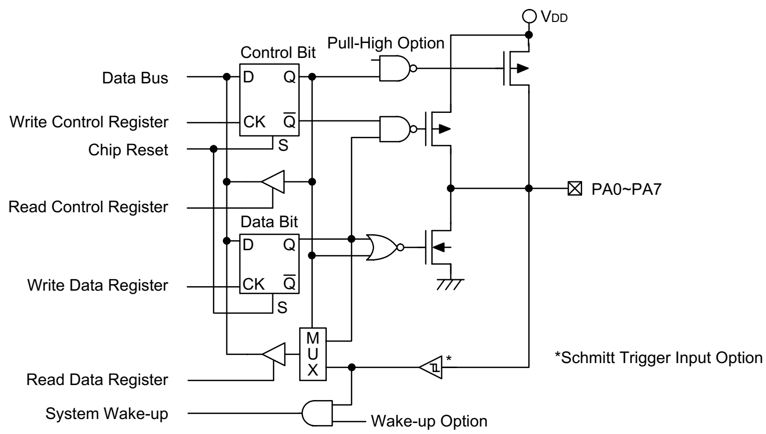
The buzzer pins BZ and $\overline{\text{BZ}}$ are pin-shared with I/O pins PB0 and PB1. If configured as buzzer pins, the correct hardware and software options must be selected.

→ **External Interrupt Input**

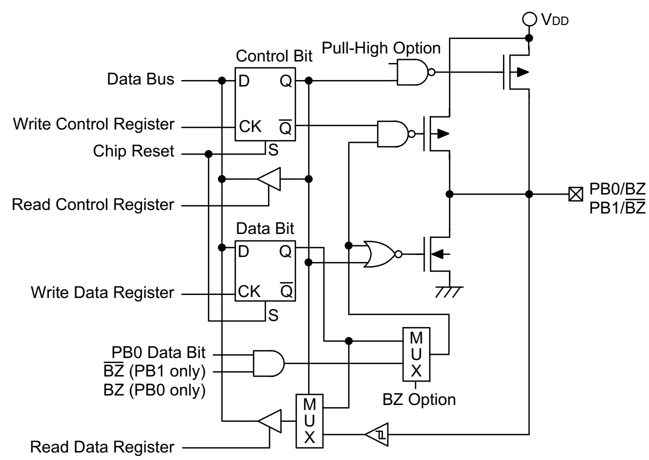
The external interrupt pin $\overline{\text{INT}}$ is pin-shared with the I/O pin PC0 or PG0 depending upon which device is used. However, for the HT48R70A-1/HT48C70-1 devices, the external interrupt pin is an independent non shared pin. For this pin to operate as an external interrupt pin and not as a normal I/O pin, the corresponding external interrupt enable bits in the INTC interrupt control register must be correctly set. For applications not requiring an external interrupt input, with the exception of the HT48R70A-1/HT48C70-1 devices, the pin can be used as a normal I/O pin, however, to do this, the external interrupt enable bits in the INTC register must be disabled.

→ **External Timer Clock Input**

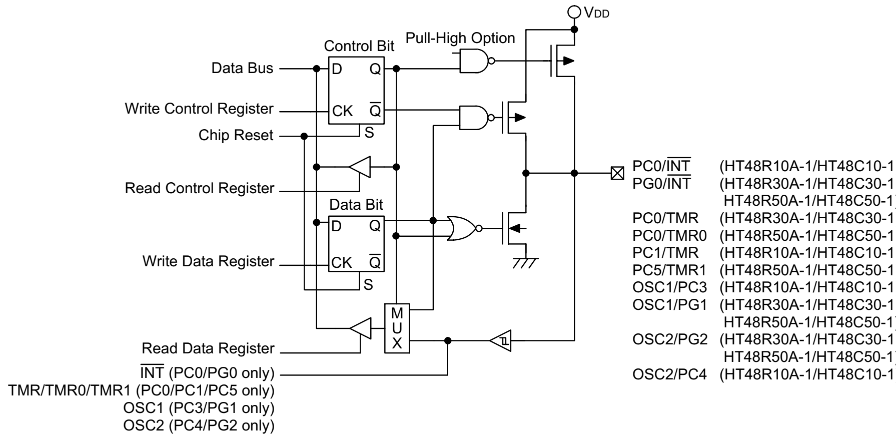
Each device contains either one or two timers depending upon which one is chosen. Each timer has an external input pin, which in the case of devices with a single timer, is known as TMR and in the case of devices with two timers are known as TMR0 and TMR1. For all devices with a single timer, the external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



PA Input/Output Ports



PB0/PB1 Input/Output Ports



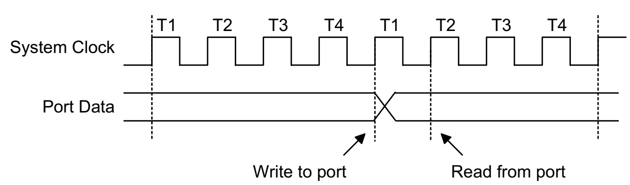
PB2~PB7, PC, PD, PE, PF and PG Input/Output Ports

→ **Oscillator**

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

Programming Considerations

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



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

Timer/Event Counters

The provision of timers form an important part of any microcontroller giving the designer a means of carrying out time related functions. The devices in the I/O range contain either one or two count up timers of either 8 or 16-bit capacity depending upon which device is selected. As each timer has three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width measurement device. In the case of 8-bit timers, the provision of an internal 8-stage prescaler to 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 can be configured to come from the internal clock source or from the external timer pin. The accompanying table lists the associated timer register names.

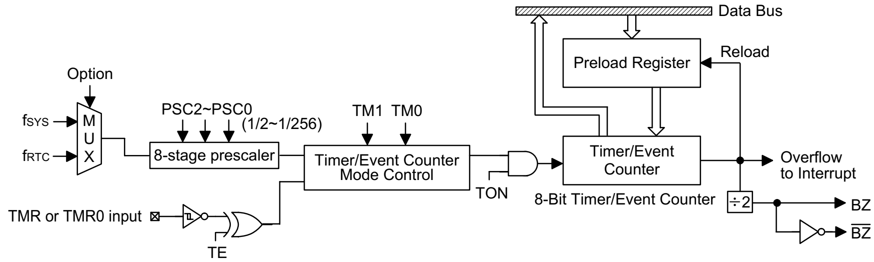
	HT48R10A-1 HT48C10-1	**HT48R30A-1 HT48C30-1**	**HT48R50A-1 HT48C50-1**	**HT48R70A-1 HT48C70-1**
No. of 8-bit Timers	1	1	1	0
Timer Register Name	TMR	TMR	TMR0	—
Timer Control Register	TMRC	TMRC	TMR0C	—
No. of 16-bit Timers	0	0	1	2
Timer Register Name	—	—	TMR1L/TMR1H	TMR0L/TMR0H TMR1L/TMR1H
Timer Control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

An external clock source is used when the timer is in the event counting mode, the clock source being provided on the external timer pin known as TMR, TMR0 or TMR1 depending on which device is selected. These external pins may be pin-shared with other I/O pins depending upon which device and package is chosen. Depending upon the condition of the TE 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.

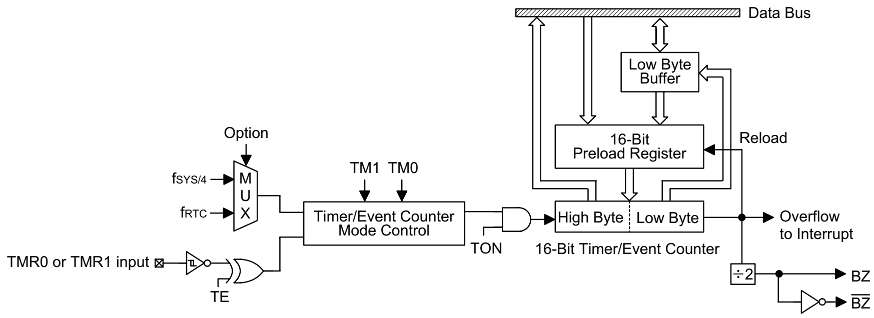
Configuring the Timer/Event Counter Input Clock Source

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 bits PSC2, PSC1 and PSC0.

An external clock source is used when the timer is in the event counting mode, the clock source being provided on an external timer pin, TMR, TMR0 or TMR1 depending upon which device and which timer is used. Depending upon the condition of the TE 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



16-bit Timer/Event Counter Structure

Timer Registers – TMR, TMR0, TMR0L/TMR0H, 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 TMR for the HT48R10A-1/HT48C10-1 and HT48R30A-1/HT48C30-1 devices and TMR0 for the HT48R50A-1/HT48C50-1 devices. In the case of the 16-bit timers, a pair of 8-bit registers is required to store the 16-bit timer value. These register pairs are known as TMR0L/TMR0H or TMR1L/TMR1H depending upon which device and timer is used. The value in the timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFH for the 8-bit timer or FFFFH for the 16-bit timers at which point the timer overflows and an internal interrupt signal 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 devices with 16-bit timers, which have both low byte and high byte timer registers, accessing these registers is carried out in a specific way. It must be noted that when using instructions to preload data into the low byte register, namely, TMRL, TMR0L or TMR1L, the data will only be placed in a low byte buffer and not directly into the low byte register. The actual transfer of the data into the low byte register is only carried out when a write to its associated high byte register, namely, TMRH, TMR0H or TMR1H, is executed. On the other hand, using instructions to preload data into the high byte timer register will result in the data being directly written to the high byte register. At the same time the data in the low byte buffer will be transferred into its associated low byte register. For this reason, when preloading data into the 16-bit timer registers, the low byte should be written first. It must also be noted that to read the contents of the low byte register, a read to the high byte register must first be executed to latch the contents of the low byte buffer into its associated low byte register. After this has been done the low byte register can be read in the normal way. Note that reading the low byte timer register will only result in reading the previously latched contents of the low byte buffer and not the actual contents of the low byte timer register.

Timer Control Registers – TMRC, TMR0C, TMR1C

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

8-bit Timer/Event Counter Control Register

b7							b0
TM1	TM0	—	TON	TE	PSC2	PSC1	PSC0

Timer prescaler rate select

PSC2	PSC1	PSC0	Timer Rate
0	0	0	1:2
0	0	1	1:4
0	1	0	1:8
0	1	1	1:16
1	0	0	1:32
1	0	1	1:64
1	1	0	1:128
1	1	1	1:256

Event Counter active edge select
1: count on falling edge
0: count on rising edge

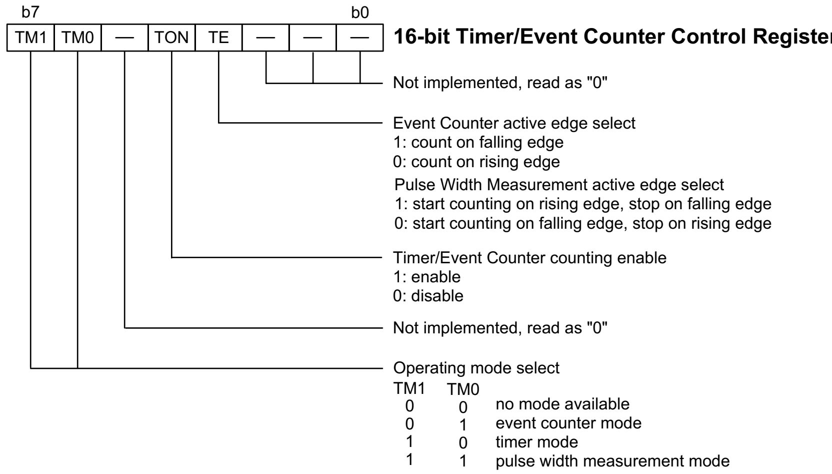
Pulse Width Measurement active edge select
1: start counting on rising edge, stop on falling edge
0: start counting on falling edge, stop on rising edge

Timer/Event Counter counting enable
1: enable
0: disable

Not implemented, read as "0"

Operating mode select

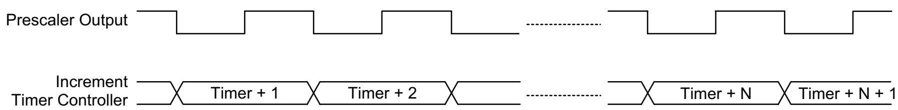
TM1	TM0	
0	0	no mode available
0	1	event counter mode
1	0	timer mode
1	1	pulse width measurement mode



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 TM0 and TM1 must be set to the required logic levels. The timer-on bit TON 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 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 TE or bit 3 of the TMRC register.

