

Technical Document

- [Tools Information](#)
- [FAQs](#)
- [Application Note](#)
 - [HA0075E MCU Reset and Oscillator Circuits Application Note](#)

Features

- Operating voltage:
f_{SYS} = 4MHz: 2.2V~5.5V, Crystal mode
f_{SYS} = 12MHz: 2.7V~5.5V, RC clock mode
- 35 bidirectional I/O lines
- Dual external interrupt inputs shared with I/O lines
- Single 8-bit programmable Timer/Event Counters with overflow interrupt and 7-stage prescaler
- Watchdog Timer function
- PFD for audio generation
- Power down and wake-up functions to reduce power consumption
- Crystal and RC oscillator
- 16 channel 8-bit resolution A/D converter
- Single channel (6+2)-bit PWM output shared with an I/O line
- Up to 0.33μs instruction cycle with 12MHz system clock at V_{DD}= 5V
- 6-level subroutine nesting
- Bit manipulation instruction
- Table read instructions
- 63 powerful instructions
- All instructions executed in one or two machine cycles
- Low voltage reset function
- Dual Integrated SPI interfaces
- Pins PB2, PB3, PD4, PD7 can be setup as either CMOS or NMOS outputs using configuration options
- 28-pin SKDIP/SOP/SSOP and 44-pin QFP packages

General Description

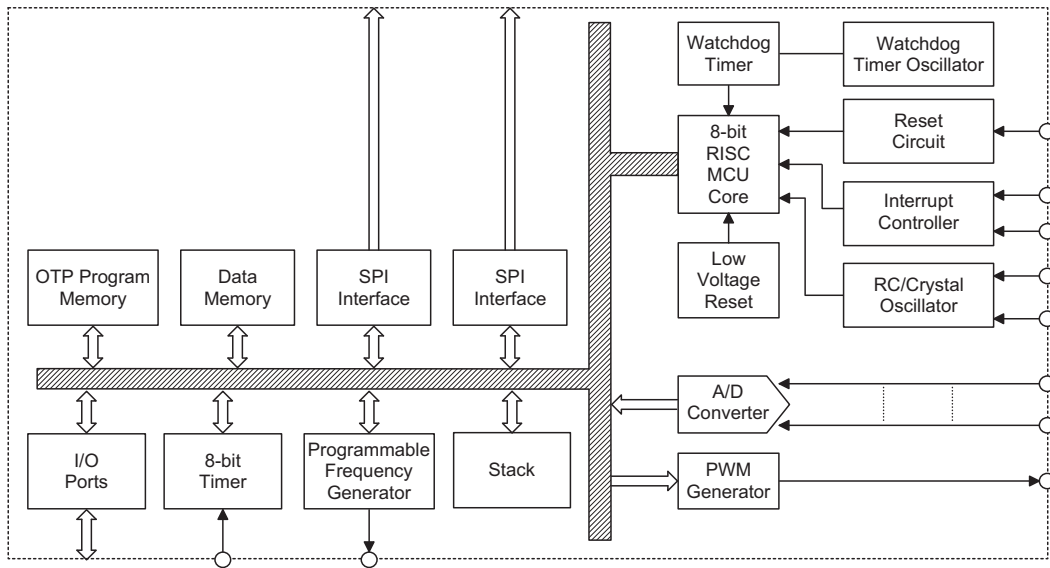
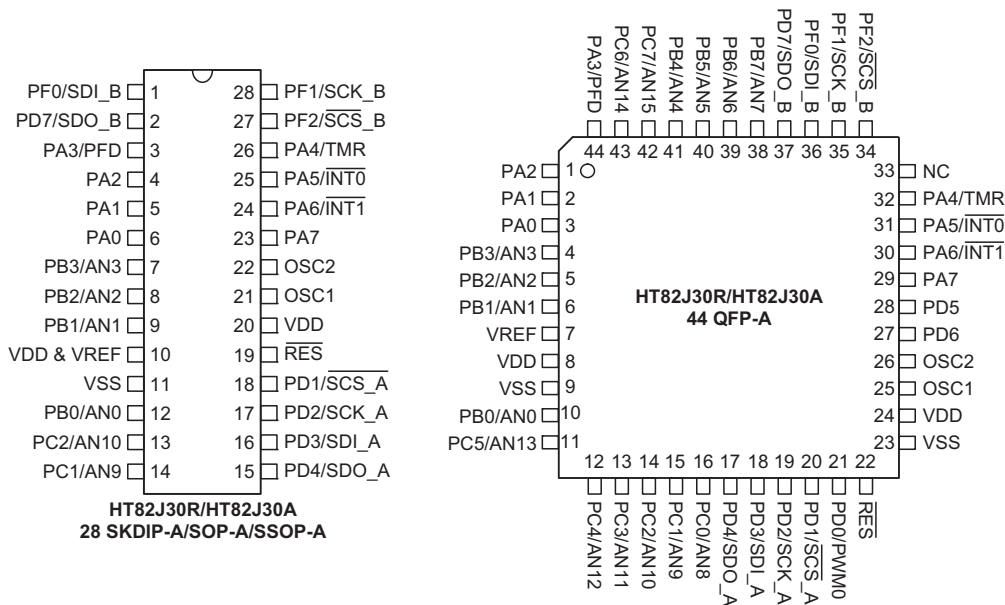
The HT82J30R and HT82J30A are 8-bit high performance, RISC architecture microcontroller devices specifically designed for A/D applications that interface directly to analog signals, such as those from sensors.

The HT82J30A mask version type is fully pin and functionally compatible with the HT82J30R OTP version device.

The advantages of low power consumption, I/O flexibility, programmable frequency divider, timer functions, oscillator options, multi-channel A/D Converter, Pulse

Width Modulation function, Watchdog timer, SPI interfaces, Power Down and wake-up functions, enhance the versatility of these devices to suit a wide range of A/D application possibilities such as sensor signal processing, motor driving, industrial control, consumer products, subsystem controllers, etc.

With the provision of dual SPI interfaces the devices are especially suitable for Joystick Encoder applications.

Block Diagram

Pin Assignment


Pin Description

Pin Name	I/O	Options	Description
PA0~PA2 PA3/PFD PA4/TMR PA5/INT0 PA6/INT1 PA7	I/O	Pull-high Wake-up PA3 or PFD	Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. The PFD, TMR and external interrupt inputs are pin-shared with PA3, PA4, and PA5, PA6 respectively.
PB0/AN0~ PB7/AN7	I/O	Pull-high PB2, PB3 either CMOS or NMOS output type	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pin on this port have pull-high resistors. PB is pin-shared with the A/D input pins. The A/D inputs are selected via software instructions. Once selected as an A/D input, the I/O function and pull-high resistor functions are disabled automatically. PB2 and PB3 can be either CMOS or NMOS output types selected via configuration option.
PC0/AN8~ PC7/AN15	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 which pin on this port have pull-high resistors. PC is pin-shared with the A/D input pins. The A/D inputs are selected via software instructions. Once selected as an A/D input, the I/O function and pull-high resistor functions are disabled automatically.
PD0/PWM0 PD1/ $\overline{\text{SCS}}_A$ PD2/SCK_A PD3/SDI_A PD4/SDO_A PD5~PD6 PD7/SDO_B	I/O	Pull-high PD0 or PWM PD4, PD7 either CMOS or NMOS output type	Bi-directional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pin on this port have pull-high resistors. PD0 is pin-shared with the PWM output selected via configuration option. PD1~PD4 are pin-shared with SPI interface A. PD7 is pin-shared with the SPI interface B. PD4 and PD7 can be either CMOS or NMOS output types selected via configuration option.
PF0/SDI_B PF1/SCK_B PF2/ $\overline{\text{SCS}}_B$	I/O	Pull-high*	Bidirectional 3-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine if the pins have pull-high resistors. PF0~PF2 are pin-shared with SPI interface B.
OSC1 OSC2	I O	Crystal or RC	OSC1, OSC2 are connected to an external RC network or external crystal, determined by configuration option, for the internal system clock. If the RC system clock is selected, pin OSC2 can be used to measure the system clock at 1/4 frequency.
$\overline{\text{RES}}$	I	—	Schmitt trigger reset input. Active low
VSS	—	—	Negative power supply, ground
VDD	—	—	Positive power supply
VREF	—	—	8-bit A/D reference voltage input pin

Note: 1. Each pin on PA can be chosen via configuration option to have a wake-up function.
2. Pins PB4~PB7, PC0, PC3~PC7, PD0, PD5 and PD6 only exist on the 44-pin package.

Absolute Maximum Ratings

Supply Voltage	$V_{SS}-0.3V$ to $V_{SS}+6.0V$	Storage Temperature	$-50^{\circ}C$ to $125^{\circ}C$
Input Voltage	$V_{SS}-0.3V$ to $V_{DD}+0.3V$	Operating Temperature	$-40^{\circ}C$ to $85^{\circ}C$
I_{OL} Total	150mA	I_{OH} Total	$-100mA$
Total Power Dissipation	500mW		

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

D.C. Characteristics
 $T_a=25^{\circ}C$

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage (Crystal OSC)	—	$f_{SYS}=4MHz$	2.2	—	5.5	V
	Operating Voltage (RC OSC)	—	$f_{SYS}=12MHz$ at $56k\Omega$	2.7	—	5.5	V
I_{DD}	Operating Current (Crystal OSC, RC OSC)	3V	No load, $f_{SYS}=12MHz$ ADC disable	—	1	2	mA
		5V		—	4	6	mA
I_{STB1}	Standby Current (WDT Enabled)	3V	No load, system HALT	—	—	5	μA
		5V		—	—	10	μA
I_{STB2}	Standby Current (WDT Disabled)	3V	No load, system HALT	—	—	1	μA
		5V		—	—	2	μA
V_{IL1}	Input Low Voltage for I/O, TMR and \overline{INT}	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O, TMR and \overline{INT}	—	—	$0.7V_{DD}$	—	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	—	—	2.8	3.0	3.2	V
I_{OL}	I/O Port Sink Current	3V	$V_{OL}=0.1V_{DD}$	4	8	—	mA
		5V		10	20	—	mA
I_{OH}	I/O Port Source Current	3V	$V_{OH}=0.9V_{DD}$	-2	-4	—	mA
		5V		-5	-10	—	mA
R_{PH}	Pull-high Resistance	3V	—	20	60	100	$k\Omega$
		5V		10	30	50	$k\Omega$
V_{AD}	A/D Input Voltage	—	—	0	—	V_{DD}	V
E_{AD}	A/D Conversion Error	3V	—	—	± 0.5	± 1	LSB
		5V		—	± 0.5	± 1	LSB
I_{ADC}	Only ADC Enable, Others Disable	3V	No load	—	0.5	1	mA
		5V		—	1.5	3	mA

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~2.7V	400	—	4000	kHz
		—	2.8V~5.5V	400	—	12000	kHz
f _{SYS2}	System Clock (RC OSC)	—	2.7V~5.5V	1000	—	12000	kHz
f _{TIMER}	Timer I/P Frequency (TMR)	—	2.2V~2.7V	0	—	4000	kHz
		—	2.8V~5.5V	0	—	12000	kHz
t _{WDTOSC}	Watchdog Oscillator Period	3V	—	45	90	180	μs
		5V	—	32	65	130	μs
t _{WDT1}	Watchdog Time-out Period (WDT OSC)	—	—	2 ¹⁵	—	2 ¹⁶	t _{WDTOSC}
t _{WDT2}	Watchdog Time-out Period (System Clock)	—	—	2 ¹⁷	—	2 ¹⁸	t _{SYS}
t _{RES}	External Reset Low Pulse Width	—	—	1	—	—	μs
t _{SST}	System Start-up Timer Period	—	Wake-up from HALT	—	1024	—	t _{SYS}
t _{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
t _{AD}	A/D Clock Period	—	—	1	—	—	μs
t _{ADC}	A/D Conversion Time	—	—	—	76	—	t _{AD}
t _{ADCS}	A/D Sample Time	—	—	—	32	—	t _{AD}
t _{CS_SK}	SPI SCS to SCK Time	—	—	50	—	—	ns
t _{SPICK}	SPI Clock Time	—	—	400	—	—	ns

 Note: t_{SYS}=1/f_{SYS1} or 1/f_{SYS2}

System Architecture

A key factor in the high-performance features of the Holtek range of 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 and A/D control system with maximum reliability and flexibility.

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

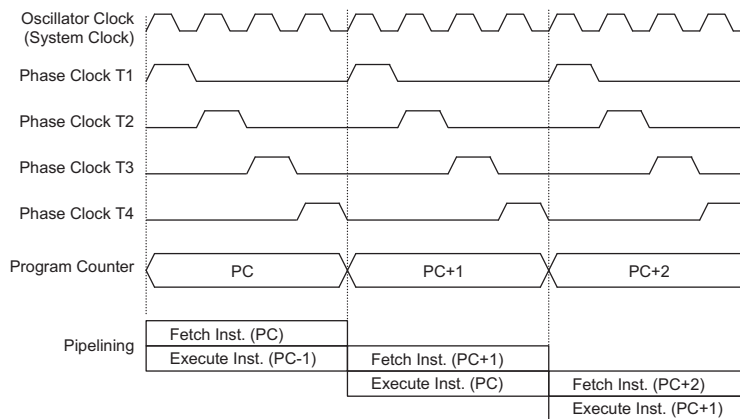
forms one instruction cycle. Although the fetching and execution of instructions takes place in consecutive instruction cycles, the pipelining structure of the microcontroller ensures that instructions are effectively executed in one instruction cycle. The exception to this are instructions where the 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.

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

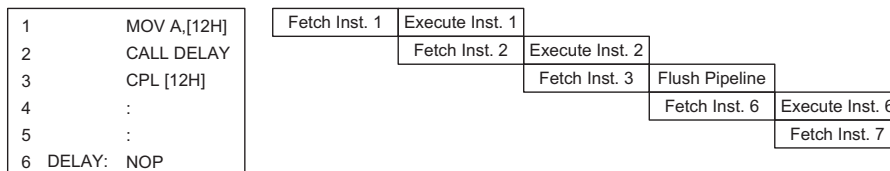
Program Counter

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as "JMP" or "CALL" that demand a jump to a non-consecutive Program Memory address. 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



System Clocking and Pipelining



Instruction Fetching

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 writeable 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.

The lower byte of the Program Counter is fully accessible under program control. Manipulating 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.

