Cost-Effective A/D 8-Bit OTP MCU

HT46R004

Revision: V1.60 Date: December 22, 2021

www.holtek.com

Table of Contents

Features

CPU Features

- Operating voltage: f_{SYS}=8MHz: 2.3V~5.5V
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD}=5V
- Power down and wake-up functions to reduce power consumption
- Two oscillators:
  Internal high speed RC -- HIRC
  Internal low speed RC -- LIRC
- Fully integrated internal 8MHz oscillator requires no external components
- All instructions executed in one or two instruction cycles
- Table read instruction
- 61 powerful instructions
- 4-level subroutine nesting
- Bit manipulation instruction

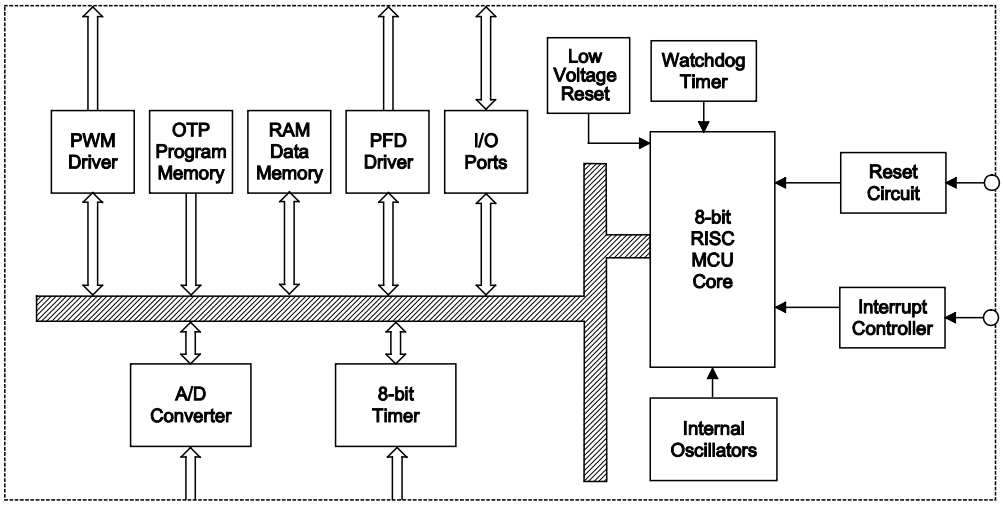
Peripheral Features

- Program Memory: 2K×14
- RAM Data Memory: 96×8
- Watchdog Timer function
- Up to 18 bidirectional I/O lines
- 8 channel 12-bit ADC
- 1 channel 8-bit PWM
- External interrupt pin shared with I/O pin
- One 8-bit programmable Timer/Event Counter with overflow interrupt and prescaler
- Low voltage reset function
- Package types: 16-pin NSOP and 20-pin SOP
- Programmable Frequency Divider – PFD

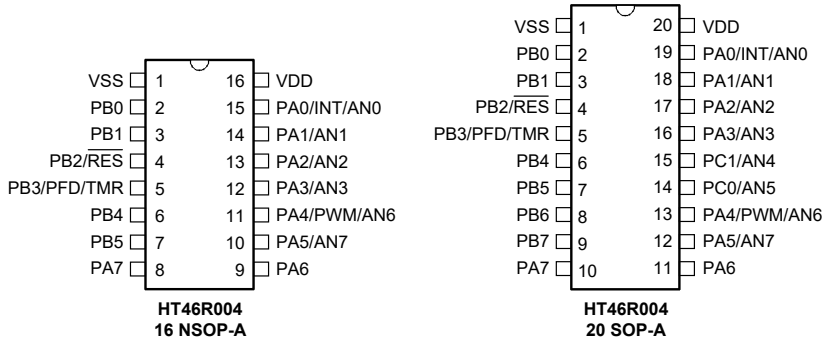
General Description

The device is an 8-bit high performance RISC architecture microcontroller device specifically designed for a wide range of applications. The advantages of low power consumption, I/O flexibility, timer functions, HALT and wake-up functions, watchdog timer, as well as low cost, enhance the versatility of the device to suit for a wide range of the I/O and A/D control application possibilities such as industrial control, consumer products and subsystem controllers, etc.

Block Diagram



Pin Assignment



Note: The PB2/$\overline{\text{RES}}$ pin is fixed to PB2 function and the $\overline{\text{RES}}$ pin is not open to user.

Pin Description

Pin Name	Function	OPT	I/T	O/T	Description
PA0/INT/AN0	PA0	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT	INTC0 CTRL0	ST	—	External interrupt input
	AN0	ADCR1	AI	—	Analog input channel 0
PA1/AN1	PA1	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	AN1	ADCR1	AI	—	Analog input channel 1
PA2/AN2	PA2	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	AN2	ADCR1	AI	—	Analog input channel 2
PA3/AN3	PA3	PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	AN3	ADCR1	AI	—	Analog input channel 3
PA4/PWM/AN6	PA4	PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PWM	CTRL0	—	CMOS	PWM output
	AN6	ADCR1	AI	—	Analog input channel 6
PA5/AN7	PA5	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	AN7	ADCR1	AI	—	Analog input channel 7
PA6~PA7	PA6	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	PA7	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
PB0~PB1	PB0~PB1	PBPU	ST	CMOS	General purpose I/O
PB2/$\overline{\text{RES}}$	PB2	—	ST	NMOS	General purpose I/O
	$\overline{\text{RES}}$	—	ST	—	Reset input (not open)
PB3/PFD/TMR	PB3	PBPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PFD	CTRL0	ST	—	PFD output
	TMR	TMR0C	ST	—	Timer/Event counter 0 input
PB4~PB7	PB4~PB7	PBPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PC0/AN5	PC0	PCPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	AN5	ADCR1	AI	—	Analog input channel 5
PC1/AN4	PC1	PCPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	AN4	ADCR1	AI	—	Analog input channel 4
VDD	VDD	—	PWR	—	Power supply
VSS	VSS	—	PWR	—	Ground

Note: OPT: Optional by register option
I/T: Input type
O/T: Output type
ST: Schmitt Trigger input
CMOS: CMOS output
NMOS: NMOS output
PWR: Power
AI: Analog Input

Absolute Maximum Ratings

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

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

D.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage (HIRC)	—	f_{SYS}=8MHz	2.3	—	5.5	V
I_{DD1}	Operating Current(HIRC on)	3V	No load, f_{SYS}=8MHz, ADC off	—	1.2	1.8	mA
		5V		—	2.4	3.6	mA
I_{STB1}	Standby Current (LIRC on)	3V	No load, system HALT	—	—	5	μA
		5V		—	—	10	μA
I_{STB2}	Standby Current (LIRC off)	3V	No load, system HALT	—	—	1	μA
		5V		—	—	2	μA
V_{IL1}	Input Low Voltage for I/O Ports, TMR and INT pin	5V	—	0	—	1.5	V
		—		0	—	$0.2V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports, TMR and INT pin	5V	—	3.5	—	5	V
		—		$0.8V_{DD}$	—	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 Voltage	—	LVR Enable, 2.1V	2.0	2.1	2.2	V
I_{OH}	I/O source current	3V	V_{OH}=0.9V_{DD}	-4	-8	—	mA
		5V		-8	-16	—	mA
I_{OL1}	I/O sink current	3V	V_{OL}=0.1V_{DD}	8	16	—	mA
		5V		16	32	—	mA
I_{OL2}	PB2 sink current	5V	V_{OL}=0.1V_{DD}	2	3	—	mA
R_{PH}	Pull-high Resistance for I/O Ports	3V	—	20	60	100	kΩ
		5V	—	10	30	50	kΩ

A.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	Operating Clock	—	2.3V~5.5V	8	8	8	MHz
f_{HIRC}	System Clock (HIRC)	3/5V	—	-2%	8	+2%	MHz
		3/5V	Ta=0°~70°C	-5%	8	+5%	MHz
		3.0~5.5V	Ta=0°~70°C	-8%	8	+8%	MHz
		3.0~5.5V	Ta=-40°~85°C	-12%	8	+12%	MHz
f_{TIMER}	Timer I/P Frequency (TMR)	3.3~5.5V	—	0	—	8	MHz
t_{WDTOSC}	Watchdog oscillator period	3V	—	45	90	180	μs
		5V	—	32	65	130	μs
t_{RES}	External reset low pulse width	—	—	1	—	—	μs
t_{RESE}	External reset low pulse width (with filter)	—	—	—	150	—	ns
t_{SST}	System start-up timer period	—	wake-up from halt	—	16	—	t_{SYS}
t_{LVR}	Low Voltage Width to Reset	—	—	0.25	1	2	ms
t_{RSTD}	System Reset Delay Time (All Reset)	—	—	25	50	100	ms

Note: 1. $t_{SYS}=1/f_{SYS}$

2. To maintain the accuracy of the internal HIRC oscillator frequency, a 0.1μF decoupling capacitor should be connected between VDD and VSS and located as close to the device as possible.

A/D Converter Characteristics

Ta=25°C

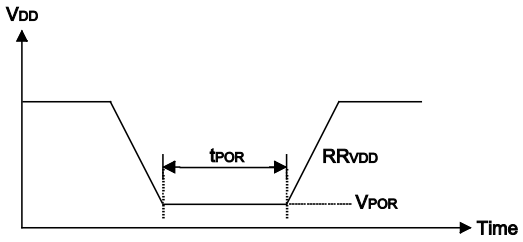
Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
AV_{DD}	Analog operating voltage	—	$V_{REF}=V_{DD}$	2.7	—	5.5	V
V_{AD}	AD Input Voltage	—	—	0	—	AV_{DD}/V_{REF}	V
DNL	A/D Differential Non-linearity	2.7V	$V_{REF}=V_{DD}=AV_{DD}$ $t_{AD}=0.5μs$	-2	—	+2	LSB
		3V					
		5V					
INL	A/D Integral non-linearity	2.7V	$V_{REF}=V_{DD}=AV_{DD}$ $t_{AD}=0.5μs$	-4	—	+4	LSB
		3V					
		5V					
I_{ADC}	Additional Power Consumption if A/D Converter is Used	3V	No load ($t_{AD}=0.5μs$)	—	0.5	—	mA
		5V		—	0.6	—	mA
t_{AD}	A/D Converter Clock Period	2.7V~5.5V	—	0.5	—	10	μs
t_{ADC}	A/D Conversion Time (Include Sample and Hold Time)	2.7V~5.5V	12-bit ADC	—	16	—	t_{AD}
t_{ON2ST}	A/D Converter On-to-Start Time	2.7V~5.5V	—	2	—	—	μs

Note: ADC conversion time (t_{AD})=n (bits ADC) + 4 (sampling time), the conversion for each bit needs one ADC clock (t_{AD}).

Power-on Reset Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{POR}	VDD Start Voltage to Ensure Power-on Reset	—	—	—	—	100	mV
RRV_{DD}	VDD Raising Rate to Ensure Power-on Reset	—	—	0.035	—	—	V/ms
t_{POR}	Minimum Time for VDD Stays at V_{POR} to Ensure Power-on Reset	—	—	1	—	—	ms

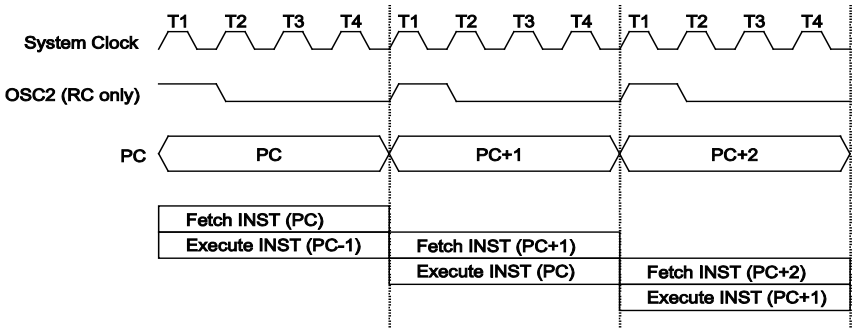


System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to the internal system architecture. The device takes 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 system with maximum reliability and flexibility.

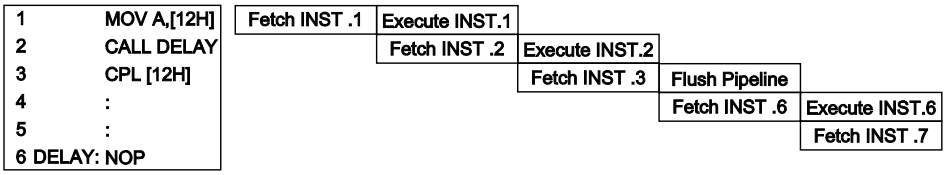
Clocking and Pipelining

The system clock derived from HIRC 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.



System Clocking and Pipelining

For instructions involving branches, such as jump or call instructions, two instruction cycles are required to complete instruction execution. An extra cycle is required as the program takes one cycle to firstly 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.



Instruction Fetching

Program Counter – PC

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.

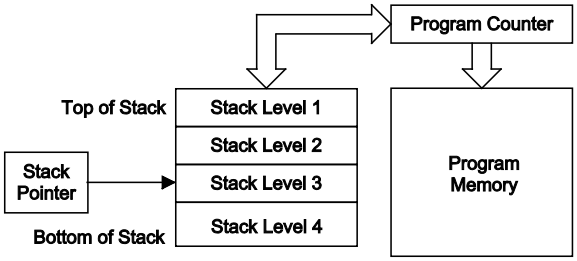
When executing instructions requiring jumping to non-consecutive addresses such as a jump instruction, a subroutine call, interrupt or reset, etc, the microcontroller manages program control by loading the required address into the Program Counter. For conditional skip instructions, once the condition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

Program Counter	
High Byte of Program	Low Byte of Program
PC10~PC8	PCL7~PCL0

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 in the present page of memory, which have 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.