Configuring the Timer Mode

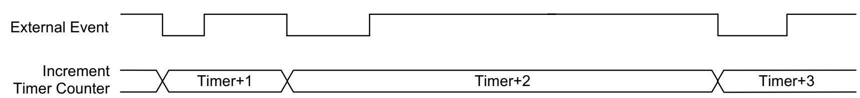
In this mode, the timer can be utilized to measure fixed time intervals, providing an internal interrupt signal each time the counter overflows. To operate in this mode, bits TM1 (bit7) and TM0 (bit6) of the TMRC register must be set to 1 and 0 respectively. In this mode, the internal clock is used as the timer clock. For the 8-bit Timer/Event Counter, the input clock frequency to the timer is f_{SYS} or f_{RTC} divided by the value programmed into the timer prescaler, the value of which is determined by bits PSC2~PSC0 of the TMRC register. For the 16-bit Timer/Event Counter, the input clock frequency to the timer is $f_{SYS}/4$ or the f_{RTC}. There is no prescaler function for the 16-bit timer. The timer-on bit, TON 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

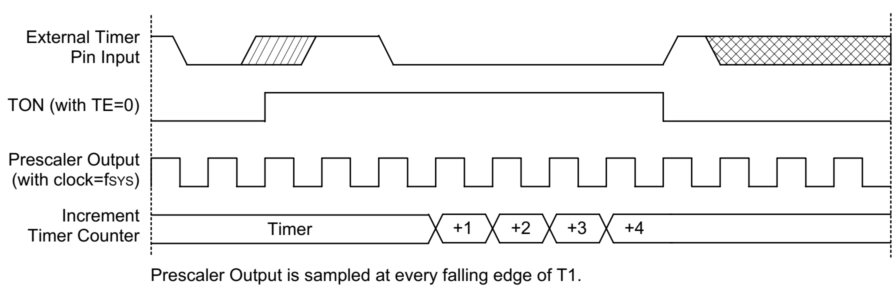
In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the internal timer. For the timer to operate in the event counting mode, bits TM1 and TM0 of the TMRC register must be set to 0 and 1 respectively. The timer-on bit, TON must be set high to enable the timer to count. With TE low, the counter will increment each time the external timer pin receives a low to high transition. If TE 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 TM0 and TM1 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 TM0 and TM1 must both be set high. If the TE 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 TON bit will be automatically



Pulse Width Measurement Mode Timing Chart

reset to zero and the timer will stop counting. If the TE bit is high, the timer will begin counting once a low to high transition has been received on the external timer pin and stop counting when the external timer pin returns to its original low level. As before, the TON bit will be automatically reset to zero and the timer will stop counting. It is important to note that in the Pulse Width Measurement Mode, the TON bit is automatically reset to zero when the external control signal on the external timer pin returns to its original level, whereas in the other two modes the TON 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 TON bit has now been reset any further transitions on the external timer pin will be ignored. Not until the TON 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 TM0 and TM1 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.

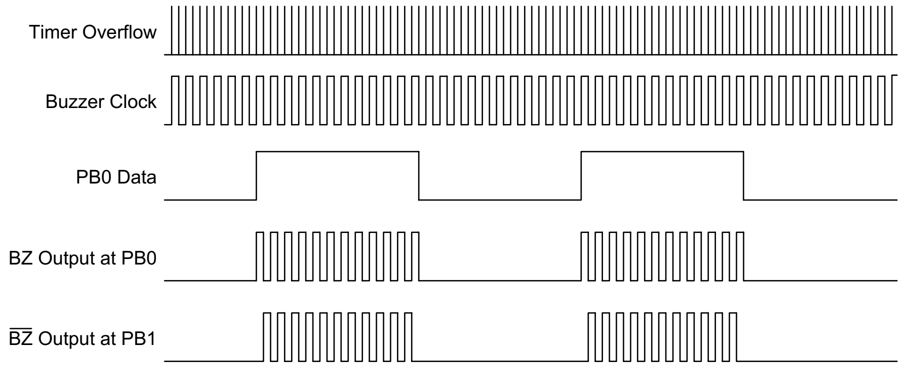
Programmable Frequency Divider (PFD) and Buzzer Application

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

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

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

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



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 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 signals for buzzer driving and as a Timer Interrupt.

I/O Interfacing

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

Programming Considerations

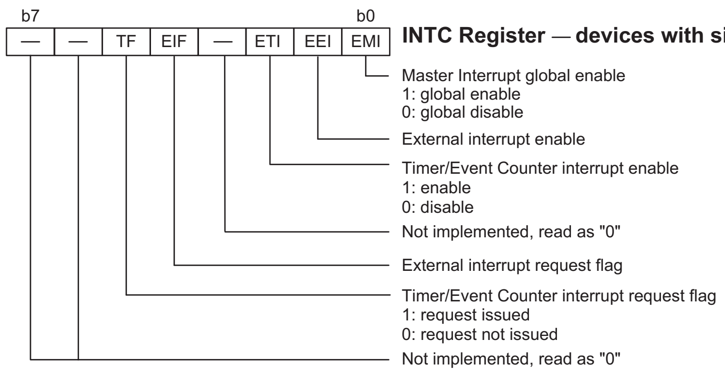
When configured to run in the timer mode, the internal system clock or RTC is used as the timer clock source and is therefore synchronized with the overall operation of the microcontroller. In this mode, when the appropriate timer register is full, the microcontroller will generate an internal interrupt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

Interrupts

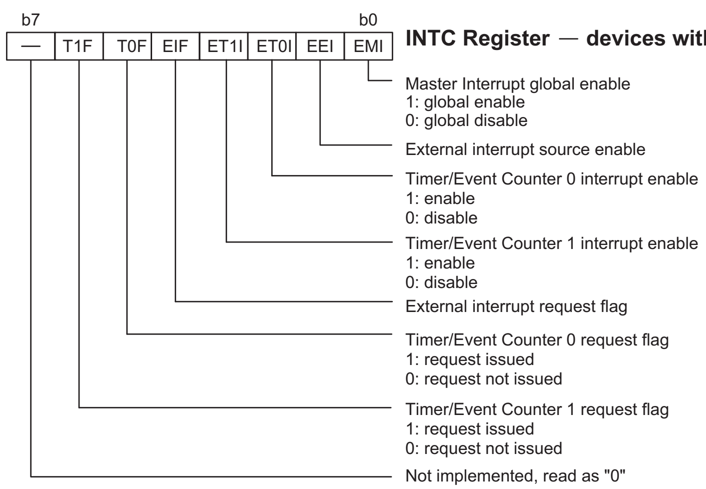
The I/O series microcontrollers provide both external interrupt and internal Timer/Event Counter interrupt functions. The Interrupt Control Register (INTC;0BH) contains the interrupt control bits to set the enable/disable and the interrupt request flags.

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 a certain interrupt requires servicing within the service routine, the EMI bit and the corresponding bit of the INTC may be set to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the SP is decremented. If immediate service is desired, the stack must be prevented from becoming full.

INTC Register — devices with single timer

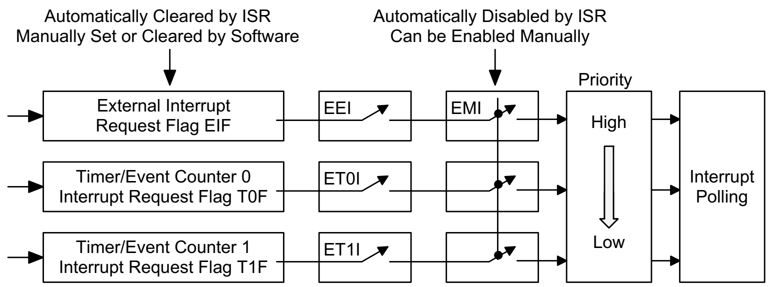


INTC Register — devices with two timers



All of these 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 register or status register (STATUS) are altered by the interrupt service program which may corrupt the desired control sequence, then the contents should be saved in advance.

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



Note In the figure, the T0F and T1F interrupt request flags and the ET0I and ET1I interrupt enable bits refer to the HT48R70A-1/ HT48C70-1 and HT48R50A-1/HT48C50-1 devices, which have two timers. For the HT48R10A-1/ HT48C10-1 and HT48R30A-1/HT48C30-1, which only have one timer, the Timer/Event Counter 0 represents the single timer, known as TMR and has an interrupt request flag known as TF and an interrupt enable bit known as ETI.

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. External interrupts are triggered by a high to low transition of the $\overline{\text{INT}}$ line, after which the related interrupt request flag (EIF; bit 4 of 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 EMI bits 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. In the case of devices with a single timer, this is bit 2 of the INTC register and is known as ETI. For devices with two timers, the Timer 0 interrupt enable is bit 2 and known as ET0I while the Timer 1 interrupt enable is bit 3 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 devices with a single timer, this is bit 5 of the INTC register and is known as TF. In the case of devices with two timers, the Timer 0 request flag is bit 5 and known as T0F, while the Timer 1 request flag is bit 6 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 interrupt will be generated when the timer overflows. This will create a subroutine call to location 08H for devices with a single timer. For devices with two timers, a subroutine call to location 08H will occur for Timer 0 and a subroutine call to location 0CH for Timer 1. When an internal interrupt occurs, the interrupt request flag, TF, T0F or T1F will be reset and the EMI bit will be cleared to disable other interrupts.

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

Note The table is applicable for the HT48R70A-1/ HT48C70-1 and HT48R50A-1/HT48C50-1 devices which have two timers, known as TMR0 and TMR1. For the HT48R30A-1/HT48C30-1 and HT48R10A-1/HT48C10-1, which only have one timer, the Timer/Event Counter 0 represents the single timer, known as TMR.

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.

Programming Considerations

The Timer/Event Counter interrupt request flags, TF, T0F and T1F, external interrupt request flag EIF, enable Timer/Event Counter interrupt bits ETI, 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 or 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.

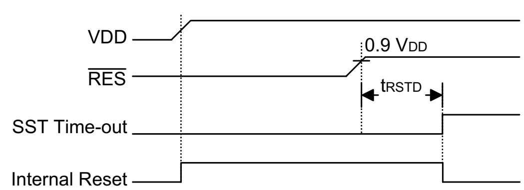
Reset

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.

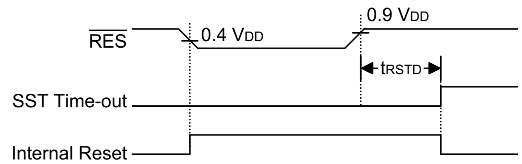
Although the microcontroller has an internal RC reset function, due to unstable power on conditions, an external RC network connected to the $\overline{\text{RES}}$ pin is generally recommended. This time delay created by the RC network ensures that the $\overline{\text{RES}}$ pin remains low for an 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

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

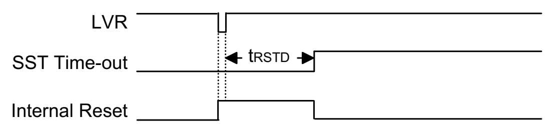
This type of reset occurs when the microcontroller is already running and the $\overline{\text{RES}}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this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



$\overline{\text{RES}}$ Reset Timing Chart

→ **Low Voltage Reset – LVR**

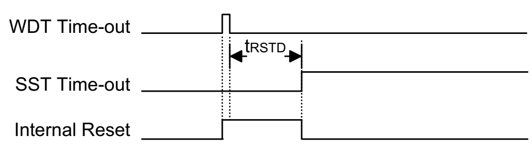
The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing the battery, the LVR will automatically reset the device internally. For a valid LVR signal, a low voltage, i.e. a voltage in the range between 0.9V~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.



