HT82A836R
USB Audio MCU

Technical Document

- Application Note
  - HA0075E MCU Reset and Oscillator Circuits Application Note

Features

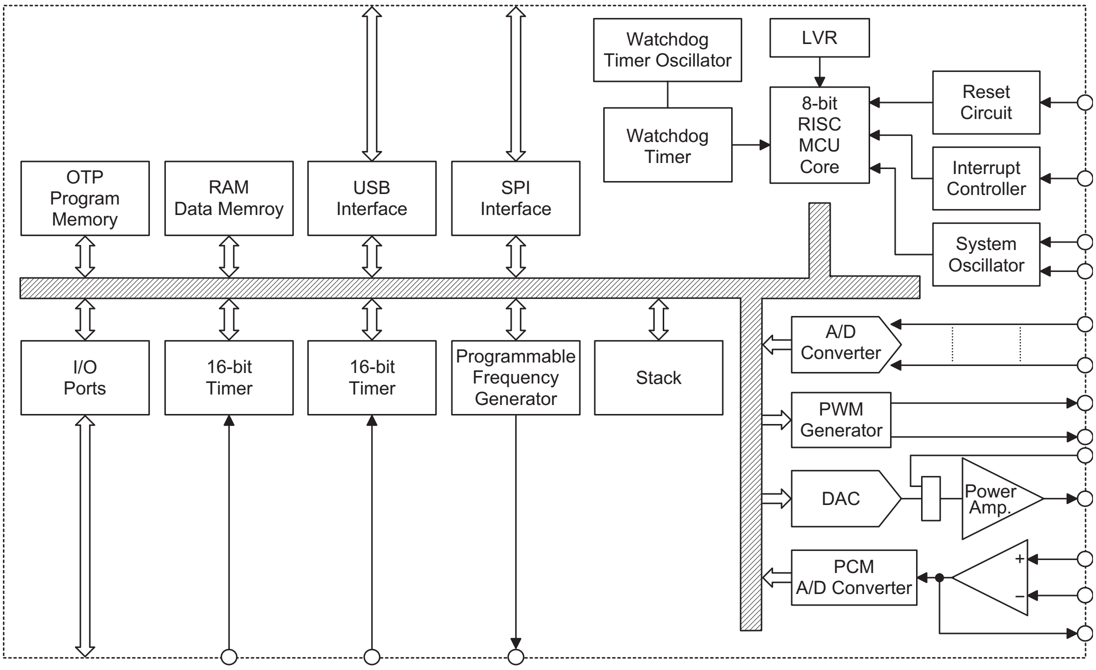
- USB 2.0 full speed compatible
- USB spec V1.1 full speed operation and USB audio device class spec V1.0
- Operating voltage at f_{SYS}= 6M/12MHz: 3.3V~5.5V
- Low voltage reset function
- Embedded high-performance 16-bit PCM ADC
- Integrated Digital PGA – Programmable Gain Amplifier
- 48kHz/8kHz sampling rate for audio playback selected by software
- 8kHz/16kHz audio recording sampling rate selected by software
- Embedded class AB power amplifier for speaker driving
- Embedded High Performance 16-bit audio DAC
- Audio playback digital volume control
- 5 endpoints supported including endpoint 0
- Supports 1 Control, 2 Interrupts and 2 Isochronous transfers
- Two hardware implemented Isochronous transfers
- Total FIFO size: 496 bytes – 8, 8, 384, 64, 32 for EP0~EP4
- 8192×16 Program Memory
- 352×8 Data Memory in two banks
- Programmable frequency divider function
- Integrated SPI hardware interface
- Port A wake-up on rising or falling transitions
- 6-channel 12-bit A/D converter
- 2-channel PWM function
- μ Law Compander
- Power-down function and wake-up reduce power consumption
- Up to of 44 bidirectional I/O lines
- Dual 16-bit programmable Timer/Event Counters with overflow interrupts
- Watchdog Timer
- 16-level subroutine nesting
- Bit manipulation instruction
- 15-bit table read instruction
- 63 powerful instructions
- All instructions executed within one or two machine cycles
- Low voltage reset function (3.0V±0.3V)
- 80-pin LQFP (10mm×10mm) package type

General Description

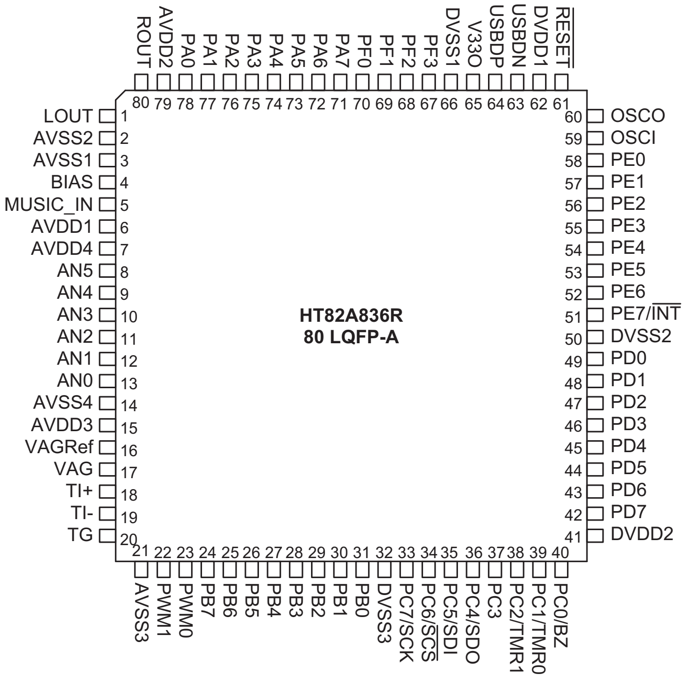
The HT82A836R is an 8-bit high-performance RISC microcontroller designed for USB phone product applications. To ensure a high level of functional integration for USB phone applications, this 8-bit microcontroller includes important features such as 16-bit PCM A/D Converter, USB transceiver, Serial Interface Engine, audio class processing unit, μ law Compander, 6-channel 12-bit ADC, 2-channel PWM and FIFO.

The DAC in the HT82A836R operates at a sampling rate of 48kHz/8kHz and the 16-bit PCM ADC operates at frequency of 8kHz/16kHz for the Microphone input, with the options selected using software. The integrated DAC also includes a digitally programmable gain amplifier with a range of –32dB to +6dB. The digital gain range of the ADC input is from 0dB to 19.5dB.

Block Diagram



Pin Assignment



Pin Description

Pin Name	I/O	Configuration Option	Description
ROUT	O	—	Right driver analog output
LOUT	O	—	Left driver analog output
AVSS4	—	—	12-bit ADC negative power supply, ground
AVSS3	—		PCM ADC negative power supply, ground
AVSS2	—	—	Audio power amplifier negative power supply, ground
AVSS1	—	—	Audio DAC negative power supply, ground
BIAS	—	—	A capacitor should be connected to ground to increase half-supply stability
MUSIC_IN	I	—	Power amplifier input signal source if register bit SELW= ″1″. The analog signal input will amplify by the power amp then output to ROUT and LOUT at the same time.
AVDD4	—	—	12-bit ADC positive power supply
AVDD3	—	—	PCM ADC positive power supply
AVDD2	—	—	Audio power amplifier positive power supply
AVDD1	—	—	Audio DAC positive power supply
AN0~AN5	I	—	12-bit ADC analog inputs
VAGRef	I	—	PCM ADC analog ground reference voltage (should left open or connected by a bypass capacitor (Ex:100pF) to ground)
VAG	O	—	PCM ADC analog ground voltage (should connected by a bypass capacitor (Ex:10μF) to ground)
TI+	I	—	OP AMP non-inverting input
TI-	I	—	OP AMP inverting input
TG	O	—	OP AMP gain setting output
PWM0~PWM1	O	—	PWM outputs
PA0~PA7	I/O	Pull-high Wake-up NMOS/CMOS Output	Bidirectional 8-bit input/output port. Each bit can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on this port have pull-high resistors. The output structure can be either NMOS or CMOS types determined via configuration option.
PB0~PB7	I/O	Pull-high Wake-up	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on this port have pull-high resistors.
PC0/BZ PC1/TMR0 PC2/TMR1 PC3 PC4/SDO PC5/SDI PC6/$\overline{\text{SCS}}$ PC7/SCK	I/O	Pull-high Wake-up	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on this port have pull-high resistors. Pin PC0 is shared with the buzzer pin BZ. Pins PC1/PC2 are shared with timer input pins TMR0/TMR1. Pins PC4/PC5 are shared with Serial Interface pins SDO/SDI. Pin PC6 is shared with the Serial Interface Slave Select pin. Pin PC7 is shared with the Serial Interface clock signal.
PD0~PD7	I/O	Pull-high Wake-up	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on this port have pull-high resistors.

Pin Name	I/O	Configuration Option	Description
PE0~PE6 PE7/$\overline{\text{INT}}$	I/O	Pull-high Wake-up	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on this port have pull-high resistors. PE7 is shared with external interrupt input $\overline{\text{INT}}$.
PF0~PF3	I/O	Pull-high Wake-up	Bidirectional 4-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on this port have pull-high resistors.
DVDD2	—	—	Positive digital power supply
DVSS2	—	—	Negative digital & I/O power supply, ground
DVDD1	—	—	Positive digital power supply
DVSS1	—	—	Negative digital power supply, ground
OSCI OSCO	I O	—	OSCI, OSCO are connected to an 6MHz or 12MHz crystal/resonator, determined by software instructions, for the internal system clock.
$\overline{\text{RESET}}$	I	—	Schmitt trigger reset input, active low
USBDN	I/O	—	USBD- line
USBDP	I/O	—	USBD+ line
V33O	O	—	3.3V regulator output

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

I_{OL} Total 150mA

Total Power Dissipation 500mW

Storage Temperature ....–50°C to 125°C

Operating Temperature..........................–40°C to 85°C

I_{OH} Total..–100mA

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

D.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	—	3.3	5.0	5.5	V
I_{DD1}	Operating Current	5V	No load, f_{SYS}=12MHz ADC on, DAC on	—	12	—	mA
I_{DD2}	Operating Current	5V	No load, f_{SYS}=12MHz ADC off, DAC off	—	8	—	mA
I_{SUS}	Suspend Current	5V	No load, system HALT, USB transceiver and 3.3V regulator on	—	330	—	μA
V_{IL1}	Input Low Voltage for I/O Ports	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports	—	—	$0.7V_{DD}$	—	V_{DD}	V
V_{IL2}	Input Low Voltage ($\overline{\text{RESET}}$)	—	—	0	—	$0.4V_{DD}$	V
V_{IH2}	Input High Voltage ($\overline{\text{RESET}}$)	—	—	$0.8V_{DD}$	—	V_{DD}	V
I_{OL}	I/O Port Sink Current	5V	V_{OL}=$0.1V_{DD}$	—	5	—	mA

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{OH}	I/O Port Source Current	5V	V_{OH}=0.9V_{DD}	—	–5	—	mA
R_{PH}	Pull-high Resistance	5V	—	30	40	80	kΩ
V_{LVR0}	Low Voltage Reset	5V	—	2.7	3.0	3.3	V
V_{V33O}	3.3V Regulator Output	5V	I_{V33O}= –5mA	3.0	3.3	3.6	V
DAC+Power Amp: Test Condition: Measurement bandwidth 20Hz to 20kHz, f_S= 48kHz. Line output series capacitor with 220μF.							
THD+N	THD+N$^{(Note)}$	5V	4Ω load	—	–30	—	dB
			8Ω load	—	–35	—	dB
SNR_{DA}	Signal to Noise Ratio Note1	5V	4Ω load	—	81	—	dB
			8Ω load	—	82	—	dB
DR	Dynamic Range	5V	4Ω load	—	87	—	dB
			8Ω load	—	88	—	dB
POUT	Output Power	5V	4Ω load, THD=10%	—	400	—	mW/ch
			8Ω load, THD=10%	—	200	—	mW/ch
PCM ADC:							
SNR_{AD}	Signal to Noise Ratio	5V	—	—	77	—	dB
VAG	Reference Voltage	5V	—	—	2.0	—	V
V_{PEAK}	Peak Single Frequency Tone Amplitude without Clipping	5V	—	—	1.575	—	V_{PK}

Note: Sine wave input at 1kHz, –6dB

A.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock (Crystal OSC)	5V	—	0.4	—	12	MHz
t_{WDTOSC}	Watchdog Oscillator Period	5V	—	—	100	—	μs
t_{RES}	Reset Low Pulse Width	—	—	1	—	—	μs
t_{SST}	System Start-up Timer Period	—	—	—	1024	—	*t_{SYS}
t_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
t_{ADC}	A/D Conversion Time	—	—	—	80	—	t_{AD}
t_{ADCS}	A/D Sampling Time	—	—	—	32	—	t_{AD}

Note: *t_{SYS}=1/f_{SYS}

System Architecture

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

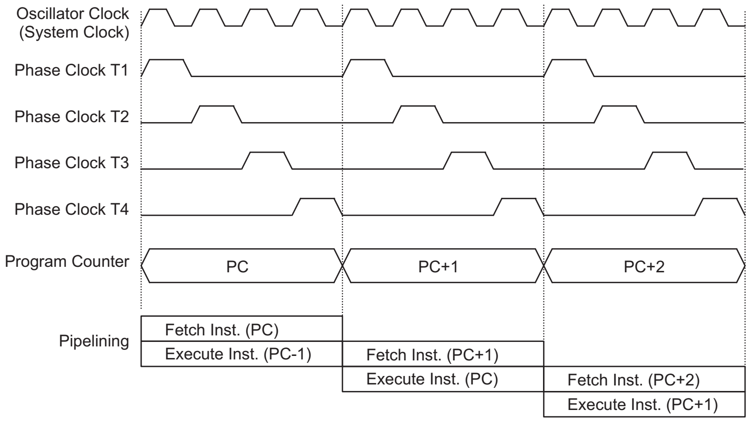
Clocking and Pipelining

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

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

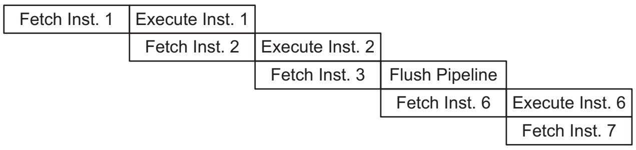
Program Counter

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as ″JMP″ or ″CALL″, that demand a jump to a non-consecutive Program Memory address. Note that the Program Counter width varies with the Program Memory capacity depending upon which device is selected.



System Clocking and Pipelining

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



Instruction Fetching

However, it must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by user.

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

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

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

Stack

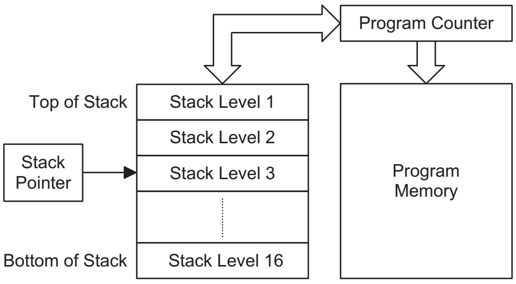
This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has 16 levels and is neither part of the data nor part of the program space, and can neither be read from nor written to. The activated level is indexed by the Stack Pointer, SP, which can also neither be read from nor written to. 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.

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

Program Counter

Note: PC12~PC8: Current Program Counter bits
@7~@0: PCL bits
#12~#0: Instruction code address bits
S12~S0: Stack register bits



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 contains One-Time Programmable, OTP, memory where users can program their application code into the device. By using the appropriate programming tools, OTP devices offer users the flexibility to freely develop their applications, which may be useful during debug or for products requiring frequent upgrades or program changes. OTP devices are also applicable for use in applications that require low or medium volume production runs.

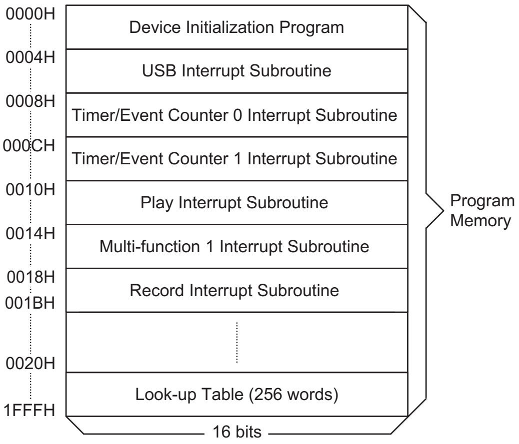
Organisation

The Program Memory has a capacity of 8K by 16 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by separate table pointer registers.

Special Vectors

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

- Location 000H
  This vector is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.
- Location 004H
  This vector is used by the USB interrupt. If a USB interrupt occurs, the program will jump to this location and begin execution if the USB interrupt is enabled and the stack is not full.
- Location 008H
  This vector is used by the Timer/Event Counter 0. If a counter overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 00CH
  This vector is used by the Timer/Event counter 1. If a counter overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 010H
  This vector is used by the play interrupt service program. If play data occurs, the program will jump to this location and begin execution if the play interrupt is enabled and the stack is not full.
- Location 014H
  This vector is used by the Multi-function interrupt. If an interrupt results from a serial interface interrupt, an end of 12-bit A/D conversion cycle or an external interrupt, the program will jump to this location and begin execution if the relevant interrupt is enabled and the stack is not full.
- Location 018H
  This area is used by the Record interrupt. If record data occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.



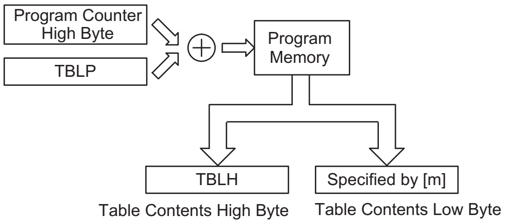
Program Memory Structure

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 pointers must first be setup which point to the data in the Program Memory which is to be read. In this device there are two table pointers, the low byte pointer, TBLP and the high byte pointer, TBHP. However, the high byte pointer, TBHP, can only be used if it is enabled using configuration options. Using both table pointers enables any area in the Program Memory to be addressed while if only the low byte pointer, TBLP, is used then only the present page or last page can be addressed.

If the configuration options do not enable the high byte pointer, then after setting up the low table pointer, TBLP, the table data can be retrieved from the current Program Memory page or last Program Memory page using the ″TABRDC[m]″ or ″TABRDL [m]″ instructions, respectively. When these instructions are executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register.

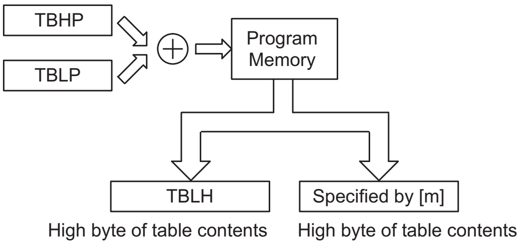
The following diagram illustrates the addressing/data flow of the look-up table using the single table address pointer TBLP:



Single Address Pointer Look-up Table

If the configuration options enable the high table pointer, TBHP, then this register together with the low table pointer, TBLP, can be used together as a pair to point to any located in the Program Memory. After setting up both the low and high byte table pointers, the table data can then be retrieved from any area of Program Memory using the ″TABRDC [m]″ instruction or from the last page of the Program Memory using the ″TABRDL [m]″ instruction. When either of these instructions are executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register.