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organized into 4 levels and 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, 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 changes, the status register will be correspondingly updated to reflect these changes. The ALU supports the following functions:

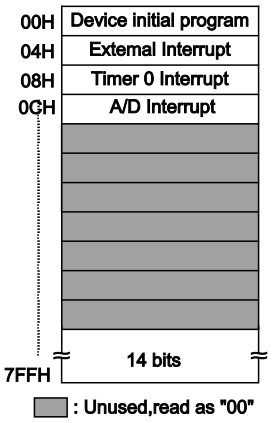
- Arithmetic operations: ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA
- Logic operations: AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA
- Rotation RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC
- Increment and Decrement INCA, INC, DECA, DEC
- Branch decision, JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET,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

Program Memory

The Program Memory is the location where the user code or program is stored. The device is supplied with One-Time Programmable, OTP, memory where users can program their application code into the device. By using the appropriate programming tools, OTP device offers users the flexibility to freely develop their applications which may be useful during debug or for products requiring frequent upgrades or program changes.

Structure

The Program Memory has a capacity of 2K×14 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries information. Table data, which can be set in any location within the Program Memory, is addressed by separate table pointer registers.



Program Memory Structure

Special Vectors

Within the Program Memory, certain locations are reserved for the reset and interrupts.

- Reset Vector
  This vector is reserved for use by the device reset for program initialization. After a device reset is initiated, the program will jump to this location and begin execution.
- External interrupt vector
  This vector is used by the external interrupt. If the external interrupt pin on the device receives an edge transition, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full. The external interrupt active edge transition type, whether high to low, low to high or both is specified in the CTRL0 register.
- Timer/Event counter 0 interrupt vector
  This internal vector is used by the Timer/Event Counter 0. If a Timer/Event Counter overflow occurs, the program will jump to its respective location and begin execution if the associated Timer/Event Counter 0 interrupt is enabled and the stack is not full.
- A/D interrupt vector
  This internal vector is used by the A/D converter. If A/D conversion complete, the program will jump to this location and begin execution if the A/D 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 set by placing the address of the look up data to be retrieved in the table pointer register, TBLP. This register defines the total address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the Program Memory using the "TABRDC [m]" or "TABRDL [m]" instructions, respectively. When the instruction is 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 accompanying diagram illustrates the addressing data flow of the look-up table.

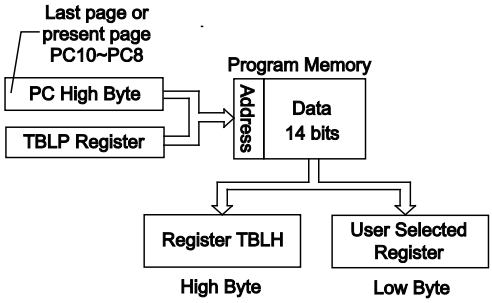


Table Program Example

The accompanying example shows how the table pointer and table data is defined and retrieved from the device. 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 "0700H" which refers to the start address of the last page within the 2K Program Memory of the microcontroller.

The table pointer is set 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 "0706H" 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.

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.

Instruction	Table Location Bits										
	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
TABRDC[m]	@10	@9	@8	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL[m]	1	1	1	@7	@6	@5	@4	@3	@2	@1	@0

Table Location

Note: PC10~PC8: Current program Counter bits

@7~@0: Table Pointer TBLP bits

Table Read Program Example

```
tempreg1 db ?          ; temporary register #1
tempreg2 db ?          ; temporary register #2
        :
        :
mov a,06h              ; initialize table pointer - note that this address is referenced
mov tblp, a            ; to the last page or present page
        :
        :
tabrdl tempreg1        ; transfers value in table referenced by table pointer to tempreg1
                       ; data at prog. memory address "0706H" 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 "0705H" 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 0700h              ; sets initial address of last page
dc 00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
        :
        :
```

RAM Data Memory

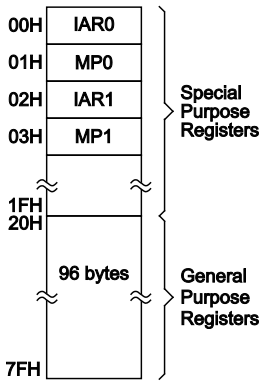
The Data Memory is a volatile area of 8-bit wide RAM internal memory and is the location where temporary information is stored.

Structure

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.

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. The start address of the device Data Memory is the address "00H".

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 reading and writing 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.



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 via the memory pointer registers.

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 writeable 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	IAR0
01H	MP0
02H	IAR1
03H	MP1
04H	EXTRESB
05H	ACC
06H	PCL
07H	TBLP
08H	TBLH
09H	WDTS
0AH	STATUS
0BH	INTC0
0CH	TMR0
0DH	TMR0C
0EH	ADRL
0FH	ADRH
10H	PA
11H	PAC
12H	PAPU
13H	PAWU
14H	PB
15H	PBC
16H	PBPU
17H	PC
18H	PCC
19H	PCPU
1AH	CTRL0
1BH	ACSR
1CH	WDTLVRC
1DH	ADCR0
1EH	ADCR1
1FH	PWM

Special Purpose Data Memory

Special Function Registers

To ensure successful operation of the microcontroller, certain internal registers are implemented in the Data Memory area. These registers ensure correct operation of internal functions such as timer, interrupts, etc., as well as external functions such as I/O data control. The locations of these registers within the Data Memory begin at the address of "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 Registers – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, although having their locations in normal RAM register, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation is using these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 and IAR1 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointers, MP0 or MP1. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointers – MP0, MP1

Two Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to indirectly address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address which the microcontroller is directed to is the address specified by the related Memory Pointer. Note that for this device, the Memory Pointers, MP0 and MP1, are both 8-bit registers and used to access the Data Memory together with their corresponding indirect addressing registers IAR0 and IAR1.

The following example shows how to clear a section of four Data Memory locations already defined as locations adres1 to adres4.