Stack

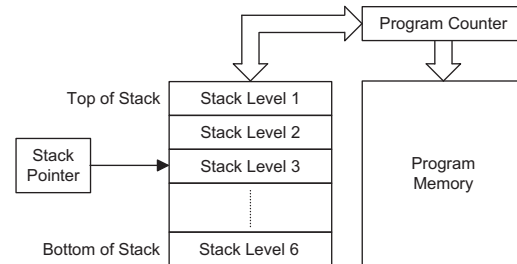
This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has 6 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, SP, and is neither readable nor writeable. 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 device reset, the Stack Pointer will point to the top of the stack.

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

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



Mode	Program Counter Bits											
	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
INT0 External Interrupt	0	0	0	0	0	0	0	0	0	1	0	0
INT1 External Interrupt	0	0	0	0	0	0	0	1	1	0	0	0
Timer/Event Counter Overflow	0	0	0	0	0	0	0	0	1	0	0	0
A/D Converter Interrupt	0	0	0	0	0	0	0	0	1	1	0	0
SPI_A Interrupt	0	0	0	0	0	0	0	1	0	0	0	0
SPI_B Interrupt	0	0	0	0	0	0	0	1	0	1	0	0
Skip	Program Counter + 2											
Loading PCL	PC11	PC10	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#11	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Program Counter

Note: PC11~PC8: Current Program Counter bits
 @7~@0: PCL bits
 #11~#0: Instruction code address bits
 S11~S0: Stack register bits

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. The HT82J30R is a One-Time Programmable, OTP, memory type device where users can program their application code into the device. 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 HT82J30A is a Mask memory type device and offers the most cost effective solution for high volume products.

Structure

The Program Memory has a capacity of 4K by 15 bits. 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.

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 device reset for program initialisation. After a device 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 $\overline{\text{INT0}}$ external input pin on the device receives a high to low transition, the program will jump to this location and begin execution, if the interrupt is enabled and the stack is not full.
- Location 008H
This vector is used by the timer/event counter. If a counter overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.

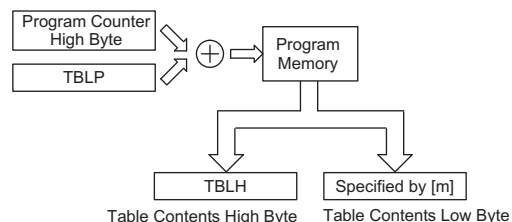
- Location 00CH
This vector is used by the A/D converter interrupt service program. If the interrupt is activated, when the A/D conversion is completed, if the interrupt is enabled and the stack is not full, the program begins execution at this location.
- Location 010H
This vector is used by serial interface A. When 8-bits of data have been received or transmitted successfully from serial interface A, the program will jump to this location and begin execution if the interrupt is enabled and the stack is not full.
- Location 014H
This vector is used by serial interface B. When 8-bits of data have been received or transmitted successfully from serial interface B, the program will jump to this location and begin execution if the interrupt is enabled and the stack is not full.
- Location 018H
This vector is used by the external interrupt. If the $\overline{\text{INT1}}$ external input pin on the device receives a high to low transition, the program will jump to this location and begin execution, if the interrupt is enabled and the stack is not full.

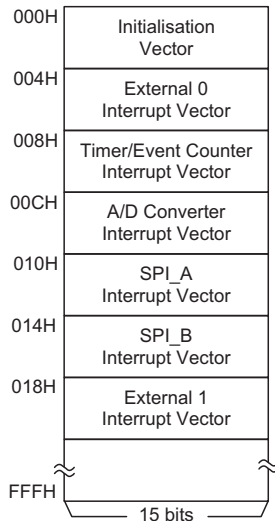
Look-up Table

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:




Program Memory Structure
Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the 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 "F00H" which refers to the start address of the last page within the 4K Program Memory of device. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "F06H" 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      ; initialise table pointer - note that this address
                ; is referenced
mov  tblp,a     ; to the last page or present page
:
:
tabrdl tempreg1 ; transfers value in table referenced by table pointer
                ; to tempreg1
                ; data at prog. memory address "F06H" 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 "F05H" transferred to
                ; tempreg2 and TBLH
                ; in this example the data "1AH" is transferred to
                ; tempreg1 and data "0FH" to register tempreg2
                ; the value "00H" will be transferred to the high byte
                ; register TBLH
:
:
org  F00h       ; sets initial address of last page
dc   00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
:
:

```

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

Table Location

Note: PC11~PC8: Current Program Counter bits
 @7~@0: Table Pointer TBLP bits

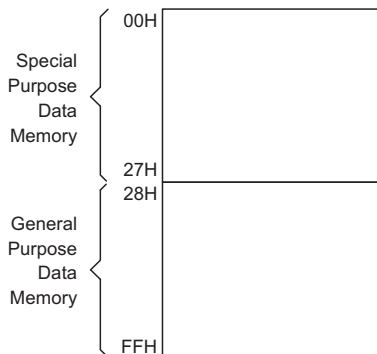
Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use the table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of 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.

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.

Structure

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". Registers which are common to all



Data Memory Structure

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.

microcontrollers, such as ACC, PCL, etc., have the same Data Memory address.

General Purpose Data Memory

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

Special Purpose Data Memory

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

00H	IAR
01H	MP
02H	
03H	
04H	
05H	ACC
06H	PCL
07H	TBLP
08H	TBLH
09H	
0AH	STATUS
0BH	INTC
0CH	
0DH	TMR
0EH	TMRC
0FH	
10H	
11H	
12H	PA
13H	PAC
14H	PB
15H	PBC
16H	PC
17H	PCC
18H	PD
19H	PDC
1AH	PWM
1BH	
1CH	PF
1DH	PFC
1EH	INTC1
1FH	
20H	
21H	ADR
22H	ADCR
23H	ACSR
24H	SBCR_A
25H	SBDR_A
26H	SBCR_B
27H	SBDR_B

 : 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, 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 and attempting to read data from these locations will return a value of 00H.

Indirect Addressing Register – IAR

The IAR register, located at Data Memory address "00H", is not physically implemented. This special register allows what is known as indirect addressing, which permits data manipulation using Memory Pointers instead of the usual direct memory addressing method

where the actual memory address is defined. Any actions on the IAR register will result in corresponding read/write operations to the memory location specified by the Memory Pointer MP. Reading the IAR register indirectly will return a result of "00H" and writing to the register indirectly will result in no operation.

Memory Pointer – MP

One Memory Pointer, known as MP, is physically implemented in the Data Memory. The Memory Pointer can be written to and manipulated in the same way as normal registers providing an easy way of addressing and tracking data. When using any operation on the indirect addressing register IAR, it is actually the address specified by the Memory Pointer that the microcontroller will be directed to.

```

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

start:
mov a,04h ; setup size of block
mov block,a
mov a,offset adres1; Accumulator loaded with first RAM address
mov mp,a ; setup memory pointer with first RAM address