The following diagram illustrates the addressing/data flow of the look-up table using the dual table address pointers TBLP and TBHP:



Dual Address Pointer Look-up Table

Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the micro controller using the single table data pointer, TBLP. 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 ″1F00H″ which refers to the start address of the last page within the 8K Program Memory of device. The table pointer is setup here to have an initial value of ″06H″. This will ensure that the first data read from the data table will be at the Program Memory address ″1F06H″ or 6locations 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.

```
tempreg1     db      ?        ; temporary register #1
tempreg2     db      ?        ; temporary register #2
             :
             :
mov          a,06h            ; initialise table pointer - note that this address
                              ; is referenced
mov          tblp,a           ; to the last page or present page
             :
             :
tabrdl       tempreg1         ; transfers value in table referenced by table pointer
                              ; to tempregl
                              ; data at prog. memory address "1F06H" 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 "1F05H" 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          1F00h            ; sets initial address of last page
dc           00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
             :
             :
```

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

Table Location

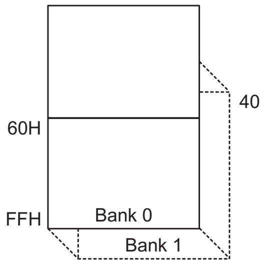
Note: PC12~PC8: Current Program Counter bits
TBHP register bit4~bit0 when TBHP option is enabled
@7~@0: Table Pointer TBLP bits

Data Memory

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

Organisation

The RAM Data Memory is subdivided into two banks, known as Bank 0 and Bank 1, all of which are implemented in 8-bit wide RAM. The Bank 0 Data Memory is subdivided into two sections, the Special Purpose Data Memory and the General Purpose Data Memory. The start address of the Bank 0 Data Memory is the address 00H and the last Data Memory address is FFH. The Bank 1 Data Memory consists only of General Purpose Data Memory. The start address of the Bank 1 Data Memory is the address 40H and the last Data Memory address is FFH. Selection of which Bank is to be used is implemented using the Bank Pointer.



Data Memory Structure

Note: Most of the RAM Data Memory bits can be directly manipulated using the ″SET [m].i″ and ″CLR [m].i″ instructions with the exception of a few dedicated bits. The RAM Data Memory can also be accessed through the Memory Pointer registers MP0 and MP1.

General Purpose Data Memory

All microcontroller programs require an area of read/write memory where temporary data can be stored and retrieved for use later. It is this area of RAM memory that is known as General Purpose Data Memory. This area of Data Memory is fully accessible by the user program for both read and write operations. By using the ″SET [m].i″ and ″CLR [m].i″ instructions individual bits can be set or reset under program control giving the user a large range of flexibility for bit manipulation in the Data Memory. As the General Purpose Data Memory exists in two banks, Bank 0 and Bank1, it is necessary to first ensure that the Bank Pointer is properly set to the correct value before accessing the General Purpose Data Memory. When the Bank Pointer is set to the value 00H, data from Bank 0 will be accessed and when set to the value 01H data from Bank 1 will be accessed. Note that Bank 1 must be accessed indirectly using the Memory Pointer MP1 and the Indirect Addressing Register IAR1.

Special Purpose Data Memory

This area of Data Memory, is located in Bank 0, where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers can be read from and written to but some are protected and are read only, the details of which are located under the relevant Special Function Register section. Note that for locations that are unused, any read instruction to these addresses will return the value ″00H″.

	Bank 0 Special Register
00H	IAR0
01H	MP0
02H	IAR1
03H	MP1
04H	BP
05H	ACC
06H	PCL
07H	TBLP
08H	TBLH
09H	WDTS
0AH	STATUS
0BH	INTC0
0CH	TMR0H
0DH	TMR0L
0EH	TMR0C
0FH	TMR1H
10H	TMR1L
11H	TMR1C
12H	PA
13H	PAC
14H	PB
15H	PBC
16H	PC
17H	PCC
18H	PD
19H	PDC
1AH	PE
1BH	PEC
1CH	PF
1DH	PFC
1EH	INTC1
1FH	TBHP
20H	USC
21H	USR
22H	UCC
23H	AWR
24H	STALL
25H	SIES
26H	MISC
27H	SETIO
28H	FIFO0
29H	FIFO1
2AH	FIFO2
2BH	FIFO3
2CH	FIFO4
2DH	DAC_LIMIT_L
2EH	DAC_LIMIT_H
2FH	DAC_WR
30H	PGA_CTRL
31H	PFDC
32H	PFDD
33H	OPER_MODE
34H	MODE_CTRL
35H	SBCR
36H	SBDR
37H	RECORD_IN_L
38H	RECORD_IN_H
39H	▨
3AH	PLAY_DATAL_L
3BH	PLAY_DATAL_H
3CH	PLAY_DATAR_L
3DH	PLAY_DATAR_H
3EH	RECORD_DATA_L
3FH	RECORD_DATA_H
40H	ADRL
41H	ADRH
42H	ADCR
43H	ACSR
44H	PA_WAKE_CTRL
45H	PWMC
46H	PWM0
47H	PWM1
48H	MFI1C
49H	USB_STATE
4AH	USVC
4BH~5FH	▨

▨ : Unused, read as "00"

Special Purpose Data Memory Structure

Special Function Registers

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

Indirect Addressing Register – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses 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 Pointer, MP0 or MP1. Acting as a pair, IAR0 and MP0 can together only access data from Bank 0, while the IAR1 and MP1 register pair can access data from both Bank 0 and Bank 1. 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 Pointer – MP0, MP1

For all devices, two Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to, is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Bank 0 only, while MP1 and IAR1 are used to access data from both Bank 0 and Bank 1.

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

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

start:
                  mov a,04h          ; setup size of block
                  mov block,a
                  mov a,offset adres1 ; Accumulator loaded with first RAM address
                  mov mp0,a          ; setup 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.

Bank Pointer – BP

The Data Memory is divided into two Banks, known as Bank 0 and Bank 1. Selecting the required Data Memory area is achieved using the Bank Pointer. If data in Bank 0 is to be accessed, then the BP register must be loaded with the value ″00″, while if data in Bank 1 is to be accessed, then the BP register must be loaded with the value ″01″.

Using Memory Pointer MP0 and Indirect Addressing Register IAR0 will always access data from Bank 0, irrespective of the value of the Bank Pointer.

The Data Memory is initialised to Bank 0 after a reset, except for the WDT time-out reset in the Power Down Mode, in which case, the Data Memory bank remains unaffected. Directly addressing the Data Memory will always result in Bank 0 being accessed irrespective of the value of the Bank Pointer.

Accumulator – ACC

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

Program Counter Low Register – PCL

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

Look-up Table Registers – TBLP, TBLH, TBHP

These three special function registers are used to control operation of the look-up table, which is stored in the Program Memory. TBLP is the table low byte pointer and indicates the lowest 8-bit address location where the table data is located. TBHP is the table high byte pointer and indicates the highest bit address location where the table data is located. The TBHP high byte table pointer can only be used if its configuration option is selected. The table pointers must be setup before any table read commands are executed. Their value can be changed, for example using the ″INC″ or ″DEC″ instructions, allowing for easy table data pointing and reading. If the TBHP configuration is enabled, then the TBLP and TBHP register pair can be used together with the ″TABRDC″ instruction to point directly to any location in the program memory. TBLH is the location where the higher order byte of the table data is stored after a table read data instruction has been executed. The lower order table data byte is transferred to a user defined location.

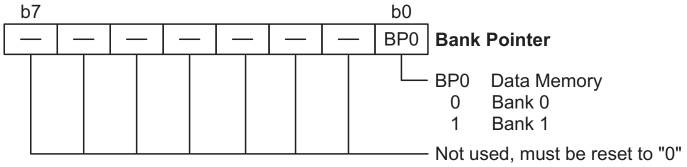
Watchdog Timer Register – WDTS

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

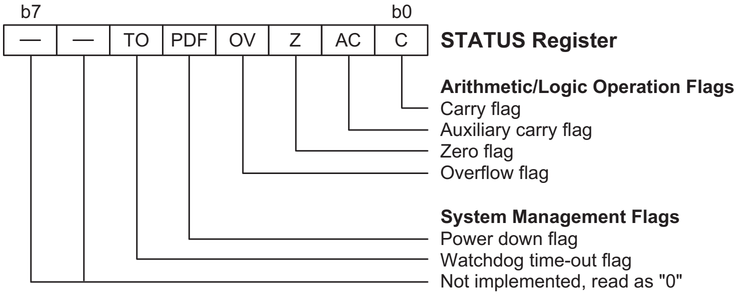
Status Register – STATUS

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, opera-



Bank Pointer



Status Register

tions related to the status register may give different results due to the different instruction operations. The TO flag can be affected only by a system power-up, a WDT time-out or by executing the ″CLR WDT″ or ″HALT″ instruction. The PDF flag is affected only by executing the ″HALT″ or ″CLR WDT″ instruction or during a system power-up.

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

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

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

Interrupt Control Registers – INTC0, INTC1, MFI1C

These three 8-bit registers, known as the INTC0, INTC1 and MFI1C control the operation of the interrupts. By setting various bits within this register using standard bit manipulation instructions, the enable/disable function of the all interrupts can be independently controlled. A master interrupt bit within this register, the EMI bit, acts like a global enable/disable and is used to set all of the interrupt enable bits on or off. This bit is cleared when an interrupt routine is entered to disable further interrupt and is set by executing the ″RETI″ instruction. Note in situations where other interrupts may require servicing within present interrupt service routines, the EMI bit can be manually set by the program after the present interrupt service routine has been entered.

Timer/Event Counter Registers – TMRL/TMRH, TMRC

The device contains two 16-bit Timer/Event Counters. Each Timer/Event Counter has an associated register pair, known as TMR0L/TMR0H and TMR1L/TMR1H which are the locations where the timer′s 16-bit value is located. Each timer also has an associated control register, known as TMR0C and TMR1C which contains the setup information for the associated timer.

Input/Output Ports and Control Registers

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

Port A Wake-up Control Register – PA_WAKE_CTRL

This register is used to select the edge type that triggers the wake-up function on the Port A pins. If the configuration options select some or all of the Port A pins to have a wake-up function then this register can be used to select either whether the active edge is a negative or positive transition. Only Port A is allowed this selection.

Pulse Width Modulator Registers – PWM0, PWM1, PWMC

The device 2 integrated Pulse Width Modulators. Each one has its own independent register, known as PWM0 and PWM1. The 8-bit contents of each of these registers define the duty cycle value for the modulation cycle of the corresponding pulse width modulator. The PWMC is the control register for the PWM functions and controls the mode selection and on/off function.

A/D Converter Registers – ADRL, ADRH, ADCR, ACSR

The device contains a single 6-channel 12-bit A/D converter. The correct operation of the A/D requires the use of two data registers, a control register and a clock source register. There are two data registers, a high byte data register known as ADRH, and a low byte data register known as ADRL. These are the register locations where the digital value is placed after the completion of an analog to digital conversion cycle. The channel selection and configuration of the A/D converter is setup via the control register ADCR while the A/D clock frequency is defined by the clock source register, ACSR.

USB Registers

The device contains an internal USB port which is controlled via several registers. These are used to setup the USB operation, the external pins, error handling etc. As this register list is too numerous to list here details can be found in the relevant USB description.

PFD Registers – PFDC, PFDD

The device contains a fully integrated Programmable Frequency Driver otherwise known as the PFD. Two registers control the overall operation of the PFD to determine the output frequency and the function enable/disable.

Other Registers

The device contains several other special function registers for control of various internal functions. As their functional description is too detailed to be described here their details will be provided in the relevant functional description section.

Input/Output Ports

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

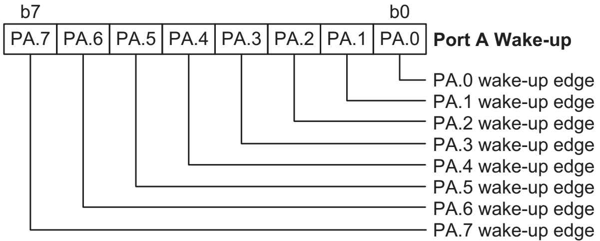
The microcontroller provides a maximum of 44 bidirectional input/output lines labeled with port names PA, PB, PC, PD, PE and PF. These I/O ports are mapped to the Data Memory with addresses as shown in the Special Purpose Data Memory table. Seven of these I/O lines can be used for input and output operations and one line as an input only. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction ″MOV A,[m]″, where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Pull-high Resistors

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

Port A Wake-up

If the HALT instruction is executed, the device will enter the Power Down Mode, where the system clock will stop resulting in power being conserved, a feature that is important for battery and other low-power applications. Various methods exist to wake-up the microcontroller, one of which is a logical transition on one of the Port A~Port F pins from high to low. After a HALT instruction forces the microcontroller into entering the Power Down Mode, the device will remain idle or in a low-power state until the logic condition of the selected wake-up pin on Port A~Port F changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Note that each pin on Port A~Port F can be selected individually using configuration options to have this wake-up feature. Additionally Port A pins have an additional selection allowing their wake-up function to be either negative or positive edge triggered. This option is provided using the PA_WAKE_CTRL register. Only Port A pins have this feature, the wake-up pins on the other ports are only negative edge triggered.



PA.x	Active edge
1	low to high
0	high to low

Port A Wake-up

I/O Port Control Registers

Each I/O port has its own control register PAC, PBC, PCC, PDC, PEC and PFC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each of the I/O ports is directly mapped to a bit in its associated port control register.

For the I/O pin to function as an input, the corresponding bit of the control register must be written as a ″1″. This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a ″0″, the I/O pin will be setup as a CMOS output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

Pin-shared Functions

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

- Serial Interface
  The serial interface pins SDO, SDI, $\overline{\text{SCS}}$ and SCK are pin-shared with the I/O pins PC4, PC5, PC6 and PC7. For applications not requiring serial interface, the pin-shared pins can be used as a normal I/O pin.

- External Interrupt
  The external interrupt pin $\overline{\text{INT}}$ is pin-shared with the I/O pin PE7. For applications not requiring an external interrupt input, the pin-shared external interrupt pin can be used as a normal I/O pin, however to do this, the external interrupt enable bits in the MFI1C register must be disabled.

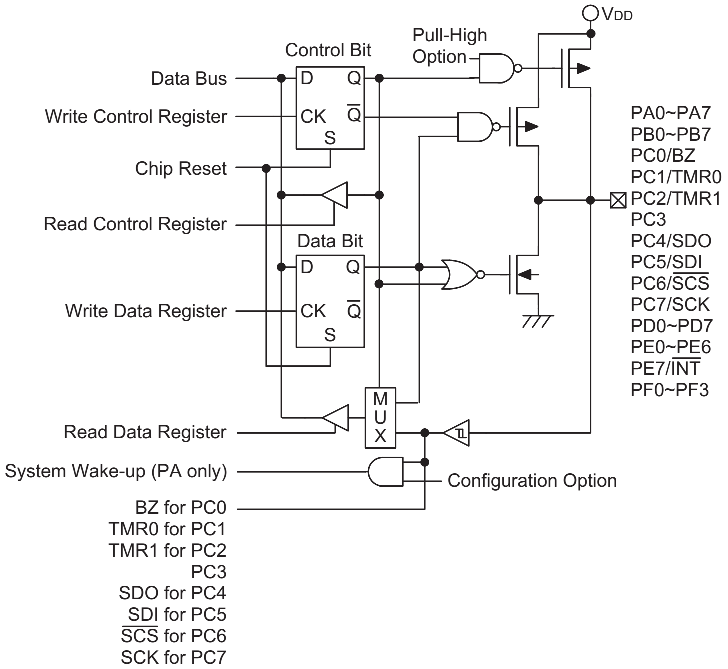
- External Timer/Event Counter Input
  The external timer pins TMR0/TMR1 are pin-shared with the I/O pins PC1/PC2. If these shared pins are to be used as a Timer/Event Counter inputs, then the Timer/Event Counter must be configured to be in the Event Counter or Pulse Width Measurement Mode. This is achieved by setting the appropriate bits in the relevant timer/Event Counter Control Register. The pin must also be setup as an input by setting the appropriate bit in the Port Control Register Pull-high resistor options can also be selected via the appropriate port pull-high configuration option. If the shared pin is to be used as a normal I/O pin, then the external timer input function must be disabled, by ensuring that the corresponding Timer/Event Counter is configured to be in the Off Mode or Timer Mode.

- PFD Output
  The device contains a PFD function whose single output is pin-shared with PC0. The output function of this pin is chosen via software. Note that the corresponding bit of the port control register, PCC.0, must setup the pin as an output to enable the PFD output. If the PCC port control register has setup the pin as an input, then the pin will function as a normal logic input with the usual pull-high option, even if the PFD configuration option has been selected.

- SPI Interface
  The device contains an internal SPI interface whose pins are shared with I/O pins PC4~PC7. The SPI Interface control register, SBCR, is used to determine if these pins are to be used as normal I/O pins or as SPI Interface pins.

I/O Pin Structures

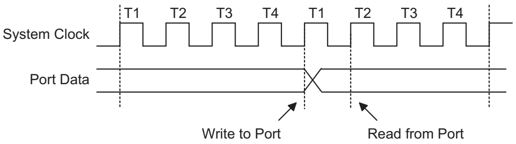
The following diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins.



Input/Output Ports

Programming Considerations

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

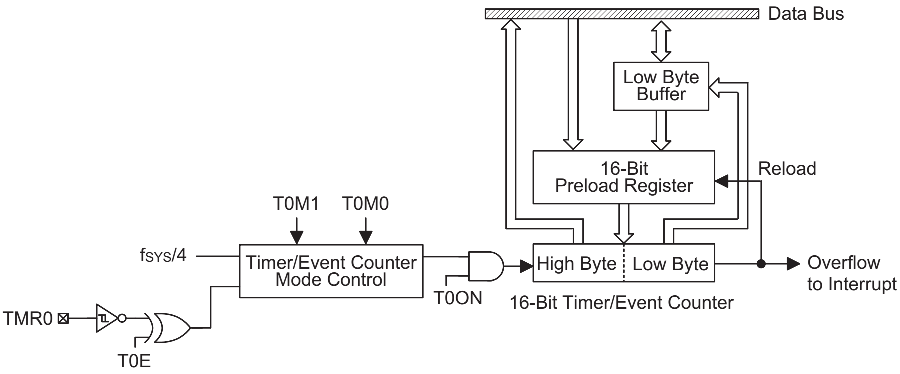


Read/Write Timing

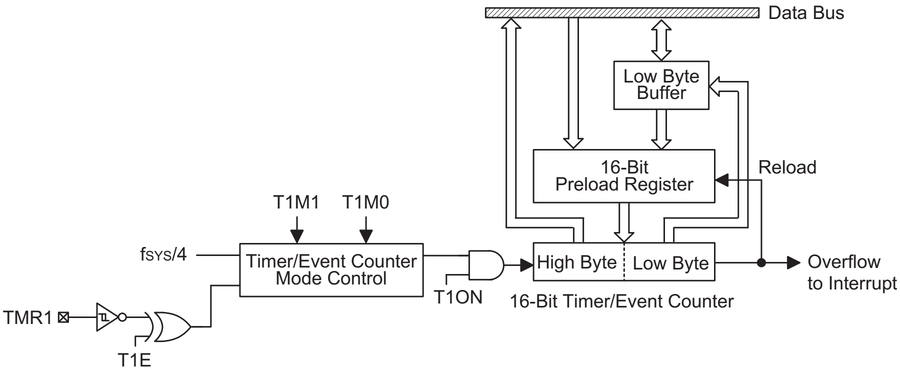
Timer/Event Counters

The provision of timers form an important part of any microcontroller giving the designer a means of carrying out time related functions. The device contains two internal 16-bit count-up timers each of which has three operating modes. The timer can be configured to operate as a general timer, external event counter or as a pulse width measurement device.