Indirect Addressing Program Example

```
data . section  'data'
adres1   db ?
adres2   db ?
adres3   db ?
adres4   db ?
block    db ?
code. section  at 0 code
org 00h
start:
mov a,04h                ; set size of block
mov block,a
mov a,offset adres1      ; Accumulator loaded with first RAM address
mov mp0,a                ; set memory pointer with first RAM address
loop:
clr IAR0                 ; clear the data at address defined by MP0
inc mp0                  ; 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.

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.

In addition, on entering an interrupt sequence or executing a subroutine call, the status register will not be pushed onto the stack automatically. If the contents of the status registers are important and if the subroutine can corrupt the status register, precautions must be taken to correctly save it. Note that bits 0~3 of the STATUS register are both readable and writeable bits.

STATUS Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	TO	PDF	OV	Z	AC	C
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	x	x	x	x

"x": unknown

Bit 7~6 Unimplemented, read as "0"

Bit 5 **TO**: Watchdog Time-Out flag
0: After power up or executing the "CLR WDT" or "HALT" instruction
1: A watchdog time-out occurred.

Bit 4 **PDF**: Power down flag
0: After power up or executing the "CLR WDT" instruction
1: By executing the "HALT" instruction

Bit 3 **OV**: Overflow flag
0: No overflow
1: An operation results in a carry into the highest-order bit but not a carry out of the highest-order bit or vice versa.

Bit 2 **Z**: Zero flag
0: The result of an arithmetic or logical operation is not zero
1: The result of an arithmetic or logical operation is zero

Bit 1 **AC**: Auxiliary flag
0: No auxiliary carry
1: 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

Bit 0 **C**: Carry flag
0: No carry out
1: An operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation

C is also affected by a rotate through carry instruction.

System Control Registers – CTRL0

These registers are used to provide control internal functions such as the PFD function, the PWM function and external interrupt edge trigger type selection.

CTRL0 Register

Bit	7	6	5	4	3	2	1	0
Name	INTES1	INTES0	PWMSEL	—	PWMC	PFDC	—	—
R/W	R/W	R/W	R/W	—	R/W	R/W	—	—
POR	1	0	0	—	0	0	—	—

Bit 7~6 **INTES1, INTES0**: External interrupt edge type selection
00: Disable
01: Rising edge trigger
10: Falling edge trigger
11: Dual edge trigger

Bit 5 **PWMSEL**: PWM type selection
0: 6+2
1: 7+1

Bit 4 Unimplemented, read as "0"

Bit 3 **PWMC**: I/O or PWM selection
0: PA4
1: PWM

Bit 2 **PFDC**: I/O or PFD selection
0: PB3
1: PFD

Bit 1~0 Unimplemented, read as "0"

Oscillator

Various oscillator options offer the user a wide range of functions according to their various application requirements. The flexible features of the oscillator functions ensure that the best optimization can be achieved in terms of speed and power saving.

System Oscillator Overview

In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer function.

Type	Name	Freq.
Internal High Speed RC	HIRC	8MHz
Internal Low Speed RC	LIRC	12kHz

Oscillator Types

System Clock Configurations

There is one system oscillator implemented in the device, internal 8MHz RC, HIRC. Also there is an internal 12kHz RC oscillator LIRC used as the clock source for the WDT function. More details are described in the accompany sections.

Internal RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has the frequency of 8MHz .Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuit is used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimized. Note that this internal system clock option requires no external pins for its operation. Refer to the A.C. Characteristics for more frequency accuracy details.

Internal 12kHz Oscillator – LIRC

The LIRC is a fully self-contained free running on-chip RC oscillator with a typical frequency of 12kHz at 5V, requiring no external components for its implementation. When the device enters the Sleep Mode, the system clock will stop running but the LIRC oscillator continues to free-run and to keep the watchdog active. However, to preserve power in certain applications the LIRC can be disabled by disabling the WDT function, Timer/Event counter and PWM function in the halt mode.

Power Down Mode and Wake-up

Power Down Mode

All of the Holtek microcontrollers have the ability to enter a Power Down Mode, also known as the HALT Mode or Sleep Mode. When the device enters this mode, the normal operating current will be reduced to an extremely low standby current level. This occurs because when the device enters the Power Down Mode, the system oscillator is stopped which reduces the power consumption to extremely low levels. However, as the device maintains its present internal condition, they 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 MCUs must have their power supply constantly maintained to keep the device in a known condition.

Entering the Power Down Mode

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

- The system oscillator will stop running and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The WDT will be cleared and resume counting if the WDT clock source comes from LIRC oscillator.
- 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 Sleep Mode is to keep the current consumption of the MCU to as low a value as possible, perhaps only in the order of several micro-amps, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimized.

Special attention must be made to the I/O pins on the device. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. Care must also be taken with the loads, which are connected to I/O pins, which are set 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.

Wake-up

After the system enters the Sleep 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.

Pins PA0~PA7 can be set via the PAWU register to permit a negative transition on the pin to wake-up the system. When a PA0~PA7 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 high before entering the SLEEP Mode, the wake-up function of the related interrupt will be ignored.

No matter what the source of the wake-up event is, once a wake-up event occurs, there will be a time delay before normal program execution resumes. Consult the table for the related time

Wake-up Source	Oscillator Type
	HIRC, LIRC
External $\overline{RES}$	$t_{RSTD} + t_{SST}$
PA Port	t_{SST}
Interrupt	
WDT Overflow	

Note: 1. t_{RSTD} (reset delay time), t_{SYS} (system clock)
2. t_{RSTD} is power-on delay, typical time=50ms
3. $t_{SST}=16t_{SYS}$

Wake-up Delay Time

Watchdog Timer

The Watchdog Timer, also known as the WDT, is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise.

Watchdog Timer Clock Source

The Watchdog Timer clock source is provided by the LIRC, the system clock f_{SYS} or f_{SYS}/4which is sourced from the HIRC oscillator. The Watchdog Timer source clock is then subdivided by a ratio of 2^8 to 2^{15} to give longer timeouts, the actual value being chosen using the WS2~WS0 bits in the WDTS register. The LIRC internal oscillator has an approximate period frequency of 12kHz at a supply voltage of 5V. However, it should be noted that this specified internal clock period can vary with V_{DD}, temperature and process variations. The Watchdog Timer can be enabled or disabled by using the WDTEN2~WDTEN0 bits in the internal WDTLVRC register.

Watchdog Timer Control Registers

WDTS Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	WS2	WS1	WS0
R/W	—	—	—	—	—	R/W	R/W	R/W
POR	—	—	—	—	—	1	1	1

Bit 7~3 Unimplemented, read as "0"

Bit 2~0 **WS2~WS0**: WDT Time-out period selection
000: $2^8/f_S$
001: $2^9/f_S$
010: $2^{10}/f_S$
011: $2^{11}/f_S$
100: $2^{12}/f_S$
101: $2^{13}/f_S$
110: $2^{14}/f_S$
111: $2^{15}/f_S$
These three bits determine the division ratio of the Watchdog Timer source clock, which in turn determines the timeout period.

WDTLVRC Register

Bit	7	6	5	4	3	2	1	0
Name	WDTCLS1	WDTCLS0	LVREN2	LVREN1	LVREN0	WDTEN2	WDTEN1	WDTEN0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **WDTCLS1~WDTCLS0**: WDT/Timer clock source
00: f_{LIRC}
01: f_{SYS}/4
10: f_{SYS}
11: f_{SYS}

Bit 5~3 Described in other section.

Bit 2~0 **WDTEN2~WDTEN0**: WDT enable control
000: Enable
101: Disable
Other values: MCU reset

Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog instruction. Note that if the Watchdog Timer function is not enabled, then any instruction related to the Watchdog Timer will result in no operation.

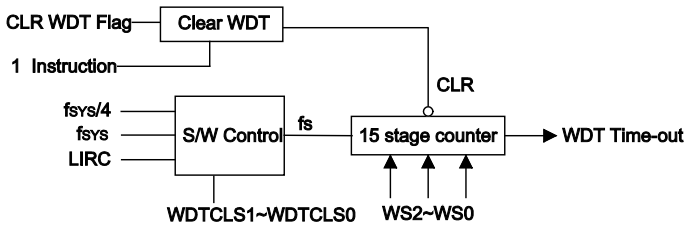
Setting the various Watchdog Timer options are controlled via the internal registers WDTLVRC and WDTS. Enabling the Watchdog Timer can be controlled by the WDTEN bits in the internal WDTLVRC register in the Data Memory.

The Watchdog Timer will be disabled if bits WDTEN2~WDTEN0 in the WDTLVRC register are written with the binary value 101B while the WDT Timer will be enabled if these bits are written with the binary value 000B. If these bits are written with the other values except 000 and 101, the MCU will be reset.

The Watchdog Timer clock can emanate from three different sources, selected by the WDTCLS1~WDTCLS0 bits in the WDTLVRC register. These sources are f_{SYS}, $f_{SYS}/4$ or LIRC. It is important to note that when the system enters the Sleep Mode the instruction clock is stopped, therefore if it has selected f_{SYS} or $f_{SYS}/4$ as the Watchdog Timer clock source, the Watchdog Timer will stop. For systems that operate in noisy environments, it's recommended to use the LIRC as the clock source. The division ratio of the prescaler is determined by bits 0, 1 and 2 of the WDTS register, known as WS0, WS1 and WS2. If the Watchdog Timer internal clock source is selected and with the WS0, WS1 and WS2 bits of the WDTS register all set high, the prescaler division ratio will be 1:32768, which will give a maximum time-out period.

Under normal program operation, a Watchdog Timer time-out will initialize a device reset and set the status bit TO. However, if the system is in the Sleep Mode, when a Watchdog Timer time-out occurs, the device will be woken up, 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 Watchdog Timer. The first is an external hardware reset, which means a low level on the external reset pin, the second is using the Clear Watchdog Timer software instructions and the third is via a "HALT" instruction.

There is only one method of using software instruction to clear the Watchdog Timer. That is to use the "CLR WDT" instruction to clear the WDT.



Watchdog Timer

Reset and Initialization

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

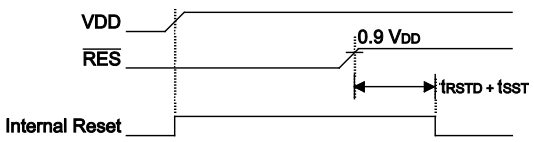
In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is running. One example of this is where after power has been applied and the microcontroller is already running, the $\overline{RES}$ line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to deal 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 set.

Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{RES}$ 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:

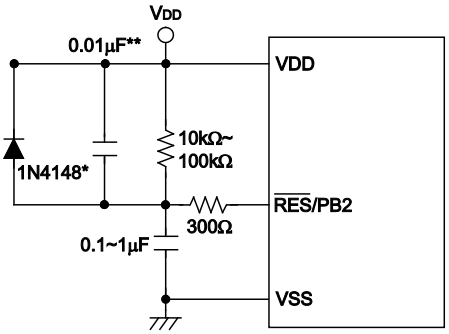
- Power-on Reset
  The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to 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 stabilize quickly at power-on, the internal reset function may be incapable of providing proper reset operation. For this reason it is recommended that an external RC network is connected to the $\overline{RES}$ pin, whose additional time delay will ensure that the $\overline{RES}$ pin remains low for an extended period to allow the power supply to stabilize. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{RES}$ 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.
  For most applications a resistor connected between VDD and the $\overline{RES}$ pin and a capacitor connected between VSS and the $\overline{RES}$ pin will provide a suitable external reset circuit. Any wiring connected to the $\overline{RES}$ pin should be kept as short as possible to minimize any stray noise interference.



Note: t_{RSTD} is power-on delay, typical time=50ms

Power-On Reset Timing Chart

For applications that operate within an environment where more noise is present the reset circuit shown is recommended.



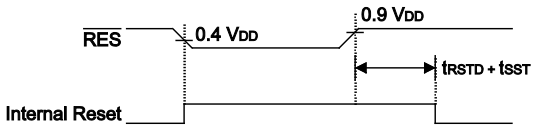
Note: "*" It is recommended that this component is added for added ESD protection.
"**" It is recommended that this component is added in environments where power line noise is significant.

External $\overline{\text{RES}}$ Circuit

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

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

  As the reset pin is shared with PB.2, the reset function must be selected by the RESBEN2~RESBEN0 bits in the EXTRESB register. This type of reset occurs when the microcontroller is already running and the $\overline{\text{RE}}$ pin is forcefully pulled low by software control using the register. In this case of other reset, the Program Counter will reset to zero and program execution initiated from this point.



Note: t_{RSTD} is power-on delay, typical time=50ms

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

- **EXTRESB Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	RESBEN2	RESBEN1	RESBEN0
R/W	—	—	—	—	—	R/W	R/W	R/W
POR	—	—	—	—	—	0	0	0

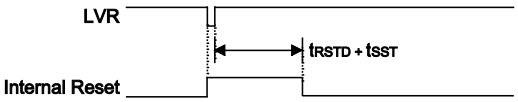
Bit 7~3 Unimplemented, read as "0"

Bit 2~0 **RESBEN2~RESBEN0**: PB2/$\overline{\text{RES}}$ selection
000: PB2
101: PB2 ($\overline{\text{RES}}$ pin not open)
Other values: MCU reset
Note: The PB2/$\overline{\text{RES}}$ pin is fixed to PB2 function and the $\overline{\text{RES}}$ pin is not open to user.

- Low Voltage Reset – LVR

  The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. This voltage is fixed at 2.1V (V_{LVR}). If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing a battery, the LVR will automatically reset the device internally.

  The LVR includes the following specifications: For a valid LVR signal, a low voltage, i.e., a voltage in the range between 0.9V~V_{LVR} must exist for greater than the value t_{LVR} specified in the A.C. characteristics. If the low voltage state does not exceed t_{LVR}, the LVR will ignore it and will not perform a reset function.



Note: t_{RSTD} is power-on delay, typical time=50ms

Low Voltage Reset Timing Chart

- **WDTLVRC Register**

Bit	7	6	5	4	3	2	1	0
Name	WDTCLS1	WDTCLS0	LVREN2	LVREN1	LVREN0	WDTEN2	WDTEN1	WDTEN0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

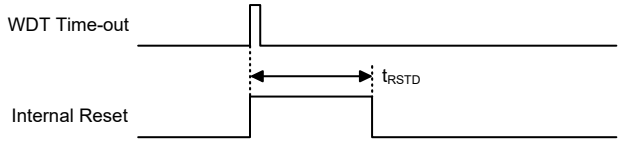
Bit 7~6 Described in other section.

Bit 5~3 **LVREN2~LVREN0**: LVR enable control
000: Enable
101: Disable
other values: MCU reset (reset will be active after 2~3 LIRC clock for debounce time)

Bit 2~0 Described in other section.

- Watchdog Time-out Reset during Normal Operation

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

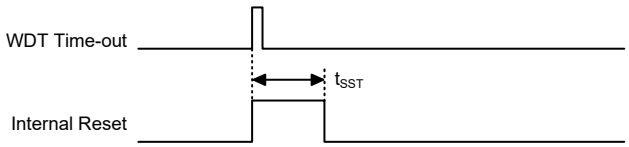


Note: t_{RSTD} is power-on delay, typical time=50ms

WDT Time-out Reset during Normal Operation Timing Chart

- Watchdog Time-out Reset during Sleep Mode

  The Watchdog time-out Reset during Sleep Mode 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.



Note: t_{SST} is 16 clock cycles for the system clock source is provided by HIRC.

WDT Time-out Reset during Sleep 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 Sleep Mode function or Watchdog Timer. The reset flags are shown in the table:

TO	PDF	RESET Conditions
0	0	Power-on reset
u	u	$\overline{RES}$ or LVR reset during NORMAL Mode operation
1	u	WDT time-out reset during NORMAL Mode operation
1	1	WDT time-out reset during Sleep Mode operation

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
Input/Output Ports	I/O ports will be set 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	Power-on Reset	$\overline{RES}$ Reset (Normal operation)	$\overline{RES}$ Reset (HALT)	WDT Time-out (Normal Operation)	WDT Time-out (HALT)*
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
MP0	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu	1uuu uuuu
MP1	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu	1uuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	--xx xxxx	--uu uuuu	--uu uuuu	--uu uuuu	--uu uuuu
WDTS	---- -111	---- -111	---- -111	---- -111	---- -uuu
STATUS	--00 xxxx	--uu uuuu	--01 uuuu	--1u uuuu	--11 uuuu
INTC0	-000 0000	-000 0000	-000 -000	-000 0000	--uu uuuu
TMR0	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAWU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PAPU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBPU	0000 0-00	0000 0-00	0000 0-00	0000 0-00	uuuu u-uu
PC	---- --11	---- --11	---- --11	---- --11	---- --uu
PCC	---- --11	---- --11	---- --11	---- --11	---- --uu
PCPU	---- --00	---- --00	---- --00	---- --00	---- --uu
CTRL0	100- 00--	100- 00--	100- 00--	100- 00--	uuu- uu--
WDTLVRC	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
PWM0	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADRL	xxxx ----	xxxx ----	xxxx ----	xxxx ----	uuuu ----
ADRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR0	01-- -000	01-- -000	01-- -000	01-- -000	uu-- -uuu
ADCR1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
ACSR	10-- -000	10-- -000	10-- -000	10-- -000	uu-- -uuu
EXTRESB	---- -000	---- -000	---- -000	---- -000	---- -uuu

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

Input/Output Ports

Holtek microcontrollers offer considerable flexibility on their I/O ports. Most pins can have either an input or output designation under user program control. Additionally, as there are pull-high resistors and wake-up software configurations, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

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

I/O Registers List

Register Name	Bit							
	7	6	5	4	3	2	1	0
PA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PAC	PAC7	PAC6	PAC5	PAC4	PAC3	PAC2	PAC1	PAC0
PAPU	PAPU7	PAPU6	PAPU5	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
PAWU	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	PAWU2	PAWU1	PAWU0
PB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
PBC	PBC7	PBC6	PBC5	PBC4	PBC3	PBC2	PBC1	PBC0
PBPU	PBPU7	PBPU6	PBPU5	PBPU4	PBPU3	—	PBPU1	PBPU0
PC	—	—	—	—	—	—	PC1	PC0
PCC	—	—	—	—	—	—	PCC1	PCC0
PCPU	—	—	—	—	—	—	PCPU1	PCPU0

Pull-high Resistors

Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, all I/O pins have the capability of being connected to an internal pull-high resistor when configured as an input. These pull-high resistors are selected using a registers PAPU~PCPU located in the Data Memory. The pull-high resistors are implemented using weak PMOS transistors. Note that pin PB2 does not have a pull-high resistor selection.

PAPU Register

Bit	7	6	5	4	3	2	1	0
Name	PAPU7	PAPU6	PAPU5	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **PAPU7~PAPU0**: Port A bit7~bit0 pull-high control
0: Disable
1: Enable

PBPU Register

Bit	7	6	5	4	3	2	1	0
Name	PBPU7	PBPU6	PBPU5	PBPU4	PBPU3	—	PBPU1	PBPU0