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

continue:

```

The important point to note here is that in the example shown above, no reference is made to specific Data Memory 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 data is located. Its value must be setup before any table read commands are executed. Its value can be changed, for example using the "INC" or "DEC" instructions, allowing for easy table data pointing and reading. 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.

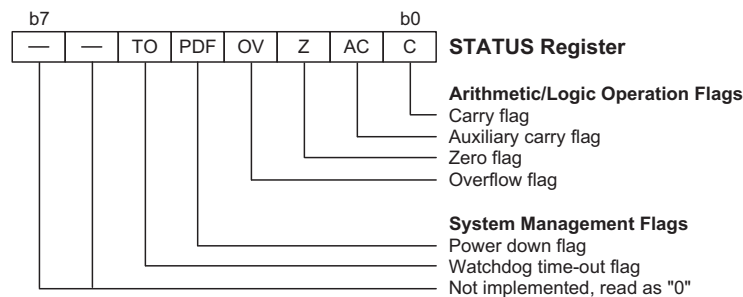
Status Register – STATUS

This 8-bit register 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). These arithmetic/logical operation and system management flags are used to record the status and operation of the microcontroller.

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.



Status Register

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 interrupt routine can change the status register, precautions must be taken to correctly save it.

Interrupt Control Registers – INTC, INTC1

The microcontrollers provide two external interrupts, an internal timer/event counter overflow interrupt, an A/D converter end-of-converter interrupt and two SPI interrupt. By setting various bits within this register using standard bit manipulation instructions, the enable/disable function of each interrupt 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.

Timer/Event Counter Registers – TMR, TMRC

Both devices possess a single internal 8-bit count-up timer. An associated register known as TMR is the location where the timers 8-bit value is located. This register can also be preloaded with fixed data to allow different time intervals to be setup. An associated control register, known as TMRC, contains the setup information for this timer, which determines in what mode the timer is to be used as well as containing the timer on/off control function.

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, PD and PF. 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, PDC and PFC, 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 initialisation, 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 ports 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 package is chosen, the microcontroller provides up to 35 bidirectional input/output lines labeled with port names PA, PB, PC, PD and PF.

This register is mapped to the Data Memory with an addresses as shown in the Special Purpose Data Memory table. Seven of these I/O lines can be used for input and output operations and one line as an input only. 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

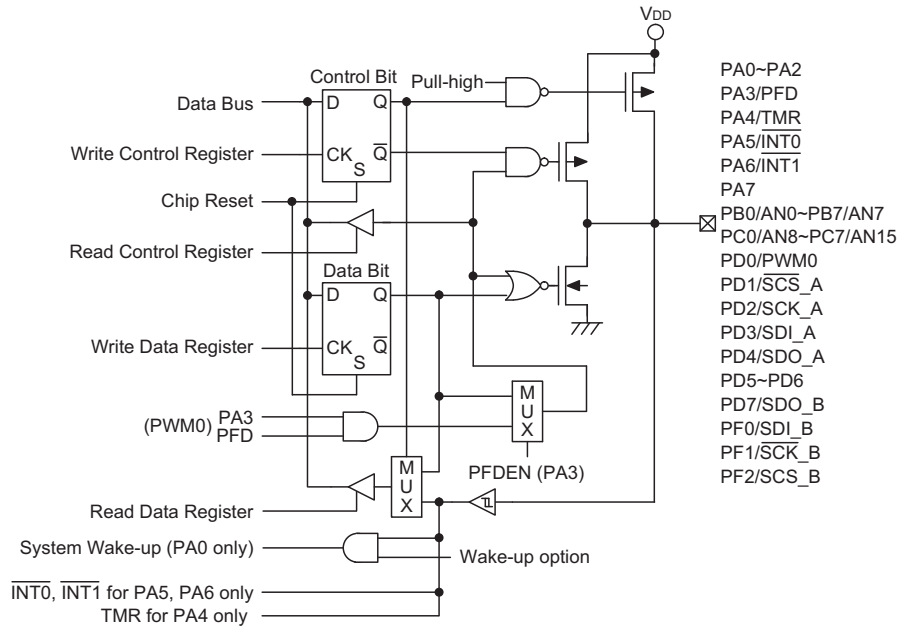
Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, I/O pins, when configured as an input have the capability of being connected to an internal pull-high resistor. The pull-high resistors are selectable via configuration options and are implemented using weak PMOS transistors.

Port A Wake-up

If the HALT instruction is executed, the device will enter the Power Down Mode, where the system clock will stop resulting in power being conserved, 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 the Power Down Mode, 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 port has its own control register PAC, PBC, PCC, PDC and PFC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each of the I/O ports is directly mapped to a bit in its associated port control register. Note that several pins can be setup to have NMOS outputs using configuration options.


Input/Output Ports

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 an output. If the pin is currently setup as an output, instructions can still be used to read the output data 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.

- **External Interrupt Input_0**
 The external interrupt pin $\overline{\text{INT0}}$ is pin-shared with the I/O pin PA5. For applications not requiring an external interrupt input, the pin-shared external interrupt pin can be used as a normal I/O pin, however to do this, the external interrupt enable bits in the INTC0 register must be disabled.
- **External Interrupt Input_1**
 The external interrupt pin $\overline{\text{INT1}}$ is pin-shared with the I/O pin PA6. For applications not requiring an external interrupt input, the pin-shared external interrupt pin can be used as a normal I/O pin, however to do this, the external interrupt enable bits in the INTC1 register must be disabled.

- **External Timer Clock Input**
 The external timer pin TMR is pin-shared with the I/O pin PA4. To configure this pin to operate as timer input, the corresponding control bits in the timer control register must be correctly set. For applications that do not require an external timer input, this pin can be used as a normal I/O pin. Note that if used as a normal I/O pin the timer mode control bits in the timer control register must select the timer mode, which has an internal clock source, to prevent the input pin from interfering with the timer operation.
- **PFD Output**
 Each device contains a PFD function whose single output is pin-shared with PA3. The output function of this pin is chosen via a configuration option and remains fixed after the device is programmed. Note that the corresponding bit of the port control register, PAC.3, must setup the pin as an output to enable the PFD output. If the PAC port control register has setup the pin as an input, then the pin will function as a normal logic input with the usual pull-high option, even if the PFD configuration option has been selected.
- **PWM Output**
 The device contains a PWM output shared with pin PD0. The PWM output functions are chosen via configuration options and remain fixed after the device is programmed. Note that the corresponding bit of the port control register, PDC, must setup the pin as an output to enable the PWM output. If the PDC port control register has setup the pin as an input, then the pin will function as a normal logic input with the usual pull-high option, even if the PWM configuration option has been selected.
- **A/D Inputs**
 These devices can have up to 16 A/D converter inputs depending upon which package type is chosen. All of these analog inputs are pin-shared with I/O pins on

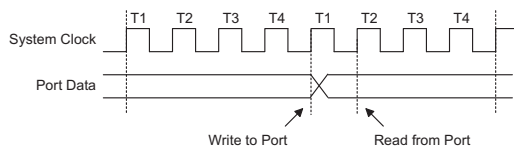
Port B and Port C. If these pins are to be used as A/D inputs and not as normal I/O pins then the corresponding bits in the A/D Converter Control Register, ADCR, must be properly set. There are no configuration options associated with the A/D function. If used as I/O pins, then full pull-high resistor configuration options remain, however if used as A/D inputs then any pull-high resistor options associated with these pins will be automatically disconnected.

I/O Pin Structures

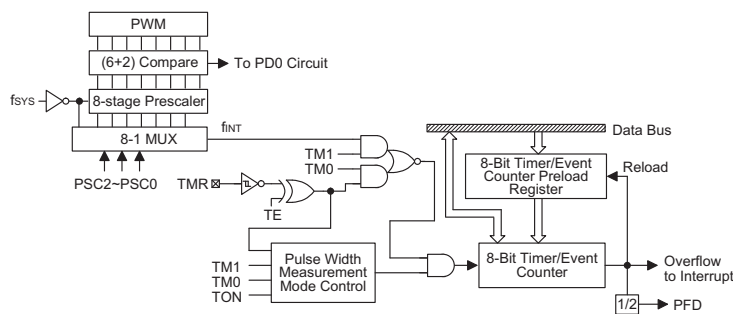
The diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins.

Programming Considerations

Within the user program, one of the first things to consider is port initialisation. After a reset, all of the data and port control register 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 PAC, PBC, PCC, PDC and PFC port control register, are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated PA, PB, PC, PD and PF port data registers are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct value into the 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 re-write this data back to the output ports.



Read/Write Timing



8-bit Timer/Event Counter Structure

Port A has the additional capability of providing wake-up functions. When the device is in the Power Down Mode, 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. These devices contains an internal 8-bit count-up timer which has three operating modes. The timer can be configured to operate as a general timer, external event counter or as a pulse width measurement device. The provision of an internal 8-stage prescaler to the timer clock circuitry gives added range to the timer.

There are two registers related to the Timer/Event Counter, TMR and TMRC. The TMR register is the register that contains the actual timing value. Writing to TMR places an initial starting value in the Timer/Event Counter preload register while reading TMR retrieves the contents of the Timer/Event Counter. The TMRC register is a Timer/Event Counter 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 system clock source or from an external clock on shared pin PA4/TMR.

Configuring the Timer/Event Counter Input Clock Source

The internal timers 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. The internal timer clock also passes through a prescaler, the value of which is conditioned by the bits PSC0, PSC1 and PSC2 in the TMRC register.

An external clock source is used when the timer is in the event counting mode, the clock source being provided on shared pin PA4/TMR. Depending upon the condition of the TE bit, each high to low, or low to high transition on the PA4/TMR pin will increment the counter by one.

Timer Register – TMR

The TMR register is an 8-bit special function register location within the special purpose Data Memory where the actual timer value is stored. The value in the timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the PA4/TMR pin. The timer will count from the initial value loaded by the preload register to the full count value of FFH 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. For a maximum full range count of 00H to FFH the preload register must first be cleared to 00H. It should be noted that after power-on the preload register will be in an unknown condition. Note that if the Timer/Event Counter is not running and data is written to its preload register, 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 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.

Timer Control Register – TMRC

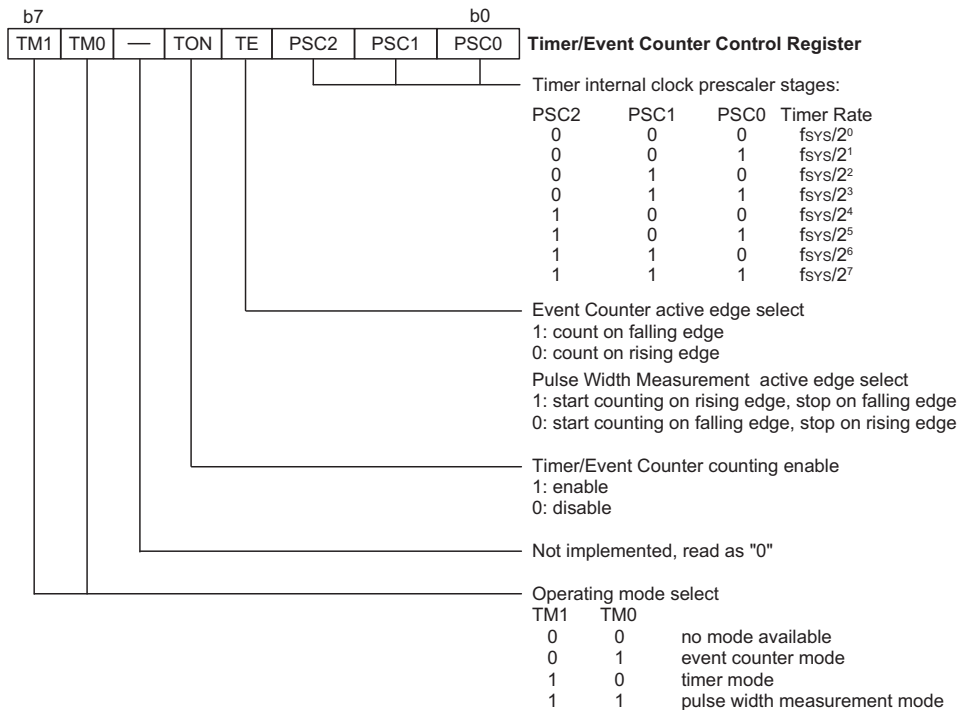
The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of the Timer Control Register TMRC. Together with the TMR register, these two registers control the full operation of the Timer/Event Counter. Before the

timer can be used, it is essential that the TMRC register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialisation.

To choose which of the three modes the timer is to operate 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 TMRC 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. Bits 0–2 of the TMRC register determine the division ratio of the input clock prescaler. The prescaler bit setting has 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 utilised to measure fixed time intervals, providing an internal interrupt signal each time the counter overflows. To operate in this mode, bits TM1 and TM0 of the TMRC register must be set to 1 and 0 respectively. In this mode, the internal clock is used as the timer clock. The input clock frequency to the timer is f_{SYS} divided by the value programmed into the timer prescaler, the value of which is determined by bits PSC0~PSC2 of the TMRC register. 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/Event Counter Control Register

timer increments by one. When the timer is full and overflows, the timer will be reset to the value already loaded into the preload register and continue counting. If the timer interrupt is enabled, an interrupt signal will also be generated. The timer interrupt can be disabled by ensuring that the ETI bit in the INTC register is cleared to zero. It should be noted that a timer overflow is one of the wake-up sources.

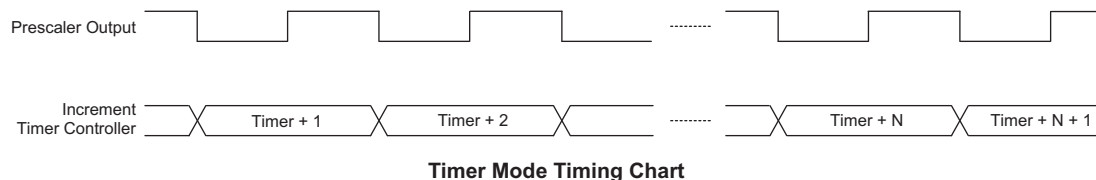
Configuring the Event Counter Mode

In this mode, a number of externally changing logic events, occurring on external pin PA4/TMR, 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 PA4/TMR pin receives a low to high transition. If the TE bit is high, the counter will increment each time PA4/TMR receives a high to low transition. As in the case of the other two modes, when the counter is full and overflows, the timer will be reset to the value already loaded into the preload register and continue counting. If the timer interrupt is enabled, an interrupt signal will also be generated. The timer interrupt can be disabled by ensuring that the ETI bit in the INTC register is cleared to zero. To ensure that the external pin PA4/TMR 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 en-

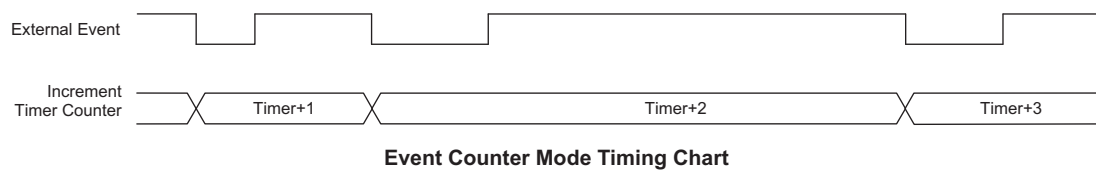
sure that the port control register configures the pin as an input. It should be noted that a timer overflow is one of the wake-up sources. In the Event Counting mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input pin, even if the microcontroller is in the Power Down Mode. As a result when the timer overflows it will generate a wake-up and if the interrupts are enabled also generate a timer interrupt signal.

Configuring the Pulse Width Measurement Mode

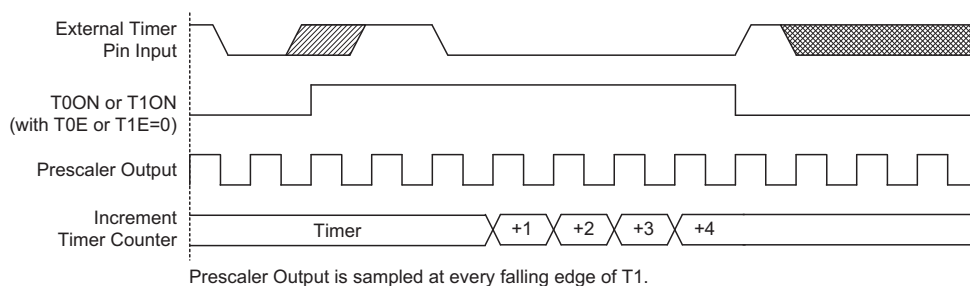
In this mode, the width of external pulses applied to the pin-shared external pin PA4/TMR can be measured. In the Pulse Width Measurement Mode, the timer clock source is supplied by the internal clock. For the timer to operate in this mode, bits TM0 and TM1 must both be set high. If the TE bit is low, once a high to low transition has been received on the PA4/TMR pin, the timer will start counting until the PA4/TMR pin returns to its original high level. At this point the TON bit will be automatically reset to zero and the timer will stop counting. If the TE bit is high, the timer will begin counting counting once a low to high transition has been received on the PA4/TMR pin and stop counting when the PA4/TMR 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



Timer Mode Timing Chart



Event Counter Mode Timing Chart



Pulse Width Measure Mode Timing Chart

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 PA4/TMR. As the TON bit has now been reset any further transitions on the PA4/TMR 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 PA4/TMR pin and not by the logic level.

As in the case of the other two modes, when the counter is full and overflows, the timer will be reset to the value already loaded into the preload register. If the timer interrupt is enabled, an interrupt signal will also be generated. To ensure that the external pin PA4/TMR 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 wake-up sources.

Programmable Frequency Divider – PFD

The PFD output is pin-shared with the I/O pin PA3. The PFD function is selected via configuration option, however, if not selected, the pin can operate as a normal I/O pin. The timer overflow signal is the clock source for the PFD circuit. The output frequency is controlled by loading the required values into the timer prescaler registers to give the required division ratio. The counter, driven by the system clock which is divided by the prescaler value, will begin to count-up from this preload register value until full, at which point an overflow signal is generated, causing the PFD output to change state. The counter will then be automatically reloaded with the preload register value and continue counting-up.

For the output to function, it is essential that the corresponding bit of the Port A control register PAC bit 3 is setup as an output. If setup as an input the PFD output will not function, however, the pin can still be used as a normal input pin. The PFD output will only be activated if bit PA3 is set to "1". This output data bit is used as the

on/off control bit for the PFD output. Note that the PFD output will be low if the PA3 output data bit is cleared to "0".

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

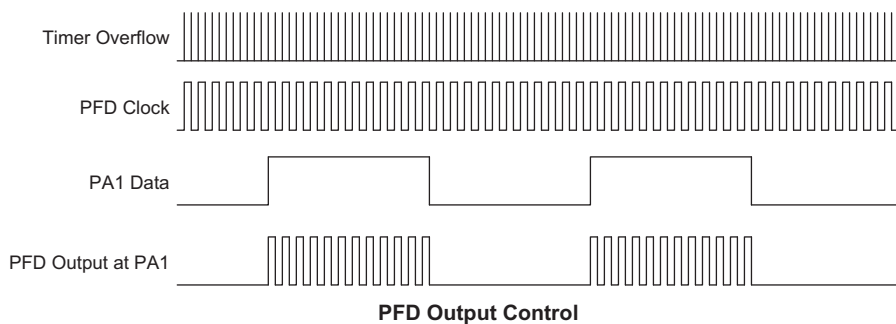
Bits PSC0~PSC2 of the TMRC register are used to define the pre-scaling stages of the internal clock source of the Timer/Event Counter.

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 PA4/ pin for correct operation. As this pin is a shared pin it must be configured correctly to ensure it is setup for use as a Timer/Event Counter input and not as a normal I/O pin. This is implemented by ensuring that the mode select bits in the Timer/Event Counter control register, select either the event counter or pulse width measurement mode. Additionally the Port Control Register PAC bit 4 must be set high to ensure that the pin is setup as an input. Any pull-high resistor configuration option on this pin will remain valid even if the pin is used as a Timer/Event Counter input.

Programming Considerations

When configured to run in the timer mode, the internal system clock is used as the timer clock source and is therefore synchronised 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 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 synchronised 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 synchronised with the internal system or timer clock.

When the Timer/Event Counter is read, or if data is written to the preload register, the clock is inhibited to avoid errors, however as this may result in a counting error, this should be taken into account by the programmer. Care must be taken to ensure that the timers are properly initialised before using them for the first time. The associated timer enable bits in the interrupt control register must be properly set otherwise the internal interrupt associated with the timer will remain inactive. The edge select, timer mode and clock source control bits in timer control register must also be correctly set to ensure the timer is properly configured for the required application. It is also important to ensure that an initial value is first loaded into the timer registers before the timer is switched on; this is because after power-on the initial values of the timer registers are unknown. After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the timer control register. Note that setting the timer enable bit high to turn the timer on, should only be executed after the timer mode bits have been properly setup. Setting the timer enable bit high together with a mode bit modification, may lead to improper timer operation if executed as a single timer control register byte write instruction.

When the Timer/Event counter overflows, its corresponding interrupt request flag in the interrupt control register will be set. If the timer interrupt is enabled this will in turn generate an interrupt signal. However irrespective of whether the interrupts are enabled or not, a Timer/Event counter overflow will also generate a wake-up signal if the device is in a Power-down condition. This situation may occur if the Timer/Event Counter is in the Event Counting Mode and if the external signal continues to change state. In such a case, the Timer/Event Counter will continue to count these external events and if an overflow occurs the device will be woken up from its Power-down condition. To prevent such a wake-up from occurring, the timer interrupt request flag should first be set high before issuing the "HALT" instruction to enter the Power Down Mode.

Timer Program Example

This program example shows how the Timer/Event Counter registers are setup, along with how the interrupts are enabled and managed. Note how the Timer/Event Counter is turned on, by setting bit 4 of the Timer Control Register. The Timer/Event Counter can be turned off in a similar way by clearing the same bit. This example program sets the Timer/Event Counter to be in the timer mode, which uses the internal system clock as the clock source.