There are three registers related to each of the Timer/Event Counters, these are a pair or timer registers and a control register. The register pairs TMR0L/TMR0H and TMR1L/TMR1H contain the 16-bit timing value. Writing to these register pairs places an initial starting value in the Timer/Event Counter preload registers while reading them retrieves the contents of the Timer/Event Counter. The TMR0C and TMR1C registers are the Timer/Event Counter control registers, which define the timer options, and determines how the timers are to be used. The timer clock source can be configured to come from the internal system clock source or from an external clock on shared pin PC1/TMR0 and PC2/TMR1.



Timer/Event Counter 0 Structure



Timer/Event Counter 1 Structure

Configuring the Timer/Event Counter Input Clock Source

The Timer/Event Counter's clock can originate from various sources. The system clock source is used when the Timer/Event Counter is in the timer mode or in the pulse width measurement mode. An external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on the external timer pin, TMR0 or TMR1. Depending upon the condition of the T0E or T1E bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.

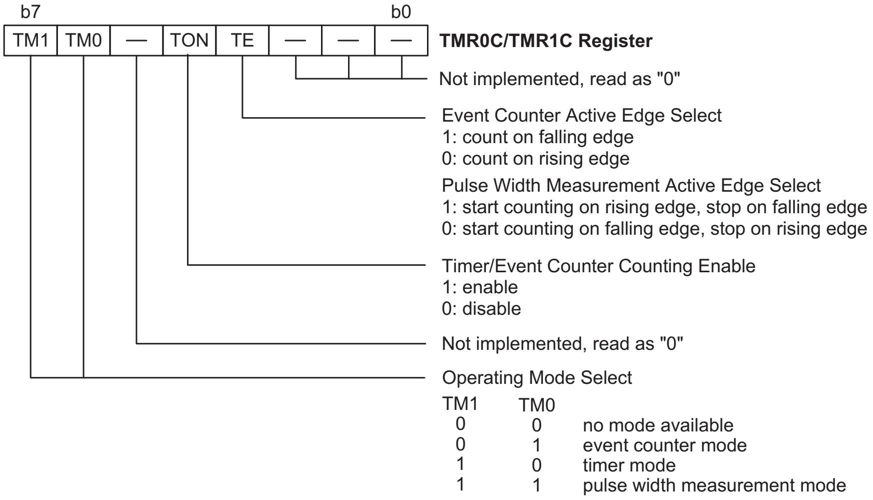
Timer Registers – TMR0H/TMR0L, TMR1L/TMR1H

The timer register are special function registers located in the Special Purpose Data Memory and is the place where the actual timer values are stored. These registers exist in pairs and are known as TMR0L/TMR0H and TMR1L/TMR1H. The value in these timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFFFH at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting. To achieve a maximum full range count of FFFFH the preload register must first be cleared to all zeros. 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 switched off and data is written to its preload register, this data will be immediately written into the actual timer register. However, if the Timer/Event 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 timer register the next time an overflow occurs.

Note that writing data to the lower byte 8-bit registers, TMR0L/TMR1L, will only put the written data into an internal lower-order byte 8-bit buffer, while writing to the high byte 8-bit registers, TMR0H/TMR1H will transfer the specified data and the contents of the lower-order byte buffer into both the TMR0/1H and TMR0/1L registers. The Timer/Event Counter preload register is modified by writing to the TMR0/1H registers. Reading the TMR0H/TMR1H registers will latch the contents of both the TMR0H/TMR1H and the TMR0L/TMR1L counters to the destination and the lower-order byte buffer. However reading the TMR0L/TMR1L will only read the contents of the low byte buffer.

Timer Control Register – TMR0C, TMR1C

The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of their control register, which has the name TMR0C and TMR1C. It is the Timer Control Register together with their corresponding timer register pair that control the full operation of the Timer/Event Counter. Before the Timer/Event Counter can be used, it is essential that the Timer Control Register pair is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialisation. To choose which of the three modes the Timer/Event Counter is to operate in, either in the timer mode, the event counting mode or the pulse width measurement mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair TM1/TM0, must be set to the required logic levels. The Timer/Event Counter on/off bit, which is bit 4 of the Timer Control Register and known as TON, provides the basic on/off control of the Timer/Event Counter. Setting the bit high allows the Timer/Event Counter to run, clearing the bit stops it running. If the Timer/Event Counter is in the event count or pulse width measurement mode, the active transition edge level type is selected by the logic level of bit 3 of the Timer control Register which is known as TE.



TMR0C/TMR1C Register

Not implemented, read as "0"

Event Counter Active Edge Select
1: count on falling edge
0: count on rising edge

Pulse Width Measurement Active Edge Select
1: start counting on rising edge, stop on falling edge
0: start counting on falling edge, stop on rising edge

Timer/Event Counter Counting Enable
1: enable
0: disable

Not implemented, read as "0"

Operating Mode Select

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

Timer/Event Counter 0/1 Control Register

Configuring the Timer Mode

In this mode, the timer can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the counter overflows. To operate in this mode, bits TM1 and TM0 of the TMRC register must be set to 1 and 0 respectively. In this mode, the internal clock is used as the timer clock. The input clock frequency to the timer is $f_{SYS}/4$. The timer-on bit, TON, must be set high to enable the timer to run. Each time an internal clock high to low transition occurs, the timer increments by one. When the timer is full and overflows, the timer will be reset to the value already loaded into the preload register and continue counting. If the timer interrupt is enabled, an interrupt signal will also be generated. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, INTC0, is reset to zero.

Configuring the Event Counter Mode

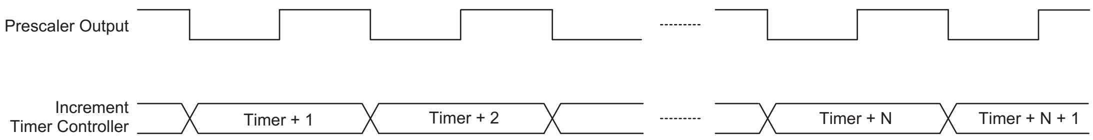
In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the Timer/Event Counter. To operate in this mode, the Operating Mode Select bit pair in the Timer Control Register must be set to the correct value. In this mode the external timer pin is used as the Timer/Event Counter clock source. After the other bits in the Timer Control Register have been setup, the enable bit, 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, 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 Active Edge Select bit 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 Interrupt Control Register, INTC0, 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 Power Down Mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.

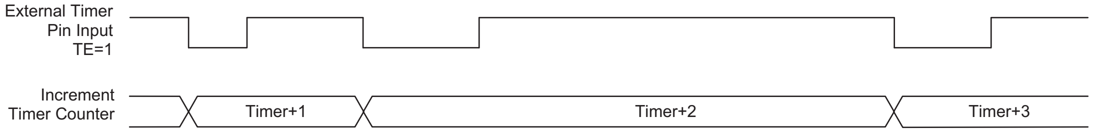
Configuring the Pulse Width Measurement Mode

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 in the Timer Control Register must be set to the correct value. In this mode the internal clock, $f_{SYS}/4$, is used as the Timer/Event Counter clock. After the other bits in the Timer Control Register have been setup, the enable bit, 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, 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 Measurement Mode, the



Timer Mode Timing Chart



Event Counter Mode Timing Chart

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 external timer pin. As the enable bit has now been reset, any further transitions on the external timer pin will be ignored. Not until the enable bit is again set high by the program can the timer begin further pulse width measurements. In this way, single shot pulse measurements can be easily made.

It should be noted that in this mode the 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 Interrupt Control Register, INTC0, 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 a pulse width measurement 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 Pulse Width Measurement Mode, the second is to ensure that the port control register configures the pin as an input.

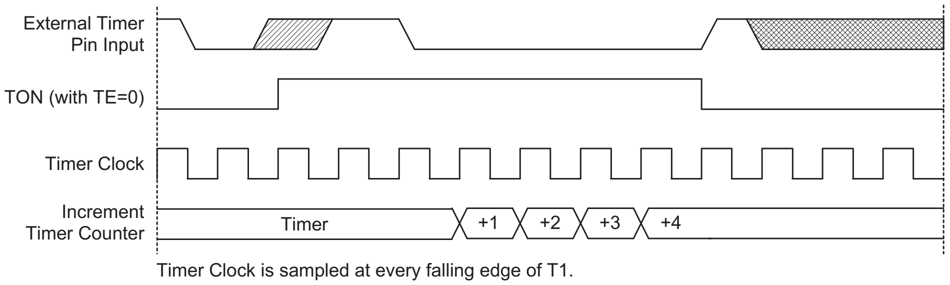
I/O Interfacing

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

Programming Considerations

When configured to run in the timer mode, the $f_{SYS}/4$ is used as the timer clock source and is therefore synchronised with the overall operation of the microcontroller. In this mode, when the appropriate timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the pulse width measurement mode, the $f_{SYS}/4$ 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 synchronized with the internal timer clock, the microcontroller will only see this external event when the next timer clock pulse arrives. As a result there may be small differences in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to be in the event counting mode which again is an external event and not synchronised with the $f_{SYS}/4$ clock.

When the Timer/Event Counter is read or if data is written to the preload registers, 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 register before the timer is switched on; this is because after power-on the initial value of the timer register is unknown. After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the timer control register. Note that setting the timer enable bit high to turn the timer on, should only be executed after the timer mode



Pulse Width Measure Mode Timing Chart

bits have been properly setup. Setting the timer enable bit high together with a mode bit modification, may lead to improper timer operation if executed as a single timer control register byte write instruction. When the Timer/Event counter overflows, its corresponding interrupt request flag in the interrupt control register will be set. If the timer interrupt is enabled this will in turn generate an interrupt signal. However irrespective of whether the timer interrupt is enabled or not, a Timer/Event counter overflow will also generate a wake-up signal if the device is in a Power-down condition. This situation may occur if the Timer/Event Counter is in the Event Counting Mode and if the external signal continues to change state. In such a case, the Timer/Event Counter will continue to count these external events and if an overflow occurs the device will be woken up from its Power-down condition. To prevent such a wake-up from occurring, the timer interrupt request flag should first be set high before issuing the HALT instruction to enter the Power Down Mode.

Timer Program Example

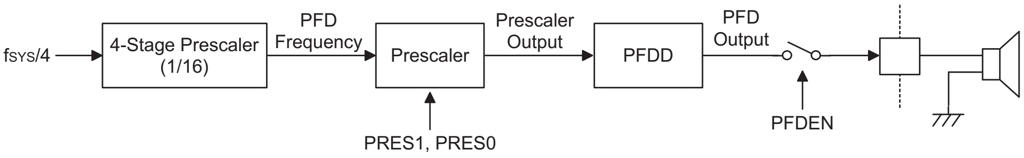
This program example shows how the Timer/Event Counter registers are setup, along with how the interrupts are enabled and managed. Note how the Timer/Event Counter is turned on, by setting bit 4 of the Timer Control Register. The Timer/Event Counter can be turned off in a similar way by clearing the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

```
org 04h             ; usb interrupt vector
jmp usbint
reti
org 08h             ; Timer/Event Counter 0 interrupt vector
jmp tmr0int         ; jump here when Timer 0 overflows
:
org 20h             ; main program
;internal Timer/Event Counter interrupt routine
tmr0int:
:
; Timer/Event Counter 0 main program placed here
:
reti
:
:
begin:
;setup Timer 0 registers
mov a,09bh          ; setup preload value – timer0 counts from this value to FFFFH
mov tmr0l,a;
mov a,00h
mov tmr0h,a;
mov a,080h          ; setup Timer 0 control register
mov tmr0c,a         ; timer mode
; setup interrupt register
mov a,005h          ; enable master interrupt and timer interrupt
mov intc0,a
set tmr0c.4         ; start Timer 0 – note mode bits must be previously setup
```

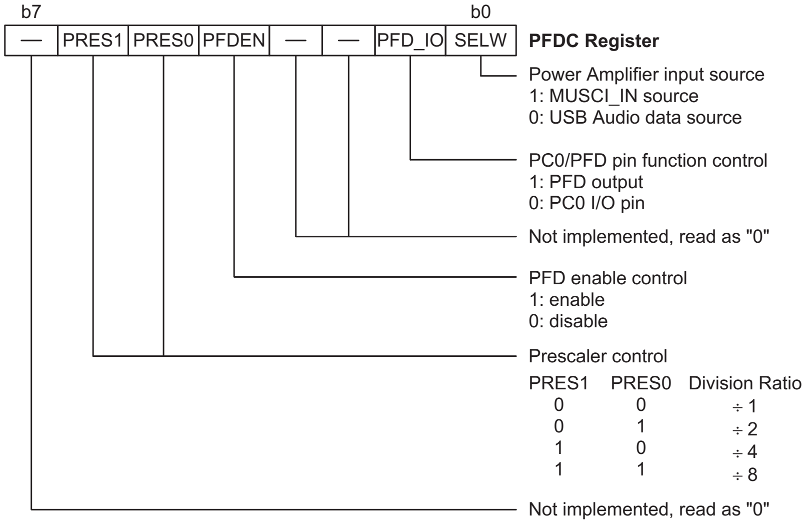
Programmable Frequency Divider – PFD

The Programmable Frequency Divider function, PFD, allows the generation of a user defined frequency. The clock source for the PFD is the system clock divided by 4, which after being divided by 16 is then passed through a programmable prescaler and the PFDD register allows a range of user defined frequencies to be generated.

Overall operation of the PFD is controlled using two registers, the PFDC register and the PFDD register. As the PFD output pin is pin-shared with I/O pin PC0, the PFD_IO bit in the PFDC register is used to select whether the pin is to be used an a normal I/O function or to be used as a PFD output. The PFDEN bit is used to control the overall on/off function of the PFD, while bits PRES0 and PRES1 are used to select the frequency division ratio of the prescaler. The PFDD register provided further division of the clock source, however this register can only be written to when the PFD function is enabled. If the PFD function is disabled, then all write operations to the PFDD register will be inhibited. When the PFD is disabled note that the PFDD register will be automatically cleared. The PFDD contents, the PFD must be enabled. When the generator is disabled, the PFDD is cleared by hardware.



PFD Block Diagram



PRES1	PRES0	Division Ratio
0	0	÷ 1
0	1	÷ 2
1	0	÷ 4
1	1	÷ 8

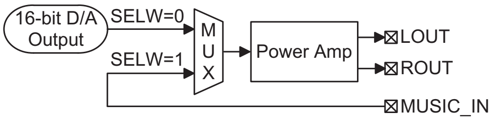
PFDC Register

The generated frequency of the PFD function is given by the following formula:

$$\text{PFD output frequency} = \frac{\text{Prescaler Output}}{2\text{x}(\text{N}+1)}\text{, where N= the value of the PFD data}$$

Power Amplifier

The SELW bit in the PFDC register is used to control the power amplifier input source. The software should set SELW = ″1″ when the power amplifier signal come from MUSIC_IN, otherwise the speaker output USB Audio data.



Interrupts

Interrupts are an important part of any microcontroller system. When a USB Interrupt, play/record data valid interrupt, a Timer/Event Counter overflow, reception of SPI data, A/D Interrupt or External Interrupt is occurs, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. The device provides a USB interrupt, two internal timer/event counter interrupts, a play/record data valid interrupt and a Multi function interrupt. This latter Multi-function Interrupt represents the Serial Interface Interrupt, A/D Interrupt or the External Interrupt.

Interrupt Registers

Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by three interrupt control registers INTC0, INTC1 and MFI1C which are located in the Data Memory. By controlling the appropriate enable bits in this 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 if cleared to zero will disable all interrupts.

Interrupt Operation

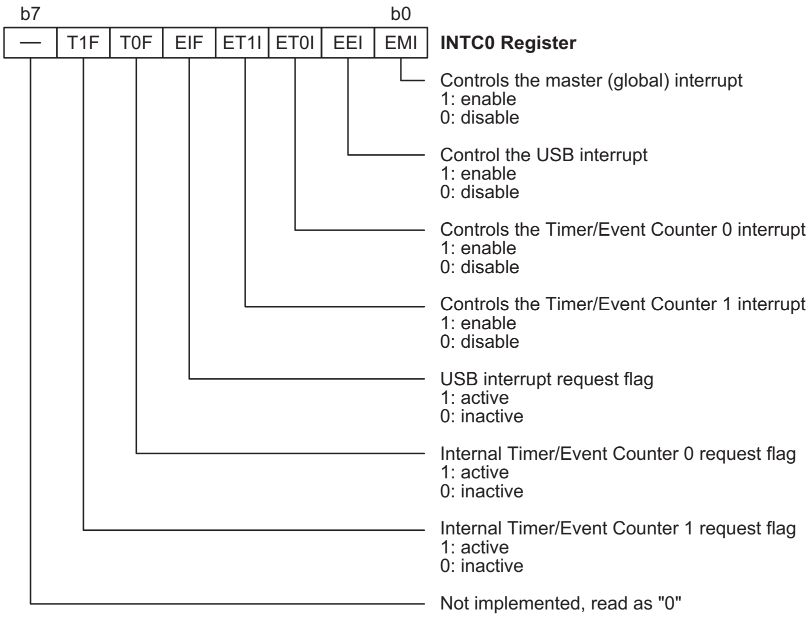
A USB interrupt, a Play or Record data valid interrupt, a Timer/Event Counter overflow, an SPI interrupt, an A/D conversion complete interrupt or an active edge on the external interrupt pin will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt 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 statement, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred. The various interrupt enable bits, together with their associated request flags, are shown in the following diagram with their order of priority. Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately serviced, the request flag will still be recorded. If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full.