Low Voltage Reset Timing Chart

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

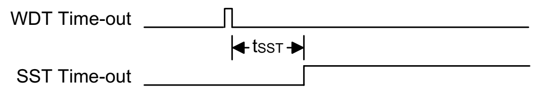
The Watchdog Time-out Reset during normal operation is the same as $\overline{\text{RES}}$ reset except that the Watchdog Time-out flag TO will be set to 1.



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	$\overline{RES}$ reset during power on
u	u	$\overline{RES}$ 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 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

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.

HT48R10A-1/HT48C10-1

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

″u″ stands for unchanged

″x″ stands for unknown

″–″ stands for unimplemented

HT48R30A-1/HT48C30-1

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

"u" stands for unchanged

"x" stands for unknown

"–" stands for unimplemented

HT48R50A-1/HT48C50-1

Register	Reset (Power On)	$\overline{RES}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
MP0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu 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	–xxx xxxx	–uuu uuuu	–uuu uuuu	–uuu uuuu
STATUS	––00 xxxx	––uu uuuu	––1u uuuu	––11 uuuu
INTC	–000 0000	–000 0000	–000 0000	–uuu uuuu
WDTS	0000 0111	0000 0111	0000 0111	uuuu uuuu
TMR0	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0C	00–0 1000	00–0 1000	00–0 1000	uu–u uuuu
TMR1H	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1L	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1C	00–0 1–––	00–0 1–––	00–0 1–––	uu–u u–––
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PD	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PG	–––– –111	–––– –111	–––– –111	–––– –uuu
PGC	–––– –111	–––– –111	–––– –111	–––– –uuu

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

HT48R70A-1/HT48C70-1

Register	Reset (Power On)	$\overline{RES}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
MP0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu 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	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
STATUS	– –00 xxxx	– –uu uuuu	– –1u uuuu	– –11 uuuu
INTC	–000 0000	–000 0000	–000 0000	–uuu uuuu
WDTS	0000 0111	0000 0111	0000 0111	uuuu uuuu
TMR0H	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0L	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0C	00–0 1– – –	00–0 1– – –	00–0 1– – –	uu–u u– – –
TMR1H	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1L	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1C	00–0 1– – –	00–0 1– – –	00–0 1– – –	uu–u u– – –
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PD	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PE	1111 1111	1111 1111	1111 1111	uuuu uuuu
PEC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PF	1111 1111	1111 1111	1111 1111	uuuu uuuu
PFC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PG	1111 1111	1111 1111	1111 1111	uuuu uuuu
PGC	1111 1111	1111 1111	1111 1111	uuuu uuuu

″u″ stands for unchanged
″x″ stands for unknown
″–″ stands for unimplemented

Oscillator

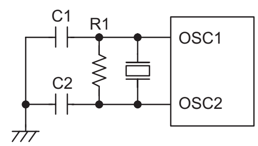
Various oscillator options offer the user a wide range of functions according to their various application requirements. Three types of system clocks can be selected while various clock source options for the Watchdog Timer as well as a real time clock function are provided for maximum flexibility. All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

System Clock Configurations

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

System Crystal/Ceramic Oscillator

For the crystal oscillator configuration, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation with no other external components required. A ceramic resonator can be used instead of a crystal but two small value capacitors should be connected between OSC1, OSC2 and ground.



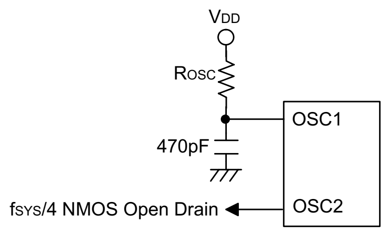
Crystal/Ceramic Oscillator

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

Crystal or Resonator	C1, C2	R1
4MHz Crystal	0pF	10kΩ
4MHz Resonator (3 pin)	0pF	12kΩ
4MHz Resonator (2 pin)	10pF	12kΩ
3.58MHz Crystal	0pF	10kΩ
3.58MHz Resonator (2 pin)	25pF	10kΩ
2MHz Crystal & Resonator (2 pin)	25pF	10kΩ
1MHz Crystal	35pF	27kΩ
480kHz Resonator	300pF	9.1kΩ
455kHz Resonator	300pF	10kΩ
429kHz Resonator	300pF	10kΩ

System RC Oscillator

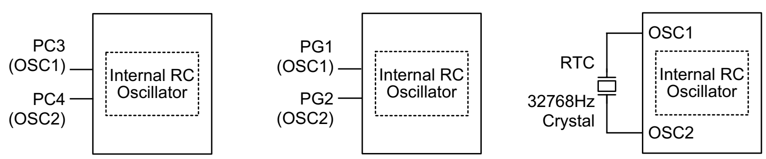
Using the external RC network as an oscillator requires that a resistor, with a value between 24kΩ and 1MΩ, is connected between OSC1 and VDD, and a 470pF capacitor is connected to ground. The generated system clock divided by 4 will be provided on OSC2 as an output which can be used for external synchronization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



RC Oscillator

Internal System RC Oscillator

In addition to the external crystal/resonator or external RC system clock configurations, the devices also contain an internal RC system clock. This oscillator is integral to the microcontroller and requires no external components. The frequency of this internal RC oscillator can be selected by configuration option to have a typical value at 5V of either 3.2MHz, 1.6MHz, 800kHz or 400kHz. The 1.6MHz, 800kHz and 400kHz frequencies are internally generated by sequentially dividing the base 3.2MHz frequency by two. Note that if this internal system clock option is selected then with the exception of the HT48R70A-1/HT48C70-1 series, the OSC1 and OSC2 pins are free for use as normal I/O pins. The OSC1 and OSC2 pins can also be connected to a 32768Hz crystal for use as an RTC oscillator on all devices. If the RTC oscillator is to be used in user applications, the internal RC oscillator must be used as the system clock. Note that the oscillation frequency of this internal system RC oscillator can vary with VDD, temperature and process variations.



- **Internal RC System Oscillator for HT48R10A-1/HT48C10-1**
- **Oscillator Pins Used as I/Os**

- **Internal RC System Oscillator for HT48R30A-1/HT48C30-1 and HT48R50A-1/HT48C50-1**
- **Oscillator Pins Used as I/Os**

- **Internal RC System Oscillator**
- **RTC 32768Hz Crystal Connected to Oscillator Pins**
- **Applicable to all Devices**

RTC Oscillator

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

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

Watchdog Timer Oscillator

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

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
- All of the I/O ports remain unchanged
- The PDF flag is set and the TO flag is cleared

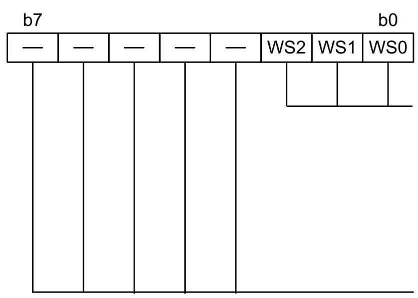
The system can leave the HALT mode by means of a reset, an interrupt, an external falling edge signal on Port A or a WDT overflow. A reset will initialize a chip reset and a WDT overflow will initialize a WDT time-out Reset from HALT but by examining the TO and PDF flags the source of the reset can be 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 PC and SP; the other flags remain in their original status.

Port A wake-up and interrupt methods can be considered as a continuation of normal execution. Each bit in Port A 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. The WDT clock is supplied by one of three sources selected by configuration option: its own self contained dedicated internal WDT oscillator, the instruction clock (system clock divided by 4) or the 32kHz RTC oscillator. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

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



WDTS Register

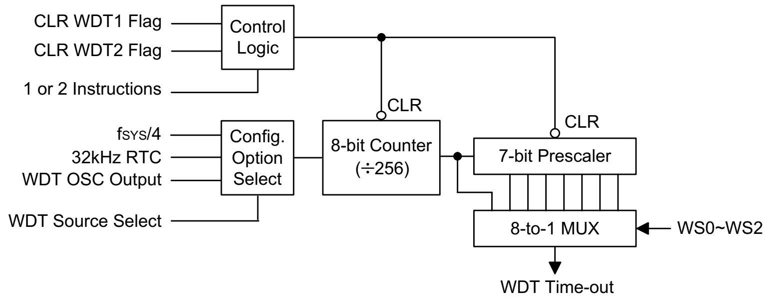
WDT prescaler rate select

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

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

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 WDT prescaler. The first is an external hard-



Watchdog Timer

ware reset (a low level on the $\overline{RES}$ pin), the second is via software instructions and the third is via a ″HALT″ instruction. There are two methods of using software instructions to clear the Watchdog Timer, one of which must be chosen by configuration option. The first option is to use the single ″CLR WDT″ instruction while the second is to use the two commands ″CLR WDT1″ and ″CLR WDT2″. For the first option, a simple execution of ″CLR WDT″ will clear the WDT while for the second option, both ″CLR WDT1″ and ″CLR WDT2″ must both be executed to successfully clear the WDT. Note that for this second option, if ″CLR WDT1″ is used to clear the WDT, successive executions of this instruction will have no effect, only the execution of a ″CLR WDT2″ instruction will clear the WDT. Similarly, after the ″CLR WDT2″ instruction has been executed, only a successive ″CLR WDT1″ instruction can clear the Watchdog Timer.

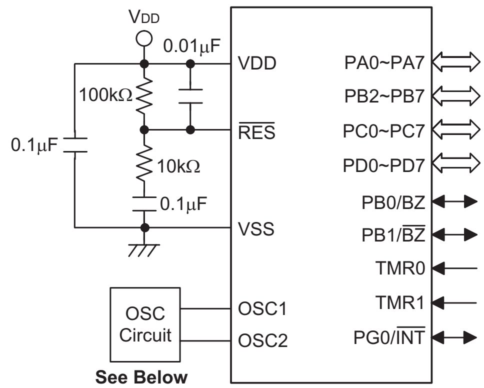
Configuration Options

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.

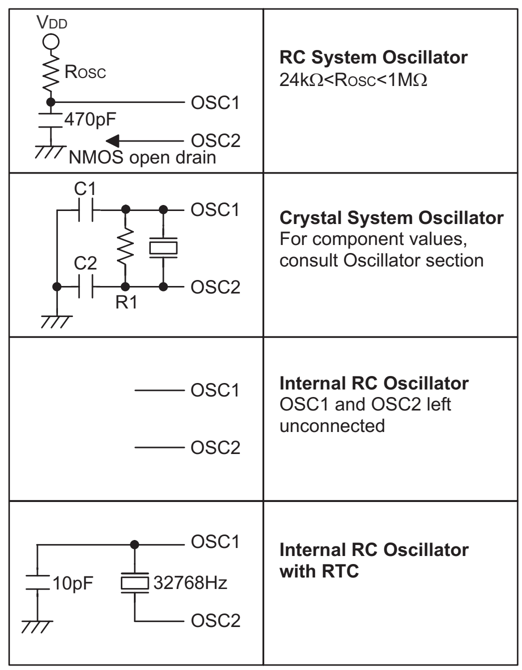
No.	Option
1	WDT clock source: WDT oscillator or $f_{SYS}/4$ or RTC oscillator or disable
2	CLRWDT instructions: 1 or 2 instructions
3	Timer/Event Counter TMR or TMR0 clock source: f_{SYS} or RTC oscillator
4	Timer/Event Counter TMR1 clock source: $f_{SYS}/4$ or RTC oscillator (HT48R50A-1/HT48C50-1 and HT48R70A-1/HT48C70-1 only)
5	PA0~PA7 wake-up: enable or disable
6	PA Schmitt Trigger or non Schmitt Trigger
7	PA, PB, PC, PD, PE, PF and PG pull-high enable or disable (port numbers is device dependent)
8	Buzzer function: single BZ enable or both BZ and $\overline{BZ}$ enable or both disable Buzzer clock source: timer 0 or timer 1 (for HT48R50A-1/HT48C50-1, HT48R70A-1/HT48C70-1)
9	LVR function: enable or disable
10	External RC system clock/External Crystal System Clock/Internal RC system clock plus RTC clock/Internal RC system plus I/O (last option not applicable to HT48R70A-1/HT48C70-1)
11	Internal RC frequency selection 3.2MHz, 1.6MHz, 800kHz or 400kHz.