```

org 04h          ; external interrupt vector
reti
org 08h          ; Timer/Event Counter interrupt vector
jmp tmrint      ; jump here when Timer overflows
:
org 20h          ; main program
;internal Timer/Event Counter interrupt routine
tmrint:
:
; Timer/Event Counter main program placed here
:
reti
:
:
begin:
;setup Timer registers
mov a,09bh      ; setup Timer preload value
mov tmr,a;
mov a,081h     ; setup Timer control register
mov tmrc,a     ; timer mode and prescaler set to /2
; setup interrupt register
mov a,005h     ; enable master interrupt and timer interrupt
mov intc,a
set tmrc.4     ; start Timer/Event Counter - note mode bits must be previously setup

```

Interrupts

Interrupts are an important part of any microcontroller system. When an external interrupt pin transition or an internal function such as a Timer/Event Counter overflow, an A/D converter conversion, or transmission or reception of SPI data occurs, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. Each device contains two external interrupts and several internal interrupts functions. The external interrupt is controlled by the action of the external interrupt pins, while the internal interrupts are controlled by the Timer/Event Counter overflow, the A/D converter and SPI data transmission or reception.

Interrupt Register

Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by the two interrupt control registers, which are located in the Data Memory. By controlling the appropriate enable bits in these registers each individual interrupt can be enabled or disabled. Also when an interrupt occurs, the corresponding request flag will be set by the microcontroller. The global enable flag if cleared to zero will disable all interrupts.

Interrupt Operation

An A/D converter end of conversion, a Timer/Event Counter overflow, 8-bits of data transmission or reception on either of the two SPI interfaces or an active edge on any of the two external interrupt pins will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a JMP statement which will jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI statement, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

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

Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be

immediately serviced, the request flag will still be recorded. If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full.

Interrupt Priority

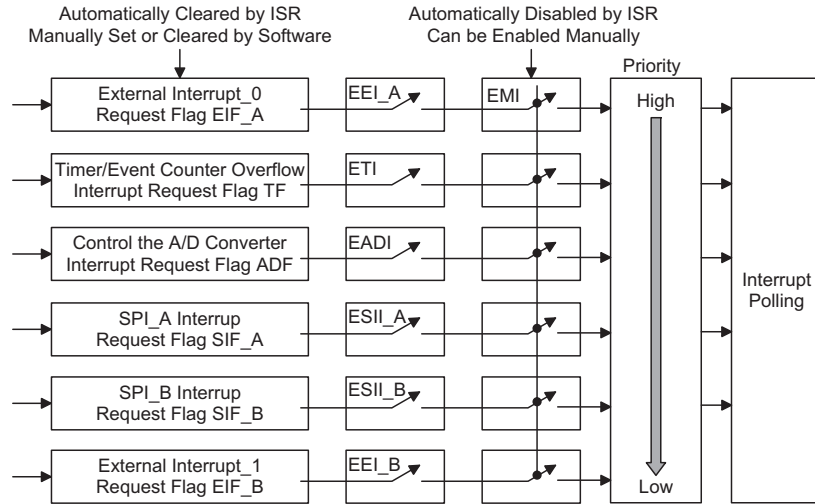
Interrupts, occurring in the interval between the rising edges of two consecutive T2 pulses, will be serviced on the latter of the two T2 pulses, if the corresponding interrupts are enabled. In case of simultaneous requests, the following table shows the priority that is applied. These can be masked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

Interrupt Source	Priority	Vector
External Interrupt $\overline{\text{INT0}}$	1	0004H
Timer/Event Counter Overflow Interrupt	2	0008H
End of A/D Converter Interrupt	3	000CH
SPI_A Interrupt	4	0010H
SPI_B Interrupt	5	0014H
External Interrupt $\overline{\text{INT1}}$	6	0018H

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 interrupt registers can prevent simultaneous occurrences.

External Interrupt

For an external interrupt to occur, the global interrupt enable bit, EMI, and external interrupt enable bit, EEI_A, or EEI_B must first be set. An actual external interrupt will take place when the external interrupt request flag, EIF_A or EIF_B is set, a situation that will occur when a high to low transition appears on the interrupt pins. The external interrupt pin is pin-shared with the I/O pins PA5 and PA6 and can only be configured as an external interrupt pin if the corresponding external interrupt enable bits in the interrupt control register INTC0 and INTC1 have been set. The pins must also be setup as inputs by setting the corresponding PAC.5 and PAC.6 bits in the port control register. When the interrupt is enabled, the stack is not full and a high to low transition appears on the external interrupt pin, a subroutine call to the external interrupt vector at location 04H or 018H will take place. When the interrupt is serviced, the external interrupt request flag, EIF_A or EIF_B will be automatically reset and the EMI bit will be automatically cleared to dis-



Interrupt Structure

able other interrupts. Note that any pull-high resistor configuration options on these pins will remain valid even if the pins are used as external interrupt inputs.

Timer/Event Counter Interrupt

For a Timer/Event Counter interrupt to occur, the global interrupt enable bit, EMI, and the corresponding timer interrupt enable bit, ETI, must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter interrupt request flag, TF, is set, a situation that will occur when the Timer/Event Counter overflows. When the interrupt is enabled, the stack is not full and a Timer/Event Counter overflow occurs, a subroutine call to the timer interrupt vector at location 08H, will take place. When the interrupt is serviced, the timer interrupt request flag, TF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

SPI Interrupt

For an SPI Interrupt to occur, the global interrupt enable bit, EMI, and the corresponding SPI interrupt enable bit, ESII_A or ESII_B, must be first set. The SBEN bit in the SBCR register must also be set. An actual SPI Interrupt will take place when one of the two SPI interrupt request flags, SIF_A or SIF_B, is set, a situation that will occur when 8-bits of data are transferred or received from either of the SPI interfaces. When the interrupt is enabled, the stack is not full and an SPI_A interrupt occurs, a subroutine call to the SPI_A interrupt vector at location 10H, will take place. For an SPI_B interrupt, a subroutine call to the SPI_B interrupt vector at location 14H, will take place. When the interrupt is serviced, the SPI interrupt request flag, SIF_A or SIF_B, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

A/D Interrupt

For an A/D converter interrupt to occur, the global interrupt enable bit, EMI, and the corresponding A/D converter interrupt enable flag, EADI, must first be set. An actual A/D Interrupt will take place when the A/D converter interrupt request flag, ADF, is set, as situation that will occur when the A/D Converter conversion process has completed. When the interrupt is enabled, the stack is not full and an A/D conversion process has completed, a subroutine call to location 0CH will take place. When the interrupt is serviced, the A/D interrupt request flag, ADF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Programming Considerations

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 interrupt control 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.

All of these interrupts have the capability of waking up the processor when in the Power Down Mode.

Only the Program Counter is pushed onto the stack. If the contents of the accumulator or status register are altered by the interrupt service program, which may corrupt the desired control sequence, then the contents should be saved in advance.

Reset and Initialisation

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

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

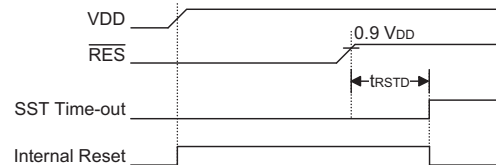
Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the RES reset is implemented in situations where the power supply voltage falls below a certain threshold.

Reset Functions

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

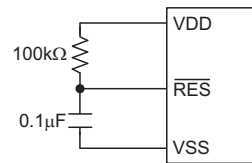
- Power-on Reset
The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to 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, if the VDD power supply rise time is not fast enough or does not stabilise quickly at power-on, the internal reset function may be incapable of providing a proper reset operation. In such cases it is recom-

mended that an external RC network is connected to the $\overline{\text{RES}}$ pin, whose additional time delay will ensure that the RES pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be 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 will begin normal operation. The abbreviation SST in the figures stands for System Start-up Timer.



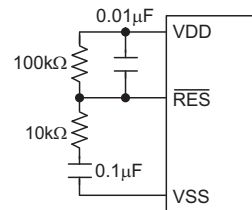
Power-On Reset Timing Chart

For most applications a resistor connected between VDD and the RES pin and a capacitor connected between VSS and the $\overline{\text{RES}}$ pin will provide a suitable external reset circuit. Any wiring connected to the RES pin should be kept as short as possible to minimise any stray noise interference.



Basic Reset Circuit

For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.

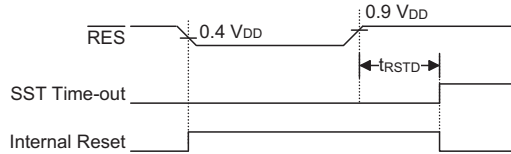


Enhanced Reset Circuit

More information regarding external reset circuits is located in Application Note HA0075E on the Holtek website.

- **RES Pin Reset**

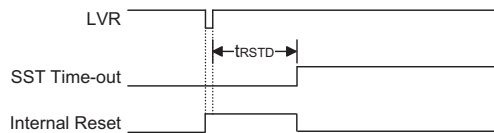
This type of reset occurs when the microcontroller is already running and the RES pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point. Note that as the external reset pin is also pin-shared with PA7, if it is to be used as a reset pin, the correct reset configuration option must be selected.



RES Reset Timing Chart

- **Low Voltage Reset – LVR**

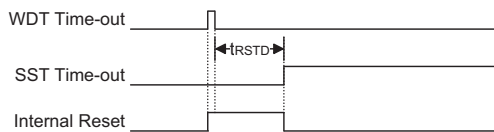
The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. The LVR function is selected via a configuration option. If the supply voltage of the device drops to within a range of $0.9V \sim V_{LVR}$ such as might occur when changing the battery, the LVR will automatically reset the device internally. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between $0.9V \sim V_{LVR}$ must exist for a time greater than that specified by t_{LVR} in the A.C. characteristics. If the low supply voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function. The actual V_{LVR} value can be selected via configuration options.



Low Voltage Reset Timing Chart

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

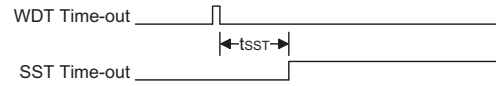
The Watchdog time-out Reset during normal operation is the same as a hardware RES pin reset except that the Watchdog time-out flag TO will be set to "1".



WDT Time-out Reset during Normal Operation Timing Chart

- **Watchdog Time-out Reset during Power Down**

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



WDT Time-out Reset during Power Down Timing Chart

Reset Initial Conditions

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

TO	PDF	RESET Conditions
0	0	RES reset during power-on
0	1	RES wake-up during Power Down
u	u	RES or LVR reset during normal operation
1	u	WDT time-out reset during normal operation
1	1	WDT time-out reset during Power Down

Note: "u" stands for unchanged

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

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

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

Register	Reset (Power-on)	WDT time-out (Normal Operation)	RES Reset (Normal Operation)	RES Reset (HALT)	WDT Time-out (HALT)*
MP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
Program Counter	000H	000H	000H	000H	000H
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	-xxx xxxx	-uuu uuuu	-uuu uuuu	-uuu uuuu	-uuu uuuu
STATUS	--00 xxxx	--1u uuuu	--uu uuuu	--01 uuuu	--11 uuuu
INTC	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMRC	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PD	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PWM	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
PF	---- -111	---- -111	---- -111	---- -111	---- -uuu
PFC	---- -111	---- -111	---- -111	---- -111	---- -uuu
INTC1	-000 -000	-000 -000	-000 -000	-000 -000	-uuu -uuu