Interrupt Priority

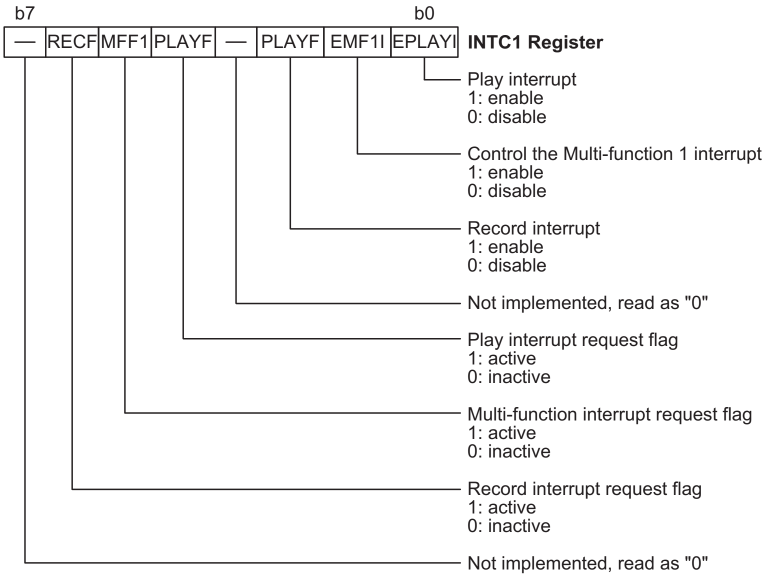
Interrupts, occurring in the interval between the rising edges of two consecutive T2 pulses, will be serviced on the latter of the two T2 pulses, if the corresponding interrupts are enabled. In case of simultaneous requests, the following table shows the priority that is applied. These can be masked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

No.	Interrupt Source	Priority	Vector
a	USB Interrupt	1	04H
b	Timer/Event Counter 0 overflow	2	08H
c	Timer/Event Counter 1 overflow	3	0CH
d	Play Interrupt	4	10H
e	Multi function 1 interrupt subroutine:Serial Interface Interrupt, A/D Interrupt, External Interrupt	5	14H
f	Record Interrupt	6	18H

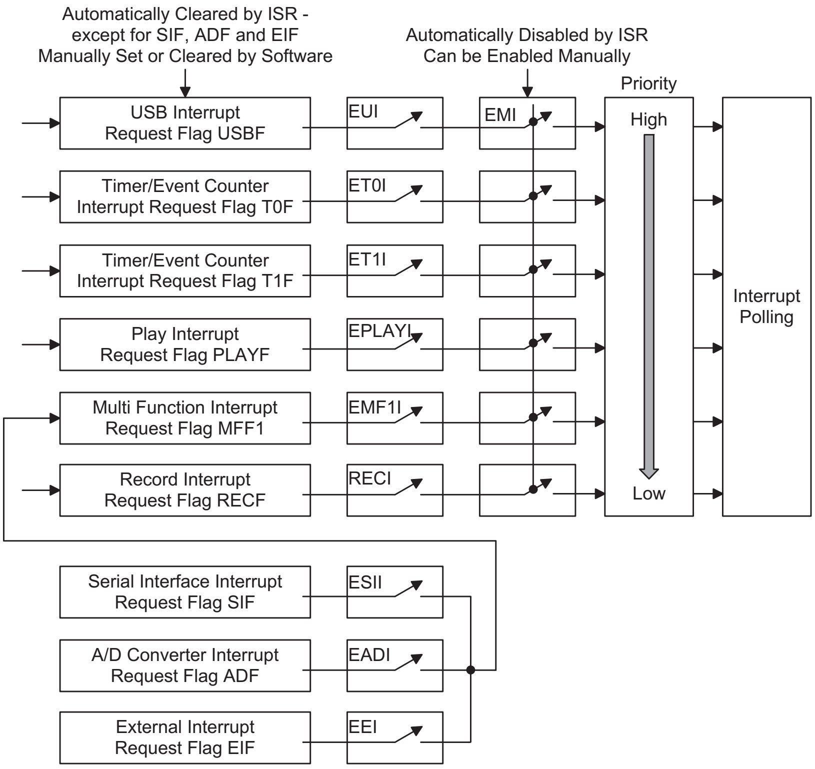
In cases where both USB and Play interrupts are enabled and where an USB and Play interrupt occurs simultaneously, the USB interrupt will always have priority and will therefore be serviced first. Suitable masking of the individual interrupts using the interrupt registers can prevent simultaneous occurrences.



INTC0 Register



INTC1 Register



Interrupt Structure

USB Interrupt

The USB interrupt will be triggered by any of the following USB events resulting in the related interrupt request flag, USBF; bit 4 of INTC0, being set:

- A PC access of the corresponding USB FIFO
- A USB suspend signal from the PC
- A USB resume signal from the PC
- A USB Reset signal

When the interrupt is enabled, the stack is not full and the interrupt is active, a subroutine call to location 04H will occur. The interrupt request flag, USBF, and EMI bits will be cleared to disable other interrupts. When the PC Host accesses the HT82A836R FIFO, the corresponding USR request bit is set, and a USB interrupt is triggered. In this way the user can determine which FIFO has been accessed. When the interrupt has been serviced, the corresponding bit will be automatically cleared. When the HT82A836R receives a USB Suspend signal from the host PC, the suspend line, bit0 of the USC register, in the HT82A836R is set and a USB interrupt is also triggered. Also when the device receives a Resume signal from the host PC, the resume line, bit3 of the USC register, is set and a USB interrupt generated.

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, ET0I or ET1I, must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter request flag, T0F or T1F, 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 0 overflow occurs, a subroutine call to the timer 0 interrupt vector at location 08H, will take place. If a Timer/Event Counter 1 overflow occurs, a subroutine call to the timer 1 interrupt vector at location 0CH will take place. When the interrupt is serviced, the timer interrupt request flag, T0F or T1F, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Play Interrupt

For a Play Interrupt to occur, the global interrupt enable bit, , and the corresponding Play Interrupt bit, EPLAI, must first be set. An actual Play Interrupt will take place when the Play Interrupt request flag, PLAYF, is set, a situation that will occur at a regular play frequency of 8kHz if the PLAY_MODE bit in the MODE_CTRL register is set high. If this bit is not high, then the play interrupt frequency will be 48kHz. When the interrupt is enabled, the stack is not full and a Play Interrupt occurs, a subroutine call to the Play Interrupt vector at location 10H, will take place. When the interrupt is serviced, the Play Interrupt request flag, PLAYF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Multi Function Interrupt

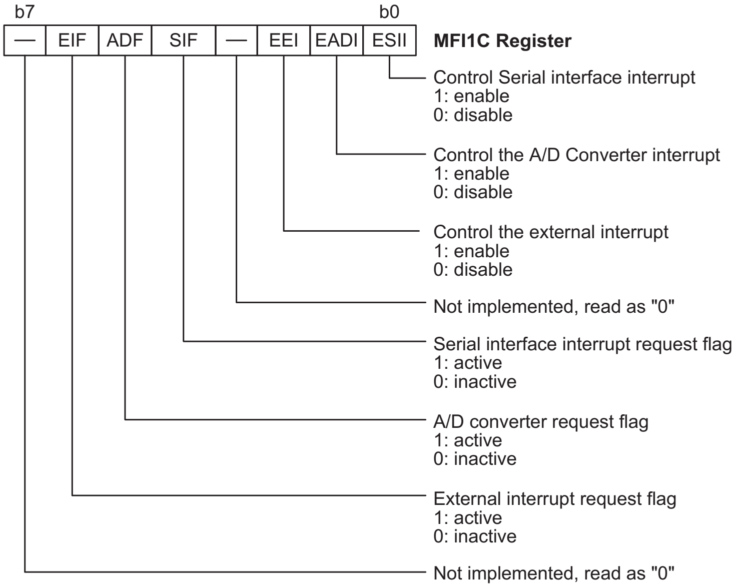
An additional interrupt known as the Multi-function interrupt is provided. Unlike the other interrupts, this interrupt has no independent source, but rather is formed from three other existing interrupt sources, namely the Serial Interface interrupt, the A/D Converter interrupt and the External interrupt. The Multi-function interrupt is enabled by setting the EMF1I bit, which is bit 1 of the INTC1 register. An actual Multi-function interrupt will be initialised when the Multi-function interrupt request flag MFF1 is set, this is bit 5 of the INTC1 register. When the master interrupt global bit is set, the stack is not full and the corresponding EMF1I interrupt enable bit is set, a Multi-Function internal interrupt will be generated when either a Serial Interface Interrupt, an A/D Converter Interrupt or an External Interrupt occurs. This will create a subroutine call to its corresponding vector location 014H. When a Multi-function internal interrupt occurs, the Multi-Function request flag MFF1 will be reset and the EMI bit will be cleared to disable other interrupts. However, it must be noted that the request flags from the original source of the Multi-function interrupt, namely the Serial Interface Interrupt, the A/D Converter or the External Interrupt will not be automatically reset and must be manually reset by the user.

External Interrupt

The device contains an external interrupt function controlled by the external pin $\overline{\text{INT}}$. For an external interrupt to occur, the pin must be setup as an interrupt input pin by ensuring that the corresponding external interrupt enable bit is first set. This is bit 2 in the MFI1C register and known as EEI. An external interrupt is triggered by a negative edge transition on the external interrupt pin $\overline{\text{INT}}$, after which the related interrupt request flag, EIF, which is bit 6 in the MFI1C register, will be set. The interrupt vector for the External Interrupt is the Multi-function interrupt located at 014H. Therefore if the Multi-function and External Interrupts are enabled, the stack is not full and a negative logical transition occurs on pin $\overline{\text{INT}}$, a subroutine call to location 014H will take place. The Multi-function Interrupt request flag MFF1 will be reset automatically and the EMI bit will be cleared to disable other interrupts. The External Interrupt flag will not be reset automatically and needs to be reset manually by the application program. The external interrupt pin $\overline{\text{INT}}$ is pin-shared with I/O pin PE7 and can only be configured as external interrupt pins if the interrupt is enabled and if the pin is programmed as an input pins.

A/D Converter Interrupt

The device contains an internal A/D converter with its own interrupt function. For an A/D Interrupt to occur, the corresponding A/D Interrupt enable bit must be first set. This is bit 1 in the MFI1C register and known as EADI. An A/D Interrupt is generated when the A/D conversion process is complete, after which the related interrupt request flag, ADF, which is bit 5 in the MFI1C register, will be set. The interrupt vector for the A/D Interrupt is the Multi-function interrupt located at 014H. Therefore if the Multi-function and A/D Interrupt are enabled, the stack is



MFI1C Register

not full and the A/D conversion completes, a subroutine call to location 014H will take place. The Multi-function Interrupt request flag MFF1 will be reset automatically and the EMI bit will be cleared to disable other interrupts. The A/D Interrupt flag will not be reset automatically and needs to be reset manually by the application program.

Serial Interface Interrupt

The device contains an internal Serial Interface with its own interrupt function. For a Serial Interface Interrupt to occur, the corresponding Serial Interface Interrupt enable bit must be first set. This is bit 0 in the MFI1C register and known as ESII. A Serial Interface Interrupt is generated when a data reception or transmission is complete, after which the related interrupt request flag, SIF, which is bit 4 in the MFI1C register, will be set. The interrupt vector for the Serial Interface Interrupt is the Multi-function Interrupt, located at 014H. Therefore if the Multi-function and Serial Interface Interrupt are enabled, the stack is not full and a serial interface data reception or transmission is complete, a subroutine call to location 014H will take place. The Multi-function Interrupt request flag MFF1 will be reset automatically and the EMI bit will be cleared to disable other interrupts. The Serial Interface Interrupt flag will not be reset automatically and needs to be reset manually by the application program.

Record Interrupt

For a Record Interrupt to occur, the global interrupt enable bit, EMI, and the corresponding Record Interrupt bit, RECI, must first be set. An actual Record Interrupt will take place when the Record Interrupt request flag, RECF, is set, a situation that will occur when the record data is valid. When the interrupt is enabled, the stack is not full and a Record Interrupt occurs, a subroutine call to the Record Interrupt vector at location 18H, will take place. When the interrupt is serviced, the Record Interrupt request flag, RECF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. If the A/D Converter is powered down (AD_ENB =1), PLL clock disabled ($\overline{\text{PLLEN}}$=1) or USB clock disabled (USBCKEN=0), the record interrupt also be disabled.

Programming Considerations

By disabling the interrupt enable bits, a requested interrupt can be prevented from being serviced, however, once an interrupt request flag is set, it will remain in this condition in the interrupt control register until the corresponding interrupt is serviced or until the request flag is cleared by a software instruction.

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

All of these interrupts have the capability of waking up the processor when in the Power Down Mode. Only the Program Counter is pushed onto the stack. If the contents of the accumulator or status register are altered by the interrupt service program, which may corrupt the desired control sequence, then the contents should be saved in advance.

Reset and Initialisation

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

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

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

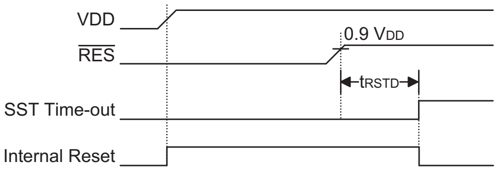
Reset Functions

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 ad-

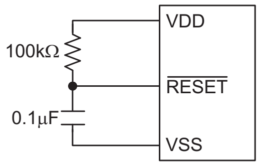
dress, a power-on reset also ensures that certain other registers are preset to known conditions. All the I/O port and port control registers will power up in a high condition ensuring that all pins will be first set to inputs.

Although the microcontroller has an internal RC reset function, if the VDD power supply rise time is not fast enough or does not stabilise quickly at power-on, the internal reset function may be incapable of providing proper reset operation. For this reason it is recommended that an external RC network is connected to the $\overline{\text{RESET}}$ pin, whose additional time delay will ensure that the $\overline{\text{RESET}}$ pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{\text{RESET}}$ line reaches a certain voltage value, the reset delay time t_{RSTD} is invoked to provide an extra delay time after which the microcontroller will begin normal operation. The abbreviation SST in the figures stands for System Start-up Timer.



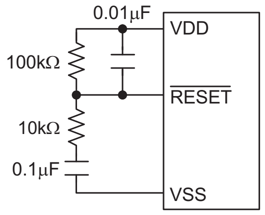
Power-On Reset Timing Chart

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



Basic Reset Circuit

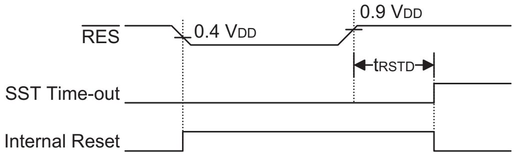
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Enhanced Reset Circuit

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

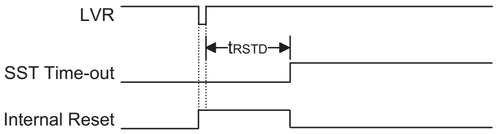
  This type of reset occurs when the microcontroller is already running and the $\overline{\text{RESET}}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from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



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

- Low Voltage Reset – LVR

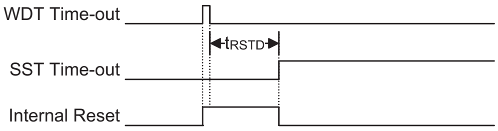
  The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device, which is selected via a configuration option. If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing the battery, the LVR will automatically reset the device internally. 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 1ms, the LVR will ignore it and will not perform a reset function.



Low Voltage Reset Timing Chart

- Watchdog Time-out Reset during Normal Operation

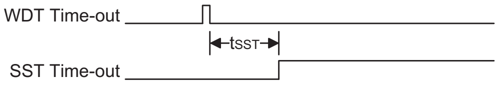
  The Watchdog time-out Reset during normal operation is the same as a hardware $\overline{\text{RES}}$ pin reset except that the Watchdog time-out flag TO will be set to ″1″.



WDT Time-out Reset during Normal Operation Timing Chart

- Watchdog Time-out Reset during Power Down

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



WDT Time-out Reset during Power Down Timing Chart

Reset Initial Conditions

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

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

Note: "u" stands for unchanged

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

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

The different kinds of resets all affect the internal registers of the microcontroller in different ways. To ensure reliable continuation of normal program execution after a reset occurs, it is important to know what condition the microcontroller is in after a particular reset occurs. The following table describes how each type of reset affects each of the microcontroller internal registers. Note that where more than one package type exists the table will reflect the situation for the larger package type.

The states of the registers are summarized in the table.

Register	Reset (Power-on)	WDT Time-out (Normal Operation)	$\overline{\text{RES}}$ Reset (Normal Operation)	$\overline{\text{RES}}$ Reset (HALT)	WDT Time-Out (HALT)*	USB Reset (Normal)	USB Reset (HALT)
MP0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
Program Counter	000H	000H	000H	000H	000H	000H	000H
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	-xxx xxxx	-uuu uuuu	-uuu uuuu	-uuu uuuu	-uuu uuuu	-uuu uuuu	-uuu uuuu
WDTS	0000 0111	0000 0111	0000 0111	0000 0111	uuuu uuuu	0000 0111	0000 0111
STATUS	--00 xxxx	--1u uuuu	--uu uuuu	--01 uuuu	--11 uuuu	--uu uuuu	--01 uuuu
INTC0	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu	-000 0000	-000 0000
TMR0H	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR0L	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR0C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu	00-0 1000	00-0 1000
TMR1H	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR1L	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR1C	00-0 1---	00-0 1---	00-0 1---	00-0 1---	uu-u u---	00-0 1---	00-0 1---
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111

Register	Reset (Power-on)	WDT Time-out (Normal Operation)	$\overline{\text{RES}}$ Reset (Normal Operation)	$\overline{\text{RES}}$ Reset (HALT)	WDT Time-Out (HALT)*	USB Reset (Normal)	USB Reset (HALT)
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PCC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PD	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PDC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PE	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PEC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PF	---- 1111	---- 1111	---- 1111	---- 1111	---- uuuu	---- 1111	---- 1111
PFC	---- 1111	---- 1111	---- 1111	---- 1111	---- uuuu	---- 1111	---- 1111
INTC1	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu	-000 0000	-000 0000
TBHP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
USC	1000 0000	uuxx uuuu	10xx 0000	10xx 0000	10xx uuuu	1000 0u00	1000 0u00
USR	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	00uu 0000	00uu 0000
UCC	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	0u00 u000	0u00 u000
AWR	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
STALL	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
SIES	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	0u00 u000	0u00 u000
MISC	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
SETIO	xxxx x010	xxxx x010	xxxx x010	xxxx x010	xxxx x010	xxxx x010	xxxx x010
FIFO0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	0000 0000	0000 0000
FIFO1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	0000 0000	0000 0000
FIFO2	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	0000 0000	0000 0000
FIFO3	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	0000 0000	0000 0000
FIFO4	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	0000 0000	0000 0000
DAC_LIMIT_L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
DAC_LIMIT_H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
DAC_WR	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
PGA_CTRL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	00uu uuuu	00uu uuuu
PFDC	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0uuu 0000	0uuu 0000
PFDD	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0uuu 0000	0uuu 0000
OPER_MODE	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
MODE_CTRL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0uuu	0000 0uuu	0000 0uuu
SBCR	0110 0000	0110 0000	0110 0000	0110 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
SBDR	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
RECORD_IN_L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
RECORD_IN_H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
PLAY_DATAL_L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu

Register	Reset (Power-on)	WDT Time-out (Normal Operation)	$\overline{\text{RES}}$ Reset (Normal Operation)	$\overline{\text{RES}}$ Reset (HALT)	WDT Time-Out (HALT)*	USB Reset (Normal)	USB Reset (HALT)
PLAY_DATAL_H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
PLAY_DATAR_L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
PLAY_DATAR_H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
RECORD_DATA_L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
RECORD_DATA_H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
ADRL	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
ADRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
ADCR	0100 0000	0100 0000	0100 0000	0100 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
ACSR	1--- --00	1--- --00	1--- --00	1--- --00	u--- --uu	u--- --uu	u--- --uu
PA_WAKE_CTRL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
PWMC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
PWM0	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
PWM1	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
MFI1C	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
USB_STATE	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
USVC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000

″*″ stands for warm reset
″u″ stands for unchanged
″x″ stands for unknown
″–″ stands for unimplemented

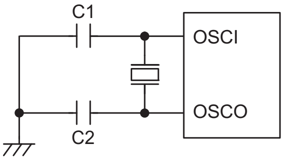
Oscillator

The device use crystal oscillator as the system clock source. Two types of crystal system clock frequencies can be selected while various clock source options for the Watchdog Timer are provided for maximum flexibility.

System Crystal/Ceramic Oscillator

For the 12MHz crystal oscillator configurations, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation. For the 6MHz crystal oscillator configuration the addition of two small value capacitors are required. The SYSCLK bit in the UCC register determines the system frequency selection.

Crystal	C1, C2
6MHz Crystal	22pF
12MHz Crystal	NC



Crystal/Ceramic Oscillator

Watchdog Timer Oscillator

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

Power Down Mode and Wake-up

Power Down Mode

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

Entering the Power Down Mode

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

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

Standby Current Considerations

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

Wake-up

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

- An external reset
- An external falling or rising edge on Port A
- An external falling edge on Port B~Port F
- A system interrupt
- A WDT overflow

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

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

Each pin on can be setup via an individual configuration option to permit a negative transition on the pin to wake-up the system. Port A has an addition function, controlled via the PA_WAKE_CTRL register in the Data Memory, allowing either a negative or positive edge to initiate a Wake-up function. Any external pin wake-up will cause the system to resume execution at the instruction following the ″HALT″ instruction.