R/W	R/W	R/W	R/W	R/W	R/W	—	R/W	R/W
POR	0	0	0	0	0	—	0	0

Bit 7~3 **PBPU7~PBPU3**: Port B bit7~bit3 pull-high control
0: Disable
1: Enable

Bit 2 Unimplemented, read as "0"

Bit 1~0 **PBPU1~PBPU0**: Port B bit1~bit0 pull-high control
0: Disable
1: Enable

PCPU Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	PCPU1	PCPU0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **PCPU1~PCPU0**: Port C bit1~bit0 pull-high control
0: Disable
1: Enable

Port A Wake-up

If the HALT instruction is executed, the device will enter the Sleep 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 PA0~PA7pins from high to low. After a HALT instruction forces the microcontroller into entering the Sleep Mode, the processor will remain 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 pins PA0~PA7 can be selected individually to have this wake-up feature using an internal register known as PAWU, located in the Data Memory.

PAWU Register

Bit	7	6	5	4	3	2	1	0
Name	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	PAWU2	PAWU1	PAWU0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **PAWU7~PAWU0**: Port A bit7~bit 0 wake-up control
0: Disable
1: Enable

I/O Port Control Registers

Each port has its own control register known as PAC, PBC and PCC, which control the input/output configuration. With this control register, each I/O pin with or without pull-high resistors can be reconfigured dynamically under software control. 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 set as a CMOS output. If the pin is currently set as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

PAC Register

Bit	7	6	5	4	3	2	1	0
Name	PAC7	PAC6	PAC5	PAC4	PAC3	PAC2	PAC1	PAC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 Port A bit 7~bit 0 Input/Output control
0: Output
1: Input

PBC Register

Bit	7	6	5	4	3	2	1	0
Name	PBC7	PBC6	PBC5	PBC4	PBC3	PBC2	PBC1	PBC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 Port B bit 7~bit 0 Input/Output control
0: Output
1: Input

PCC Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	PCC1	PCC0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	1	1

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 Port C bit 1~bit 0 Input/Output control
0: Output
1: Input

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 application program control.

External Interrupt Input

The external interrupt pin, INT, is pin-shared with an I/O pin. To use the pin as an external interrupt input the correct bits in the INTC0 register must be programmed. The pin must also be set as an input by setting the PAC0 bit in the Port Control Register. A pull-high resistor can also be selected via the appropriate port pull-high resistor register. Note that even if the pin is set as an external interrupt input the I/O function still remains.

External Timer/Event Counter Input

The Timer/Event Counter pin TMR is pin-shared with I/O pins. For this shared pin to be used as Timer/Event Counter input, the Timer/Event Counter must be configured to be in the Event Counter or Pulse Width Capture Mode. This is achieved by setting the appropriate bits in the Timer/Event Counter Control Register. The pin must also be set as input by setting the appropriate bit in the Port Control Register. Pull-high resistor options can also be selected using the port pull-high resistor registers. Note that even if the pin is set as an external timer input the I/O function still remains.

PFD Output

The PFD function output is pin-shared with an I/O pin. The output function of this pin is chosen using the CTRL0 register. Note that the corresponding bit of the port control register must be set the pin as an output to enable the PFD output. If the port control register has set the pin as an input, then the pin will function as a normal logic input with the usual pull-high selection, even if the PFD function has been selected.

PWM Output

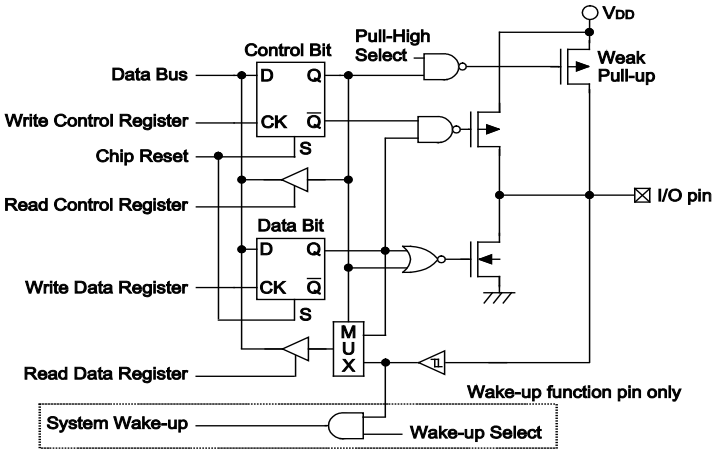
For the device the PWM function is included. The PWM function whose outputs are pin-shared with I/O pins. The PWM output functions are chosen using the CTRL0 register. Note that the corresponding bit of the port control registers, for the output pin, must setup the pin as an output to enable the PWM output. If the pins are setup as inputs, then the pin will function as a normal logic input with the usual pull-high selections, even if the PWM registers have enabled the PWM function.

A/D Input

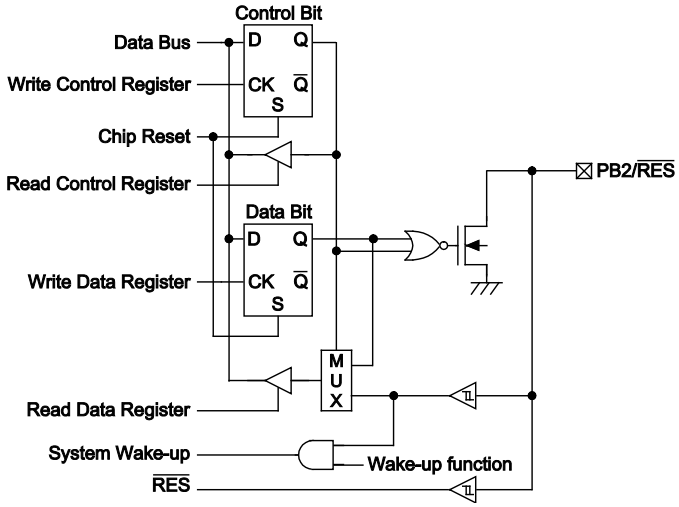
The device has eight inputs to the A/D converter. All of these analog inputs are pin-shared with I/O pins. If these pins are to be used as A/D inputs and not as I/O pins, then the corresponding PCRn bits in the A/D converter control register, ADCR1, must be properly setup. If chosen as I/O pins, then full pull-high resistor control remains, however if used as A/D inputs then any pull-high resistor control associated with these pins will be automatically disconnected.

I/O Pin Structures

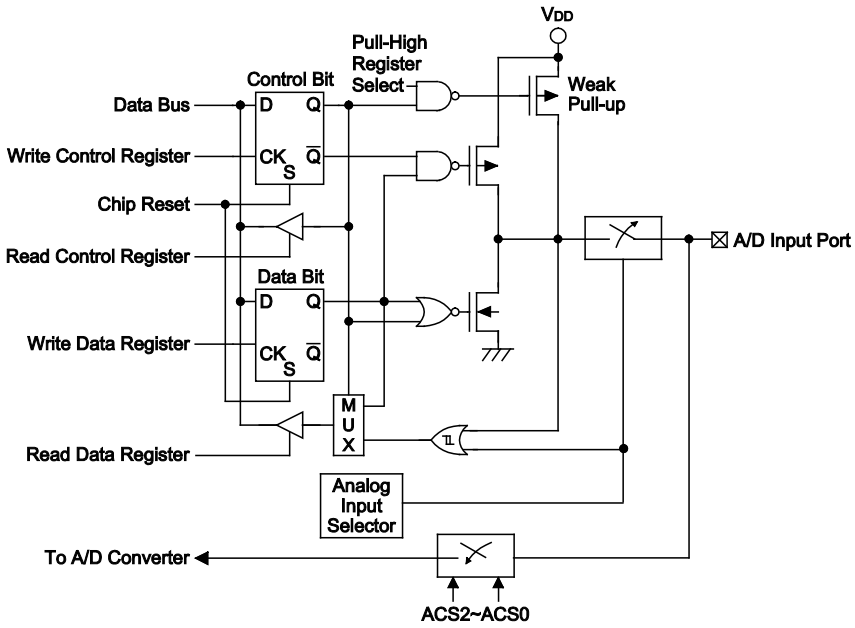
The accompanying diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins.



Generic Input/Output Ports



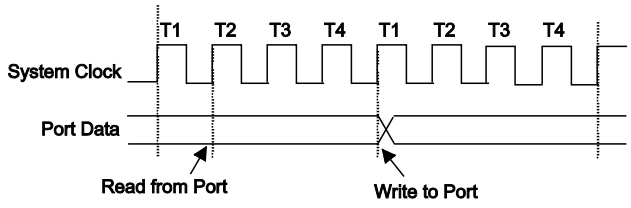
PB2 NMOS Input/Output Port



A/D Input/Output Port

Programming Considerations

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



Read Modify Write Timing

Pins PA0~PA7 each have wake-up functions, selected via the PAWU register. When the device is in the Sleep Mode, various methods are available to wake the device up. One of these is a high to low transition of any pins. Single or multiple pins on Port A can be set to have this function.

Timer/Event Counter

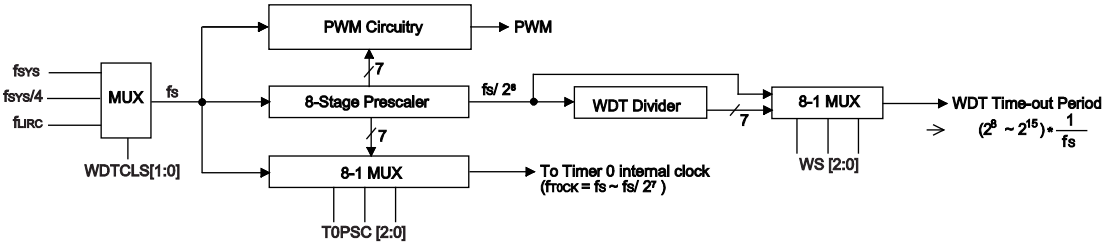
The provision of timer form an important part of any microcontroller, giving the designer a means of carrying out time related functions. The device contains from an 8-bit count-up timer. As the timer has three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width capture device. The provision of an internal prescaler to the clock circuitry on gives added range to the timer.

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

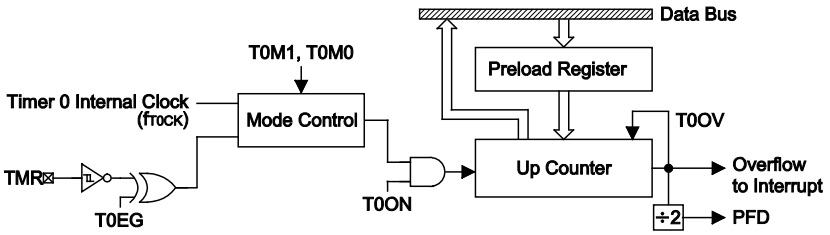
Configuring the Timer/Event Counter Input Clock Source

The Timer/Event Counter clock source can originate from various sources, an internal clock or an external pin. The internal clock source is used when the timer is in the timer mode. For the Timer/Event Counter 0, this internal clock source is first divided by a prescaler, the division ratio of which is conditioned by the Timer Control Register bits T0PSC2~T0PSC0. The internal clock source can be derived from the system clock f_{SYS} or from the instruction clock $f_{SYS}/4$ or the internal low speed oscillator LIRC for Timer/Event Counter selected by the clock selection bits WDTCLS1~WDTCLS0 in the register WDTLVRC.

An external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on an external timer pin TMR. Depending upon the condition of the T0EG bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.



Clock Source for Timer/PWM/WDT



8-bit Timer/Event Counter 0 Structure

Timer Register – TMR0

The timer register is special function register located in the Special Purpose Data Memory and is the place where the actual timer value is stored. The register is known as TMR0. The value in the timer register increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFH at which point the timer overflows and an internal interrupt signal is generated. The timer value will then reset with the initial preload register value and continue counting.

Note that to achieve a maximum full range count of FFH, the preload register must first be cleared. 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 in an OFF condition 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 data register during this period will remain in the preload register and will only be written into the actual counter the next time an overflow occurs.

Timer Control Register – TMR0C

The flexible features of the Holtek microcontroller Timer/Event Counter enable it to operate in three different modes, the options of which are determined by the contents of their respective control register.

The Timer Control Register is known as TMR0C. It is the Timer Control Register together with its corresponding timer register that controls the full operation of the Timer/Event Counter. Before the timer can be used, it is essential that the Timer Control Register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialization.

To select which of the three modes the timer is to operate in, either in the timer mode, the event counting mode or the pulse width capture mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair T0M1/T0M0, must be set to the required logic levels. The timer-on bit, which is bit 4 of the Timer Control Register and known as T0ON, provides the basic on/off control of the respective timer. Setting the bit to high allows the counter to run. Clearing the bit stops the counter. Bits 0~2 of the Timer Control Register determine the division ratio of the input clock prescaler. The prescaler bit settings have no effect if an external clock source is used. If the timer is in the event count or pulse width capture mode, the active transition edge level type is selected by the logic level of bit 3 of the Timer Control Register which is known as T0EG.

TMR0C Register

Bit	7	6	5	4	3	2	1	0
Name	T0M1	T0M0	—	T0ON	T0EG	T0PSC2	T0PSC1	T0PSC0
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	0	0	—	0	1	0	0	0

Bit 7~6 **T0M1~T0M0**: Timer operation mode selection
00: No mode available
01: Event counter mode
10: Timer mode
11: Pulse width capture mode

Bit 5 Unimplemented, read as "0"

Bit 4 **T0ON**: Timer/event counter counting enable
0: Disable
1: Enable

Bit 3 **T0EG**: Timer/Event Counter active edge selection
In event counter mode (T0M1~T0M0=01)
0: Count on rising edge
1: Count on falling edge
In pulse width measurement mode (T0M1~T0M0=11)
0: Start counting on falling edge, stop on the rising edge
1: Start counting on rising edge, stop on the falling edge

Bit 2~0 **T0PSC2~T0PSC0**: Timer prescalar rate selection
000: f_S
001: $f_S/2$
010: $f_S/4$
011: $f_S/8$
100: $f_S/16$
101: $f_S/32$
110: $f_S/64$
111: $f_S/128$

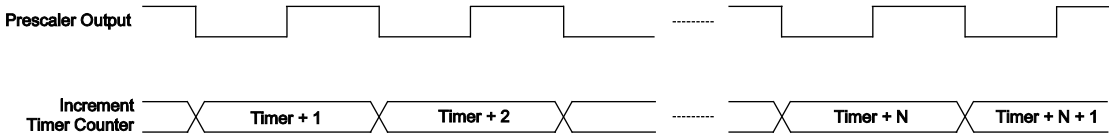
Timer Mode

In this mode, the Timer/Event Counter can be utilized to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. To operate in this mode, the Operating Mode Select bit pair, TnM1/TnM0, in the Timer Control Register must be set to the correct value as shown.

Bit7	Bit6
1	0

Control Register Operating Mode Select Bits for the Timer Mode.

In this mode the internal clock is used as the timer clock. The timer input clock source is f_{SYS}, $f_{SYS}/4$ or f_{LIRC}. However, this timer clock source is further divided by a prescaler, the value of which is determined by the bits T0PSC2~T0PSC0 in the Timer Control Register. The timer-on bit, T0ON must be set high to enable the timer to run. Each time an internal clock high to low transition occurs, the timer increments by one. When the timer is full and overflows, an interrupt signal is generated and the timer will reload the value already loaded into the preload register and continue counting. A timer overflow condition and corresponding internal interrupts are two of the wake-up sources. However, the internal interrupts can be disabled by ensuring that the T0E bits of the INTC0 register are reset to zero.



Timer Mode Timing Chart

Event Counter Mode

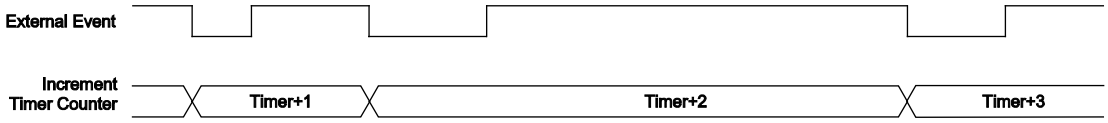
In this mode, a number of externally changing logic events, occurring on the external timer TMR pin, can be recorded by the Timer/Event Counter. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0, in the Timer Control Register must be set to the correct value as shown.

Bit7	Bit6
0	1

Control Register Operating Mode Select Bits for the Timer Mode.

In this mode, the external timer TMR pin is used as the Timer/Event Counter clock source, however it is not divided by the internal prescaler. After the other bits in the Timer Control Register have been set, the enable bit T0ON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. If the Active Edge Select bit, T0EG, which is bit 3 of the Timer Control Register, is low, the Timer/Event Counter will increment each time the external timer pin receives a low to high transition. If the T0EG is high, the counter will increment each time the external timer pin receives a high to low transition. When it is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the corresponding Interrupt Control Register. It is reset to zero.

As the external timer pin is shared with an I/O pin, to ensure that the pin is configured to operate as an event counter input pin, two things have to happen. The first is to ensure that the Operating Mode Select bits in the Timer Control Register place the Timer/Event Counter in the Event Counting Mode. The second is to ensure that the port control register configures the pin as an input. It should be noted that in the event counting mode, even if the microcontroller is in the Sleep Mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input TMR pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.



Event Counter Mode Timing Chart (T0EG=1)

Pulse Width Capture Mode

In this mode, the Timer/Event Counter can be utilised to measure the width of external pulses applied to the external timer pin. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0, in the Timer Control Register must be set to the correct value as shown.

Bit7	Bit6
1	1

Control Register Operating Mode Select Bits for the Pulse Width Capture Mode

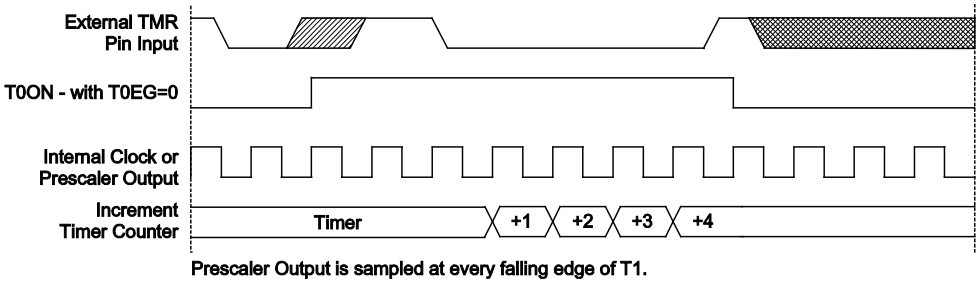
In this mode the internal clock, f_{SYS}, $f_{SYS}/4$ or f_{LIRC} is used as the internal clock for the 8-bit Timer/Event Counter. However, the clock source, f_{SYS}, for the 8-bit timer is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits T0PSC2~T0PSC0, which are bit 2~0 of the Timer Control Register, After other bits in the Timer Control Register have been set, the enable bit T0ON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter, however it will not actually start counting until an active edge is received on the external timer pin.

If the Active Edge Select bit T0EG which is bit 3 of the Timer Control Register is low, once a high to low transition has been received on the external timer pin, the Timer/Event Counter will start counting until the external timer pin returns to its original high level. At this point the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. If the Active Edge Select bit is high, the Timer/Event Counter will begin counting once a low to high transition has been received on the external timer pin and stop counting when the external timer pin returns to its original low level. As before, the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. It is important to note that in the pulse width capture mode, the enable 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 enable bit can only be reset to zero under program control.

The residual value in the Timer/Event Counter, which can now be read by the program, therefore represents the length of the pulse received on the TMR pin. As the enable bit has now been reset, any further transitions on the external timer pin will be ignored. The timer cannot begin further pulse width capture until the enable bit is set high again by the program. In this way, single shot pulse measurements can be easily made.

It should be noted that in this mode the Timer/Event Counter is controlled by logical transitions on the external timer pin and not by the logic level. When the Timer/Event Counter is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the corresponding Interrupt Control Register, it is reset to zero.

As the TMR pin is shared with an I/O pin, to ensure that the pin is configured to operate as a pulse width capture pin, two things have to be implemented. The first is to ensure that the Operating Mode Select bits in the Timer Control Register place the Timer/Event Counter in the pulse width capture mode, the second is to ensure that the port control register configure the pin as an input.



Pulse Width Capture Mode Timing Chart (T0EG=0)

Prescaler

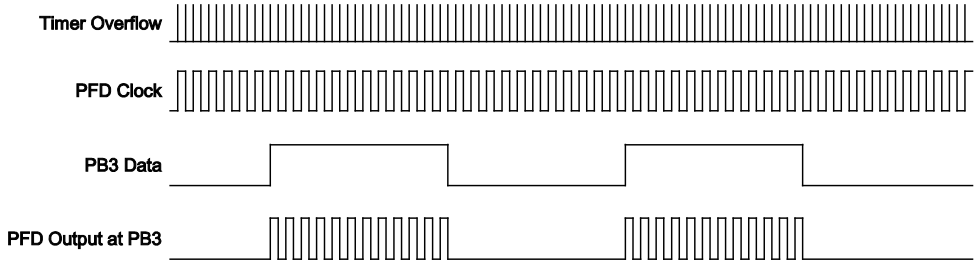
Bits T0PSC2~T0PSC0 of the TMR0C register can be used to define a division ratio for the internal clock source of the Timer/Event Counter enabling longer time out periods to be set.

PFD Function

The Programmable Frequency Divider provides a means of producing a variable frequency output suitable for application, such as some interfaces requiring a precise frequency generator.

The Timer/Event Counter overflow signal is the clock source for the PFD function, which is controlled by PFDC bit in CTRL0. For this device the clock source can come from Timer/Event Counter 0. The output frequency is controlled by loading the required values into the timer prescaler and timer registers to give the required division ratio. The counter will begin to count-up from this preload register value until full, at which point an overflow signal is generated, causing both the PFD outputs to change state. Then the counter will be automatically reloaded with the preload register value and continue counting-up.

If the CTRL0 register has selected the PFD function, then for PFD output to operate, it is essential for the Port B control register PBC to set the PFD pins as outputs. PB3 must be set high to activate the PFD. The output data bits can be used as the on/off control bit for the PFD outputs. Note that the PFD outputs will all be low if the output data bit is cleared to zero.



PFD Function

I/O Interfacing

The Timer/Event Counter, when configured to run in the event counter or pulse width capture mode, requires the use of an external timer pin for its operation. As this pin is a shared pin it must be configured correctly to ensure that it is set for use as a Timer/Event Counter input pin. This is achieved by ensuring that the mode selects bits in the Timer/Event Counter control register, either the event counter or pulse width capture mode. Additionally the corresponding Port Control Register bit must be set high to ensure that the pin is set as an input. Any pull-high resistor connected to this pin will remain valid even if the pin is used as a Timer/Event Counter input.

Programming Considerations

When running 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 capture 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.

When the Timer/Event Counter overflows, its corresponding interrupt request flag in the interrupt control register will be set. If the Timer/Event Counter 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 Sleep Mode.

Timer Program Example

The program shows how the Timer/Event Counter registers are set 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 Counters to be in the timer mode, which uses the internal system clock as their clock source.

PFD Programming Example