ADR	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR	0100 0000	0100 0000	0100 0000	0100 0000	uuuu uuuu
ACSR	1--- --00	1--- --00	1--- --00	1--- --00	u--- --uu
SBCR_A	0110 0000	0110 0000	0110 0000	0110 0000	uuuu uuuu
SBDR_A	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
SBCR_B	0110 0000	0110 0000	0110 0000	0110 0000	uuuu uuuu
SBDR_B	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu

Note: "*" means "warm reset"
 "-" not implemented
 "u" means "unchanged"
 "x" means "unknown"

Oscillator

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

The two methods of generating the system clock are:

- External crystal/resonator oscillator
- External RC oscillator

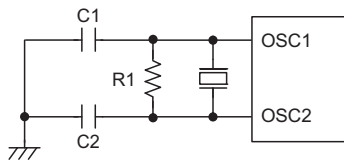
One of these two methods must be selected using the configuration options.

More information regarding the oscillator is located in Application Note HA0075E on the Holtek website.

External Crystal/Resonator Oscillator

After selecting the correct configuration option, for most crystal oscillator configurations, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation. However, to ensure oscillation, it is recommended that two small value capacitors and a resistor, the values of which are shown in the table, should be connected as shown in the diagram.

The following table shows the C1, C2 and R1 values according to different crystal values.



Crystal/Ceramic Oscillator

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

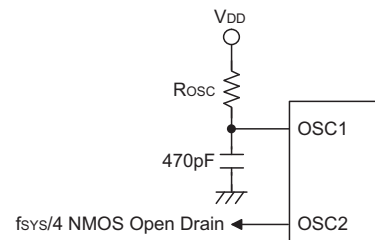
Note: The resistance and capacitance for reset circuit should be designed in such a way as to ensure

that the V_{DD} is stable and remains within a valid operating voltage range before bringing RES to high.

*** Make the length of the wiring, which is connected to the RES pin as short as possible, to avoid noise interface.

External RC Oscillator

Using the external system RC oscillator requires that a resistor, with a value between 4.7kΩ and 750kΩ, is connected between OSC1 and VDD, and a 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. Note that as the OSC2 output is an NMOS open-drain type, a pull high resistor should be connected if it to be used to monitor the internal frequency. Although this is a cost effective oscillator configuration, the oscillation frequency can vary with VDD, temperature and process variations 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} refer to the Holtek website for typical RC Oscillator vs. Temperature and VDD characteristics graphics. Note that it is the only microcontroller internal circuitry together with the external resistor, that determine the frequency of the oscillator. The external capacitor shown on the diagram does not influence the frequency of oscillation.



RC Oscillator

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.

Power Down Mode and Wake-up

Power Down Mode

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

Entering the Power Down Mode

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

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

Standby Current Considerations

As the main reason for entering the Power Down Mode is to keep the current consumption of the microcontroller to as low a value as possible, perhaps only in the order of several micro-amps, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimised.

Special attention must be made to the I/O pins on the device. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. Care must also be taken with the loads, which are connected to I/O pins, which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs.

If the configuration options have enabled the Watchdog Timer internal oscillator then this will continue to run when in the Power Down Mode and will thus consume some power. For power sensitive applications it may be therefore preferable to use the system clock source for the Watchdog Timer. If any I/O pins are configured as A/D analog inputs using the channel configuration bits in the ADCR register, then the A/D converter will be turned on and a certain amount of power will be consumed. It may be therefore desirable before entering the Power Down Mode to ensure that the A/D converter is powered down by ensuring that any A/D input pins are setup as normal logic inputs with pull-high resistors.

Wake-up

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

- An external reset
- An external falling edge on Port A
- A system interrupt
- A WDT overflow

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

Each pin on Port A can be setup via an individual configuration option to permit a negative transition on the pin to wake-up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction.

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

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

Watchdog Timer

The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by one of two sources selected by configuration option: its own self contained dedicated internal WDT oscillator or $f_{SYS}/4$. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

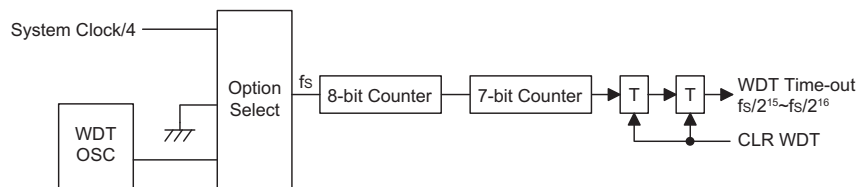
All Watchdog Timer options, such as enable/disable, WDT clock source and clear instruction type all selected through configuration options. There are no internal registers associated with the WDT in this device. One of the WDT clock sources is an internal oscillator which has an approximate period of $65\mu s$ at a supply voltage of 5V. However, it should be noted that this specified internal clock period can vary with VDD, temperature and process variations. The other WDT clock source option is the $f_{SYS}/4$ clock. Whether the WDT clock source is its own internal WDT oscillator, or from $f_{SYS}/4$, it is further divided by an internal 15-bit counter and a clearable single bit counter to give longer Watchdog time-outs. As this ratio is fixed it gives an overall Watchdog Timer time-out value of $2^{15}/f_s$ to $2^{16}/f_s$. As the clear instruction only resets the last stage of the divider chain, for this reason the actual division ratio and corresponding Watchdog Timer time-out can vary by a factor of two.

The exact division ratio depends upon the residual value in the Watchdog Timer counter before the clear instruction is executed. It is important to realise that as there are no independent internal registers or configuration options associated with the length of the Watchdog Timer time-out, it is completely dependent upon the frequency of $f_{SYS}/4$ or the internal WDT oscillator.

If the $f_{SYS}/4$ clock is used as the WDT clock source, it should be noted that when the system enters the Power Down Mode, then the instruction clock is stopped and the WDT will lose its protecting purposes. For systems that operate in noisy environments, using the internal WDT oscillator is strongly recommended.

Under normal program operation, a WDT time-out will initialise a device reset and set the status bit TO. However, if the system is in the Power Down Mode, when a WDT time-out occurs, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the WDT. The first is an external hardware reset, which means a low level on the RES pin, the second is using the watchdog 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.



Watchdog Timer

Pulse Width Modulator

The device provides a single channel Pulse Width Modulator function. Useful for such applications such as motor speed control, the Pulse Width Modulator function provides an output with a fixed frequency but with a duty cycle that can be varied by setting particular values into the corresponding PWM register.

Channels	PWM Mode	Output Pins	Register Name
1	6+2	PD0	PWM

A single register, located in the Data Memory is assigned to the Pulse Width Modulator and is known as the PWM register. It is in this register that the 8-bit value, which represents the overall duty cycle of one modulation cycle of the output waveform, should be placed. To increase the PWM modulation frequency, each modulation cycle is modulated into four individual modulation sub-sections, known as the 6+2 mode. Note that it is only necessary to write the required modulation value into the corresponding PWM register as the subdivision of the waveform into its sub-modulation cycles is implemented automatically within the microcontroller hardware. The PWM clock source is the system clock f_{SYS} .

This method of dividing the original modulation cycle into a further 4 sub-cycles enables the generation of higher PWM frequencies, which allow a wider range of applications to be served. As long as the periods of the generated PWM pulses are less than the time constants of the load, the PWM output will be suitable as such long time constant loads will average out the pulses of the PWM output. The difference between what is known as the PWM cycle frequency and the PWM modulation fre-

quency should be understood. As the PWM clock is the system clock, f_{SYS} , and as the PWM value is 8-bits wide, the overall PWM cycle frequency is $f_{SYS}/256$. However, while the PWM modulation frequency for the 6+2 mode of operation will be $f_{SYS}/64$.

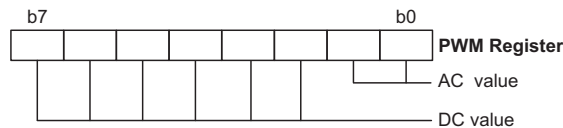
6+2 PWM Mode

Each full PWM cycle, as it is controlled by an 8-bit PWM, PWM0 register, has 256 clock periods. However, in the 6+2 PWM Mode, each PWM cycle is subdivided into four individual sub-cycles known as modulation cycle 0~modulation cycle 3, denoted as "i" in the table. Each one of these four sub-cycles contains 64 clock cycles. In this mode, a modulation frequency increase by a factor of four is achieved. The 8-bit PWM register value, which represents the overall duty cycle of the PWM waveform, is divided into two groups. The first group which consists of bit2~bit7 is denoted here as the DC value. The second group which consists of bit0~bit1 is known as the AC value. In the 6+2 PWM mode, the duty cycle value of each of the four modulation sub-cycles is shown in the following table.

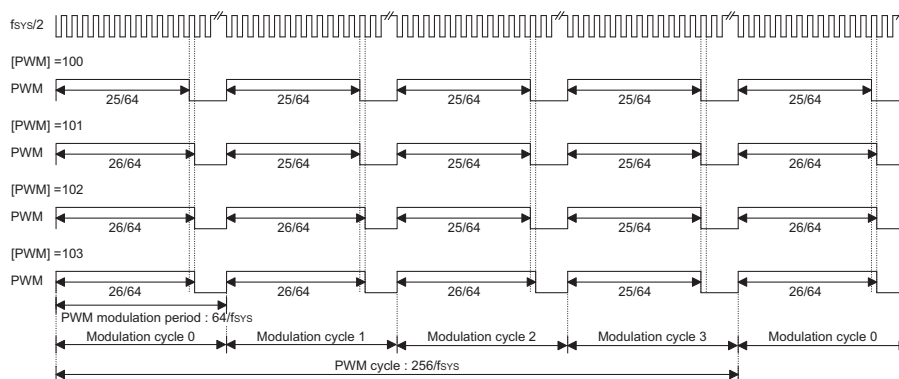
Parameter	AC (0~3)	DC (Duty Cycle)
Modulation cycle i (i=0~3)	$i < AC$	$\frac{DC + 1}{64}$
	$i \geq AC$	$\frac{DC}{64}$

6+2 Mode Modulation Cycle Values

The following diagram illustrates the waveforms associated with the 6+2 mode of PWM operation. It is impor-



6+2 Mode PWM Register



6+2 PWM Mode

tant to note how the single PWM cycle is subdivided into 4 individual modulation cycles, numbered from 0~3 and how the AC value is related to the PWM value.

PWM Output Control

PWM Modulation Frequency	PWM Cycle Frequency	PWM Cycle Duty
$f_{\text{SYS}}/64$	$f_{\text{SYS}}/256$	(PWM register value)/256

The PWM0 output is shared with pin PD0. To operate as a PWM0 output and not as I/O pins, the correct PWM configuration option must be selected. A "0" must also be written to the corresponding bit in the I/O port control register, PDC.0, to ensure that the PWM output pin is setup as an output. After these two initial steps have been carried out, and of course after the required PWM value has been written into the PWM register, writing a "1" to the corresponding PD.0 bit in the PD output data register will enable the PWM data to appear on the pin. Writing a "0" to the bit will disable the PWM output function and force the output low. In this way, the Port D data output register bit, PD.0, can be used as an on/off control for the PWM function. Note that if the configuration options have selected the PWM function, but a "1" has been written to its corresponding bit in the PDC control register to configure the pin as an input, then the pin can still function as a normal input line, with pull-high resistor options.

PWM Programming Example

The following sample program shows how the PWM output is setup and controlled. Before use the corresponding PWM output configuration options must first be selected.