Application Circuits

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



HT48R50A-1/HT48C50-1



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 0.5μs and branch or call instructions would be implemented within 1μs. 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″ or ″CLR WDT2″ instructions are 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 ← 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] ← 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 ← 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 ← 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] ← 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. Repetitively 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. Repetitively 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 ← [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 ← 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 ← [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 ← 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] ← 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 ← ACC ″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 ← [m].(i+1); (i = 0~6)
[m].7 ← C
C ← [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 ← [m].(i+1); (i = 0~6)
ACC.7 ← C
C ← [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 ← ACC − [m] − $\overline{C}$

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] ← ACC − [m] − $\overline{C}$

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] ← [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 ≠ 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 ← 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] ← 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 ← 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~[m].0 ↔ [m].7 ~ [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 ~ ACC.0 ← [m].7 ~ [m].4
ACC.7 ~ ACC.4 ← [m].3 ~ [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
[*optional items*]	Syntax elements that are enclosed by a pair of brackets are optional. For example, the syntax of the command line is as follows: **HASM** [*options*] *filename* [;] In the above command line, *options* and semicolon; are both optional, but *filename* is required, except for the following case: Brackets in the instruction operands. In this case, the brackets refer to memory address.
{*choice1* | *choice2*}	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...	Three dots following an item signify that more items with the same form may be entered. For example, the directive PUBLIC has the following form: **PUBLIC** *name1* [,*name2* [,...]] In the above form, the three dots following *name2* indicate that many names can be entered as long as each is preceded by a comma.

Statement Syntax

The construction of each statement is as follows:

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

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

Name

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

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

with the following restrictions :

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

Operation

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

Operand

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

Comment

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

Assembly Directives

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

Conditional Assembly Directives

The conditional block has the following form:

IF
statements
[**ELSE**
statements]
ENDIF

→ **Syntax**
IF *expression*
IFE *expression*

- Description
  The directives **IF** and **IFE** test the *expression* following them.
  The **IF** directive grants assembly if the value of the *expression* is true, i.e. non-zero.
  The **IFE** directive grants assembly if the value of the *expression* is false, i.e. zero.
- Example

```
IF  debugcase
        ACC1    equ 5
        extern  username: byte
ENDIF
```

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

→ **Syntax**
IFDEF *name*
IFNDEF *name*

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

```
IFDEF       buf_flag
            buffer  DB  20 dup(?)
ENDIF
```

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

File Control Directives

→ **Syntax**
INCLUDE *file-name*
or
INCLUDE "*file-name*"

- Description
  This directive inserts source codes from the source file given by `file-name` into the current source file during assembly. Cross Assembler supports at most 7 nesting levels.
- Example
```
INCLUDE macro.def
```
  In this example, the Cross Assembler inserts the source codes from the file `macro.def` into the current source file.

→ **Syntax**
PAGE *size*

- Description
  This directive specifies the number of the lines in a page of the program listing file. The page size must be within the range from 10 to 255, the default page size is 60.
- Example
```
PAGE 57
```
  This example sets the maximum page size of the listing file to 57 lines.

→ **Syntax**
.LIST
.NOLIST

- Description
  The directives **.LIST** and **.NOLIST** decide whether or not the source program lines are to be copied to the program listing file. **.NOLIST** suppresses copying of subsequent source lines to the program listing file. **.LIST** restores the copying of subsequent source lines to the program listing file. The default is **.LIST**.
- Example
```
.NOLIST
mov a, 1
mov b1, a
.LIST
```
In this example, the two instructions in the block enclosed by **.NOLIST** and **.LIST** are suppressed from copying to the source listing file.

→ **Syntax**
.LISTMACRO
.NOLISTMACRO

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

→ **Syntax**

.LISTINCLUDE
.NOLISTINCLUDE

- Description

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

→ **Syntax**

MESSAGE '*text-string*'

- Description

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

→ **Syntax**

ERRMESSAGE '*error-string*'

- Description

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

Program Directives

→ **Syntax (comment)**

; *text*

- Description

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

→ **Syntax**

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

- Description

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

  The optional *align* type defines the alignment of the given section. It can be one of the following:

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

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

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

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

– COMMON
  Creates overlapping sections by placing the start of all sections with the same complete name at the same address. The length of the resulting area is the length of the longest section.
– AT *address*
  Causes all label and variable addresses defined in a section to be relative to the given address. The *address* can be any valid expression except a forward reference. It is an absolute address in a specified ROM/RAM bank and must be within the ROM/RAM range.

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

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

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

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

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

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

→ **Syntax**
END

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

→ **Syntax**

ORG *expression*

- Description

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

- Example

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

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

→ **Syntax**

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

- Description

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

- Example

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

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

→ **Syntax**

```
name   PROC
name   ENDP
```

- Description

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

- Example

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

→ **Syntax**

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

- Description

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

- Example

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

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

Data Definition Directives

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

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

→ **Syntax**

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

- Description

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

- Example

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

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

→ **Syntax**

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

- Description

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

- Example

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

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

→ **Syntax**

```
name EQU expression
```

- Description

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

- Example

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

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

Macro Directives

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

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

→ **Syntax**

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

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

→ **Syntax**

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

- Description

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

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

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

The following source program refers to the macro Delay ...

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

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

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

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

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

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

        0 Errors
```

Assembly Instructions

The syntax of an instruction has the following form:

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

where

name:	→ label name
mnemonic	→ instruction name (keywords)
operand1	→ registers memory address
operand2	→ 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
                jmp $1
        $0:                          ;; local label
                jmp Label1
        $1:     jmp $0
Label3:
```

Reserved Assembly Language Words

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

- Reserved Names (directives, operators)

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

- Reserved Names (instruction mnemonics)

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

- Reserved Names (registers names)

A	WDT	WDT1	WDT2

Cross Assembler Options

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

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

→ **Syntax**

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

- Example,

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

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

Assembly Listing File Format

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

Source Program Listing

Each line in the source program has the following syntax:

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

- `Line-number` is the number of the line starting from the first statement in the assembly source file (4 decimal digits).
- The 2nd field – `offset` – is the offset from the beginning of the current section to the code (4 hexadecimal digits)
- The 3rd field – `code` – is present only if the statement generates code or data (two hexadecimal 4-digit data)
  The `code` shows the numeric value in hexadecimal if the value is known at assembly time. Otherwise, a proper flag will indicate the action required to compute the value. The following two flags may appear behind the code field.
  R → relocatable address (Cross Linker must resolve)
  E → external symbol (Cross Linker must resolve)
  The following flag may appear before the code field
  = → **EQU** or equal-sign directive
  The following 2 flags may appear in the code field
  ---- → section address (Cross Linker must resolve)
  nn[xx] → **DUP** expression: nn **DUP**(?)
- The 4th field – `statement` – is the source statement shown exactly as it appears in the source file, or as expanded by a macro. The following flags may appear before a statement.
  n → Macro-expansion nesting level
  C → line from **INCLUDE** file

• Summary

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

l l l l → 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 extb1 : 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  b1, a
  30  0002  0080     E    mov  extb1, a
  31  0003  0FAA          mov  a, 0aah
  32  0004  0093          mov  pac, a
  33  0005                clrpa
  33  0005  0F00      1 mov a, 00h
  33  0006  0092      1 mov [12h], a
  33  0007            1 clrpb
  33  0007  1F14      2 clr [14h]
  34  0008  0700     R    mov  a, b1
  35  0009  0F00     E    mov  a, bank extlab
  36  000A  0F00     E    mov  a, offset extb1
  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. Known as the HT-IDE3000 development system, the software provides a user friendly Windows based approach for program editing and debugging while the 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-IDE3000 ensures that designers have the best tools to maximize efficiency in the design and release of their microcontroller applications.

HT-IDE3000 Development Environment

The HT-IDE3000 (Holtek Integrated Development Environment) is a high performance integrated development environment designed around Holtek's series of 8-bit microcontroller (MCU) chips. 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-IDE3000 system is the HT-ICE or In-Circuit Emulator, capable of emulating the Holtek 8-bit MCU in real time, in addition to providing powerful debugging and trace features.

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 using 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 software updates, Holtek provides regular HT-IDE3000 Service Packs. These Service Packs 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

→ **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

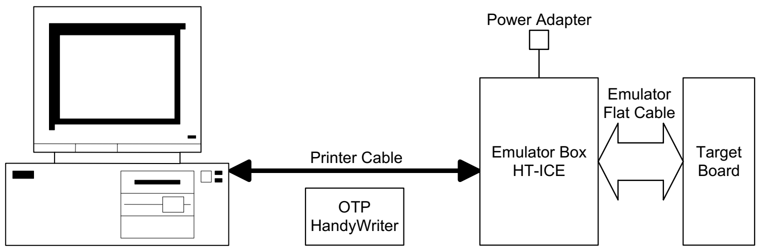
Holtek's OTP devices are fully supported by a range of programmers. For engineering level OTP device programming, Holtek supplies its HandyWriter programming tool which provides a quick and efficient means for low volume OTP programming. More programmers from other suppliers are available which provide more efficient and higher volume production capability.

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.

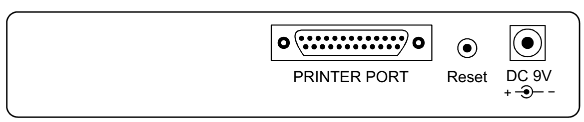
System Configuration

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 must be installed in the Supervisor Privilege mode.

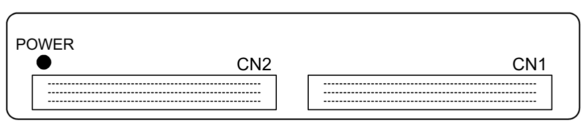


→ **The HT-IDE3000 system contains the following hardware components**

- 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, Fig below
- I/O interface card for connecting the target board to the HT-ICE box
- Power Adapter, output 9V
- 25-pin D-type printer cable
- HandyWriter



HT-ICE Rear View

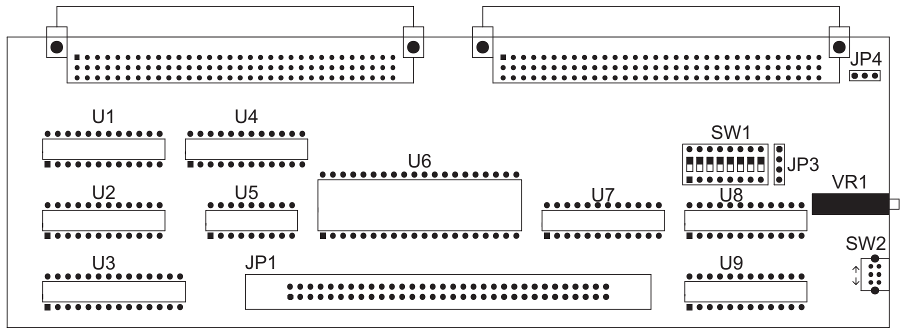


HT-ICE Front View

→ **HT-ICE Interface Card Settings**

The HT-ICE interface card 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. 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.	Package	Socket	SW1							
			1	2	3	4	5	6	7	8
HT48R10A-1	24SKDIP	U1	OFF	OFF	ON	OFF	ON	OFF	OFF	—
HT48R30A-1	24SKDIP	U2	ON	OFF	OFF	ON	OFF	OFF	ON	—
HT48R30A-1	28SKDIP	U3	ON	OFF	OFF	ON	OFF	OFF	ON	—
HT48R50A-1	28SKDIP	U3	ON	OFF	OFF	ON	OFF	OFF	ON	—

Switch SW2 should be set to the lower position.