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

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

Watchdog Timer

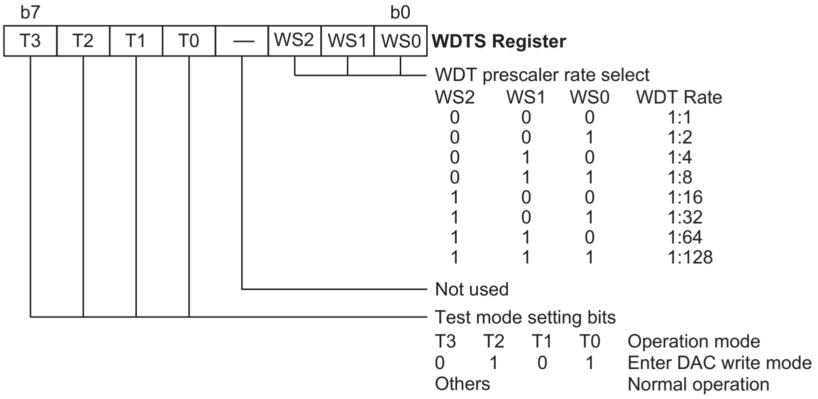
The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by two sources selected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

The internal WDT oscillator has an approximate period of 65μs at a supply voltage of 5V. If selected, it is first divided by 256 via an 8-stage counter. Note that this period can vary with VDD, temperature and process variations. For longer WDT time-out periods the WDT prescaler can be utilized. By writing the required value to bits 0, 1 and 2 of the WDTS register, known as WS0, WS1 and WS2, longer time-out periods can be achieved. With WS0, WS1 and WS2 all equal to 1, the division ratio is 1:128 which gives a maximum time-out period.

A configuration option can select the instruction clock, which is the system clock divided by 4, as the WDT clock source instead of the internal WDT oscillator. If the instruction clock is used as the clock source, it must be noted that when the system enters the Power Down Mode, as the system clock is stopped, then the WDT clock source will also be stopped. Therefore the WDT will lose its protecting purposes. In such cases the system cannot be restarted by the WDT and can only be restarted using external signals. For systems that operate in noisy environments, using the internal WDT oscillator is therefore the recommended choice.

Under normal program operation, a WDT time-out will initialise a device reset and set the status bit TO. However, if the system is in the Power Down Mode, when a WDT time-out occurs, only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the WDT and the WDT prescaler. The first is an external hardware reset, which means a low level on the $\overline{\text{RESET}}$ pin, the second is using the watchdog software instructions and the third is via a ″HALT″ instruction.

There are two methods of using software instructions to clear the Watchdog Timer, one of which must be chosen by configuration option. The first option is to use the single ″CLR WDT″ instruction while the second is to use the two commands ″CLR WDT1″ and ″CLR WDT2″. For the first option, a simple execution of ″CLR WDT″ will clear the WDT while for the second option, both ″CLR WDT1″ and ″CLR WDT2″ must both be executed to successfully clear the WDT. Note that for this second option, if ″CLR WDT1″ is used to clear the WDT, successive executions of this instruction will have no effect, only the execution of a ″CLR WDT2″ instruction will clear the WDT. Similarly, after the ″CLR WDT2″ instruction has been executed, only a successive ″CLR WDT1″ instruction can clear the Watchdog Timer.



WDTS Register

WDT prescaler rate select

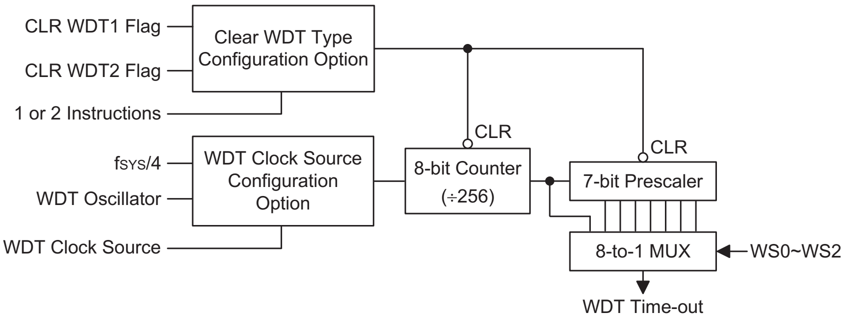
WS2	WS1	WS0	WDT Rate
0	0	0	1:1
0	0	1	1:2
0	1	0	1:4
0	1	1	1:8
1	0	0	1:16
1	0	1	1:32
1	1	0	1:64
1	1	1	1:128

Not used

Test mode setting bits

T3	T2	T1	T0	Operation mode
0	1	0	1	Enter DAC write mode
Others				Normal operation

Watchdog Timer Register



Watchdog Timer

USB Function

The device includes a USB 1.1 interface which can be used for data application data transfer. Five endpoints are included in the USB function of this device.

USB Interface

The Interface in the HT82A836R device has 5 Endpoints, known as EP0~EP4. Endpoint 0, EP0, is used for Control transfer. Endpoints EP1 and EP4 are for Interrupt transfer, while EP2 supports the Isochronous out transfer. EP3 supports Isochronous in transfer. A set of registers stored in the Data Memory is used for overall control of the USB function. These control registers include, USC, USR, UCC, AWR, STALL, SIES and MISC. There are also five FIFO registers with the names FIFO0~FIFO4. The size of each FIFO is as follows:, FIFO0-8 bytes, FIFO1-8 bytes, FIFO2-384 bytes, FIFO3-32 bytes and FIFO4-64 bytes, giving a total of 496 bytes. The URD bit, which is bit7 of the USC register is the USB reset signal control function definition bit.

USB Interface Registers

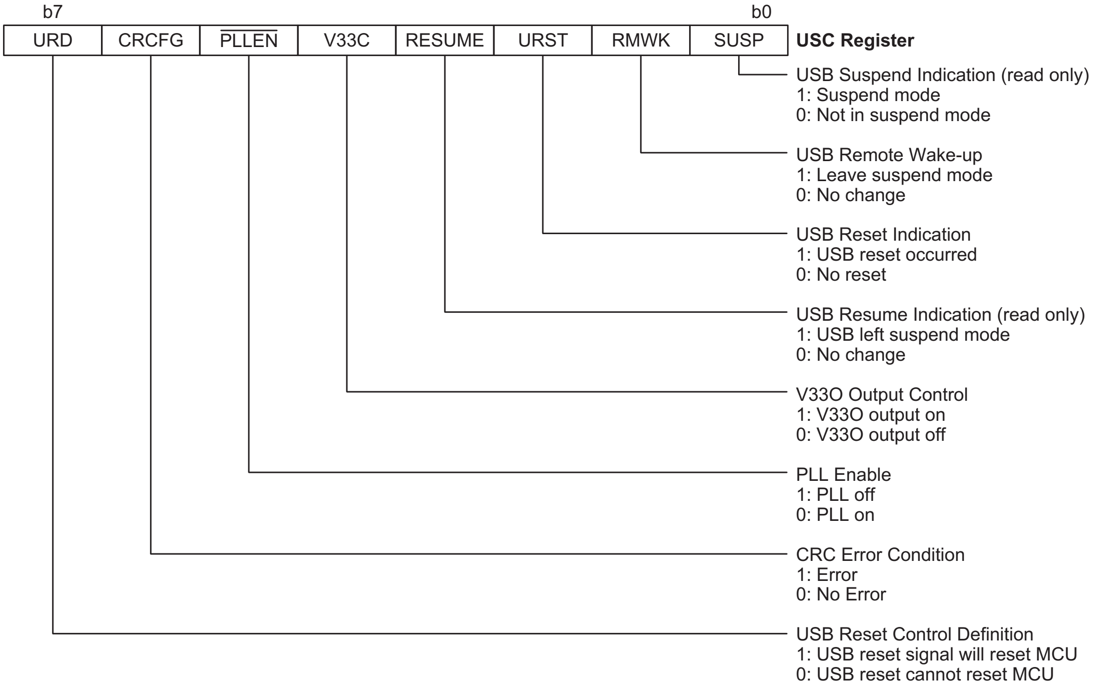
The USB setup, data management and endpoint control in the device is controlled via a series of registers in the Data Memory.

USC Register

The USC register is the register for the overall control of the USB function. The initial status of this register is 80H.

Further explanation of each of the bits is given below:

- SUSP
  The SUSP bit is the USB Suspend Indicator bit. When this read-only bit is set to ″1″ by the SIE, it indicates that the USB bus has entered the suspend mode. The USB interrupt is also triggered when this bit changes from low to high.
- RMWK
  The RMWK read/write bit is the USB remote wake-up command. It is set by the MCU to allow the USB host to leave the suspend mode after an external wake-up.
- URST
  The URST read/write bit is the USB reset indication bit. This bit is set and cleared by the USB SIE and indicates a USB reset event on the USB bus. When this bit is set to ″1″, this indicates that a USB reset has occurred and that a USB interrupt will be generated.
- RESUME
  The RESUME read only bit is used to indicate that the USB has left the Suspend Mode. When the USB has left the Suspend Mode, this read-only bit is set to ″1″ by the SIE. When the RESUME bit is set by SIE, an interrupt will be generated to wake-up the MCU. In order to detect the suspend state, the MCU should set the USBCKEN bit and clear SUSP2 in the UCC register, to enable the SIE detect function. The RESUME bit will be cleared when SUSP goes to ″0″. When the MCU is detecting the SUSP, the condition of the RESUME bit, which will wake-up the MCU, should be noted and taken into consideration.
- V33C
  The V33C read/write bit is the control bit for the internally generated 3.3V supply for the USB interface.
- $\overline{\text{PLLEN}}$
  The $\overline{\text{PLLEN}}$ read/write bit is the control bit for the internal Phase Locked Loop function.
- CRCFG
  The CRCFG read/write bit is the CRC error condition failure flag. The CRCFG bit will be set by the hardware however the CRCFG bit needs to be cleared using firmware.
- URD
  The URD read/write bit is the USB reset signal control function definition bit.



USB Control Register – USC

USR Register

The USR register is the USB endpoint interrupt status register and is used to indicate which endpoint is accessed and to select the USB bus. The endpoint request flags, EP0F, EP1F, EP2F, EP3F and EP4F are used to indicate which endpoints are accessed. If an endpoint is accessed, the related endpoint request flag will be set to ″1″ and the USB interrupt will be generated if the USB interrupt is enabled and the stack is not full. When the active endpoint request flag is serviced, the endpoint request flag has to be cleared to ″0″ by software.

Further explanation of each of the bits is given below:

- ESP0F~ESP4F
  The ESP0F~ESP4F read/write bits are set by the SIE an indicate whether the associated endpoint has been accessed and a USB interrupt generated. After the interrupt has been serviced the bits should be cleared by the application program.

UCC Register

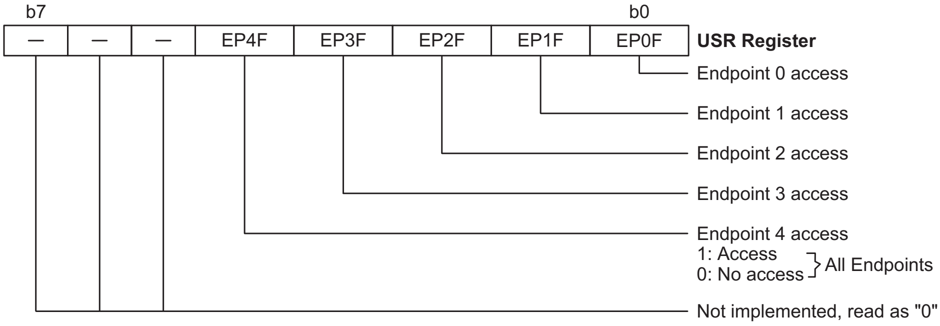
The UCC register is the system clock control register and is used to select the clock that is used in the MCU. This register consists of a USB clock control bit, USBCKEN, a second suspend mode control bit, SUSP2, and a system clock selection bit, SYSCLK. The register also controls the endpoint selection, which is determined by bits EPS0, EPS1 and EPS2.

Further explanation of each of the bits is given below:

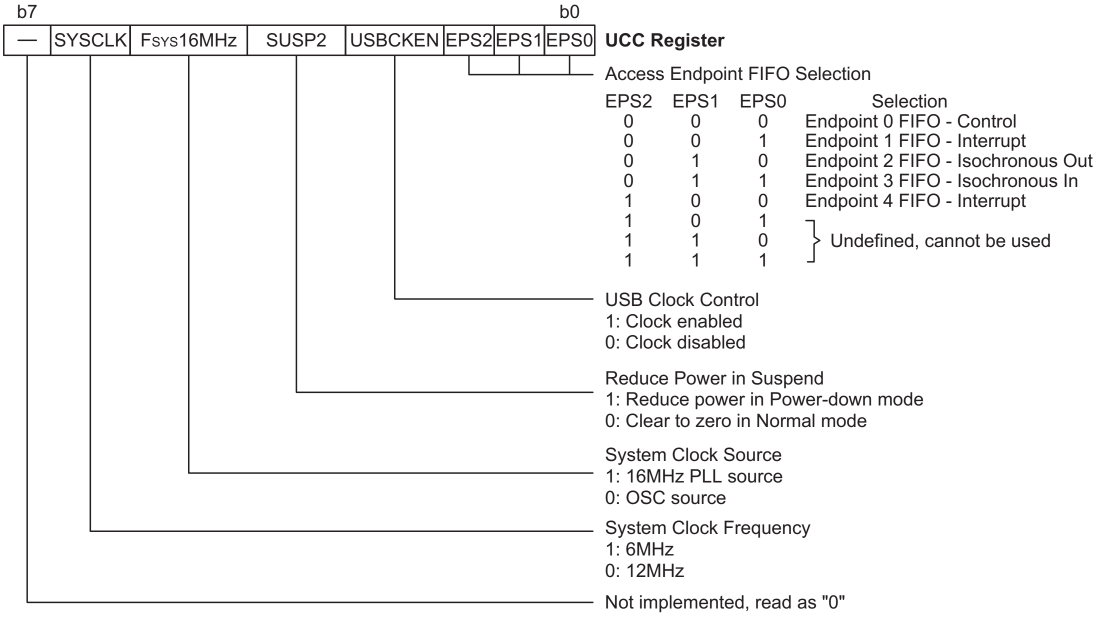
- EPS0~EPS2
  These three read/write bits are for the endpoint FIFO selection. It should be noted that Isochronous endpoints 2 and 3 are implemented in hardware, therefore FIFO2 and FIFO3 cannot be read from or written to using firmware.
- USBCKEN
  The USBCKEN read/write bit enables the USB clock.
- SUSP2
  The SUSP2 read/write bit is the second suspend bit and is used to select a power reducing function when the device is in the Suspend Mode. In the Normal Mode this bit should be cleared to zero.
- FSYS16MHz
  This read/write bit is used to determine if the system clock is derived from an external oscillator or from the internal PLL 16MHz clock.
- SYSCLK
  The SYSCLK read/write bit is used to determine if the system clock is either 6MHz or 12MHz.

AWR Register

The AWR register is used to store the current USB device address and also for control of the Remote Wake-up function. The initial value of the AWR register is ″00H″. The address value extracted from the USB command must not be loaded into this register until the SETUP stage has finished.

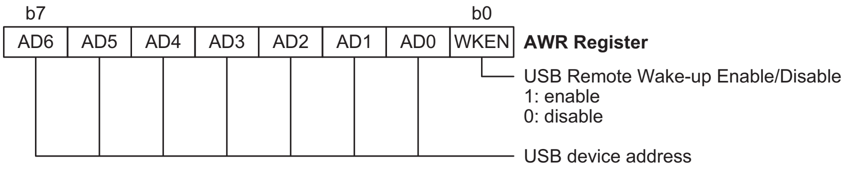


USB Endpoint Status Register – USR



EPS2	EPS1	EPS0	Selection
0	0	0	Endpoint 0 FIFO - Control
0	0	1	Endpoint 1 FIFO - Interrupt
0	1	0	Endpoint 2 FIFO - Isochronous Out
0	1	1	Endpoint 3 FIFO - Isochronous In
1	0	0	Endpoint 4 FIFO - Interrupt
1	0	1	Undefined, cannot be used
1	1	0	
1	1	1	

System Clock Control Register – UCC



Device Address Register – AWR

STALL Register

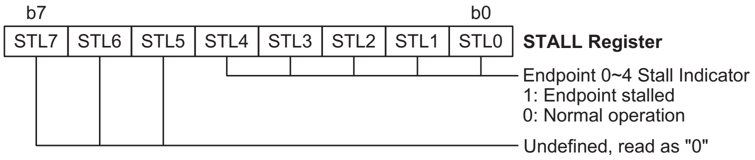
The STALL register shows whether the corresponding endpoint has operated correctly or not. As soon as the endpoint has operated incorrectly, the related read/write bit in the STALL has to be set to ″1″. The STALL register will be cleared by a USB reset signal and a setup token event. The initial value of the STALL register is ″00H″.

SIES Register

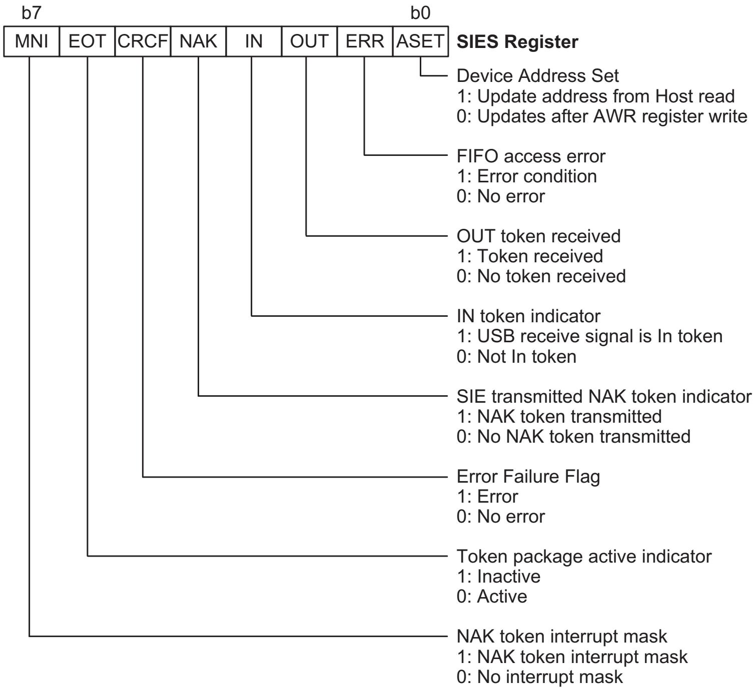
The SIES register is the setup register for the Serial Interface Engine.

Further explanation of each of the bits is given below:

- ASET
  The read/write ASET bit is used to configure the SIE to automatically change the device address to the value presently stored in the AWR register. When this bit is set to ″1″ by, the SIE will update the device address with the value stored in the AWR register after the PC host has successfully read the data from the device with an IN operation. Otherwise, when this bit is cleared to ″0″, the SIE will update the device address immediately after an address is written to the AWR register. Therefore in order to work properly, the program has to clear this bit after a next valid SETUP token is received.
- ERR
  The read/write ERR bit is used to indicate that errors have occurred when the FIFO is accessed. This bit is set by SIE and should be cleared by the program. This bit is used for all endpoints.
- OUT
  The read/write OUT bit is used to indicate the reception of an OUT token, except for the OUT zero length token. The device will clear this bit after the OUT data has been read. Also, this bit will be cleared by the SIE after the next valid SETUP token is received.
- IN
  The read only IN bit is used to indicate that the current USB receiving signal from PC host is an IN token.
- NAK
  This read only bit is used to indicate that the SIE has transmitted a NAK signal to the host in response to the PC host IN or OUT token.