```
org  04h           ; external interrupt vector
org  08h           ; Timer Counter 0 interrupt vector
jmp  tmr0int       ; jump here when Timer 0 overflows
:
:
org  20h           ; main program
:
:
                   ; internal Timer 0 interrupt routine
tmr0int:
:
                   ; Timer 0 main program placed here
:
:
begin:
                   ; set Timer 0 registers
mov  a,09bh        ; set Timer 0 preload value
mov  tmr0,a
mov  a,081h        ; set Timer 0 control register
mov  tmr0c,a       ; timer mode and prescaler set to/2
mov  a, 0c0H       ; select fSYS for the TMR0 clock source
mov  wdtlvrc, a
                   ; set interrupt register
mov  a,05h         ; enable master interrupt and both timer interrupts
mov  intc0,a
:
:
set  tmr0c.4       ; start Timer 0
:
:
```

Pulse Width Modulator

The device includes one 8-bit PWM function. Useful for the applications such as motor speed control, the PWM function provides outputs with a fixed frequency but with a duty cycle that can be varied by setting particular values into the corresponding PWM register.

PWM Operation

The register, known as PWM and located in the Data Memory is assigned to each Pulse Width Modulator channel. It is here 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 subdivided into two or four individual modulation subsections, known as the 7+1 mode or 6+2 mode respectively. The required mode and the on/off control for each PWM channel is selected using the CTRL0 register. Note that when using the PWM, it is only necessary to write the required value into the PWM register and select the required mode set and on/off control using the CTRL0 register, the subdivision of the waveform into its sub-modulation cycles is implemented automatically within the microcontroller hardware. The PWM clock source f_S comes from the system clock f_{SYS}, $f_{SYS}/4$ or f_{LIRC}.

This method of dividing the original modulation cycle into a further 2 or 4 sub-cycles enable the generation of higher PWM frequencies which allow a wider range of applications to be served. The difference between what is known as the PWM cycle frequency and the PWM modulation frequency should be understood. As the PWM clock source comes from the system clock f_{SYS} and the PWM value is 8-bits wide, the overall PWM cycle frequency is $f_{SYS}/256$. However, when in the 7+1 mode of operation the PWM modulation frequency will be $f_S/128$, while the PWM modulation frequency for the 6+2 mode of operation will be $f_S/64$.

PWM Modulation	PWM Cycle Frequency	PWM Cycle Duty
$f_S/64$ for (6+2) bits mode $f_S/128$ for (7+1) bits mode	$f_S/256$	[PWM]/256

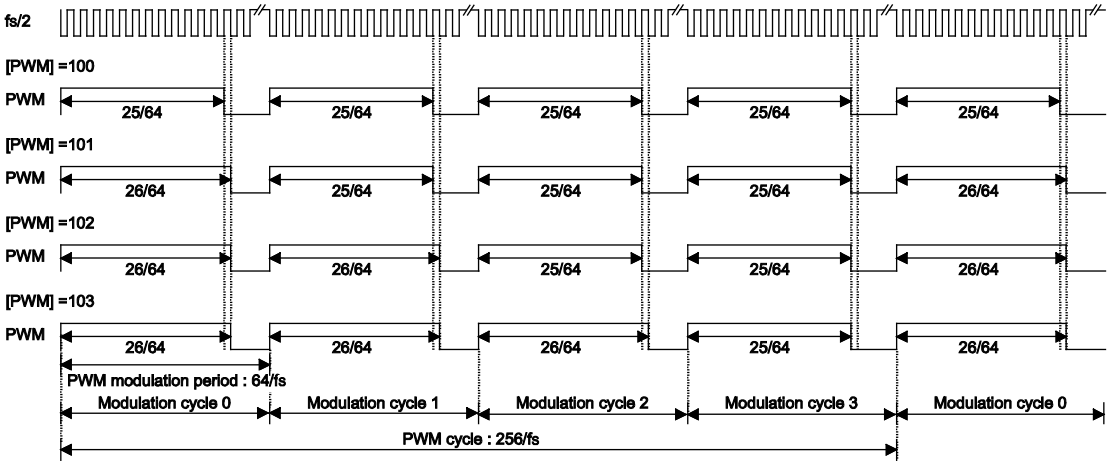
6+2 PWM Mode

Each full PWM cycle, as it is controlled by an 8-bit PWM 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 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.

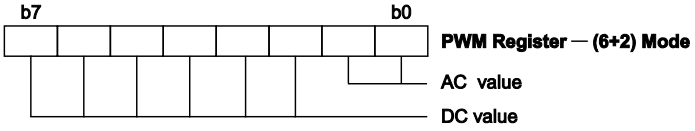
Parameter AC (0~3)	DC	(Duty Cycle)
Modulation cycle i (i=0~3)	i<AC	(DC+1)/64
	i>AC	DC/64

6+2 Mode Modulation Cycle Values

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



6+2 PWM Mode



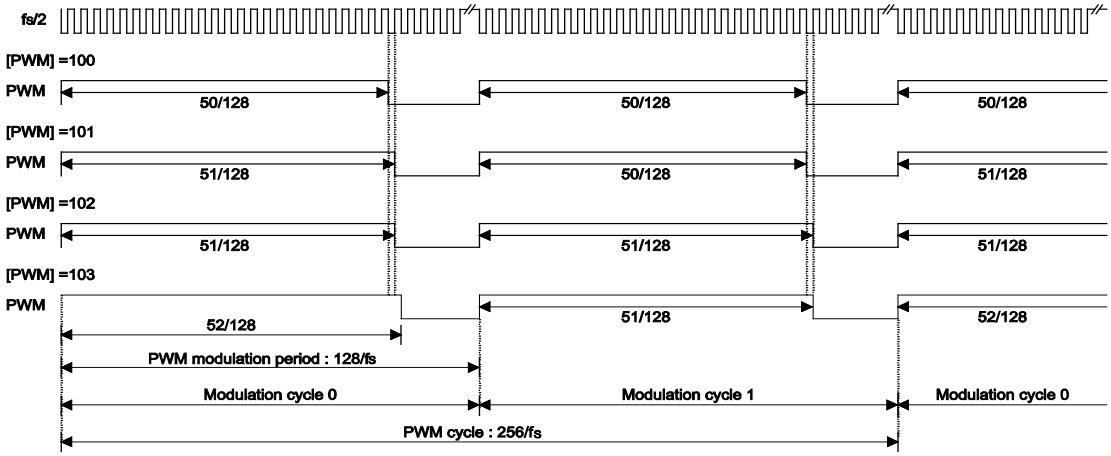
PWM Register for 6+2 Mode

7+1 PWM Mode

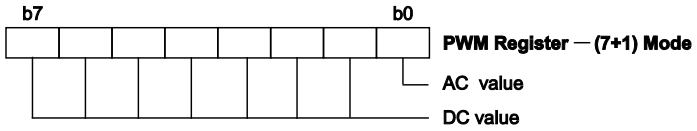
Each full PWM cycle, as it is controlled by an 8-bit PWM register, has 256 clock periods. However, in the 7+1 PWM mode, each PWM cycle is subdivided into two individual sub-cycles known as modulation cycle 0~modulation cycle 1, denoted as i in the table. Each one of these two sub-cycles contains 128 clock cycles. In this mode, a modulation frequency increase of two 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 bit1~bit7 is denoted here as the DC value. The second group which consists of bit0 is known as the AC value. In the 7+1 PWM mode, the duty cycle value of each of the two modulation sub-cycles is shown in the following table.

Parameter	AC(0~1)	DC (Duty Cycle)
Modulation cycle i (i=0~1)	i<AC	(DC+1)/128
	i>=AC	DC/128

The following diagram illustrates the waveforms associated with the 7+1 mode PWM operation. It is important to note how the single PWM cycle is subdivided into 2 individual modulation cycles, numbered 0 and 1 and how the AC value is related to the PWM value.



7+1 Mode



PWM Register for 7+1 Mode

PWM Output Control

The PWM outputs are pin-shared with the I/O pin PA4. To operate as a PWM output and not as an I/O pin, the correct bits must be set in the CTRL0 register. A zero value must also be written to the corresponding bit in the I/O port control register PAC.4 to ensure that the corresponding PWM output pin is set 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 high value to the corresponding bit in the output data register PA.4 will enable the PWM data to appear on the pin. Writing a zero value will disable the PWM output function and force the output low. In this way, the Port data output registers can be used as an on/off control for the PWM function. Note that if the CTRL0 register has selected the PWM function, but a high value has been written to its corresponding bit in the PAC 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