SW2	Function
↑	HT48XA6 carrier output
↓	Other MCU

The pin assignments in locations U1, U2, U3 and U6 are defined so as to match the datasheet pin assignments for the I/O 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

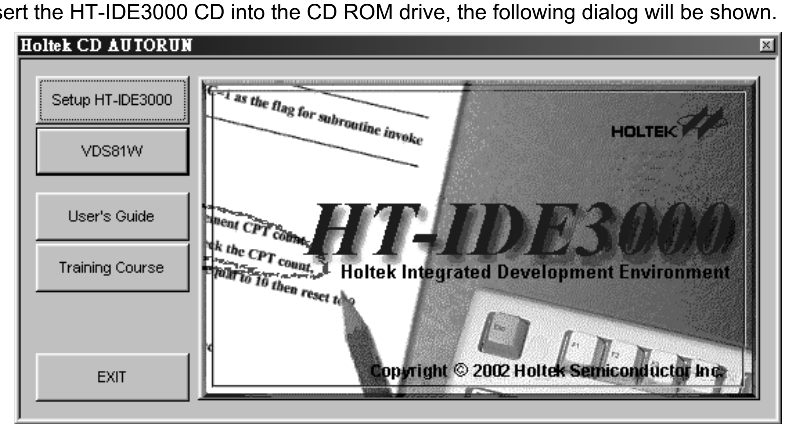
- Step 1
  Plug the power adapter into the power connector of the HT-ICE
- Step 2
  Connect the target board to the HT-ICE by using the I/O interface card or flat cable
- Step 3
  Connect the HT-ICE to the host machine using the printer cable

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

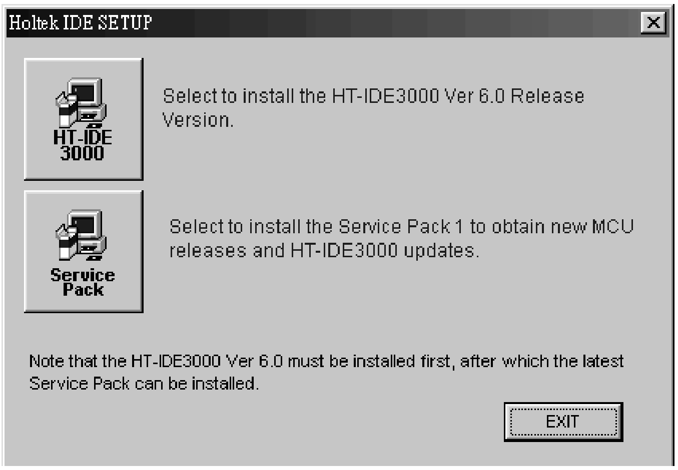
Caution Exercise care when using the power adapter. Do not use a power adapter whose output voltage is not 9V, otherwise the HT-ICE may be damaged. It is strongly recommended that only the power adapter supplied by Holtek be used. First plug the power adapter to the power connector of the HT-ICE.

Software Installation

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



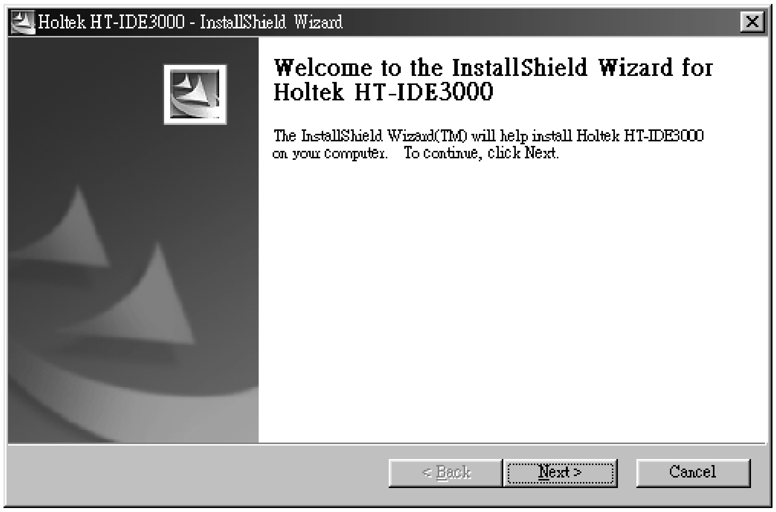
Click <HT-IDE3000> button and the following dialog will be shown.



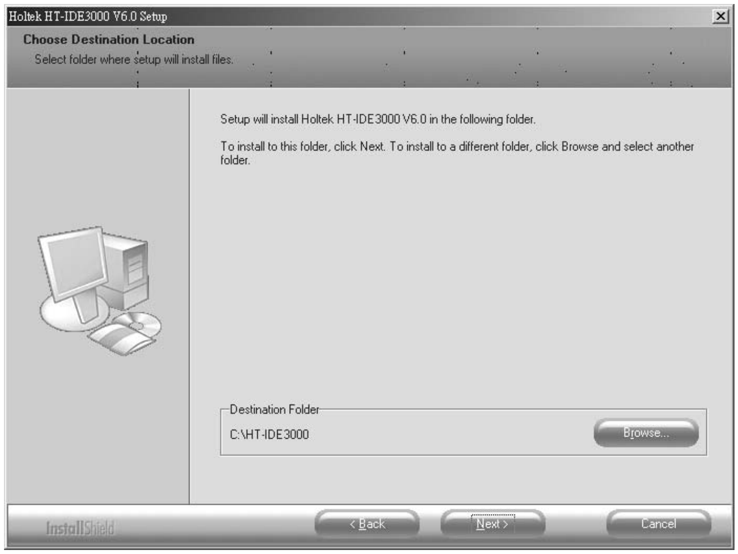
Click <HT-IDE3000> or <Service Pack> as you want.

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

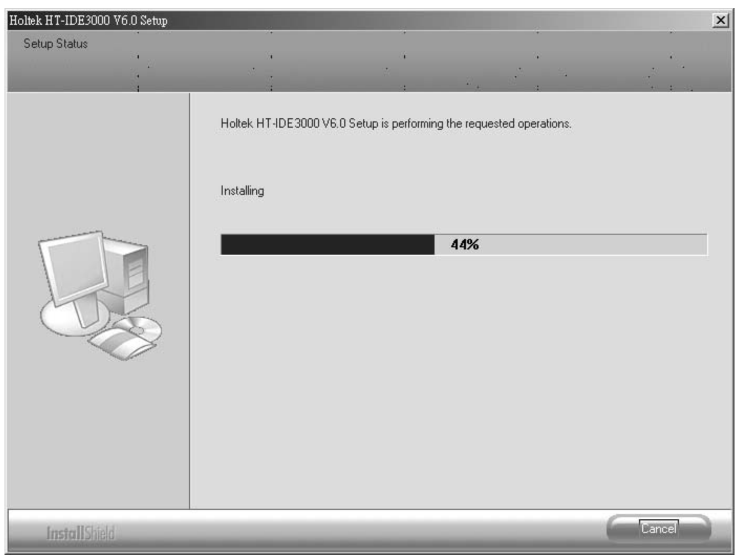
- Step 2
  Press the <Next> button to continue setup or press <Cancel> button to abort.



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

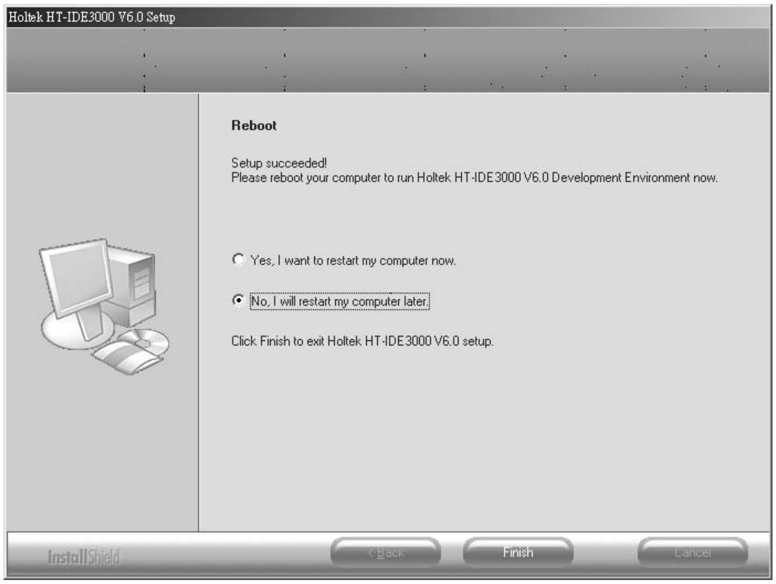


- Step 4
  Specify the path you want to install the HT-IDE3000 and click <Next> button.
- Step 5
  SETUP will copy all files to the specified directory.



- Step 6

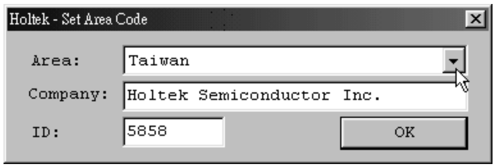
  If the process is successful a dialog will be shown.



- 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.



OTP HandyWriter

The Holtek HandyWriter was specifically developed to program the range of Holtek OTP (One Time Programmable) MCU devices allowing users to easily and efficiently burn their programming code into the OTP devices. The advantages of this writer include its small and easy to manage size, ease of installation and easy to use special features. The structure of the writer includes the following components:

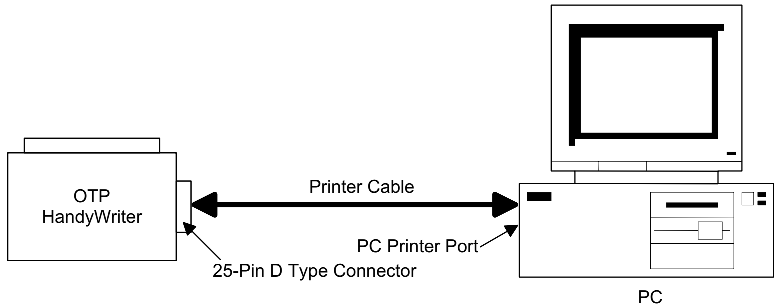
- Single 40-pin DIP TEXTOOL
- Single 25-pin printer port D-type female connector
- Single 96-pin VME connector

To use the HandyWriter requires the following:

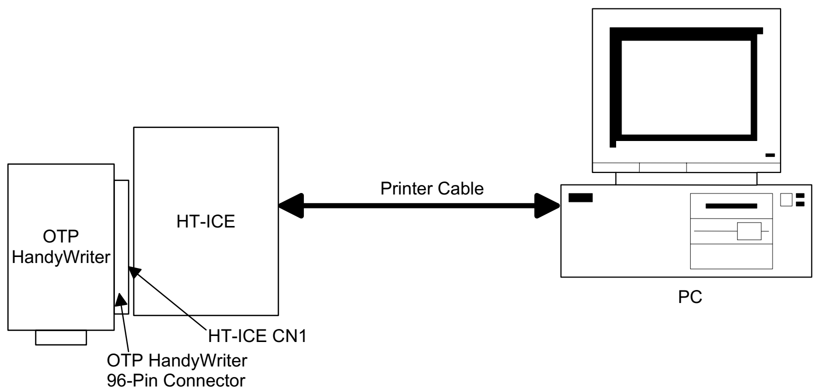
- 16V power adapter with minimum current rating of 500mA. For best purposes, please use the adapter included with the HandyWriter carton.
- IBM386 compatible or higher spec. PC
- Win95/98/NT/2000/XP Windows operating system
- HT-IDE3000 Integrated Development Environment
- If the writer is directly connected to the PC, the HT-ICE is not required.

Installation of the HandyWriter:

- To directly connect to a PC, use the printer cable to connect from the HandyWriter′s 25-pin D-type connector to the printer port of the PC as shown in the figure below.



- To connect via the HT-ICE, first connect the HandyWriter to the VME 96-pin socket CN1 on the HT-ICE then connect the HT-ICE to the PC′s printer port using the printer cable as shown in the figure below.