Endpoint Stall Register – STALL



Serial Interface Setup Register – SIES

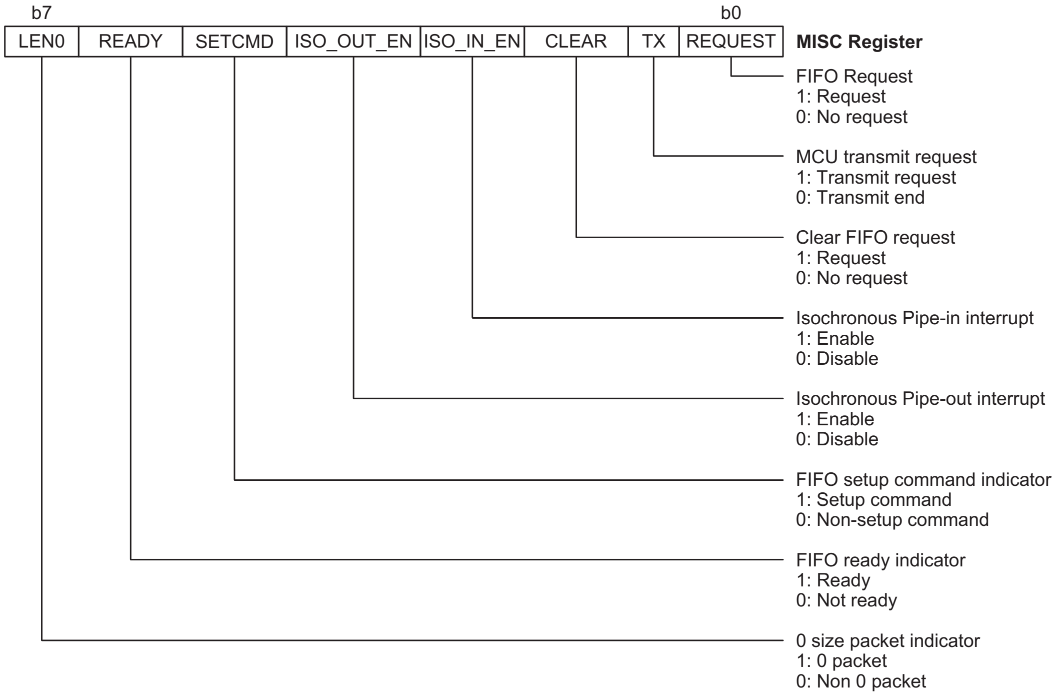
- CRCF
  The CRCF read/write is an error condition failure flag that includes CRC, PID and no integrate token error. CRCF will be set by the hardware but needs to be cleared by the firmware.
- EOT
  The EOT read only read only flag is the Token Package active flag. Note that this flag is active low.
- NMI
  The read/write NMI bit is the NAK token interrupt mask flag. If this bit set, when the device sends a NAK token to the host, the interrupt will be disabled. Otherwise if this bit is cleared, when the device sends a NAK token to the host, it will enter the interrupt subroutine.

MISC Register

The MISC register combines command and status to control the desired endpoint FIFO action and to show the status of the desired endpoint FIFO. The MISC register will be cleared by a USB reset signal.

Further explanation of each of the bits is given below:

- REQUEST
  The read/write REQUEST, if set high, can request the FIFO after the corresponding status has been set. When finished this bit must be set low.
- TX
  The read/write TX bit represents the direction and MCU access transition end. When set high, the MCU desires to write data to the FIFO. After finishing, this bit must be set low before terminating the request to represent a transition end. For an MCU read operation, this bit must be set low and then high after finishing.
- CLEAR
  The read/write CLEAR bit MCU is used to request a FIFO clear, even if the FIFO is not ready. After clearing the FIFO, the USB interface will send a force_tx_err to tell the Host that data under-run if the Host wants to read data.
- ISO_IN_EN
  The read/write ISO_IN_EN bit enables the isochronous in pipe interrupt.
- ISO_OUT_EN
  The read/write ISO_OUT_EN bit enables the isochronous out pipe interrupt.
- SETCMD
  The read/write SETCMD bit is used to show that the data in the FIFO is a setup command. The bit will remain in the same state until the following data enters the FIFO.
- READY
  The read only READY bit is used to indicate that the desired FIFO is ready.
- LEN0
  The read only LEN0 bit is used to indicate that the host has sent a 0-sized packet to the MCU. This bit must be cleared by a read action to the corresponding FIFO.



Miscellaneous Register – MISC

SETIO Register

The SETIO register is used to setup the endpoints to either input or output pipe type. The DATA token toggle bit is also contained within this register. Note that for USB definition, when the host sends a ″set Configuration″, the Data pipe should send DATA0, about the Data toggle, first. Therefore, when the device receives a ″set configuration″ setup command, the user needs to toggle this bit as the following data will send DATA0 first. It is only required to set the data pipe as an input pile or output pile. The purpose of this function is to avoid the host sending an abnormal IN or OUT token and disabling the endpoint. All bits are read/write.

SETIO (27H) register, USB Endpoint 1~Endpoint 4 set IN/OUT pipe register.

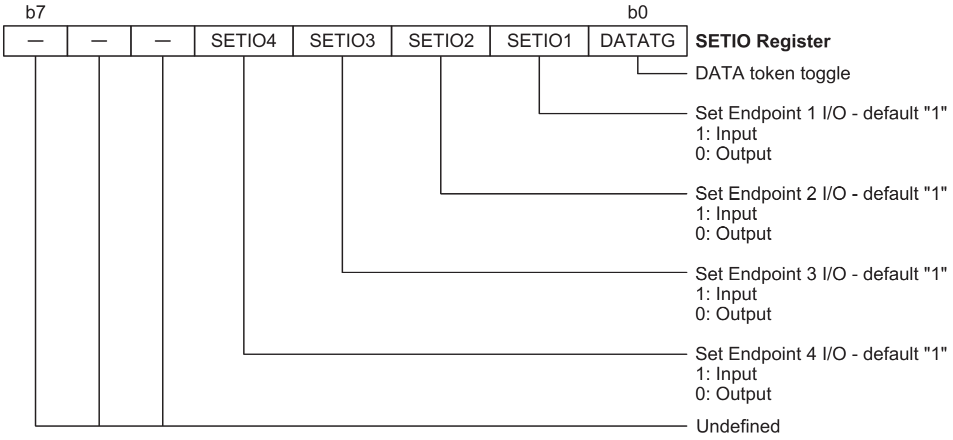
USB_STATE Register

This register is used to indicate the error state due to SE0 or SE1 noise as well as the USBD- and USBD+ input signals. The SE0 and SE1 bits are set by the SIE and cleared with the program.

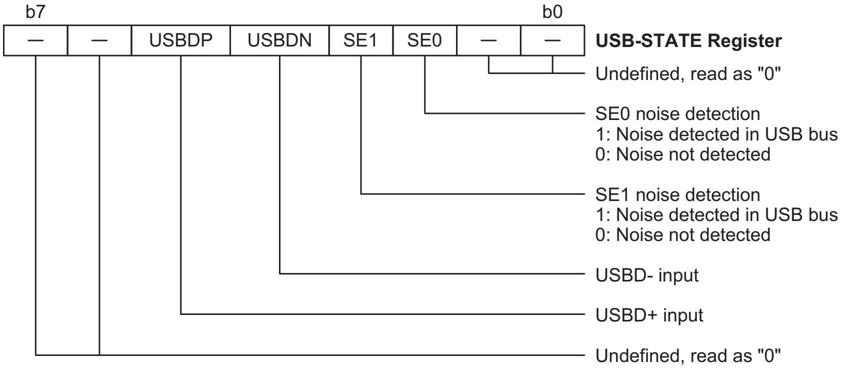
Suspend Wake-Up Remote Wake-Up

The device includes a Suspend mode. If there is no signal on the USB bus for over 3ms, the device will enter a suspend mode. When this happens, the SUSPEND bit, which is bit 0 of the USC register, will be set to ″1″ and a USB interrupt will be generated to indicate that the device should jump to the suspend state to meet the requirements of the USB suspend current spec. In order to meet the requirements of the suspend current, the program should disable the USB clock by clearing the USBCKEN bit, which is bit3 of the UCC register, to ″0″.

The suspend current can be further decreased by setting the SUSP2 bit, which is bit4 of the UCC register. When the resume signal is sent out by the host, the HT82A836R will be woken up the by the USB interrupt and the RESUME bit, which is bit 3 of the USC register, will be set. In order to make the device operate correctly, the program must set the USBCKEN bit and clear the SUSP2 bit. Since the Resume signal will be cleared before the Idle signal is sent out by the host and the SUSPEND bit, will change to ″0″. Therefore when the MCU is detecting the Suspend line, the condition of the Resume line should be noted and taken into consideration.

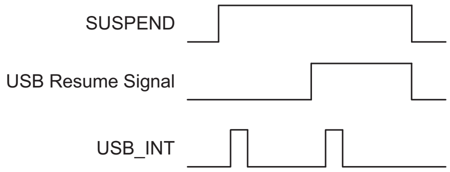


USB Endpoint Setup IN/OUT Pipe Register – SETIO



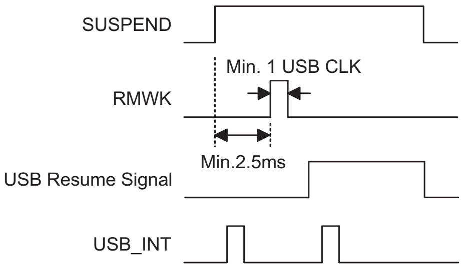
USB State Register – USB-STATE

The following shows the related timing of this operation:



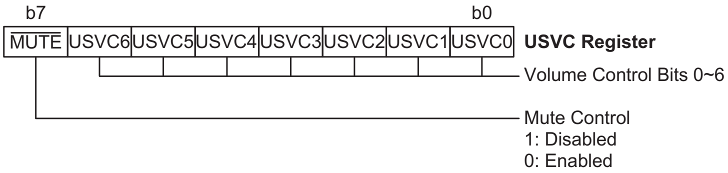
The device contains a remote wake-up function which can wake-up the USB Host by sending a wake-up pulse through the RMWK bit, which is bit 1 of the USC register. Once the USB Host receives a wake-up signal from the device, it will send a Resume signal to the device.

The following shows the related timing:



USB Speaker Volume Control

The speaker output volume as well as the speaker mute/un-mute function are controlled by the USB Speaker Volume Control USVC register. The volume range can be set between a range of 6dB to –32dB by software. The relationship between the USVC volume control bits and the amplification or attenuation values are shown in the Volume Control Table. The mute control will be enabled when the $\overline{\text{MUTE}}$ bit is low and both the DAC and the power amplifier will be muted.



USB Speaker Volume Control Register – USVC

Result (dB)	USVC	Result (dB)	USVC	Result (dB)	USVC	Result (dB)	USVC
6	000_1100	–2	111_1100	–10	110_1100	–24	101_1100
5.5	000_1011	–2.5	111_1011	–10.5	110_1011	–25	101_1011
5	000_1010	–3	111_1010	–11	110_1010	–26	101_1010
4.5	000_1001	–3.5	111_1001	–11.5	110_1001	–27	101_1001
4	000_1000	–4	111_1000	–12	110_1000	–28	101_1000
3.5	000_0111	–4.5	111_0111	–13	110_0111	–29	101_0111
3	000_0110	–5	111_0110	–14	110_0110	–30	101_0110
2.5	000_0101	–5.5	111_0101	–15	110_0101	–31	101_0101
2	000_0100	–6	111_0100	–16	110_0100	–32	101_0100
1.5	000_0011	–6.5	111_0011	–17	110_0011	—	—
1	000_0010	–7	111_0010	–18	110_0010	—	—
0.5	000_0001	–7.5	111_0001	–19	110_0001	—	—
0	000_0000	–8	111_0000	–20	110_0000	—	—
–0.5	111_1111	–8.5	110_1111	–21	101_1111	—	—
–1	111_1110	–9	110_1110	–22	101_1110	—	—
–1.5	111_1101	–9.5	110_1101	–23	101_1101	—	—

Speaker Volume Control Table

FIFO Registers

Registers	R/W	Power-on	Function
FIFO0~ FIFO4	R/W	xxH	EPi accessing register – EPSX bits in the UCC register. (i = 0~4). When an end-point is disabled, the corresponding accessing register should be disabled.

FIFO0~FIFO4 (28H~2CH) USB Endpoint Accessing Registers Definitions

DAC Limit Registers

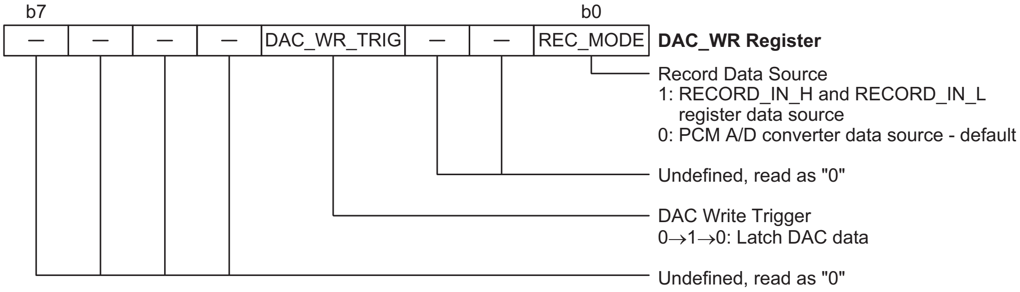
The DAC_Limit_L and DAC_Limit_H registers are used to define the 16-bit DAC output limits. The values in the DAC_Limit_L and DAC_Limit_H registers are unsigned values. If the 16-bit data from the Host exceeds that of the range defined by the two DAC_Limit_L and DAC_Limit_H registers then the output digital code to DAC will be clamped within these register values.

DAC_Limit_L	DAC output limit low byte
DAC_Limit_H	DAC output limit high byte

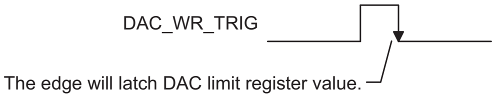
Example to set the DAC output limit values:

```
;------------------------------------------------------------
; Set DAC Limit Value=FF00H
;------------------------------------------------------------
clr         [02DH] ; Set DAC Limit low byte=00H
set         [02EH] ; Set DAC Limit high byte=FFH
;------------------------------------------------------------
```

In order to prevent a popping noise from the speaker output, the power amplifier should output a value of $V_{DD}/2$, which means a value of 8000H should be sent to the DAC during the initial power on state. Generating a pulse on the DAC_WR_TRIG bit will write the values into the DAC. If the DAC_WR_TRIG, bit 3 of the DAC_WR register, is already high then clearing the DAC_WR_TRIG bit, will write the values into the DAC_Limit_L and DAC_Limit_H registers for the DAC.



DAC Write and Record Source Register – DAC_WR



Note: In the DAC write data mode (high nibble of WDTS register is 0101b), the DAC_Limit_L and DAC_Limit_H registers will be the 16-bit DAC input data register at the falling edge of DAC_WR_TRIG. Otherwise, these two registers are used to define the 16-bit DAC output limit (repeated below).

REC_MODE=1:Writing to RECORD_IN_L register will only put the written data to an internal lower-order byte buffer (8-bit) and writing RECORD_IN_H will transfer the RECORD_IN_L and RECORD_IN_H registers content to isochronous in buffer. When record interrupt happened, firmware should write 16-bit 2′s complement value to RECORD_IN_L and RECORD_IN_H (RECORD_U_EN=0) registers or write 8-bit μ law value to RECORD_IN_H register (RECORD_U_EN=1).

REC_MODE= ″1″	Record data comes from the RECORD_IN_H and RECORD_IN_L registers
REC_MODE= ″0″	Record data comes from the PCM ADC (Default =0)

The record write data format will be controlled by bit RECORD_U_EN of OPER_MODE register when REC_MODE= ″1″.

The record data write format as follow:

	RECORD_IN_L	**RECORD_IN_H**
RECORD_U_EN= 0	PCM (Low byte)	PCM (High byte)
RECORD_U_EN= 1	N/A	μ Law

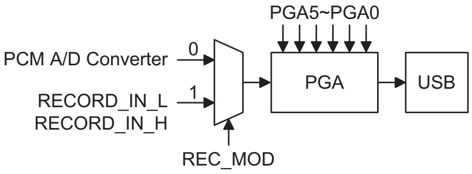
Example Program to Eliminate Pop Noise:

```
System_Initial:
;-----------------------------------------------------------
; Avoid Pop Noise
;-----------------------------------------------------------
mov a,WDTS
mov FIFO_TEMP,a       ;Save WDTS value
mov a,00001111b
mov a,WDTS
mov a,01010000b
orm a,WDTS           ;Enter DAC Write Data mode, high nibble of WDTS=0101b
clr [02DH]           ;Set DAC data low byte=00H
mov a,80H
mov [02EH],a         ;Set DAC data high byte=80H

;Write 8000H to DAC
set [02FH].3

clr [02FH].3

;-----------------------------------------------------------
mov a,FIFO_TEMP ;Restore WDTS value
mov WDTS,a ;Quit DAC Write Data mode
;-----------------------------------------------------------
```

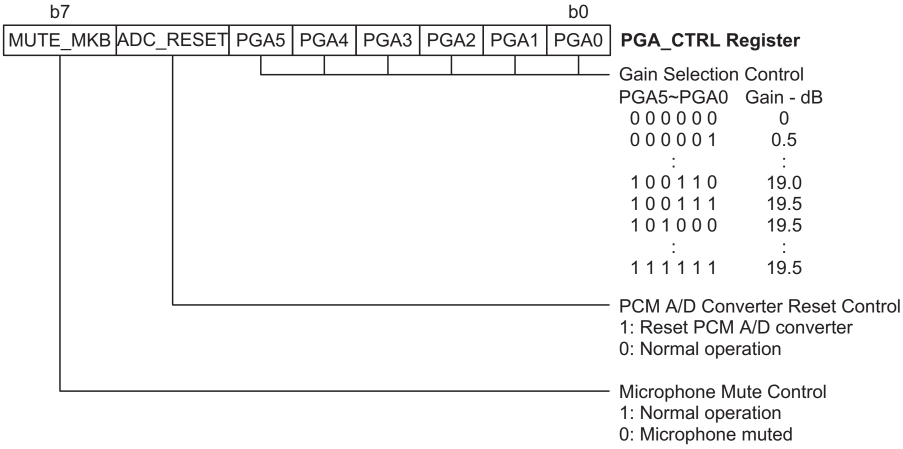
Digital Programmable Gain Amplifier – PGA

The device includes a fully integrated Programmable Gain Amplifier, otherwise known as the PGA. The PGA is a digital amplifier used to amplify the 16-bit data that comes from the PCM A/D Converter (REC_MODE (2FH.0)=0) or from the RECORD_IN_H and RECORD_IN_L (REC_MODE (2FH.0)=1).

The PGA function is controlled using the PGA_CTRL register within which there is six bits to control the gain value. This gain value ranges from 0dB up to a maximum of 19.5dB, in steps of 0.5dB, and is selected using the PGA0~PGA5 bits.



PGA Block Diagram



Gain Selection Control

PGA5~PGA0	Gain - dB
0 0 0 0 0 0	0
0 0 0 0 0 1	0.5
:	:
1 0 0 1 1 0	19.0
1 0 0 1 1 1	19.5
1 0 1 0 0 0	19.5
:	:
1 1 1 1 1 1	19.5

PCM A/D Converter Reset Control
1: Reset PCM A/D converter
0: Normal operation

Microphone Mute Control
1: Normal operation
0: Microphone muted

Programmable Gain Amplifier Control Register – PGA_CTRL

SPI Serial Interface

The device includes a single SPI Serial Interface. The SPI interface is a full duplex serial data link, originally designed by Motorola, which allows multiple devices connected to the same SPI bus to communicate with each other. The devices communicate using a master/slave technique where only the single master device can initiate a data transfer. A simple four line signal bus is used for all communication.