```
mov  a,64h          ; set PWM value of decimal 100
mov  pwm0,a
set  ctrl0.5        ; select the 7+1 PWM mode
set  ctrl0.3        ; select pin PA4 to have a PWM function
clr  pac.4          ; set pin PA4 as an output
set  pa.4           ; enable the PWM output
:       :
clr  pa.4           ; disable the PWM output_ pin PA4 forced low
```

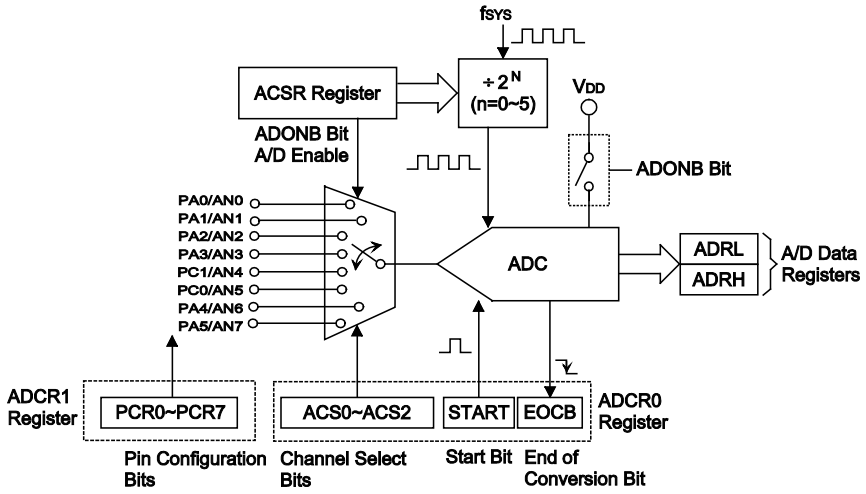
Analog to Digital Converter

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

The device contains an 8-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 either a 12-bit digital value.

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



A/D Converter Structure

A/D Converter Data Registers – ADRL, ADRH

The device, which has an internal 12-bit A/D converter, require two data registers, a high byte register, known as ADRH, and a low byte register, known as ADRL. After the conversion process takes place, these registers can be directly read by the microcontroller to obtain the digitised conversion value. Only the high byte register, ADRH, utilises its full 8-bit contents. The low byte register utilises only 4 bit of its 8-bit contents as it contains only the lowest bits of the 12-bit converted value.

In the following table, D0~D11 is the A/D conversion data result bits.

ADRH, ADRL Register

Bit	ADRH								ADRL							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Name	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	—	—	—	—
R/W	R	R	R	R	R	R	R	R	R	R	R	R	—	—	—	—
POR	x	x	x	x	x	x	x	x	x	x	x	x	—	—	—	—

"x" unknown

Unimplemented, read as "0"

D11~D0: ADC conversion data

A/D Converter Control Registers – ACSR, ADCR0, ADCR1

To control the function and operation of the A/D converter, three control registers known as ADCR0, ADCR1 and ACSR are provided. These 8-bit registers define functions such as the on/off function, 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, the A/D clock source as well as controlling the start function and monitoring the A/D converter end of conversion status.

The ACS2~ACS0 bits in the ADCR0 register define the channel number. As the device contains only one actual analog to digital converter circuit, each of the individual 8 analog inputs must be routed to the converter. It is the function of the ACS2~ACS0 bits in the ADCR0 register to determine which analog channel is actually connected to the internal A/D converter.

The PCR7~PCR0 bits contained in the ADCR1 register which determine which pins on PA0~PA5, PC0~PC1 are used as analog inputs for the A/D converter and which pins are to be used as normal I/O pins. If the PCRn bit has a value of 1, then the corresponding pin, namely one of the AN0~AN7 analog inputs, will be set as analog inputs. Note that if the PCRn bit is set to zero, then the corresponding pin on PA0~PA5, PC0~PC1 will be set as a normal I/O pin, the analog input channels will be all disabled and the A/D converter circuitry will be powered off.

ADCR0 Register

Bit	7	6	5	4	3	2	1	0
Name	START	EOCB	—	—	—	ACS2	ACS1	ACS0
R/W	R/W	R	—	—	—	R/W	R/W	R/W
POR	0	1	—	—	—	0	0	0

Bit 7 **START**: Start the A/D conversion
0→1→0: Start
0→1: Reset the A/D converter and set EOCB to "1"
This bit is used to initiate an A/D conversion process. The bit is normally low but if set high and then cleared low again, the A/D converter will initiate a conversion process. When the bit is set high the A/D converter will be reset.

Bit 6 **EOCB**: End of A/D conversion flag
0: A/D conversion ended
1: A/D conversion in progress
This read only flag is used to indicate when an A/D conversion process has completed. When the conversion process is running the bit will be high.

Bit 5~3 Unimplemented, read as "0"

Bit 2~0 **ACS2~ACS0**: A/D channel selection
000: AN0
001: AN1
010: AN2
011: AN3
100: AN4
101: AN5
110: AN6
111: AN7

ADCR1 Register

Bit	7	6	5	4	3	2	1	0
Name	PCR7	PCR6	PCR5	PCR4	PCR3	PCR2	PCR1	PCR0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **PCR7~PCR0**: select I/O or ADC analog input
0: I/O
1: Analog input
If PCR0~PCR7 are all zero, the ADC circuit is power off to reduce power consumption.

ACSR Control Register

Bit	7	6	5	4	3	2	1	0
Name	TEST	ADONB	—	—	—	ADCS2	ADCS1	ADCS0
R/W	R/W	R/W	—	—	—	R/W	R/W	R/W
POR	1	0	—	—	—	0	0	0

Bit 7 **TEST**: For test mode use only

Bit 6 **ADONB**: ADC MODULE on/off control bit
0: ADC module is on
1: ADC module is off
Note: 1. it is recommended to set ADONB=1 before entering sleep for saving power.
2. ADONB=1 will power down the ADC module.

Bit 5~3 Unimplemented, read as "0"

Bit 2~0 **ADCS2~ADCS0**: Select ADC clock source
000: $f_{SYS}/2$
001: $f_{SYS}/8$
010: $f_{SYS}/32$
011: Undefined
100: f_{SYS}
101: $f_{SYS}/4$
110: $f_{SYS}/16$
111: Undefined
These three bits are used to select the clock source for the A/D converter.

A/D Operation

The START bit in the 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, the EOCB bit in the ADCR0 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 start operation of the internal analog to digital converter.

The EOCB bit in the ADCR0 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 ADCR0 register to check whether it has been cleared as an alternative method of detecting the end of an A/D conversion cycle.

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 ADCS2, ADCS1 and ADCS0 bits in the ACSR register.

The A/D converter overall on/off control is a function of both the ADONB bit in the ACSR register and the PCR7~PCR0 bits in the ADCR1 register as shown in the table. Either the ADONB bit cleared to zero or the PCR7~PCR0 bits set to a zero value will switch off the A/D converter. These are important consideration in power sensitive applications and must be taken into account if power consumption is to be minimised. As the table illustrates, execution of the HALT instruction has no effect on the A/D converter on/off control and subsequently its power consumption.

PCR7~PCR0 Bits	HALT Instruction	ADONB Bit	ADC On/Off
=0	X	X	Off
> 0	X	0	On
> 0	X	1	Off

X: Don't care

A/D Converter On/Off Control

Although the A/D clock source is determined by the system clock f_{SYS}, and by bits ADCS2, ADCS1 and ADCS0, there are some limitations on the maximum A/D clock source speed that can be selected. As the range value of permissible A/D clock period, t_{AD}, is 0.5μs~10us, care must be taken for system clock speeds in excess of 4MHz. For system clock speeds in excess of 4MHz, the ADCS2, ADCS1 and ADCS0 bits should not be set to "000". Doing so will give A/D clock periods that are exceeds the limited range of 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.

f_{SYS}	A/D Clock Period(t_{AD})						
	ADCS2, ADCS1, ADCS0 =000 (f_{SYS}/2)	ADCS2, ADCS1, ADCS0 =001 (f_{SYS}/8)	ADCS2, ADCS1, ADCS0 =010 (f_{SYS}/32)	ADCS2, ADCS1, ADCS0 =100 (f_{SYS})	ADCS2, ADCS1, ADCS0 =101 (f_{SYS}/4)	ADCS2, ADCS1, ADCS0 =110 (f_{SYS}/16)	ADCS2, ADCS1, ADCS0 =011,111
1MHz	2μs	8μs	32μs	1μs	4μs	16μs*	Undefined
2MHz	1μs	4μs	16μs	500ns	2μs	8μs	Undefined
4MHz	500ns	2μs	8μs	250ns*	1μs	4μs	Undefined
8MHz	250ns*	1μs	4μs	125ns*	500ns	2μs	Undefined
12MHz	167ns*	667ns	2.67μs	83ns*	333ns*	1μs	Undefined

A/D Clock Period Examples

A/D Input Pins

All of the A/D analog input pins are pin-shared with the I/O pins on Port A and PC. Bits PCR7~PCR0 in the ADCR1 register determine whether the input pins are set as normal input/output pins or whether they are set 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 set through register programming, apply to the input pins only when they are used as normal I/O pins, if set as A/D inputs the pull-high resistors will be automatically disconnected. Note that it is not necessary to first set the A/D pin as an input in the I/O port control registers to enable the A/D input as when the PCR7~PCR0 bits enable an A/D input, the status of the port control register will be overridden.

Summary of A/D Conversion Steps

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

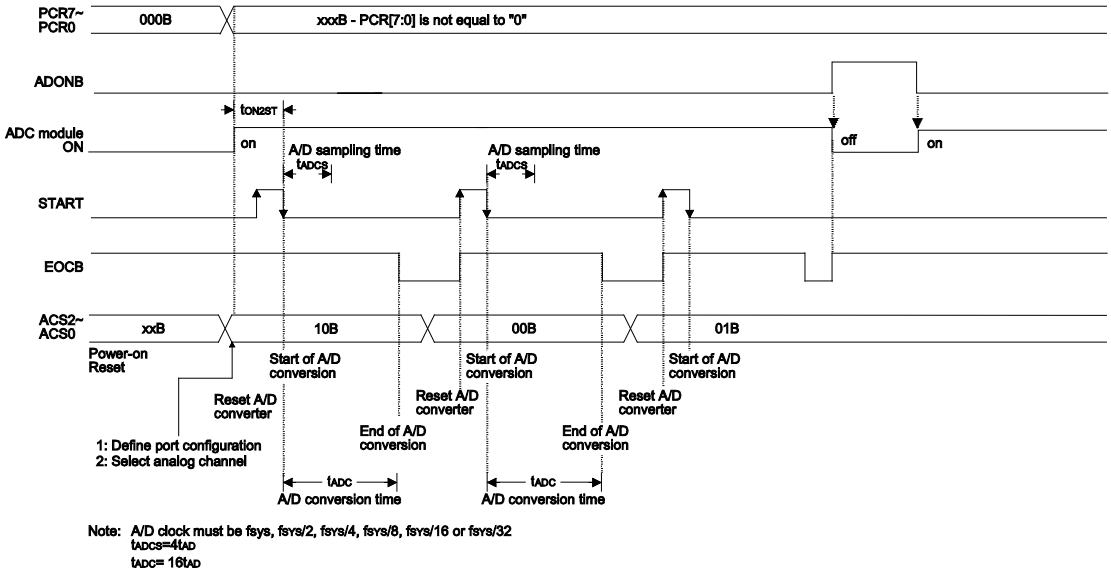
- Step 1
  Select the required A/D conversion clock by correctly programming bits ADCS2~ADCS0 in the ACSR register.
- Step 2
  Select which pins are to be used as A/D inputs and configure them as A/D input pins by correctly programming the PCR7~PCR0 bits in the ADCR1 register.
- Step 3
  Enable the A/D by clearing the ADONB in the ACSR register to zero.
- Step 4
  Select which channel is to be connected to the internal A/D converter by correctly programming the ACS2~ACS0 bits which are also contained in the register.
- Step 5
  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, the INTC0 interrupt control register must be set to “1”, the A/D converter interrupt bit, ADE , must also be set to “1”.
- Step 6
  The analog to digital conversion process can now be initialized by setting the START bit in the ADCR0 register from low to high and then low again. Note that this bit should have been originally cleared to zero.

- Step 7

  To check when the analog to digital conversion process is complete, the EOCB bit in the ADCR0 register can be polled. The conversion process is complete when this bit goes low. When this occurs the A/D data registers ADRL and ADRH 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 ADCR0 register is used, the interrupt enable step above can be omitted.

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



A/D Conversion 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 $16t_{AD}$ where t_{AD} is equal to the A/D clock period.

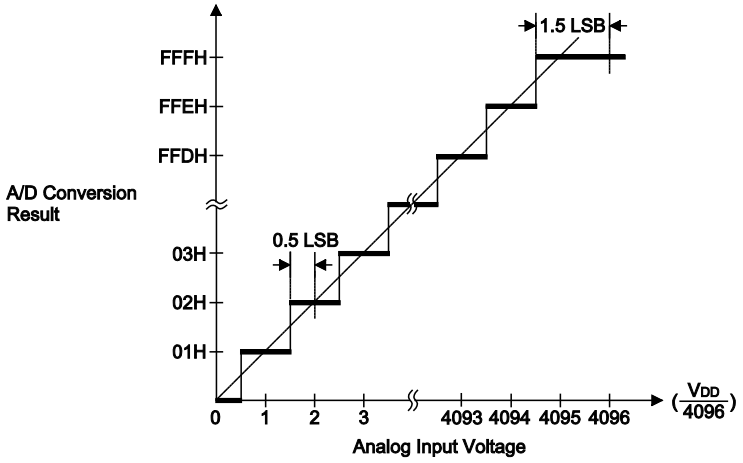
Programming Considerations

When programming, the special attention must be given to the PCR [7:0] bits in the 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 internal A/D circuitry will be power down. Setting the ADONB bit high has the ability to power down the internal A/D circuitry, which may be an important consideration in power sensitive applications.

A/D Transfer Function

As the device contains a 12-bit A/D converter, its full-scale converted digitized value is equal to FFFH. 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} divided by 4096. The diagram shows the ideal transfer function between the analog input value and the digitized output value for the A/D converter.

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



Ideal A/D Transfer Function

A/D Programming Example

The following two programming examples illustrate how to set and implement an A/D conversion. In the first example, the method of polling the EOCB bit in the ADCR0 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  ADE              ; disable ADC interrupt
mov  a,00000001B
mov  ACSR,a           ; select fSYS/8 as A/D clock and ADONB=0
mov  a,00000001B      ; set ADCR1 register to configure Port as A/D inputs
mov  ADCR1,a
mov  a, 00000000B     ; select AN0 to be connected to the A/D converter
mov  ADCR0, a
:
:
Start_conversion:
clr  START
set  START            ; reset A/D
clr  START            ; start A/D
Polling_EOC:
sz   EOCB             ; poll the ADCR0 register EOCB bit to detect end of A/D conversion
jmp  polling_EOC      ; continue polling
mov  a,ADRL           ; read low byte conversion result value
mov  adrl_buffer,a    ; save result to user defined register
mov  a,ADRH           ; read high byte conversion result value
mov  adrh_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  ADE              ; disable ADC interrupt
mov  a,00000001B
mov  ACSR,a           ; select fSYS/8 as A/D clock and ADONB=0
mov  a,00000001B      ; set ADCR1 register to configure Port as A/D inputs
mov  ADCR1,a
mov  a, 00000000B     ; select AN0 to be connected to the A/D converter
mov  ADCR0, a
:
:
Start_conversion:
clr  START
set  START            ; reset A/D
clr  START            ; start A/D
clr  ADF              ; clear ADC interrupt request flag
set  ADE              ; enable ADC interrupt
set  EMI              ; enable global interrupt
:
:
:
                      ; ADC interrupt service routine
ADC_:
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,ADRL           ; read low byte conversion result value
mov  adrl_buffer,a    ; save result to user defined register
mov  a,ADRH           ; read high byte conversion result value
mov  adrh_buffer,a    ; save result to user defined register
:
:
EXIT_ISR:
mov  a,status_stack
mov  STATUS,a         ; restore STATUS from user defined memory
mov  a, acc_stack     ; restore ACC from user defined memory
clr  ADF              ; clear ADC interrupt flag
reti
```