```

mov a, 64h          ; setup PWM value of 100
                   ; decimal which is 64H
mov pwm, a
clr pdc.0          ; setup pin PD0 as an output
set pd.0           ; PD.0=1; enable the PWM
                   ; output
: :
: :
clr pd.0           ; disable the PWM output -
                   ; PD0 will remain low

```

Analog to Digital Converter

The need to interface to real world analog signals is a common requirement for many electronic systems. However, to properly process these signals by a microcontroller, they must first be converted into digital signals by A/D converters. By integrating the A/D conversion electronic circuitry into the microcontroller, the need for external components is reduced significantly with the corresponding follow-on benefits of lower costs and reduced component space requirements.

A/D Overview

Depending upon which package type is chosen, the devices contain either a 6 or 16-channel analog to digital converter which can directly interface to external analog signals, such as that from sensors or other control signals and convert these signals directly into an 8-bit digital value.

Package	Channels	Resolution	Input Pins
28-pin	6	8-bit	PB0~PB3 PC1, PC2
44-pin	16	8-bit	PB0~PB7 PC0~PC7

The A/D block diagram shows the overall internal structure of the A/D converter, together with its associated registers.

A/D Converter Data Registers – ADR

The devices, which contain a single 8-bit A/D converter, require one data register, known as ADR. After the conversion process takes place, this register can be directly read by the microcontroller to obtain the digitised conversion value.

In the following tables, D0~D7 are the A/D conversion data result bits.

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADR	D7	D6	D5	D4	D3	D2	D1	D0

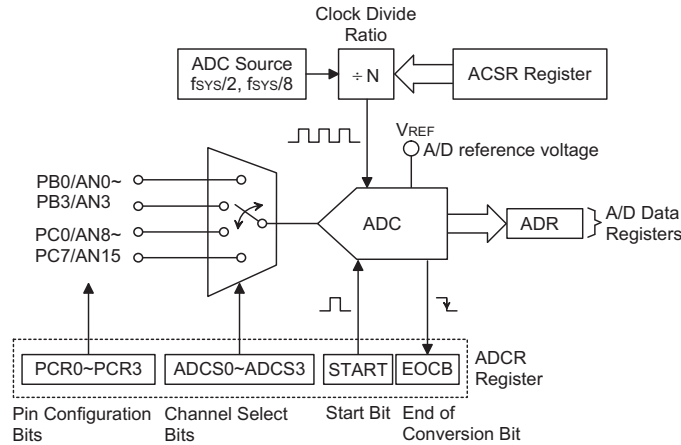
A/D Data Register

A/D Converter Control Register – ADCR

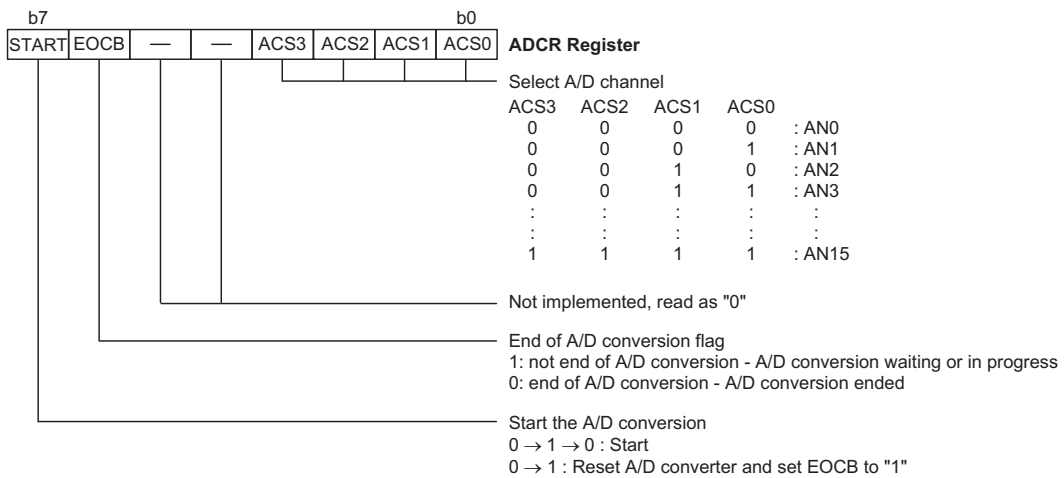
To control the function and operation of the A/D converter, control registers known as ADCR and ADSR are provided. These 8-bit registers define functions such as the selection of which analog channel is connected to the internal A/D converter, which pins are used as analog inputs and which are used as normal I/Os as well as controlling the start function and monitoring the A/D converter end of conversion status.

One section of this register contains the bits ACS3~ACS0 which define the channel number. As each of the devices contains only one actual analog to digital converter circuit, each of the individual 4 analog inputs must be routed to the converter. It is the function of the ACS3~ACS0 bits in the ADCR register to determine which analog channel is actually connected to the internal A/D converter. Note that the ACS3 bit must always be assigned a zero value.

The START bit in the ADCR register is used to start and reset the A/D converter. When the microcontroller sets this bit from low to high and then low again, an analog to digital conversion cycle will be initiated. When the START bit is brought from low to high but not low again,



A/D Converter Structure



A/D Converter Control Register

the EOCB bit in the ADCR register will be set to a "1" and the analog to digital converter will be reset. It is the START bit that is used to control the overall on/off operation of the internal analog to digital converter.

The EOCB bit in the ADCR register is used to indicate when the analog to digital conversion process is complete. This bit will be automatically set to "0" by the microcontroller after a conversion cycle has ended. In addition, the corresponding A/D interrupt request flag will be set in the interrupt control register, and if the interrupts are enabled, an appropriate internal interrupt signal will be generated. This A/D internal interrupt signal will direct the program flow to the associated A/D internal interrupt address for processing. If the A/D internal interrupt is disabled, the microcontroller can be used to poll the EOCB bit in the ADCR register to check whether it has been cleared as an alternative method of detecting the end of an A/D conversion cycle.

A/D Converter Clock Source Register – ACSR

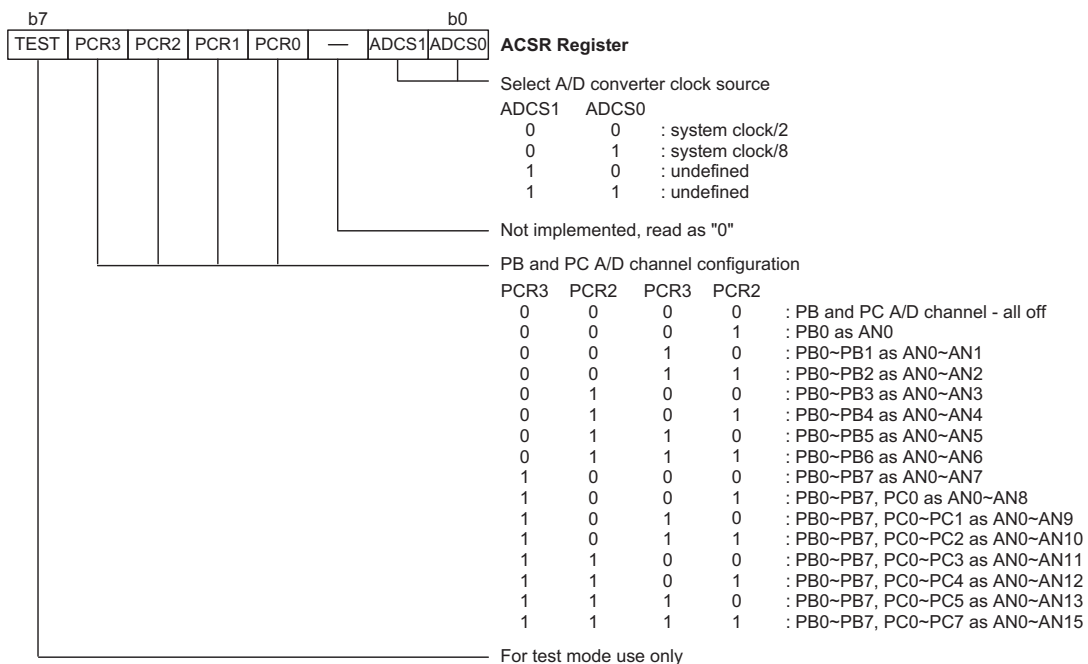
The clock source for the A/D converter, which originates

from the system clock f_{SYS} , is first divided by a division ratio, the value of which is determined by the ADCS1 and ADCS0 bits in the ACSR register.

The ACSR control register also contains the PCR3~PCR0 bits which determine which pins on Port B and Port C are used as analog inputs for the A/D converter and which pins are to be used as normal I/O pins. If the 3-bit address on PCR3~PCR0 has a value of "1111" or higher, then all 16 pins, namely AN0, AN1, AN2 and AN15 will all be set as analog inputs. Note that if the PCR3~PCR0 bits are all set to zero, then all the Port B and Port C pins will be setup as normal I/Os and the internal A/D converter circuitry will be powered off to reduce the power consumption.

Although the A/D clock source is determined by the system clock f_{SYS} , and by bits ADCS1 and ADCS0, there are some limitations on the maximum A/D clock source speed that can be selected. As the minimum value of permissible A/D clock period, t_{AD} , is 0.5 μ s, care must be taken for system clock speeds in excess of 4MHz. For system clock speeds in excess of 4MHz, the ADCS1 and ADCS0 bits

f _{sys}	A/D Clock Period (t _{AD})			
	ADCS1, ADCS0=00 (f _{sys} /2)	ADCS1, ADCS0=01 (f _{sys} /8)	ADCS1, ADCS0=10	ADCS1, ADCS0=11
1MHz	2μs	8μs	Undefined	Undefined
2MHz	1μs	4μs	Undefined	Undefined
4MHz	500ns*	2μs	Undefined	Undefined
8MHz	250ns*	1μs	Undefined	Undefined
12MHz	167ns*	667ns*	Undefined	Undefined

A/D Clock Period Examples

A/D Converter Clock Source Register

should not be set to "00". Doing so will give A/D clock periods that are less than the minimum A/D clock period which may result in inaccurate A/D conversion values. Refer to the following table for examples, where values marked with an asterisk * show where, depending upon the device, special care must be taken, as the values may be less than the specified minimum A/D Clock Period.

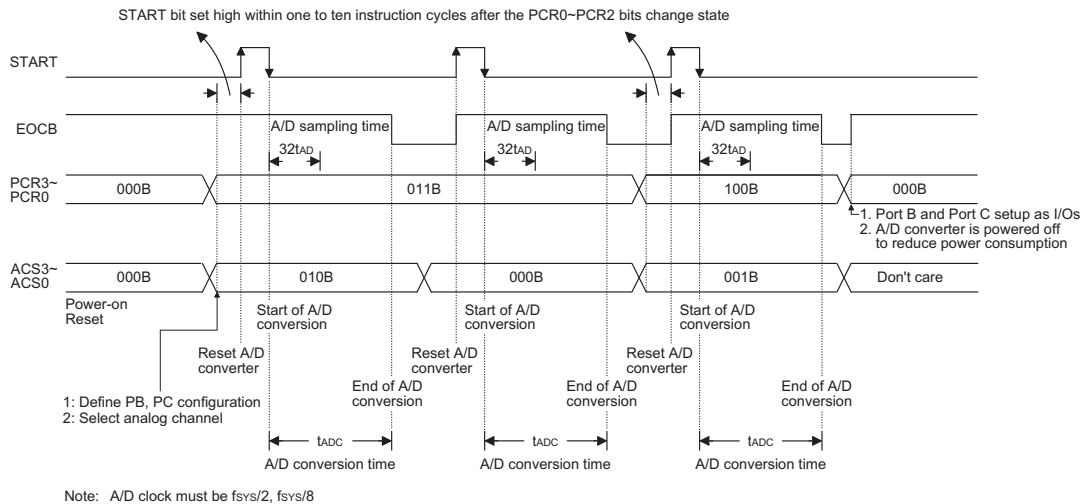
A/D Input Pins

All of the A/D analog input pins are pin-shared with the I/O pins on Port B and Port C. Bits PCR3~PCR0 in the ACSR register, not configuration options, determine whether the input pins are setup as normal Port B and Port C input/output pins or whether they are setup as analog inputs. In this way, pins can be changed under program control to change their function from normal I/O operation to analog inputs and vice versa. Pull-high resistors, which are setup through configuration options, apply to the input pins only when they are used as normal I/O

pins, if setup as A/D inputs the pull-high resistors will be automatically disconnected. Note that it is not necessary to first setup the A/D pin as an input in the PBC or PCC port control register to enable the A/D input as when the PCR3~PCR0 bits enable an A/D input, the status of the port control register will be overridden. The VREF pin is used as the A/D converter reference voltage. The value of VREF must not exceed the VDD value. Appropriate measures should also be taken to ensure that the VREF pin remains as stable and noise free as possible. Note that on the 28-pin package, there is no VREF pin as it is internally connected to VDD.

Initialising the A/D Converter

The internal A/D converter must be initialised in a special way. Each time the Port B and Port C A/D channel selection bits are modified by the program, the A/D converter must be re-initialised. If the A/D converter is not initialised after the channel selection bits are changed,



A/D Conversion Timing

the EOCB flag may have an undefined value, which may produce a false end of conversion signal. To initialise the A/D converter after the channel selection bits have changed, then, within a time frame of one to ten instruction cycles, the START bit in the ADCR register must first be set high and then immediately cleared to zero. This will ensure that the EOCB flag is correctly set to a high condition.

Summary of A/D Conversion Steps

The following summarises the individual steps that should be executed in order to implement an A/D conversion process.

- Step 1
Select the required A/D conversion clock by correctly programming bits ADSC1 and ADSC0 in the ACSR register.
- Step 2
Select which channel is to be connected to the internal A/D converter by correctly programming the ACS3-ACS0 bits which are also contained in the ADCR register.
- Step 3
Select which pins on Port B and Port C are to be used as A/D inputs and configure them as A/D input pins by correctly programming the PCR3-PCR0 bits in the ACSR register.
- Step 4
If the interrupts are to be used, the interrupt control registers must be correctly configured to ensure the A/D converter interrupt function is active. The master interrupt control bit, EMI, in the INTC interrupt control register must be set to "1" and the A/D converter interrupt bit, EADI, in the INTC register must also be set to "1".
- Step 5
The analog to digital conversion process can now be initialised by setting the START bit in the ADCR regis-

ter from "0" to "1" and then to "0" again. Note that this bit should have been originally set to "0".

- Step 6
To check when the analog to digital conversion process is complete, the EOCB bit in the ADCR register can be polled. The conversion process is complete when this bit goes low. When this occurs the A/D data registers ADR can be read to obtain the conversion value. As an alternative method if the interrupts are enabled and the stack is not full, the program can wait for an A/D interrupt to occur.

Note: When checking for the end of the conversion process, if the method of polling the EOCB bit in the ADCR register is used, the interrupt enable step above can be omitted.

The A/D conversion timing diagram shows graphically the various stages involved in an analog to digital conversion process and its associated timing.

The setting up and operation of the A/D converter function is fully under the control of the application program as there are no configuration options associated with the A/D converter. After an A/D conversion process has been initiated by the application program, the microcontroller internal hardware will begin to carry out the conversion, during which time the program can continue with other functions. The time taken for the A/D conversion is 76t_{AD} where t_{AD} is equal to the A/D clock period.

Programming Considerations

When programming, special attention must be given to the A/D channel selection bits in the ADSR register. If these bits are all cleared to zero no external pins will be selected for use as A/D input pins allowing the pins to be used as normal I/O pins. When this happens the power supplied to the internal A/D circuitry will be reduced resulting in a reduction of supply current. This ability to re-

duce power by turning off the internal A/D function by clearing the A/D channel selection bits may be an important consideration in battery powered applications.

Another important programming consideration is that when the A/D channel selection bits change value, the A/D converter must be re-initialised. This is achieved by pulsing the START bit in the ADCR register immediately after the channel selection bits have changed state. The exception to this is where the channel selection bits are all cleared, in which case the A/D converter is not required to be re-initialised.

A/D Programming Example

The following two programming examples illustrate how to setup and implement an A/D conversion. In the first example, the method of polling the EOCB bit in the ADCR register is used to detect when the conversion cycle is complete, whereas in the second example, the A/D interrupt is used to determine when the conversion is complete.

Example: using an EOCB polling method to detect the end of conversion

```

    clr  EADI                ; disable ADC interrupt
    mov  a,00011001
    mov  acsr,a              ; setup the ACSR register to select fsys/8 as the A/D clock
                                ; setup the ACSR register to configure Port PB0~PB2 as A/D
                                ; Inputs

    mov  a,00000000
    mov  adcr,a              ; setup the ADCR register and select AN0 to be connected to
                                ; the A/D converter
                                ; As the Port B and Port C channel bits have changed the
                                ; following START signal (0→1→0) must be issued within
                                ; 10 instruction cycles
:
Start_conversion:
    clr  START
    set  START                ; reset A/D
    clr  START                ; start A/D
Polling_EOC:
    sz   EOCB                 ; poll the ADCR register EOCB bit to detect end
                                ; of A/D conversion
    jmp  polling_EOC         ; continue polling
    mov  a,ADR                ; read conversion result value
    mov  adrl_buffer,a       ; save result to user defined register
:
    jmp  start_conversion    ; start next A/D conversion

```

Example: using the interrupt method to detect the end of conversion

```

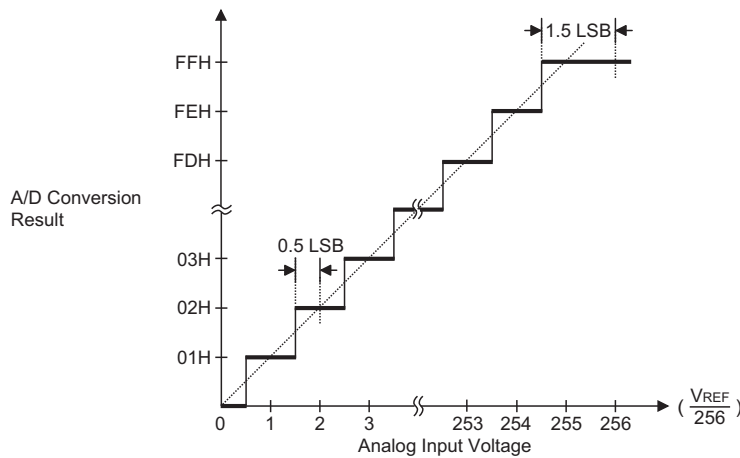
    clr  EADI                ; disable ADC interrupt
    mov  a,00000001B
    mov  ACSR,a              ; setup the ACSR register to select fSYS/8 as
                                ; the A/D clock

    mov  a,00000010B        ; setup ADCR register to configure Port PB0~PB3
                                ; as A/D inputs
    mov  ADCR,a              ; and select AN0 to be connected to the A/D
:
                                ; As the Port B channel bits have changed the
                                ; following START signal (0→1→0) must be issued
                                ; within 10 instruction cycles
:
Start_conversion:
    clr  START
    set  START                ; reset A/D
    clr  START                ; start A/D
    clr  ADF                  ; clear ADC interrupt request flag
    set  EADI                 ; enable ADC interrupt
    set  EMI                  ; enable global interrupt
:
:
:

```

```

; ADC interrupt service routine
ADC_ISR:
    mov  acc_stack,a           ; save ACC to user defined memory
    mov  a,STATUS
    mov  status_stack,a       ; save STATUS to user defined memory
    :
    :
    mov  a,ADR                 ; read conversion result value
    mov  adrl_buffer,a        ; save result to user defined register
    :
    :
EXIT_INT_ISR:
    mov  a,status_stack
    mov  STATUS,a             ; restore STATUS from user defined memory
    mov  a,acc_stack          ; restore ACC from user defined memory
    reti
    