SPI Interface Communication

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

SPI Registers

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

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

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

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

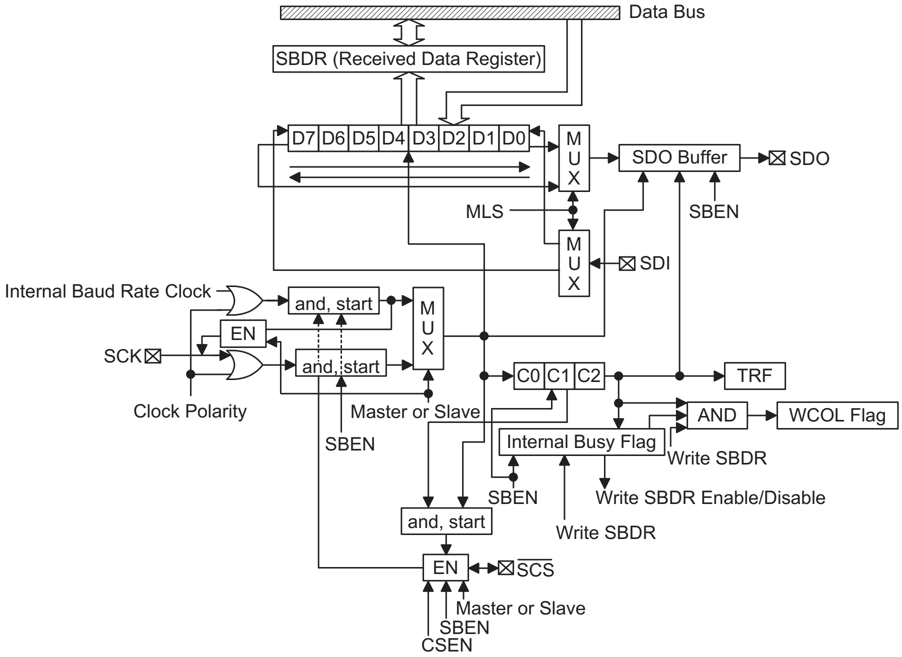
SPI Bus Enable/Disable

To enable the SPI bus then CSEN=1 and SBEN=1, the SCK, SDI, SDO and $\overline{\text{SCS}}$ lines should all be zero, then wait for data to be written to the SBDR (TXRX buffer) register. For the Master Mode, after data has been written to the SBDR (TXRX buffer) register then transmission or reception will start automatically. When all the data has been transferred the TRF bit should be set. For the Slave Mode, when clock pulses are received on SCK, data in the TXRX buffer will be shifted out or data on SDI will be shifted in.

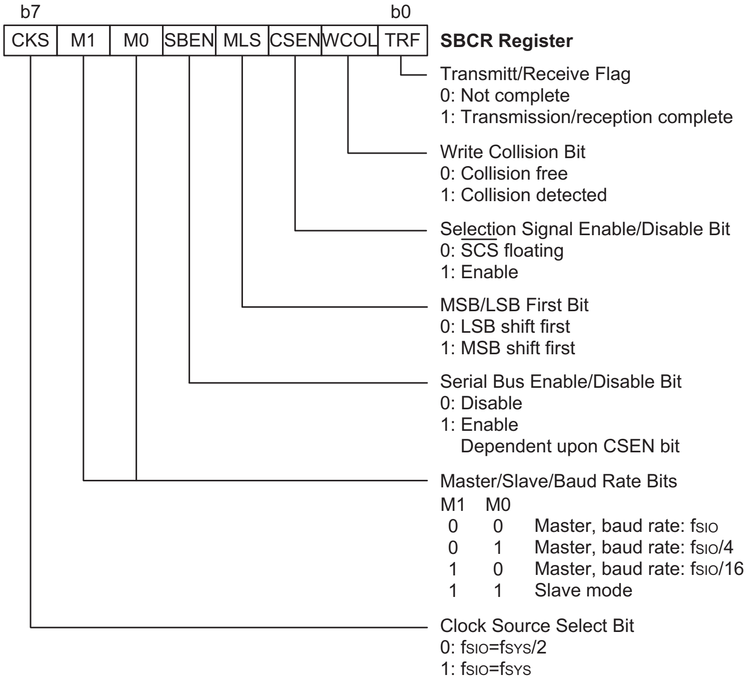
To Disable the SPI bus SCK, SDI, SDO, $\overline{\text{SCS}}$ floating.

SPI Operation

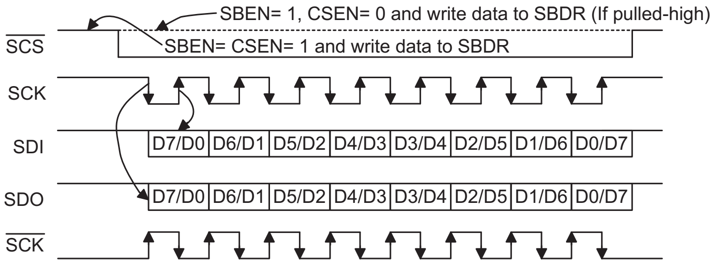
All communication is carried out using the 4-line interface for both Master or Slave Mode. The timing diagram shows the basic operation of the bus.



SPI Block Diagram



SPI Interface Control Register



SPI Bus Timing

The SBEN bit determines if pins PC4~PC7 are used as normal I/O pins or as SPI function pins. If this bit is high then the pins will be SPI function pins and here pin $\overline{\text{SCS}}$ will go low if CSEN=1. If the bit is low then the pins will function as normal I/O pins. Note that when SBEN=1, then any pull-high resistors connected to pins PC4~PC7 will be disconnected therefore the user hardware should ensure that external pull-high resistors are added to the SPI pins if necessary. If CSEN = 0 then the $\overline{\text{SCS}}$ pin will be in a floating state.

The SPI clock polarity is controlled using the SIO_CPOL bit in the MODE_CTRL register. If SIO_CPOL = 1, then the rising edge will be selected. Otherwise if SIO_CPOL = 0, the falling edge will be selected.

The CSEN bit in the SBCR register controls the overall function of the SPI interface. Setting this bit high, will enable the SPI interface by allowing the $\overline{\text{SCS}}$ line to be active, which can then be used to control the SPI interface. If the CSEN bit is low, the SPI interface will be disabled and the $\overline{\text{SCS}}$ line will be in a floating condition and can therefore not be used for control of the SPI interface. The SBEN bit in the SBCR register must also be high which will change the pin function from a standard I/O to an SPI function pin. If in the Master Mode the SCK line will be either high or low depending upon the clock polarity configuration option. If in the Slave Mode the SCK line will be in a floating condition. If SBEN is low then the bus will be disabled.

In the Master Mode the Master will always generate the clock signal. The clock and data transmission will be initiated after data has been written to the SBDR register.

In the Slave Mode, the clock signal will be received from an external master device for both data transmission or reception. The following sequences show the order to be followed for data transfer in both Master and Slave Mode:

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

SBEN= ″1″	PC4~PC7 are SPI function pins (pin $\overline{\text{SCS}}$ will go low if CSEN=1).
SBEN= ″0″	PC4~PC7 are general purpose I/O Port pins (Default)

Note: (1) If SBEN=″1″, the pull-high resistor of PC4~PC7 will be disable. When this happens, the user should add external pull-high resistors to the SPI related pins if necessary (EX: pin $\overline{\text{SCS}}$).

(2) If CSEN=″0″, the $\overline{\text{SCS}}$ pin will enter a floating state.

The SPI cock polarity controlled by SIO_CPOL bit of MODE_CTRL register. If SIO_CPOL =″1″, rising edge ($\overline{\text{CLK}}$) will be selected. Otherwise SIO_CPOL=″0″, falling edge (CLK) will be selected.

Error Detection

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

Programming Considerations

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

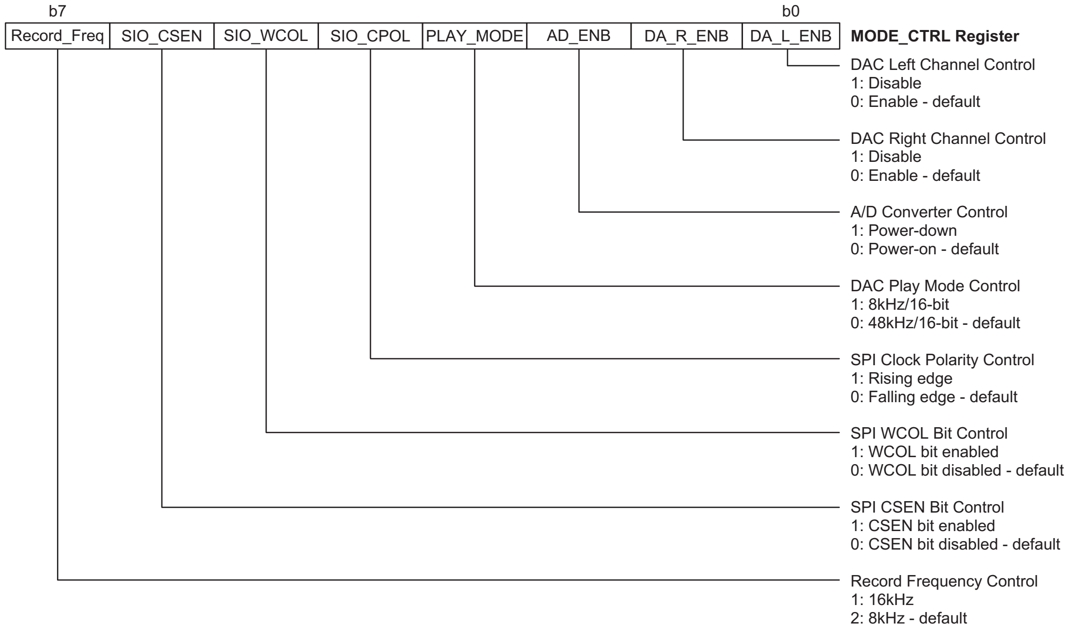
Mode Control Register

The MODE_CTRL register is used to control DAC and ADC operation mode and SPI function.

Note that the WCOL and CSEN bits are in the SBCR register.

SPI usage example:

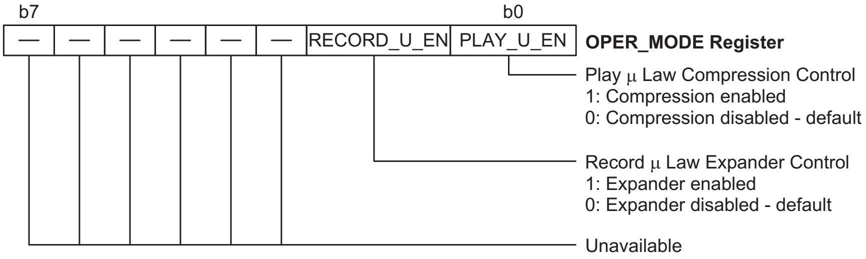
```
SPI_Test:
            clr    UCC.@UCC_SYSCLK     ;12MHz SYSCLK
set         SIO_CSEN                   ;SPI Chip Select Function Enable
            clr    SIO_CPOL            ;falling edge change data
            ;Master Mode, SCLK=fSIO
            clr       M1
            clr       M0
            ;--------------
            clr    CKS                 ;fSIO=fSYS/2
            clr    TRF                 ;clear TRF flag
            clr    TRF_INT             ;clear Interrupt SPI flag
            set    MLS                 ;MSB shift first
            set    CSEN                ;Chip Select Enable
            set    SBEN                ;SPI Enable, SCS will go low
if POLLING_MODE
            clr    ESII                ;SPI Interrupt Disable
;WRITE INTO "WRITE ENABLE" INSTRUCTION
            MOV    A,OP_WREN
            MOV    SBDR,A
$0:
            snz    TRF
            jmp    $0
            clr    TRF
else
            set    ESII                ;SPI Interrupt Enable
;WRITE INTO "WRITE ENABLE" INSTRUCTION
            MOV    A,OP_WREN
            MOV    SBDR,A
$0:
            snz    TRF_INT             ;set at SPI Interrupt
            jmp    $0
            clr    TRF_INT
endif
```



Mode Control Register – MODE_CTRL

Operation Mode Control Register

The OPER_MODE register is used to control certain operational operational modes. The operation mode is used to control the µ law compander enable/disable for the speaker and microphone data.



Operation Mode Control Register – OPER_MODE

Play/Record Data

The Play and Record data for the device is contained in 4 Play registers and 2 Record registers. The play/record interrupt will be activated when play/record data in the PLAY_DATA or RECORD_DATA registers is valid. The PLAY_DATA/RECORD_DATA registers will latch data until the next interrupt is generated. The DAC PLAY_DATA register contains an unsigned value with a range of 0~FFFFH. The RECORD_DATA is stored in 2′s complement format with a range of 8000H~7FFFH.

The update rate of the PCM ADC RECORD_DATA is 8kHz with the Record_Freq bit in the MODE_CTRL register is equal to 0, or 16kHz if the bit is set to 1. The update rate for the PLAY_DATA is 48kHz, if the PLAY_MODE bit in the MODE_CTRL register is equal to 0, or 8kHz if the bit is equal to 1. All of the PLAY and RECORD registers are read only.

Name	b7	b6	b5	b4	b3	b2	b1	b0
PLAY_DATAL_L	PL_D7	PL_D6	PL_D5	PL_D4	PL_D3	PL_D2	PL_D1	PL_D0
PLAY_DATAL_H	PL_D15	PL_D14	PL_D13	PL_D12	PL_D11	PL_D10	PL_D9	PL_D8
PLAY_DATAR_L	PR_D7	PR_D6	PR_D5	PR_D4	PR_D3	PR_D2	PR_D1	PR_D0
PLAY_DATAR_H	PR_D15	PR_D14	PR_D13	PR_D12	PR_D11	PR_D10	PR_D9	PR_D8
RECORD_DATA_L	R_D7	R_D6	R_D5	R_D4	R_D3	R_D2	R_D1	R_D0
RECORD_DATA_H	R_D15	R_D14	R_D13	R_D12	R_D11	R_D10	R_D9	R_D8

The play data format is controlled by bit PLAY_U_EN in the OPER_MODE register.

	PLAY_DATAL_L	PLAY_DATAL_H	PLAY_DATAR_L	PLAY_DATAR_H
PLAY_U_EN=0	PCM (Left Channel Low Byte)	PCM (Left Channel High Byte)	PCM (Right Channel Low Byte)	PCM (Right Channel High Byte)
PLAY_U_EN=1	N/A	µ Law (Left Channel)	N/A	µ Law (Right Channel)

The record data registers RECORD_DATA_L/RECORD_DATA_H will not be affected by bit RECORD_U_EN in the OPER_MODE register. The record data registers RECORD_DATA_L/RECORD_DATA_H are in PCM format.

Pulse Width Modulator

The device contains a 2 channel Pulse Width Modulator function, more commonly known as PWM. Useful for such 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 registers.

The device has two PWM outputs for which two 8-bit PWM registers are provided and are known as PWM0 and PWM1. It is in these registers, that the 8-bit value, which represents the overall duty cycle of one modulation cycle of the output waveform, should be placed. To increase the PWM modulation frequency, each modulation cycle is modulated into two or four individual modulation sub-sections, known as the 7+1 mode or 6+2 respectively. The mode selection is made using the PWMC register. Note that it is only necessary to write the required modulation value into the corresponding PWM0 or PWM1 register, as the subdivision of the waveform into its sub-modulation cycles is implemented automatically within the microcontroller hardware. For all devices, the PWM clock source is the system clock f_{SYS}.

This method of dividing the original modulation cycle into a further 2 or 4 sub-cycles enables the generation of higher PWM frequencies, which allow a wider range of applications to be served. As long as the periods of the generated PWM pulses are less than the time constants of the load, the PWM output will be suitable for driving, as such long time constant loads will average out the pulses of the PWM output. The difference between what is known as the PWM cycle frequency and the PWM modulation frequency should be understood. As the PWM clock is the system clock, f_{SYS}, and as the PWM value is 8-bits wide, the overall PWM cycle frequency is f_{SYS}/256. However when in the 7+1 mode of operation, the PWM modulation frequency will be f_{SYS}/128, while the PWM modulation frequency for the 6+2 mode of operation will be f_{SYS}/64.

The overall PWM output enable/disable is controlled using the PWMC register which acts like an on/off switch for each PWM output.

PWM Modulation	PWM Cycle Frequency	PWM Cycle Duty
f_{SYS}/64 for (6+2) bits mode f_{SYS}/128 for (7+1) bits mode	f_{SYS}/256	[PWM]/256

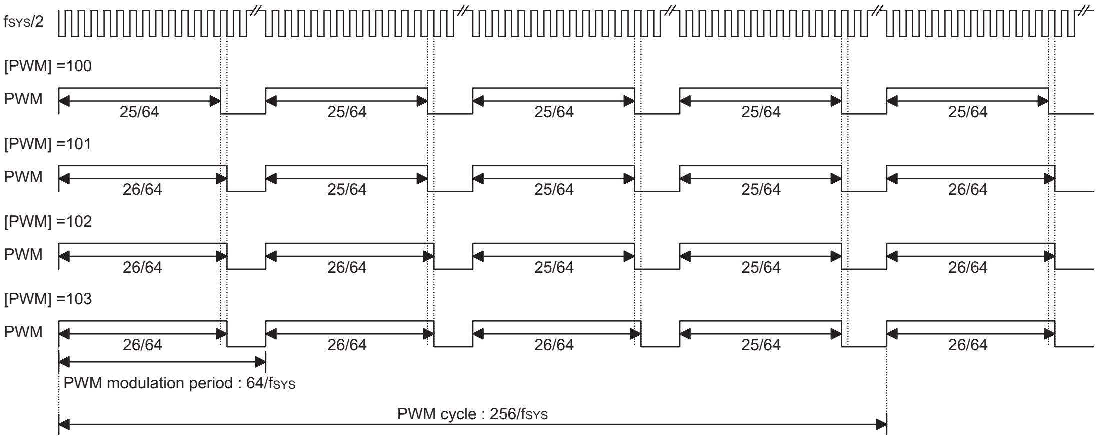
6+2 PWM Mode

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.