Note: To power off ADC module, it is necessary to set ADONB as “1”.

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer/Event Counter requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs.

The device contains only one external interrupt and multiple internal interrupts. The external interrupts are controlled by the action of the external interrupt pin, while the internal interrupt is controlled by the Timer/Event Counter and the A/D converter interrupt.

Interrupt Register

Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by using the register, INTC0. By controlling the appropriate enable bits in the register 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 cleared to zero will disable all interrupts.

Function	Enable Bit	Request Flag
Global	EMI	—
INT Pin	INTE	INTF
Timer	T0E	T0F
A/D Converter	ADE	ADF

INTC0 Register

Bit	7	6	5	4	3	2	1	0
Name	—	ADF	T0F	INTF	ADE	T0E	INTE	EMI
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as "0"

Bit 5 **ADF**: A/D interrrupt request flag
0: No request
1: Interrupt request

Bit 5 **T0F**: Timer/Event Counter 0 interrrupt request flag
0: No request
1: Interrupt request

Bit 4 **INTF**: INT pin interrupt request flag
0: No request
1: Interrupt request

Bit 3 **ADE**: A/D interrupt control
0: Disable
1: Enable

Bit 2 **T0E**: Timer/Event Counter 0 interrupt control
0: Disable
1: Enable

Bit 1 **INTE**: INT interrupt control
0: Disable
1: Enable

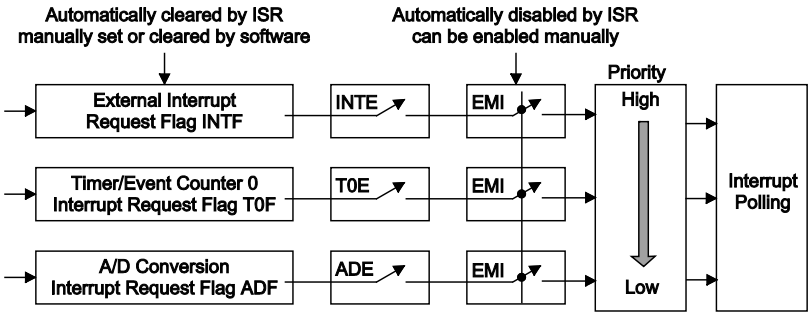
Bit 0 **EMI**: Global interrupt control
0: Disable
1: Enable

Interrupt Operation

A Timer/Event Counter overflow, a completion of A/D conversion or an active edge on the external interrupt pin 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 instruction, 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.

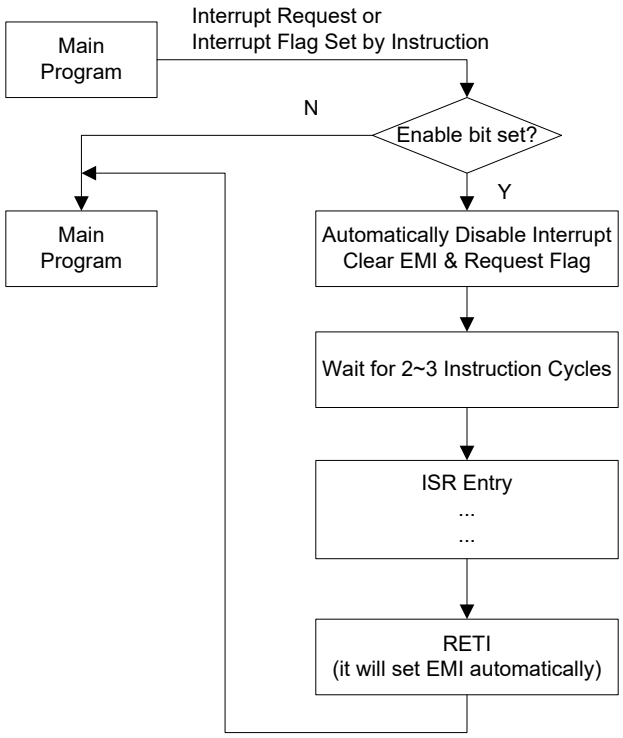
The various interrupt enable bits, together with their associated request flags, are shown in the following diagram with their order of priority.



Interrupt Scheme

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.

When an interrupt request is generated it takes 2 or 3 instruction cycles before the program jumps to the interrupt vector. If the device is in the Sleep Mode and is woken up by an interrupt request then it will take 3 cycles before the program jumps to the interrupt vector.



Interrupt Flow

Interrupt Priority

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

Interrupt Source	Priority	Vector
External interrupt	1	04H
Timer/Event Counter 0 overflow	2	08H
A/D converter complete	3	0CH

In cases where both external and internal interrupts are enabled and where an external and internal interrupt occurs simultaneously, the external interrupt will always have priority and will therefore be serviced first. Suitable masking of the individual interrupts using the 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, INTE, must first be set. An actual external interrupt will take place when the external interrupt request flag, INTF is set, a situation that will occur when an edge transition appears on the external INT line. The type of transition that will trigger an external interrupt, whether high to low, low to high or both is determined by the INTES0 and INTES1 bits, which are bits 6 and 7 respectively in the CTRL0 control register. These two bits can also disable the external interrupt function.

INTES1	INTES0	Request Flag
0	0	External interrupt disable
0	1	Rising edge trigger
1	0	Falling edge trigger
1	1	Dual edge trigger

The external interrupt pin is pin-shared with the I/O pin PA0 and can only be used as an external interrupt pin if the corresponding external interrupt enable bit in the INTC0 register has been set and the edge trigger type has been selected using the CTRL0 register. The pin must also be set as an input by setting the corresponding PAC.0 bit in the port control register. When the interrupt is enabled, the stack is not full and a transition appears on the external interrupt pin, a subroutine call to the external interrupt vector at location 04H, will take place. When the interrupt is serviced, the external interrupt request flag, INTF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor connections on this pin will remain valid even if the pin is used as an external interrupt input

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 T0E must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter request flag T0F is set, a situation that will occur when the relevant 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 relevant timer interrupt vector, will take place. When the interrupt is serviced, the timer interrupt request flag T0F will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

A/D Converter Interrupt

The device includes A/D interrupt. For an A/D interrupt to occur, the global interrupt enable bit EMI and the corresponding interrupt enable bit ADE must be first set. An actual A/D interrupt will take place when the A/D converter request flag ADF is set, a situation that will occur when an A/D conversion process has completed. When the interrupt is enabled, the stack is not full and an A/D conversion process finishes execution, a subroutine call to the relevant A/D interrupt vector, 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.

Interrupt Wake-up Function

Each of the interrupt functions has the capability of waking up the microcontroller when in the Sleep Mode. A wake-up is generated when an interrupt request flag changes from low to high and is independent of whether the interrupt is enabled or not. Therefore, even though the device is in the Sleep Mode and its system oscillator is stopped, situations such as external edge transitions on the external interrupt pins or timer/event counter overflow may cause their respective interrupt flag to be set high and consequently generate an interrupt. Care must therefore be taken if spurious wake-up situations are to be avoided. If an interrupt wake-up function is to be disabled then the corresponding interrupt request flag should be set high before the device enters the Sleep Mode. The interrupt enable bits have no effect on the interrupt wake-up function.

Programming Considerations

By disabling the relevant 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 register until the corresponding interrupt is serviced or until the request flag is cleared by the application program.

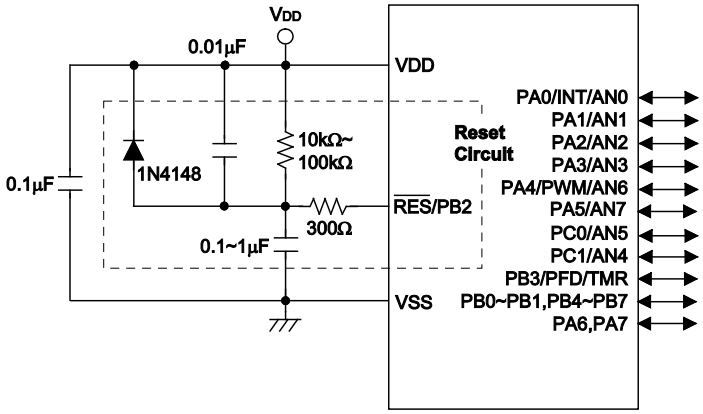
It is recommended that programs do not use the "CALL" instruction within the interrupt service subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately. If only one stack is left and the interrupt is not well controlled, the original control sequence will be damaged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

All of these interrupts have the capability of waking up the microcontroller when it is in Sleep Mode, the wake up being generated when the interrupt request flag changes from low to high. If it is required to prevent a certain interrupt from waking up the microcontroller then its respective request flag should be first set high before entering the Sleep Mode.

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

To return from an interrupt subroutine, either a RET or RETI instruction may be executed. The RETI instruction in addition to executing a return to the main program also automatically sets the EMI bit high to allow further interrupts. The RET instruction however only executes a return to the main program leaving the EMI bit in its present zero state and therefore disabling the execution of further interrupts.

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 microcontroller, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

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

Instruction Timing

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

Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of several 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 such as 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 Operation

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 which 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 instructions related to the data memory access in the following table can be used when the desired data memory is located in Data Memory sector 0.