```



Ideal A/D Transfer Function

A/D Transfer Function

As the device contain a 8-bit A/D converter, its full-scale converted digitised value is equal to FFH. Since the full-scale analog input value is equal to the V_{DD} voltage, this gives a single bit analog input value of V_{DD}/256. The diagram show the ideal transfer function between the analog input value and the digitised output value for the A/D converter.

Note that to reduce the quantisation error, a 0.5 LSB offset is added to the A/D Converter input. Except for the digitised zero value, the subsequent digitised values will change at a point 0.5 LSB below where they would change without the offset, and the last full scale digitised value will change at a point 1.5 LSB below the V_{DD} level.

SPI Serial Interface

The device includes two SPI Serial Interfaces. The SPI interface is a full duplex serial data link, originally designed by Motorola, which allows multiple devices connected to the same SPI bus to communicate with each other. The devices communicate using a master/slave technique where only the single master device can initi-

ate a data transfer. A simple four line signal bus is used for all communication.

SPI Interface Communication

Four lines are used for SPI communication known as SDI - Serial Data Input, SDO - Serial Data Output, SCK - Serial Clock and SCS - Slave Select. Note that the condition of the Slave Select line is conditioned by the CSEN bit in the SBCR control register. If the CSEN bit is high then the SCS line is active while if the bit is low then the SCS line will be in a floating condition. The following timing diagram depicts the basic timing protocol of the SPI bus.

SPI Registers

There are two registers associated with the SPI Interface. These are the SBCR register which is the control register and the SBDR which is the data register. The SBCR register is used to setup the required setup parameters for the SPI bus and also used to store associated operating flags, while the SBDR register is used for data storage.

After Power on, the contents of the SBDR register will be in an unknown condition while the SBCR register will default to the condition below:

CKS	M1	M0	SBEN	MLS	CSEN	WCOL	TRF
0	1	1	0	0	0	0	0

Note that data written to the SBDR register will only be written to the TXRX buffer, whereas data read from the SBDR register will actual be read from the register.

SPI Bus Enable/Disable

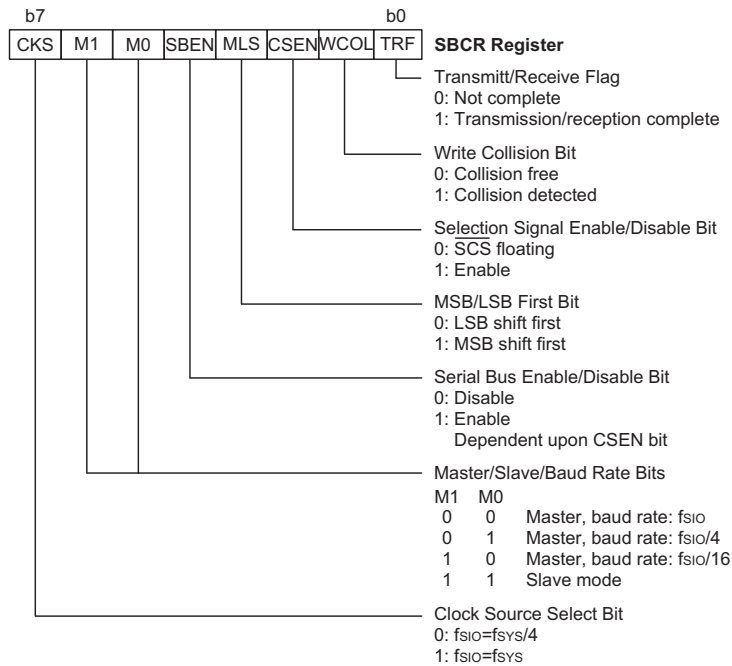
To enable the SPI bus, CSEN should be set high, then SCK, SDI, SDO and SCS lines should all be zero, then wait for data to be written to the SBDR (TXRX buffer) register. For the Master Mode, after data has been written to the SBDR (TXRX buffer) register then transmission or reception will start automatically. When all the data has been transferred the TRF bit should be set. For the Slave Mode, when clock pulses are received on SCK, data in the TXRX buffer will be shifted out or data on SDI will be shifted in.

To Disable the SPI bus SCK, SDI, SDO, \overline{SCS} should be floating.

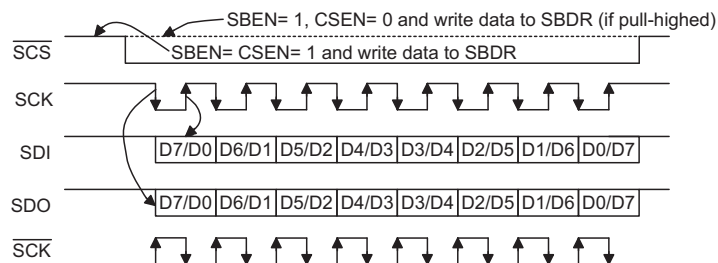
SPI Operation

All communication is carried out using the 4-line interface for both Master or Slave Mode. The timing diagram shows the basic operation of the bus.

The CSEN bit in the SBCR register controls the overall function of the SPI interface. Setting this bit high, will enable the SPI interface by allowing the SCS line to be active, which can then be used to control the SPI interface. If the CSEN bit is low, the SPI interface will be disabled and the SCS line will be in a floating condition and can therefore not be used for control of the SPI interface. The SBEN bit in the SBCR register must also be high which will place the SDI line in a floating condition and the SDO line high. If in the Master Mode the SCK line will be either high or low depending upon the clock polarity configuration option. If in the Slave Mode the SCK line will be in a floating condition. If SBEN is low then the bus will be disabled and SCS, SDI, SDO and SCK will all be in a floating condition.



SPI Interface Control Register



In the Master Mode, the Master will always generate the clock signal. The clock and data transmission will be initiated after data has been written to the SBDR register. In the Slave Mode, the clock signal will be received from an external master device for both data transmission or reception. The following sequences show the order to be followed for data transfer in both Master and Slave Mode:

- Master Mode

- Step 1. Select the clock source using the CKS bit in the SBCR control register
- Step 2. Setup the M0 and M1 bits in the SBCR control register to select the Master Mode and the required Baud rate. Values of 00, 01 or 10 can be selected.
- Step 3. Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Slave device.
- Step 4. Setup the SBEN bit in the SBCR control register to enable the SPI interface.
- Step 5. For write operations: write the data to the SBDR register, which will actually place the data into the TXRX buffer. Then use the SCK and SCS lines to output the data.
Goto to step 6. For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
- Step 6. Check the WCOL bit, if set high then a collision error has occurred so return to step5. If equal to zero then go to the following step.
- Step 7. Check the TRF bit or wait for an SBI serial bus interrupt.
- Step 8. Read data from the SBDR register.
- Step 9. Clear TRF.
- Step10. Goto step 5.

- Slave Mode:

- Step 1. The CKS bit has a don't care value in the slave mode.
- Step 2. Setup the M0 and M1 bits to 00 to select the Slave Mode. The CKS bit is don't care.
- Step 3. Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Master device.
- Step 4. Setup the SBEN bit in the SBCR control register to enable the SPI interface.
- Step 5. For write operations: write data to the SBCR register, which will actually place the data into the TXRX register, then wait for the master clock and SCS signal. After this goto step 6.

For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.

- Step 6. Check the WCOL bit, if set high then a collision error has occurred so return to step5. If equal to zero then go to the following step.
- Step 7. Check the TRF bit or wait for an SBI serial bus interrupt.
- Step 8. Read data from the SBDR register.
- Step 9. Clear TRF
- Step10. step 5

SPI Configuration Options

Several configuration options exist for the SPI Interface function which must be setup during device programming. One option is to enable the operation of the WCOL, write collision bit, in the SBCR register. Another option exists to select the clock polarity of the SCK line. A configuration option also exists to disable or enable the operation of the CSEN bit in the SBCR register. If the configuration option disables the CSEN bit then this bit cannot be used to affect overall control of the SPI Interface.

Error Detection

The WCOL bit in the SBCR register is provided to indicate errors during data transfer. The bit is set by the Serial Interface but must be cleared by the application program. This bit indicates a data collision has occurred which happens if a write to the SBDR register takes place during a data transfer operation and will prevent the write operation from continuing. The bit will be set high by the Serial Interface but has to be cleared by the user application program. The overall function of the WCOL bit can be disabled or enabled by a configuration option.

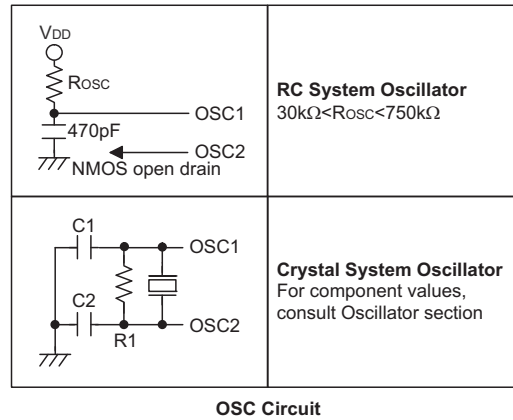
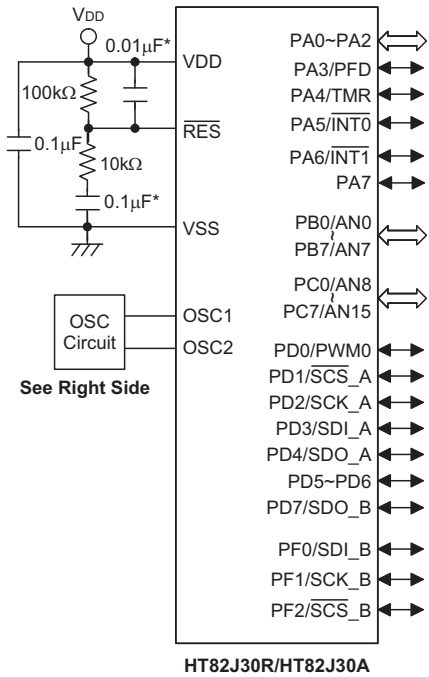
Programming Considerations

When the device is placed into the Power Down Mode note that data reception and transmission will continue. The TRF bit is used to generate an interrupt when the data has been transferred or received.

Configuration Options

Options
I/O Options
PA0~PA7: wake up enable or disable
PA0~PA7: pull-high enable or disable
PB0~PB7: pull-high enable or disable
PC0~PC7: pull-high enable or disable
PD0~PD7: pull-high enable or disable
PF0~PF7: pull-high enable or disable
PB2, PB3, PD4, PD7: CMOS or NMOS type
Oscillator Options
OSC type selection: RC or crystal
PWM Options
PWM: enable or disable
PFD Options
PFD: enable or disable
Watchdog Options
WDT: enable or disable
CLRWDT instructions: one or two instructions
WDT Clock Source: Fsys/4 or WDT oscillator
LVR Options
LVR function: enable or disable
SPI Options
SPI_A: enable or disable
SPI_A WCOL bit: enable or disable
SPI_A CSEN bit: enable or disable
SPI_A SCK clock polarity: rising edge or falling edge
SPI_B: enable or disable
SPI_B WCOL bit: enable or disable
SPI_B CSEN bit: enable or disable
SPI_B SCK clock polarity: rising edge or falling edge

Application Circuits



Instruction Set

Introduction

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

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

Instruction Timing

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

Moving and Transferring Data

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

Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and

subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

Logical and Rotate Operations

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

Branches and Control Transfer

Program branching takes the form of either jumps to specified locations using the JMP instruction or to a subroutine using the CALL instruction. They differ in the sense that in the case of a subroutine call, the program must return to the instruction immediately when the subroutine has been carried out. This is done by placing a return instruction RET in the subroutine which will cause the program to jump back to the address right after the CALL instruction. In the case of a JMP instruction, the program simply jumps to the desired location. There is no requirement to jump back to the original jumping off point as in the case of the CALL instruction. 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 the Data 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 the "HALT" instruction for Power-down operations and instructions to control the operation of the Watchdog Timer for reliable program operations under extreme electric or electromagnetic environments. For their relevant operations, refer to the functional related sections.

Instruction Set Summary

The following table depicts a summary of the instruction set categorised according to function and can be consulted as a basic instruction reference using the following listed conventions.

Table conventions:

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
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

Mnemonic	Description	Cycles	Flag Affected
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	↑ ^{Note}	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	↑ ^{Note}	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	↑ ^{Note}	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	↑ ^{Note}	C