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

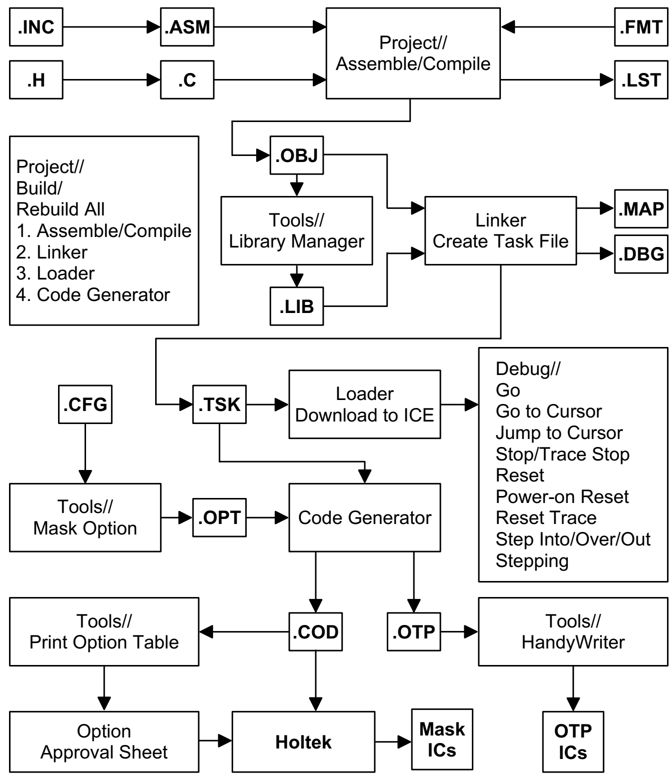
Step 4 – Transmit Code to Holtek

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

Step 5 – Programming the OTP Device

- Build the project for creating the .OTP file
- Click on Tools menu and select HandyWriter command or use HT-HandyWriter in program group to program the OTP devices

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



Appendix

Appendix A

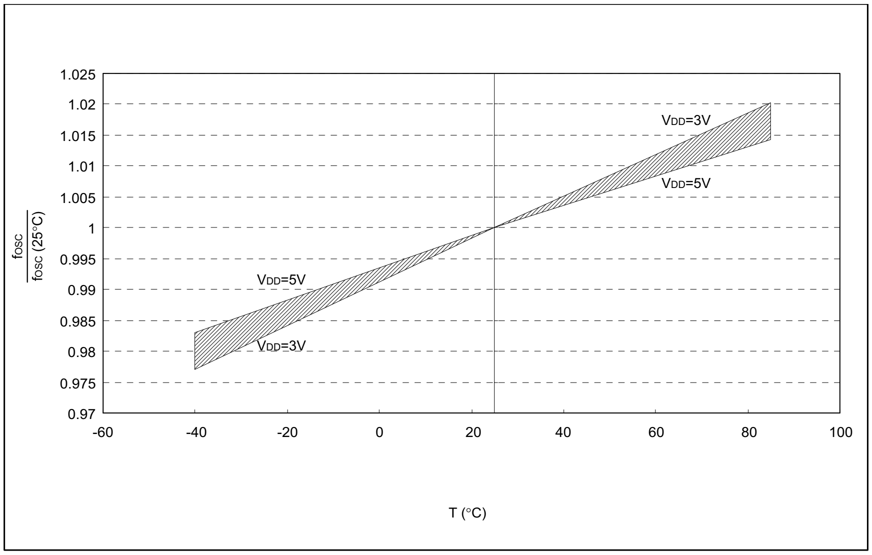
Device Characteristic Graphics

A

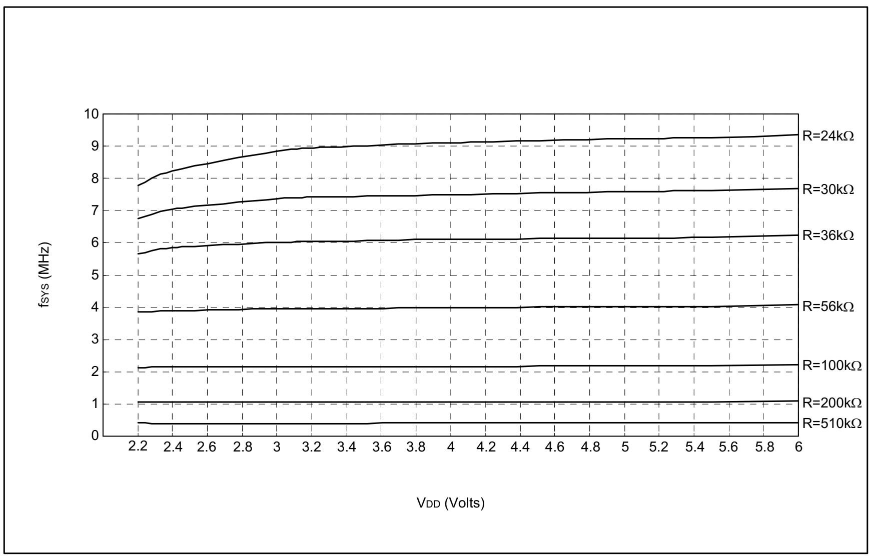
The following characteristic graphics depicts typical device behavior. The data presented here is a statistical summary of data gathered on units from different lots over a period of time. This is for information only and the figures were not tested during manufacturing.

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.

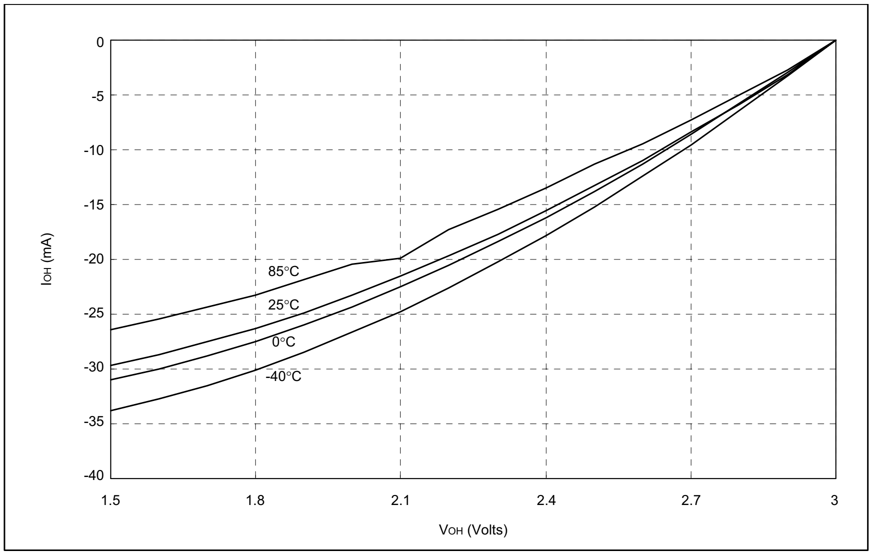
Typical RC OSC vs. Temperature



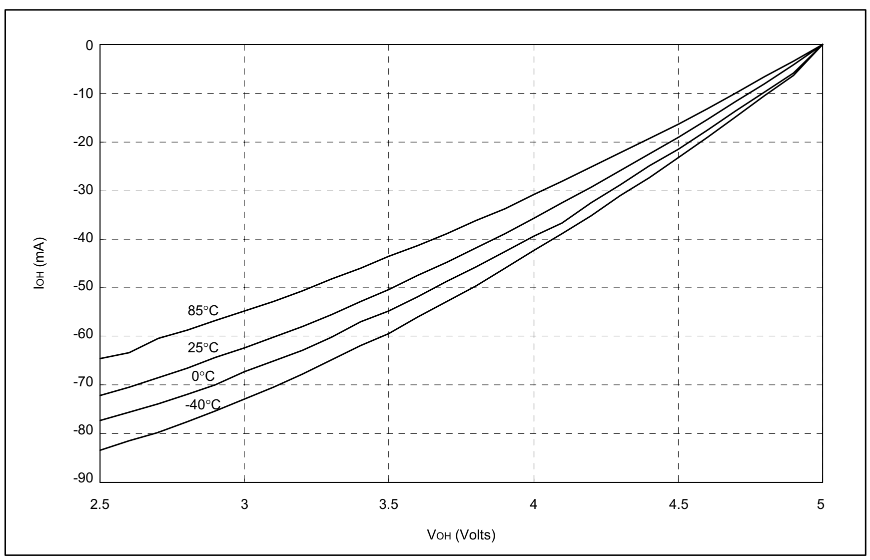
Typical RC Oscillator Frequency vs. V_{DD}