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, SC
ADDM A,[m]	Add ACC to Data Memory	1Note	Z, C, AC, OV, SC
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV, SC
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV, SC
ADCM A,[m]	Add ACC to Data memory with Carry	1Note	Z, C, AC, OV, SC
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV, SC, CZ
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV, SC, CZ
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1Note	Z, C, AC, OV, SC, CZ
SBC A,x	Subtract immediate data from ACC with Carry	1	Z, C, AC, OV, SC, CZ
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV, SC, CZ
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1Note	Z, C, AC, OV, SC, CZ
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1Note	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	1Note	Z
ORM A,[m]	Logical OR ACC to Data Memory	1Note	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1Note	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	1Note	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	1Note	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1Note	Z
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	1Note	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	1Note	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	1Note	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	1Note	C

Mnemonic	Description	Cycles	Flag Affected
Data Move			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	1Note	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	1Note	None
SET [m].i	Set bit of Data Memory	1Note	None
Branch Operation			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	1Note	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	1Note	None
SZ [m].i	Skip if bit i of Data Memory is zero	1Note	None
SNZ [m]	Skip if Data Memory is not zero	1Note	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	1Note	None
SIZ [m]	Skip if increment Data Memory is zero	1Note	None
SDZ [m]	Skip if decrement Data Memory is zero	1Note	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	1Note	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	1Note	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 Operation			
TABRD [m]	Read table (specific page) to TBLH and Data Memory	2Note	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2Note	None
ITABRD [m]	Increment table pointer TBLP first and Read table to TBLH and Data Memory	2Note	None
ITABRDL [m]	Increment table pointer TBLP first and Read table (last page) to TBLH and Data Memory	2Note	None
Miscellaneous			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	1Note	None
SET [m]	Set Data Memory	1Note	None
CLR WDT	Clear Watchdog Timer	1	TO, PDF
SWAP [m]	Swap nibbles of Data Memory	1Note	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 up to three 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 WDT” instruction the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after the “CLR WDT” instructions is 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 ← ACC + [m] + C

Affected flag(s): OV, Z, AC, C, SC

ADCM A,[m] Add ACC to Data Memory with Carry

Description: The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.

Operation: [m] ← ACC + [m] + C

Affected flag(s): OV, Z, AC, C, SC

ADD A,[m] Add Data Memory to ACC

Description: The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.

Operation: ACC ← ACC + [m]

Affected flag(s): OV, Z, AC, C, SC

ADD A,x Add immediate data to ACC

Description: The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.

Operation: ACC ← ACC + x

Affected flag(s): OV, Z, AC, C, SC

ADDM A,[m] Add ACC to Data Memory

Description: The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.

Operation: [m] ← ACC + [m]

Affected flag(s): OV, Z, AC, C, SC

AND A,[m] Logical AND Data Memory to ACC

Description: Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.

Operation: ACC ← ACC "AND" [m]

Affected flag(s): Z

AND A,x Logical AND immediate data to ACC

Description: Data in the Accumulator and the specified immediate data perform a bit wise logical AND operation. The result is stored in the Accumulator.

Operation: ACC ← ACC "AND" x

Affected flag(s): Z

ANDM A,[m] Logical AND ACC to Data Memory

Description: Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.

Operation: [m] ← ACC "AND" [m]

Affected flag(s): Z

CALL addr Subroutine call

Description: Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.

Operation: Stack ← Program Counter + 1
Program Counter ← addr

Affected flag(s): None

CLR [m] Clear Data Memory

Description: Each bit of the specified Data Memory is cleared to 0.

Operation: [m] ← 00H

Affected flag(s): None

CLR [m].i Clear bit of Data Memory

Description: Bit i of the specified Data Memory is cleared to 0.

Operation: [m].i ← 0

Affected flag(s): None

CLR WDT Clear Watchdog Timer

Description: The TO, PDF flags and the WDT are all cleared.

Operation: WDT cleared
TO ← 0
PDF ← 0

Affected flag(s): TO, PDF

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] ← $\overline{[m]}$

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 ← $\overline{[m]}$

Affected flag(s): Z

DAA [m] Decimal-Adjust ACC for addition with result in Data Memory

Description: Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition.

Operation: [m] ← ACC + 00H or
[m] ← ACC + 06H or
[m] ← ACC + 60H or
[m] ← ACC + 66H

Affected flag(s): C

DEC [m] Decrement Data Memory

Description Data in the specified Data Memory is decremented by 1.

Operation [m] ← [m] − 1

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

Description Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation ACC ← [m] − 1

Affected flag(s) Z

HALT Enter power down mode

Description This instruction stops the program execution and turns off the system clock. The contents of the Data Memory and registers are retained. The WDT and prescaler are cleared. The power down flag PDF is set and the WDT time-out flag TO is cleared.

Operation TO ← 0
PDF ← 1

Affected flag(s) TO, PDF

INC [m] Increment Data Memory

Description Data in the specified Data Memory is incremented by 1.

Operation [m] ← [m] + 1

Affected flag(s) Z

INCA [m] Increment Data Memory with result in ACC

Description Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation ACC ← [m] + 1

Affected flag(s) Z

JMP addr Jump unconditionally

Description The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction.

Operation Program Counter ← addr

Affected flag(s) None

MOV A,[m] Move Data Memory to ACC

Description The contents of the specified Data Memory are copied to the Accumulator.

Operation ACC ← [m]

Affected flag(s) None

MOV A,x Move immediate data to ACC

Description The immediate data specified is loaded into the Accumulator.

Operation ACC ← x

Affected flag(s) None

MOV [m],A Move ACC to Data Memory

Description The contents of the Accumulator are copied to the specified Data Memory.

Operation [m] ← ACC

Affected flag(s) None

NOP No operation

Description No operation is performed. Execution continues with the next instruction.

Operation No operation

Affected flag(s) None

OR A,[m] Logical OR Data Memory to ACC

Description Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″OR″ [m]

Affected flag(s) Z

OR A,x Logical OR immediate data to ACC

Description Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″OR″ x

Affected flag(s) Z

ORM A,[m] Logical OR ACC to Data Memory

Description Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.

Operation [m] ← ACC ″OR″ [m]

Affected flag(s) Z

RET Return from subroutine

Description The Program Counter is restored from the stack. Program execution continues at the restored address.

Operation Program Counter ← Stack

Affected flag(s) None

RET A,x Return from subroutine and load immediate data to ACC

Description The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address.

Operation Program Counter ← Stack
ACC ← x

Affected flag(s) None

RETI Return from interrupt

Description The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the 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) ← [m].i; (i=0~6)
[m].0 ← C
C ← [m].7

Affected flag(s): C

RLCA [m] Rotate Data Memory left through Carry with result in ACC

Description: Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation:
ACC.(i+1) ← [m].i; (i=0~6)
ACC.0 ← C
C ← [m].7

Affected flag(s): C

RR [m] Rotate Data Memory right

Description: The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7.

Operation:
[m].i ← [m].(i+1); (i=0~6)
[m].7 ← [m].0

Affected flag(s): None

RRA [m] Rotate Data Memory right with result in ACC

Description: Data in the specified Data Memory is rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation:
ACC.i ← [m].(i+1); (i=0~6)
ACC.7 ← [m].0

Affected flag(s): None

RRC [m] Rotate Data Memory right through Carry

Description: The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.

Operation:
[m].i ← [m].(i+1); (i=0~6)
[m].7 ← C
C ← [m].0

Affected flag(s): C

RRCA [m] Rotate Data Memory right through Carry with result in ACC

Description: Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation:
ACC.i ← [m].(i+1); (i=0~6)
ACC.7 ← C
C ← [m].0

Affected flag(s): C

SBC A,[m] Subtract Data Memory from ACC with Carry

Description: The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation: $ACC \leftarrow ACC - [m] - \overline{C}$

Affected flag(s): OV, Z, AC, C, SC, CZ

SBC A, x Subtract immediate data from ACC with Carry

Description: The immediate data 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: $ACC \leftarrow ACC - [m] - \overline{C}$

Affected flag(s): OV, Z, AC, C, SC, CZ

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: $[m] \leftarrow ACC - [m] - \overline{C}$

Affected flag(s): OV, Z, AC, C, SC, CZ

SDZ [m] Skip if decrement Data Memory is 0

Description: The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation:
[m] ← [m] − 1
Skip if [m]=0

Affected flag(s): None

SDZA [m] Skip if decrement Data Memory is zero with result in ACC

Description: The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation:
ACC ← [m] − 1
Skip if ACC=0

Affected flag(s): None

SET [m] Set Data Memory

Description Each bit of the specified Data Memory is set to 1.

Operation [m] ← FFH

Affected flag(s) None

SET [m].i Set bit of Data Memory

Description Bit i of the specified Data Memory is set to 1.

Operation [m].i ← 1

Affected flag(s) None

SIZ [m] Skip if increment Data Memory is 0

Description The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

SIZA [m] Skip if increment Data Memory is zero with result in ACC

Description The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

SNZ [m].i Skip if Data Memory is not 0

Description If the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m].i ≠ 0

Affected flag(s) None

SNZ [m] Skip if Data Memory is not 0

Description If 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]≠ 0

Affected flag(s) None

SUB A,[m] Subtract Data Memory from ACC

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − [m]

Affected flag(s) OV, Z, AC, C, SC, CZ

SUBM A,[m] Subtract Data Memory from ACC with result in Data Memory

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation [m] ← ACC − [m]

Affected flag(s) OV, Z, AC, C, SC, CZ

SUB A,x Subtract immediate data from ACC

Description The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − x

Affected flag(s) OV, Z, AC, C, SC, CZ

SWAP [m] Swap nibbles of Data Memory

Description The low-order and high-order nibbles of the specified Data Memory are interchanged.

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

SWAPA [m] Swap nibbles of Data Memory with result in ACC

Description The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation ACC.3~ACC.0 ← [m].7~[m].4
ACC.7~ACC.4 ← [m].3~[m].0

Affected flag(s) None

SZ [m] Skip if Data Memory is 0

Description If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation Skip if [m]=0

Affected flag(s) None

SZA [m] Skip if Data Memory is 0 with data movement to ACC

Description The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

SZ [m].i Skip if bit i of Data Memory is 0

Description If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation Skip if [m].i=0

Affected flag(s) None

TABRD [m] Read table (specific page) to TBLH and Data Memory

Description The low byte of the program code (specific page) addressed by the table pointer pair (TBLP and TBHP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

TABRDL [m] Read table (last page) to TBLH and Data Memory

Description The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

ITABRD [m] Increment table pointer low byte first and read table to TBLH and Data Memory

Description Increment table pointer low byte, TBLP, first and then the program code addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

ITABRDL [m] Increment table pointer low byte first and read table (last page) to TBLH and Data Memory

Description Increment table pointer low byte, TBLP, first and then the low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

XOR A,[m] Logical XOR Data Memory to ACC

Description Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s) Z

XORM A,[m] Logical XOR ACC to Data Memory

Description Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory.

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s) Z

XOR A,x Logical XOR immediate data to ACC

Description Data in the Accumulator and the specified immediate data perform a bitwise logical XOR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″XOR″ x

Affected flag(s) Z

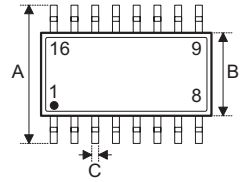
Package Information

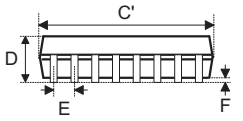
Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the Package/Carton Information.

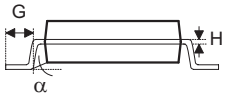
Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

- Package Information (include Outline Dimensions, Product Tape and Reel Specifications)
- The Operation Instruction of Packing Materials
- Carton information

16-pin NSOP (150mil) Outline Dimensions



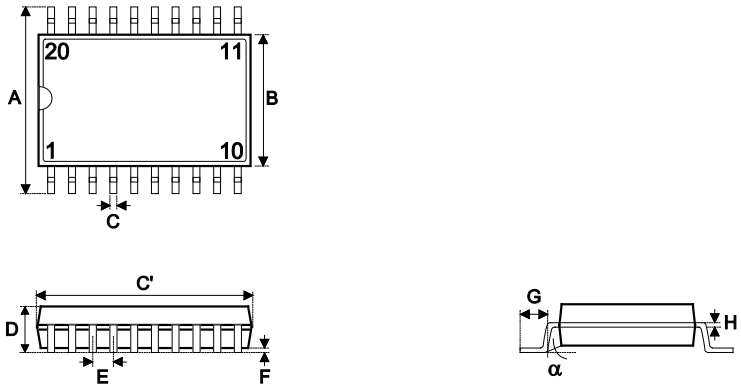




Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.236 BSC	—
B	—	0.154 BSC	—
C	0.012	—	0.020
C'	—	0.390 BSC	—
D	—	—	0.069
E	—	0.050 BSC	—
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6.000 BSC	—
B	—	3.900 BSC	—
C	0.31	—	0.51
C'	—	9.900 BSC	—
D	—	—	1.75
E	—	1.270 BSC	—
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°

20-pin SOP (300mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.406 BSC	—
B	—	0.295 BSC	—
C	0.012	—	0.020
C’	—	0.504 BSC	—
D	—	—	0.104
E	—	0.050 BSC	—
F	0.004	—	0.012
G	0.016	—	0.050
H	0.008	—	0.013
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	10.30 BSC	—
B	—	7.50 BSC	—
C	0.31	—	0.51
C’	—	12.80 BSC	—
D	—	—	2.65
E	—	1.27 BSC	—
F	0.10	—	0.30
G	0.40	—	1.27
H	0.20	—	0.33
α	0°	—	8°