Data Move			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	↑ ^{Note}	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	↑ ^{Note}	None
SET [m].i	Set bit of Data Memory	↑ ^{Note}	None
Branch			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	↑ ^{Note}	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	↑ ^{note}	None
SZ [m].i	Skip if bit i of Data Memory is zero	↑ ^{Note}	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	↑ ^{Note}	None
SIZ [m]	Skip if increment Data Memory is zero	↑ ^{Note}	None
SDZ [m]	Skip if decrement Data Memory is zero	↑ ^{Note}	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	↑ ^{Note}	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	↑ ^{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	↑ ^{Note}	None
SET [m]	Set Data Memory	↑ ^{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	↑ ^{Note}	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC	1	None
HALT	Enter power down mode	1	TO, PDF

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

Instruction Definition

ADC A,[m]	Add Data Memory to ACC with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.
Operation	$ACC \leftarrow ACC + [m] + C$
Affected flag(s)	OV, Z, AC, C
ADCM A,[m]	Add ACC to Data Memory with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.
Operation	$[m] \leftarrow ACC + [m] + C$
Affected flag(s)	OV, Z, AC, C
ADD A,[m]	Add Data Memory to ACC
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.
Operation	$ACC \leftarrow ACC + [m]$
Affected flag(s)	OV, Z, AC, C
ADD A,x	Add immediate data to ACC
Description	The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.
Operation	$ACC \leftarrow ACC + x$
Affected flag(s)	OV, Z, AC, C
ADDM A,[m]	Add ACC to Data Memory
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.
Operation	$[m] \leftarrow ACC + [m]$
Affected flag(s)	OV, Z, AC, C
AND A,[m]	Logical AND Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	$ACC \leftarrow ACC \text{ "AND" } [m]$
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	$ACC \leftarrow ACC \text{ "AND" } x$
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	$[m] \leftarrow ACC \text{ "AND" } [m]$
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 \leftarrow Program Counter + 1 Program Counter \leftarrow addr
Affected flag(s)	None
CLR [m]	Clear Data Memory
Description	Each bit of the specified Data Memory is cleared to 0.
Operation	[m] \leftarrow 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 \leftarrow 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 \leftarrow 0 PDF \leftarrow 0
Affected flag(s)	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repeatedly executing this instruction without alternately executing CLR WDT2 will have no effect.
Operation	WDT cleared TO \leftarrow 0 PDF \leftarrow 0
Affected flag(s)	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repeatedly executing this instruction without alternately executing CLR WDT1 will have no effect.
Operation	WDT cleared TO \leftarrow 0 PDF \leftarrow 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	$[m] \leftarrow ACC + 00H$ or $[m] \leftarrow ACC + 06H$ or $[m] \leftarrow ACC + 60H$ or $[m] \leftarrow ACC + 66H$
Affected flag(s)	C
DEC [m]	Decrement Data Memory
Description	Data in the specified Data Memory is decremented by 1.
Operation	$[m] \leftarrow [m] - 1$
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	$ACC \leftarrow [m] - 1$
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	$TO \leftarrow 0$ $PDF \leftarrow 1$
Affected flag(s)	TO, PDF

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

OR A,x	Logical OR immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC "OR" x
Affected flag(s)	Z
ORM A,[m]	Logical OR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC "OR" [m]
Affected flag(s)	Z
RET	Return from subroutine
Description	The Program Counter is restored from the stack. Program execution continues at the restored address.
Operation	Program Counter ← Stack
Affected flag(s)	None
RET A,x	Return from subroutine and load immediate data to ACC
Description	The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address.
Operation	Program Counter ← Stack ACC ← x
Affected flag(s)	None
RETI	Return from interrupt
Description	The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. 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	$[m].(i+1) \leftarrow [m].i; (i = 0\sim6)$ $[m].0 \leftarrow C$ $C \leftarrow [m].7$
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	$ACC.(i+1) \leftarrow [m].i; (i = 0\sim6)$ $ACC.0 \leftarrow C$ $C \leftarrow [m].7$
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	$[m].i \leftarrow [m].(i+1); (i = 0\sim6)$ $[m].7 \leftarrow [m].0$
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	$ACC.i \leftarrow [m].(i+1); (i = 0\sim6)$ $ACC.7 \leftarrow [m].0$
Affected flag(s)	None
RRC [m]	Rotate Data Memory right through Carry
Description	The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.
Operation	$[m].i \leftarrow [m].(i+1); (i = 0\sim6)$ $[m].7 \leftarrow C$ $C \leftarrow [m].0$
Affected flag(s)	C
RRCA [m]	Rotate Data Memory right through Carry with result in ACC
Description	Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation	$ACC.i \leftarrow [m].(i+1); (i = 0\sim6)$ $ACC.7 \leftarrow C$ $C \leftarrow [m].0$
Affected flag(s)	C

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

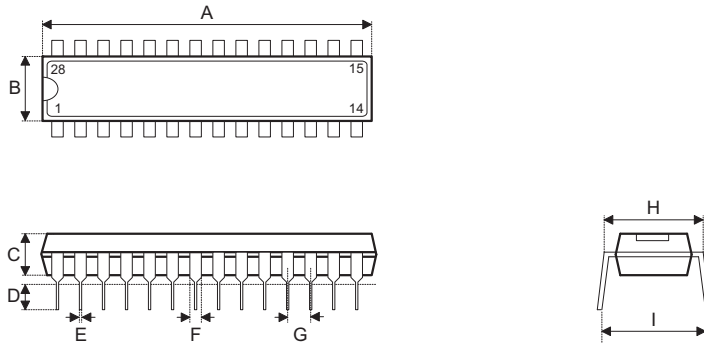
SIZ [m]	Skip if increment Data Memory is 0
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	$[m] \leftarrow [m] + 1$ 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	$ACC \leftarrow [m] + 1$ Skip if $ACC = 0$
Affected flag(s)	None
SNZ [m].i	Skip if bit i of Data Memory is not 0
Description	If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.
Operation	Skip if $[m].i \neq 0$
Affected flag(s)	None
SUB A,[m]	Subtract Data Memory from ACC
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$ACC \leftarrow ACC - [m]$
Affected flag(s)	OV, Z, AC, C
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$[m] \leftarrow ACC - [m]$
Affected flag(s)	OV, Z, AC, C
SUB A,x	Subtract immediate data from ACC
Description	The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$ACC \leftarrow ACC - x$
Affected flag(s)	OV, Z, AC, C

SWAP [m]	Swap nibbles of Data Memory
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged.
Operation	$[m].3 \sim [m].0 \leftrightarrow [m].7 \sim [m].4$
Affected flag(s)	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	$ACC.3 \sim ACC.0 \leftarrow [m].7 \sim [m].4$ $ACC.7 \sim ACC.4 \leftarrow [m].3 \sim [m].0$
Affected flag(s)	None
SZ [m]	Skip if Data Memory is 0
Description	If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	Skip if $[m] = 0$
Affected flag(s)	None
SZA [m]	Skip if Data Memory is 0 with data movement to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	$ACC \leftarrow [m]$ 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	$[m] \leftarrow$ program code (low byte) $TBLH \leftarrow$ 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	$[m] \leftarrow$ program code (low byte) $TBLH \leftarrow$ 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

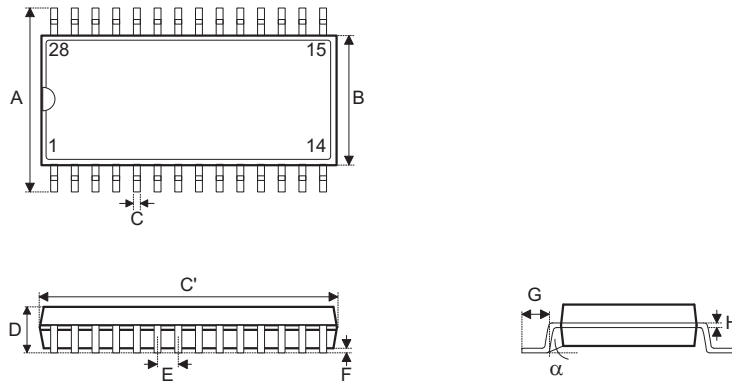
Package Information

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	—	—	375

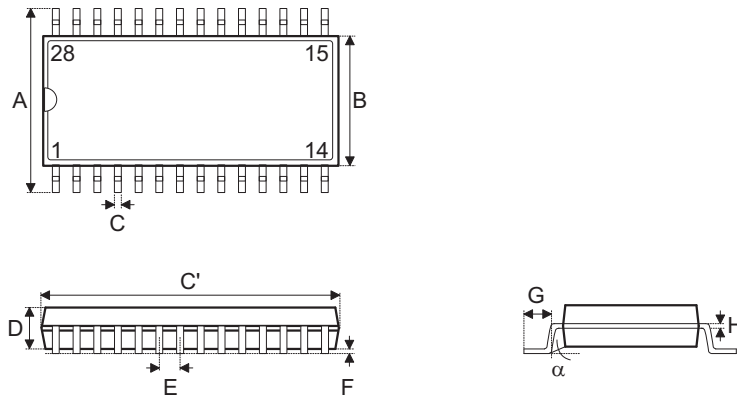
28-pin SOP (300mil) Outline Dimensions



• MS-013

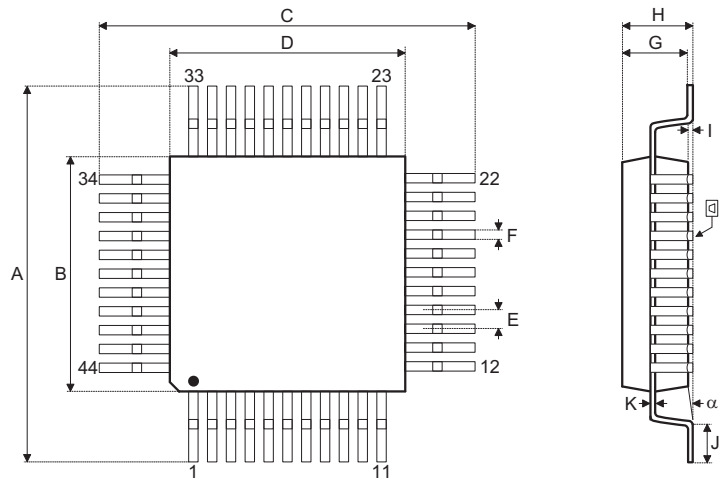
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	393	—	419
B	256	—	300
C	12	—	20
C'	697	—	713
D	—	—	104
E	—	50	—
F	4	—	12
G	16	—	50
H	8	—	13
α	0°	—	8°

28-pin SSOP (150mil) Outline Dimensions

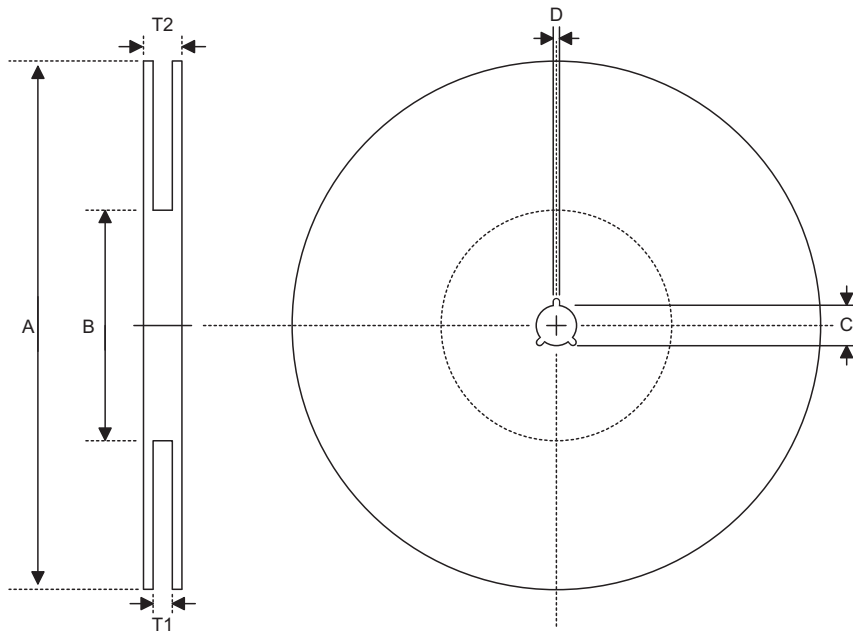


Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	228	—	244
B	150	—	157
C	8	—	12
C'	386	—	394
D	54	—	60
E	—	25	—
F	4	—	10
G	22	—	28
H	7	—	10
α	0°	—	8°

44-pin QFP (10mm×10mm) Outline Dimensions



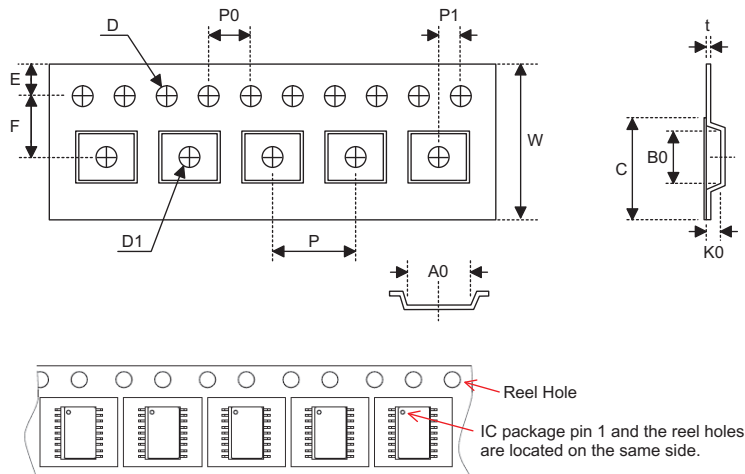
Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	13.00	—	13.40
B	9.90	—	10.10
C	13.00	—	13.40
D	9.90	—	10.10
E	—	0.80	—
F	—	0.30	—
G	1.90	—	2.20
H	—	—	2.70
I	0.25	—	0.50
J	0.73	—	0.93
K	0.10	—	0.20
L	—	0.10	—
α	0°	—	7°

Product Tape and Reel Specifications
Reel Dimensions

SOP 28W (300mil)

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330.0±1.0
B	Reel Inner Diameter	100.0±1.5
C	Spindle Hole Diameter	13.0 ^{+0.5/-0.2}
D	Key Slit Width	2.0±0.5
T1	Space Between Flange	24.8 ^{+0.3/-0.2}
T2	Reel Thickness	30.2±0.2

SSOP 28S (150mil)

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330.0±1.0
B	Reel Inner Diameter	100.0±1.5
C	Spindle Hole Diameter	13.0 ^{+0.5/-0.2}
D	Key Slit Width	2.0±0.5
T1	Space Between Flange	16.8 ^{+0.3/-0.2}
T2	Reel Thickness	22.2±0.2

Carrier Tape Dimensions

SOP 28W (300mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	24.0±0.3
P	Cavity Pitch	12.0±0.1
E	Perforation Position	1.75±0.10
F	Cavity to Perforation (Width Direction)	11.5±0.1
D	Perforation Diameter	1.5 ^{+0.1/-0.0}
D1	Cavity Hole Diameter	1.50 ^{+0.25/-0.00}
P0	Perforation Pitch	4.0±0.1
P1	Cavity to Perforation (Length Direction)	2.0±0.1
A0	Cavity Length	10.85±0.10
B0	Cavity Width	18.34±0.10
K0	Cavity Depth	2.97±0.10
t	Carrier Tape Thickness	0.35±0.01
C	Cover Tape Width	21.3±0.1

SSOP 28S (150mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	16.0±0.3
P	Cavity Pitch	8.0±0.1
E	Perforation Position	1.75±0.1
F	Cavity to Perforation (Width Direction)	7.5±0.1
D	Perforation Diameter	1.55 ^{+0.10/-0.00}
D1	Cavity Hole Diameter	1.50 ^{+0.25/-0.00}
P0	Perforation Pitch	4.0±0.1
P1	Cavity to Perforation (Length Direction)	2.0±0.1
A0	Cavity Length	6.5±0.1
B0	Cavity Width	10.3±0.1
K0	Cavity Depth	2.1±0.1
t	Carrier Tape Thickness	0.30±0.05
C	Cover Tape Width	13.3±0.1

Holtek Semiconductor Inc. (Headquarters)

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

Holtek Semiconductor Inc. (Taipei Sales Office)

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

Holtek Semiconductor Inc. (Shanghai Sales Office)

G Room, 3 Floor, No.1 Building, No.2016 Yi-Shan Road, Minhang District, Shanghai, China 201103
Tel: 86-21-5422-4590
Fax: 86-21-5422-4705
<http://www.holtek.com.cn>

Holtek Semiconductor Inc. (Shenzhen Sales Office)

5F, Unit A, Productivity Building, Gaoxin M 2nd, Middle Zone Of High-Tech Industrial Park, ShenZhen, China 518057
Tel: 86-755-8616-9908, 86-755-8616-9308
Fax: 86-755-8616-9722

Holtek Semiconductor Inc. (Beijing Sales Office)

Suite 1721, Jinyu Tower, A129 West Xuan Wu Men Street, Xicheng District, Beijing, China 100031
Tel: 86-10-6641-0030, 86-10-6641-7751, 86-10-6641-7752
Fax: 86-10-6641-0125

Holtek Semiconductor (USA), Inc. (North America Sales Office)

46729 Fremont Blvd., Fremont, CA 94538, USA
Tel: 1-510-252-9880
Fax: 1-510-252-9885
<http://www.holtek.com>

Copyright © 2009 by HOLTEK SEMICONDUCTOR INC.

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