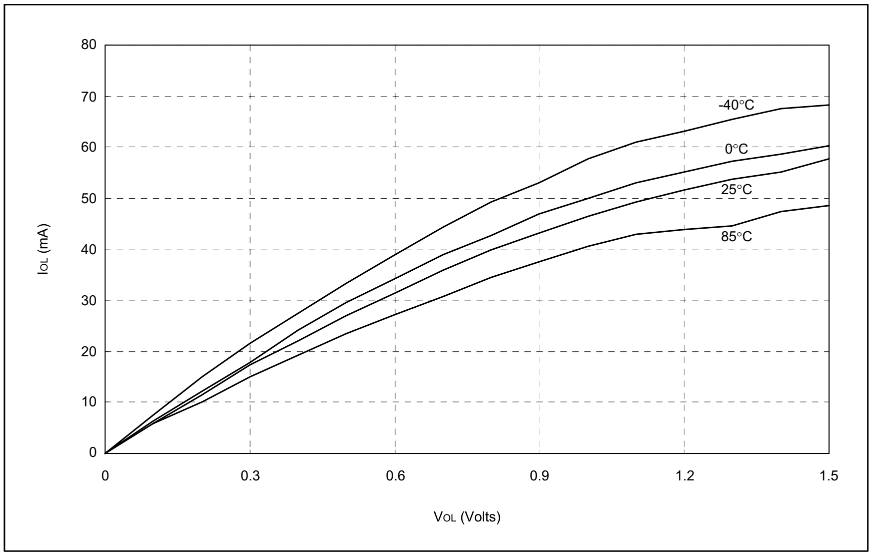
I_{OH} vs. V_{OH}, V_{DD}=3V



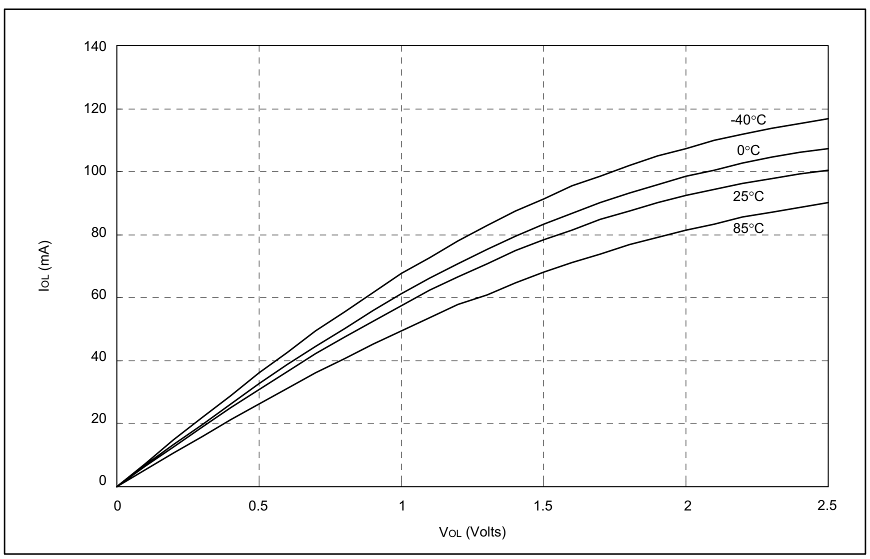
I_{OH} vs. V_{OH}, V_{DD}=5V



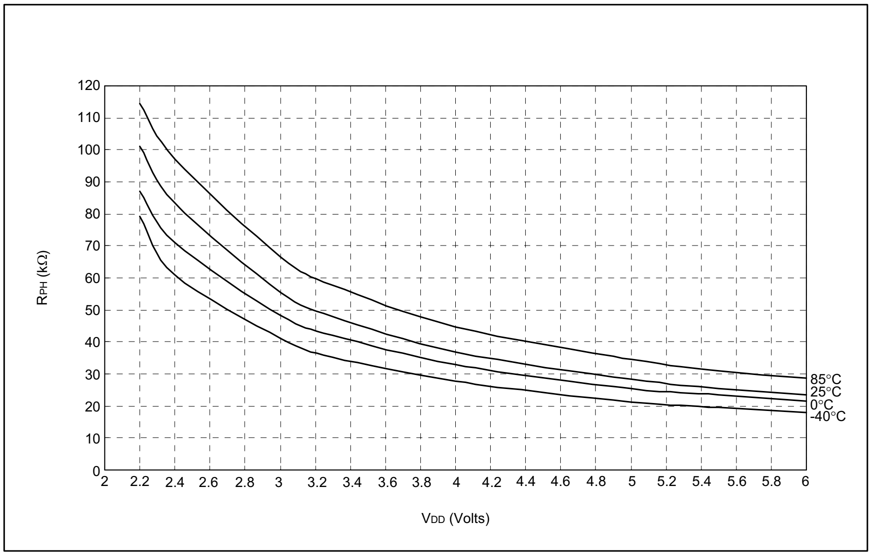
I_{OL} vs. V_{OL}, V_{DD}=3V



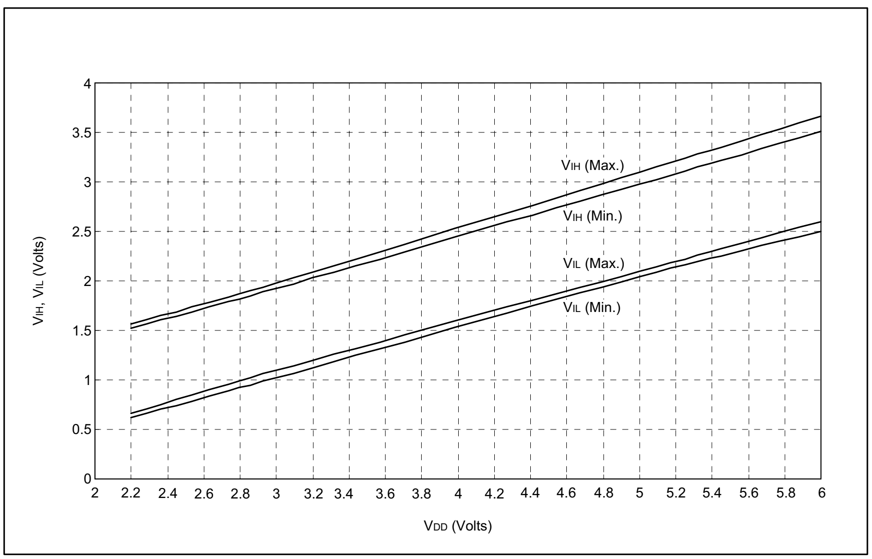
I_{OL} vs. V_{OL}, V_{DD}=5V



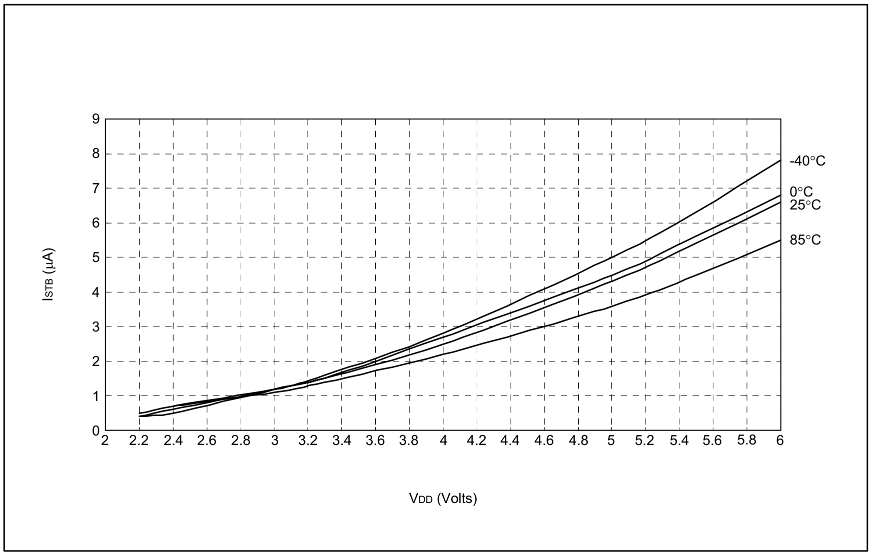
Typical R_{PH} vs. V_{DD}



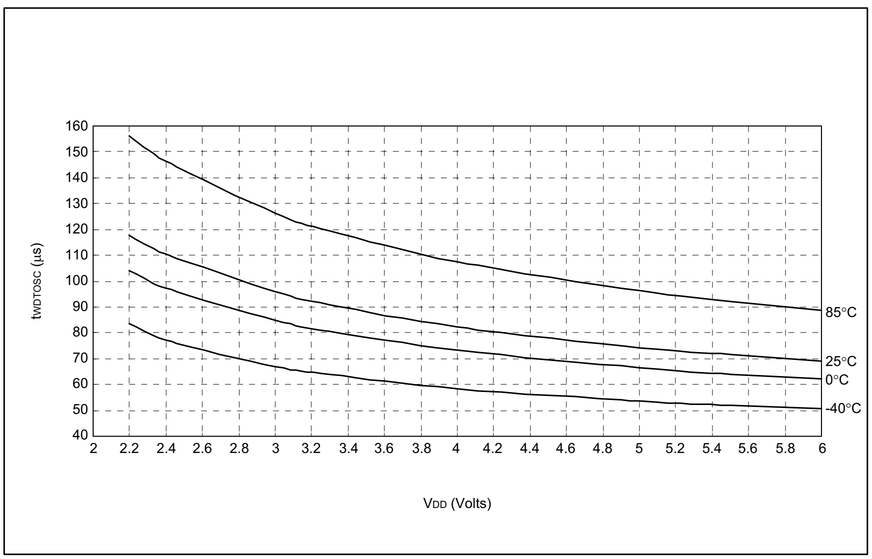
Typical V_{IH}, V_{IL} vs. V_{DD} in -40°C to +85°C



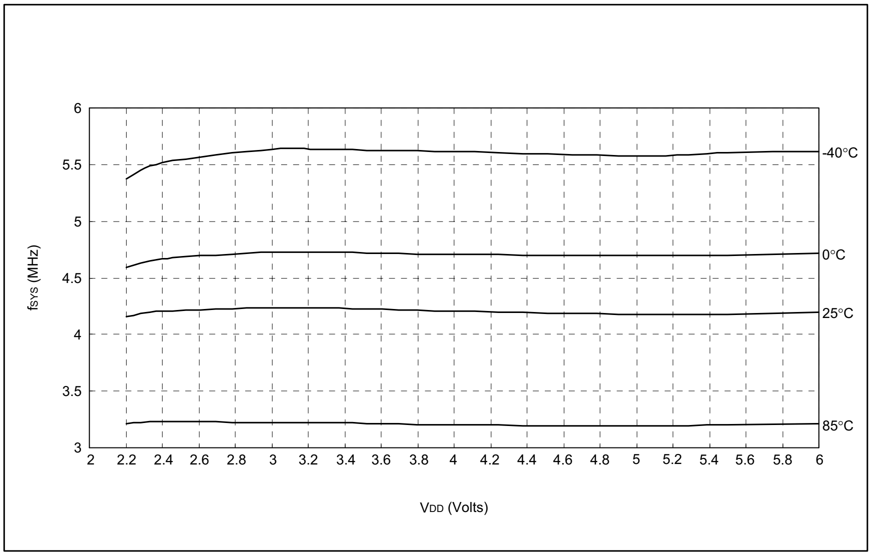
Typical I_{STB} vs. V_{DD} Watchdog Enable



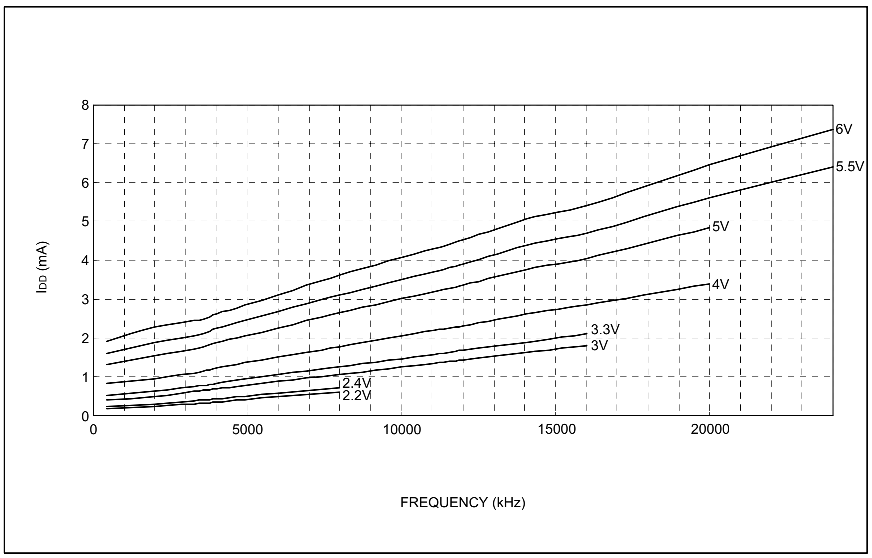
Typical t_{WDTOSC} vs. V_{DD}



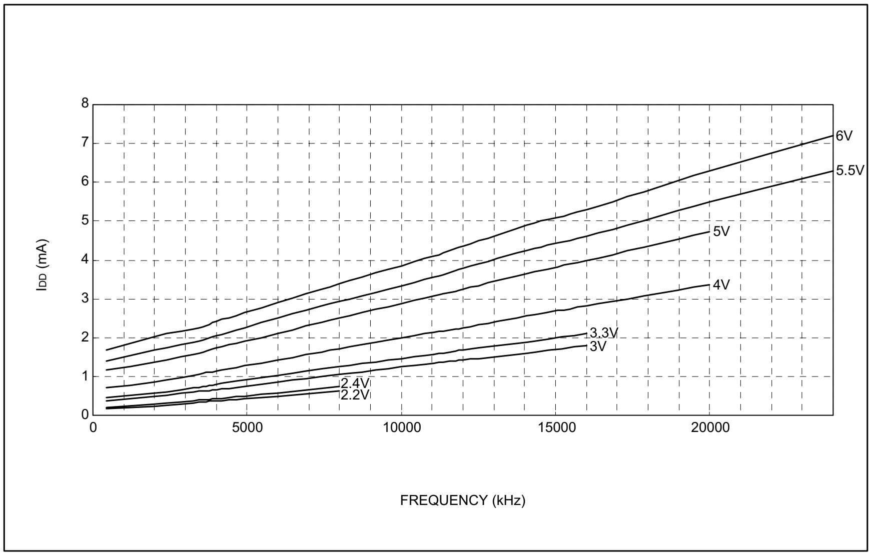
Typical Internal RC OSC vs. V_{DD}



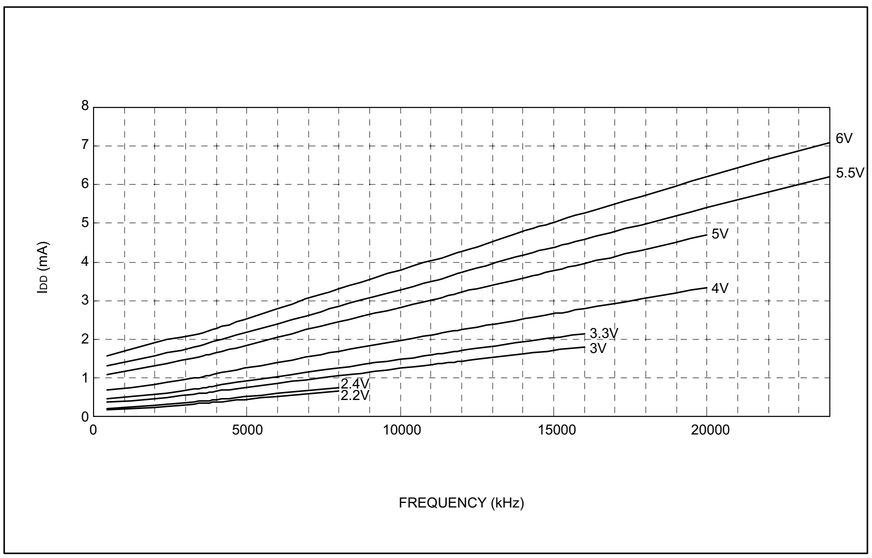
Typical I_{DD} vs. Frequency (External Clock, Ta=-40°C)



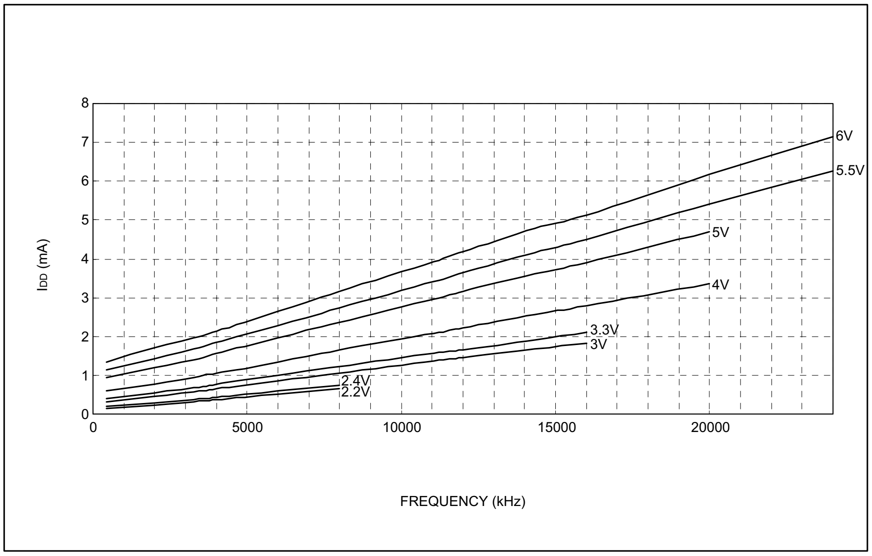
Typical I_{DD} vs. Frequency (External Clock, Ta=0°C)



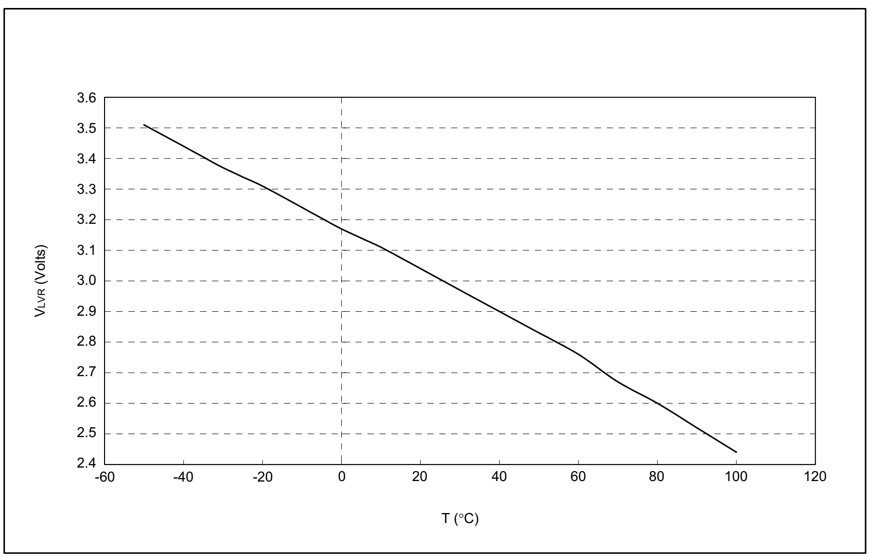
Typical I_{DD} vs. Frequency (External Clock, Ta=+25°C)



Typical I_{DD} vs. Frequency (External Clock, Ta=+85°C)



Typical V_{LVR} vs. Temperature

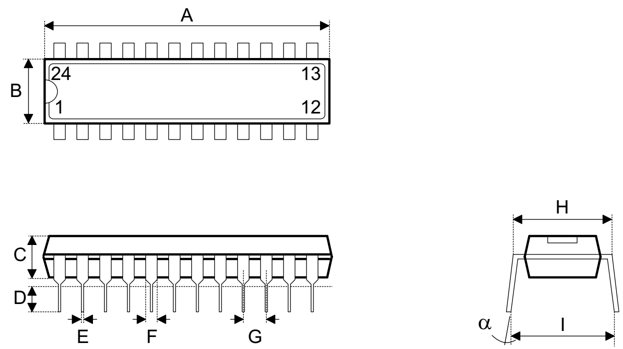


A p p e n d i x   B

Package Information

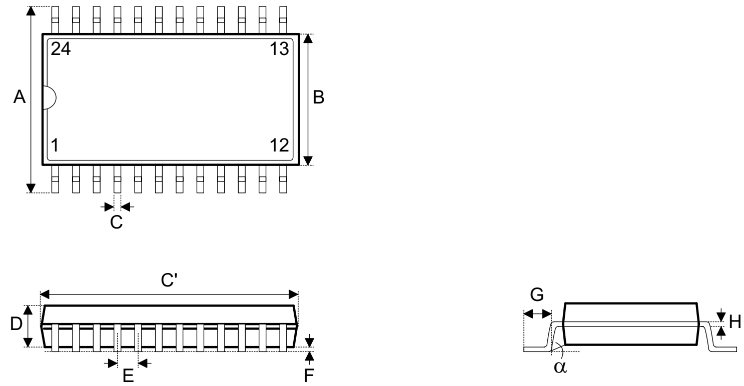
B

24-pin SKDIP (300mil) Outline Dimensions



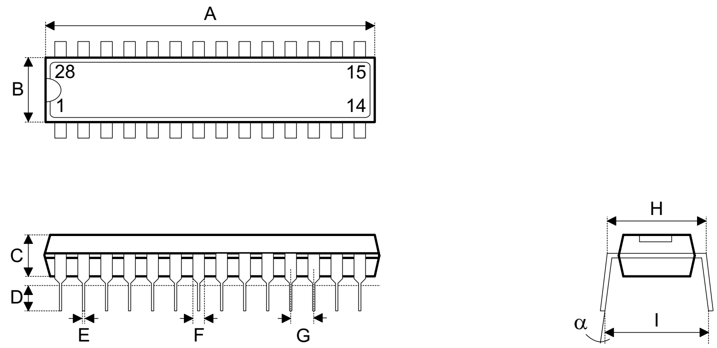
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	1235	—	1265
B	255	—	265
C	125	—	135
D	125	—	145
E	16	—	20
F	50	—	70
G	—	100	—
H	295	—	315
I	345	—	360
α	0°	—	15°

24-pin SOP (300mil) Outline Dimensions



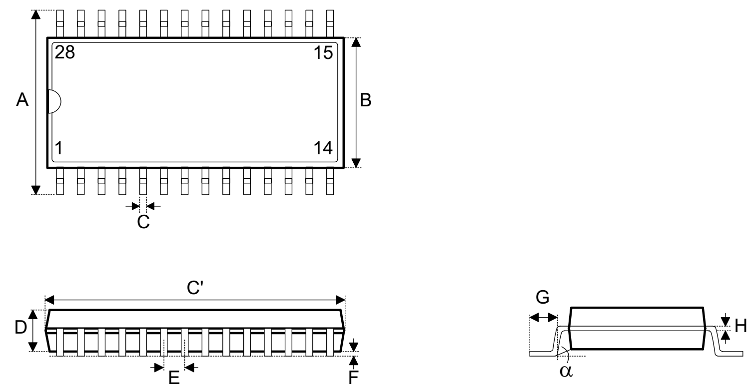
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	394	—	419
B	290	—	300
C	14	—	20
C′	590	—	614
D	92	—	104
E	—	50	—
F	4	—	—
G	32	—	38
H	4	—	12
α	0°	—	10°

28-pin SKDIP (300mil) Outline Dimensions



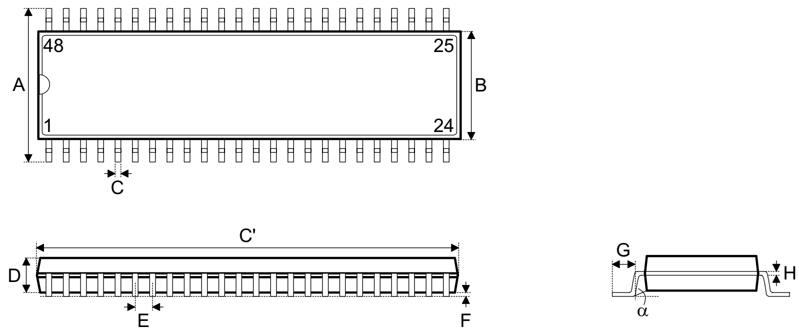
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	1375	—	1395
B	278	—	298
C	125	—	135
D	125	—	145
E	16	—	20
F	50	—	70
G	—	100	—
H	295	—	315
I	330	—	375
α	0°	—	15°

28-pin SOP (300mil) Outline Dimensions



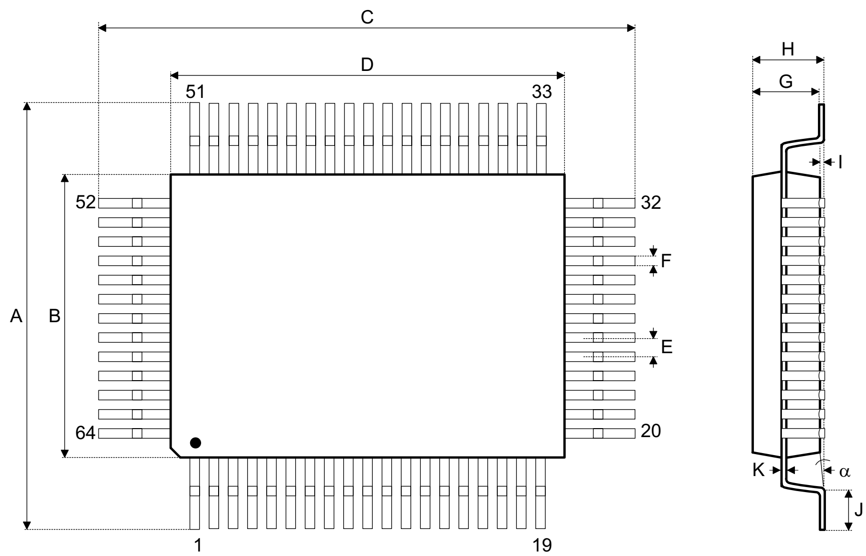
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	394	—	419
B	290	—	300
C	14	—	20
C′	697	—	713
D	92	—	104
E	—	50	—
F	4	—	—
G	32	—	38
H	4	—	12
α	0°	—	10°

48-pin SSOP (300mil) Outline Dimensions



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

64-pin QFP (14x20) Outline Dimensions



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

Holtek Semiconductor Inc. (Headquarters)
No.3, Creation Rd. II, Science-based Industrial Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
http://www.holtek.com.tw

Holtek Semiconductor Inc. (Sales Office)
11F, No.576, Sec.7 Chung Hsiao E. Rd., Taipei, Taiwan
Tel: 886-2-2782-9635
Fax: 886-2-2782-9636
Fax: 886-2-2782-7128 (International sales hotline)

Holtek Semiconductor (Shanghai) Inc.
7th Floor, Building 2, No.889, Yi Shan Rd., Shanghai, China
Tel: 021-6485-5560
Fax: 021-6485-0313
http://www.holtek.com.cn

Holtek Semiconductor (Hong Kong) Ltd.
Block A, 3/F, Tin On Industrial Building, 777-779 Cheung Sha Wan Rd., Kowloon, Hong Kong
Tel: 852-2-745-8288
Fax: 852-2-742-8657

Holmate Semiconductor, Inc.
46712 Fremont Blvd., Fremont, CA94538
Tel: 510-252-9880
Fax: 510-252-9885
http://www.holmate.com