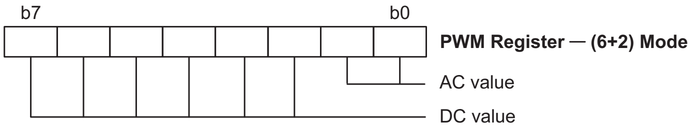
Parameter	AC (0~3)	DC (Duty Cycle)
Modulation cycle i (i=0~3)	i<AC	$\frac{DC+1}{64}$
	i≥AC	$\frac{DC}{64}$

6+2 Mode Modulation Cycle Values

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



Pulse Width Modulation Registers for (6+2) PWM Mode

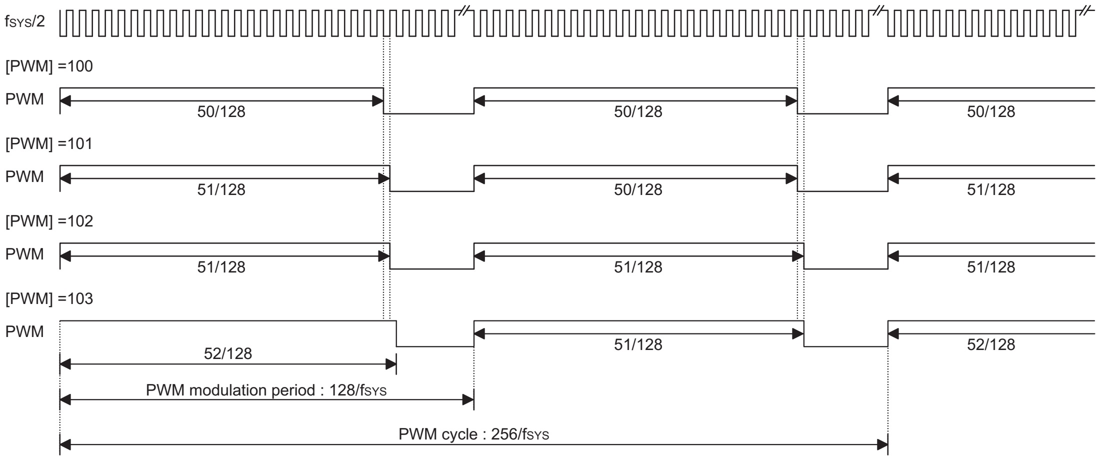
7+1 PWM Mode

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.

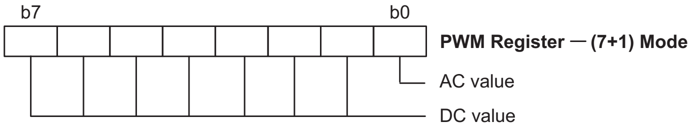
Parameter	AC (0~1)	DC (Duty Cycle)
Modulation cycle i (i=0~1)	i<AC	$\frac{DC+1}{128}$
	i≥AC	$\frac{DC}{128}$

7+1 Mode Modulation Cycle Values

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



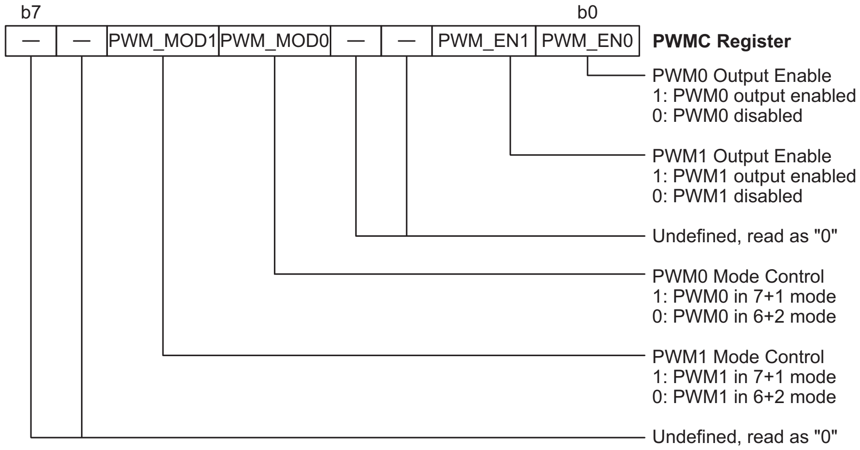
(7+1) PWM Mode Output Waveform



Pulse Width Modulation Registers for (7+1) PWM Mode

PWM Output Control

Control of the two PWM outputs is achieved using the PWMC register. Bits within this register control the on/off function of the individual PWM outputs as well as their chosen mode type. Note than when the PWM outputs are disabled they will remain in a low state.



PWM Control Register – PWMC

PWM Programming Example

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

```
mov a,64h           ; setup PWM0 value of 100 decimal which is 64H
mov PWM0,a
clr PWMC.PWM_MOD0   ; setup pin PWM0 to the 6+2 PWM Mode
set PWMC.PWM_EN0    ; enable PWM0 output
 :
 :
clr PWMC.PWM_EN0    ; disable PWM0 output
```

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 a 6-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 a 12-bit digital value.

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

A/D Converter Data Registers – ADRL, ADRH

For the HT82A836R device, which has a 12-bit A/D converter, two registers are required, a high byte register, known as ADRH, and a low byte register, known as ADRL, to store the 12-bit analog to digital conversion value. After the conversion process takes place, these registers can be directly read by the microcontroller to obtain the digitised conversion value. Note that only the high byte register ADRH utilises its full 8-bit contents. The low byte register utilises only 4 bits of its 8-bit contents as it contains only the lowest bit of the 12-bit converted value.

In the following tables, D0~D11 are the A/D conversion data result bits.

Register	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRL	D3	D2	D1	D0	—	—	—	—
ADRH	D11	D10	D9	D8	D7	D6	D5	D4

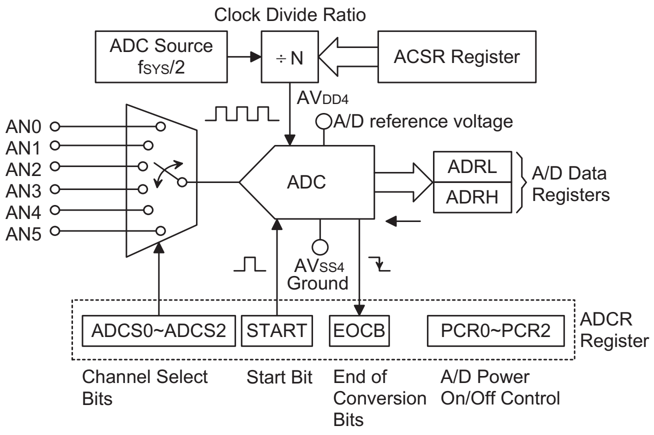
A/D Converter Data Register

A/D Converter Control Register – ADCR

To control the function and operation of the A/D converter, a control register known as ADCR is provided. This 8-bit register defines functions such as the selection of which analog channel is connected to the internal A/D converter, power on/off the A/D converter, control the start function and monitoring the A/D converter end of conversion status.

One section of this register contains the bits ACS2~ACS0 which define the channel number. As each of the devices contains only one actual analog to digital converter circuit, each of the individual 6 analog inputs must be routed to the converter. It is the function of the ACS2~ACS0 bits in the ADCR register to determine which analog channel is actually connected to the internal A/D converter.

The ADCR control register also contains the PCR2~PCR0 bits which determine power on/off the A/D converter and. If the PCR2~PCR0 bits are all set to zero, then the internal A/D converter circuitry will be powered off to reduce the power consumption. Any other non-zero combination on the PCR2~PCR0 bits will power-on the the A/D converter will be power on which will consume a certain amount of power.



A/D Converter Structure

The START bit in the ADCR register is used to start and reset the A/D converter. When the microcontroller sets this bit from low to high and then low again, an analog to digital conversion cycle will be initiated. When the START bit is brought from low to high but not low again, the EOCB bit in the ADCR register will be set to a ″1″ and the analog to digital converter will be reset. It is the START bit that is used to control the overall on/off operation of the internal analog to digital converter.

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

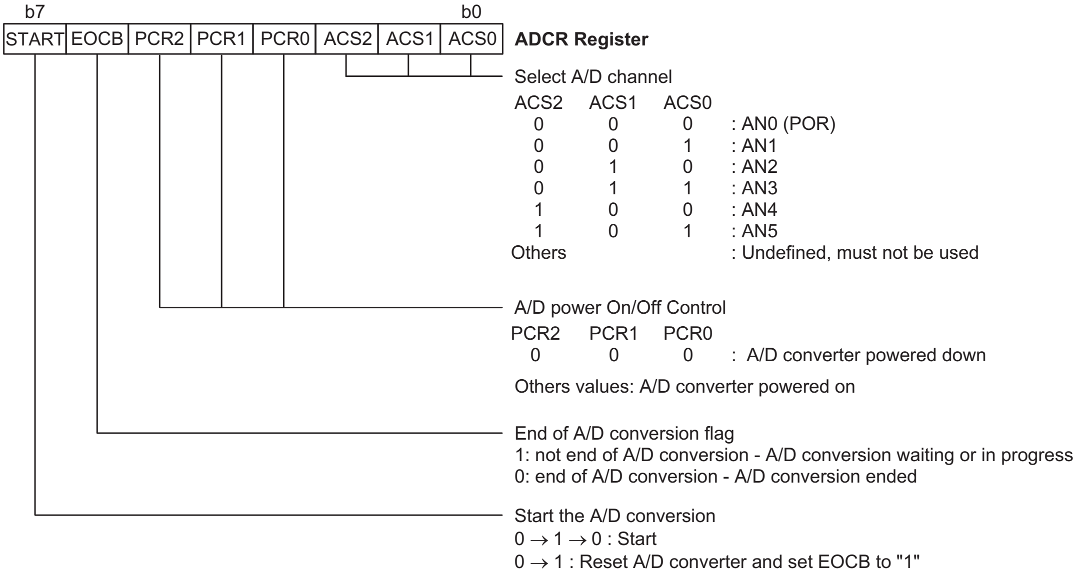
A/D Converter Clock Source Register – ACSR

The clock source for the A/D converter, which originates from the system clock f_{SYS}, is first divided by a division ratio, the value of which is determined by the ADCS1 and ADCS0 bits in the ACSR register.

Although the A/D clock source is determined by the system clock f_{SYS}, and by bits ADCS1 and ADCS0, there are some limitations on the maximum A/D clock source speed that can be selected.

A/D Input Pins

All of the A/D analog input pins are independent analog inputs and not shared with other I/O pins. Bits PCR2~PCR0 in the ADCR register, not configuration options, determine whether the A/D converter is powered on or powered down. The AVDD4 power supply pin is used as the A/D converter reference voltage, and as such analog inputs must not be allowed to exceed this value. Appropriate measures should also be taken to ensure that the AVDD4 pin remains as stable and noise free as possible.



Select A/D channel

ACS2	ACS1	ACS0	
0	0	0	: AN0 (POR)
0	0	1	: AN1
0	1	0	: AN2
0	1	1	: AN3
1	0	0	: AN4
1	0	1	: AN5
Others			: Undefined, must not be used

A/D power On/Off Control

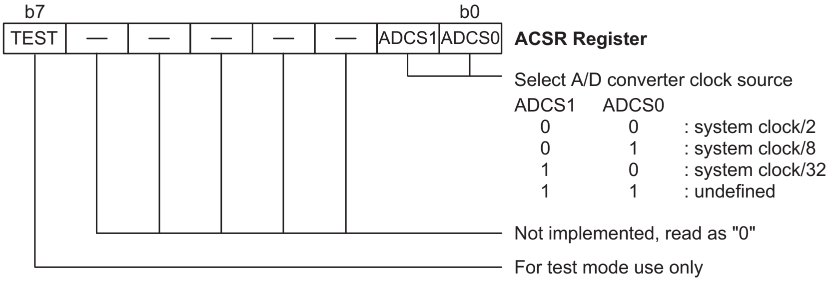
PCR2	PCR1	PCR0	
0	0	0	: A/D converter powered down

Others values: A/D converter powered on

End of A/D conversion flag
1: not end of A/D conversion - A/D conversion waiting or in progress
0: end of A/D conversion - A/D conversion ended

Start the A/D conversion
0 → 1 → 0 : Start
0 → 1 : Reset A/D converter and set EOCB to "1"

ADCR Register



Select A/D converter clock source

ADCS1	ADCS0	
0	0	: system clock/2
0	1	: system clock/8
1	0	: system clock/32
1	1	: undefined

Not implemented, read as "0"

For test mode use only

A/D Converter Clock Source Register

Initialising the A/D Converter

The internal A/D converter must be initialised in a special way. Each time the A/D channel selection bits are modified by the program, the A/D converter must be re-initialised. If the A/D converter is not initialised after the channel selection bits are changed, the EOCB flag may have an undefined value, which may produce a false end of conversion signal. To initialise the A/D converter after the channel selection bits have changed, then, within a time frame of one to ten instruction cycles, the START bit in the ADCR register must first be set high and then immediately cleared to zero. This will ensure that the EOCB flag is correctly set to a high condition.

Summary of A/D Conversion Steps

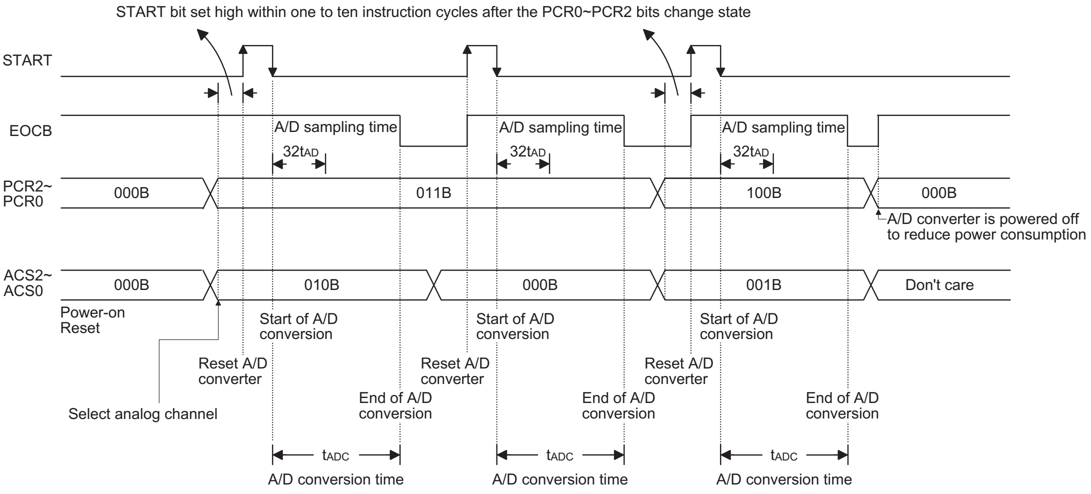
The following summarises the individual steps that should be executed in order to implement an A/D conversion process.

- Step 1
  Select the required A/D conversion clock by correctly programming bits ADCS1 and ADCS0 in the ACSR register.
- Step 2
  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 ADCR register.
- Step 3
  Select A/D converter power on or power down by programming the PCR2~PCR0 bits in the ADCR register. Note that this step can be combined with Step 2 into a single ADCR register programming operation.
- Step 4
  If the interrupts are to be used, the interrupt control registers must be correctly configured to ensure the A/D converter interrupt function is active. The master interrupt control bit, EMI, in the INTC0 interrupt control register must be set to ″1″, the multi-function 1 interrupt control bit, EMF1I, in the INTC1 register and the A/D converter interrupt bit, EADI, in the MFI1C register must also be set to ″1″.
- Step 5
  The analog to digital conversion process can now be initialised by setting the START bit in the ADCR register from ″0″ to ″1″ and then to ″0″ again. Note that this bit should have been originally set to ″0″.
- Step 6
  To check when the analog to digital conversion process is complete, the EOCB bit in the ADCR register can be polled. The conversion process is complete when this bit goes low. When this occurs the A/D data registers 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 ADCR register is used, the interrupt enable step above can be omitted.

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

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



Note: A/D clock must be $f_{SYS}/2$, $f_{SYS}/8$ or $f_{SYS}/32$

A/D Conversion Timing

Programming Considerations

When programming, special attention must be given to the A/D channel selection bits in the ADCR register. This ability to reduce power by turning off the internal A/D function by clearing the A/D channel selection bits may be an important consideration in battery powered applications.

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

A/D Programming Example

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

Example: using an EOCB polling method to detect the end of conversion for the HT82A836R.

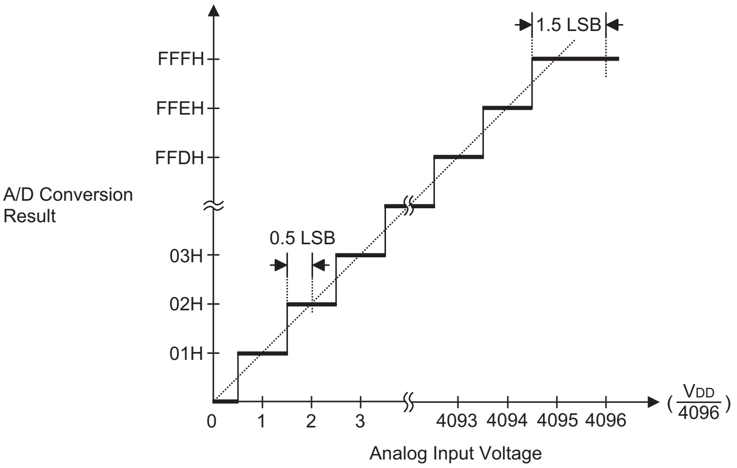
```
            clr EADI                  ; disable ADC interrupt
            mov a,00000001 B
            mov ACSR,a                ; setup the ACSR register to select fSYS/8 as the A/D
                                      ; clock
            mov a,00100000B           ; setup the ADCR register to power up the A/D
                                      ; converter
            mov ADCR,a                ; and select AN0 to be connected to the A/D
                                      ; converter
                   :                  ; the following START signal (0-1-0) must be issued
                   :                  ; within 10 instruction cycles
                   :
Start_conversion:
            clr START
            Set START                 ; reset A/D
            clr START                 ; start A/D
Polling_EOC:
            sz EOCB                   ; poll the ADCR register EOCB bit to detect end
                                      ; of A/D conversion
            jmp polling_EOC           ; continue polling
            mov a,ADRH                ; read conversion result high byte value from the
                                      ; ADRH register
            Adr_buffer_h,a            ; save result to user defined memory
            mov a,ADRL                ; read conversion result low byte value from the
                                      ; ADRL register
            mov adr_buffer_l,a        ; save result to user defined memory
                   :
                   :
            jmp start_conversion      ; start next A/D conversion
```

Example: using an interrupt method to detect the end of conversion for the HT82A836R.

```
            clr EADI ; disable ADC interrupt
            mov a,00000001B
            mov ACSR,a                ; setup the ACSR register to select fSYS/8 as the A/D
                                      ; clock
            mov a,00100000B           ; setup the ADCR register to power up the A/D
                                      ; converter
            mov ADCR,a                ; and select AN0 to be connected to the A/D
                                      ; converter
            :                         ; the following START signal (0-1-0) must be issued

            :                         ; within 10 instruction cycles
            :
Start_conversion:
            clr ADF                   ; clear ADC interrupt request flag
            set EMF1I                 ; Multi function 1 interrupt Enable
            set EADI                  ; enable ADC interrupt
            set EMI                   ; enable global interrupt
            clr START
            set START                 ; reset A/D
            clr START                 ; start A/D

                   :
                   :
                   :
; ADC interrupt service routine
ADC_ISR:
            clr ADF                   ; clear ADC interrupt request flag
            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,ADRH                ; read conversion result high byte value from the
                                      ; ADRH register
            mov adr_buffer_h,a        ; save result to user defined register
            mov a,ADRL                ; read conversion result low byte value from the
                                      ; ADRL register
            mov adr_buffer_l,a        ; save result to user defined register
                   :
                   :
EXIT_INT_ISR:
            mov a,status_stack
            mov STATUS,a              ; restore STATUS from user defined memory
            mov a,acc_stack           ; restore ACC from user defined memory
            reti
```

A/D Transfer Function

As the HT82A836R device contains a 12-bit A/D converter, their full-scale converted digitised value is equal to FFFH. Since the full-scale analog input value is equal to the voltage, this gives a single bit analog input value of $V_{DD}/4096$. The following graphs show the ideal transfer function between the analog input value and the digitised output value for the A/D converters. Note that to reduce the quantisation error, a 0.5 LSB offset is added to the A/D Converter input. Except for the digitised zero value, the subsequent digitised values will change at a point 0.5 LSB below where they would change without the offset, and the last full scale digitised value will change at a point 1.5 LSB below the VDD level.



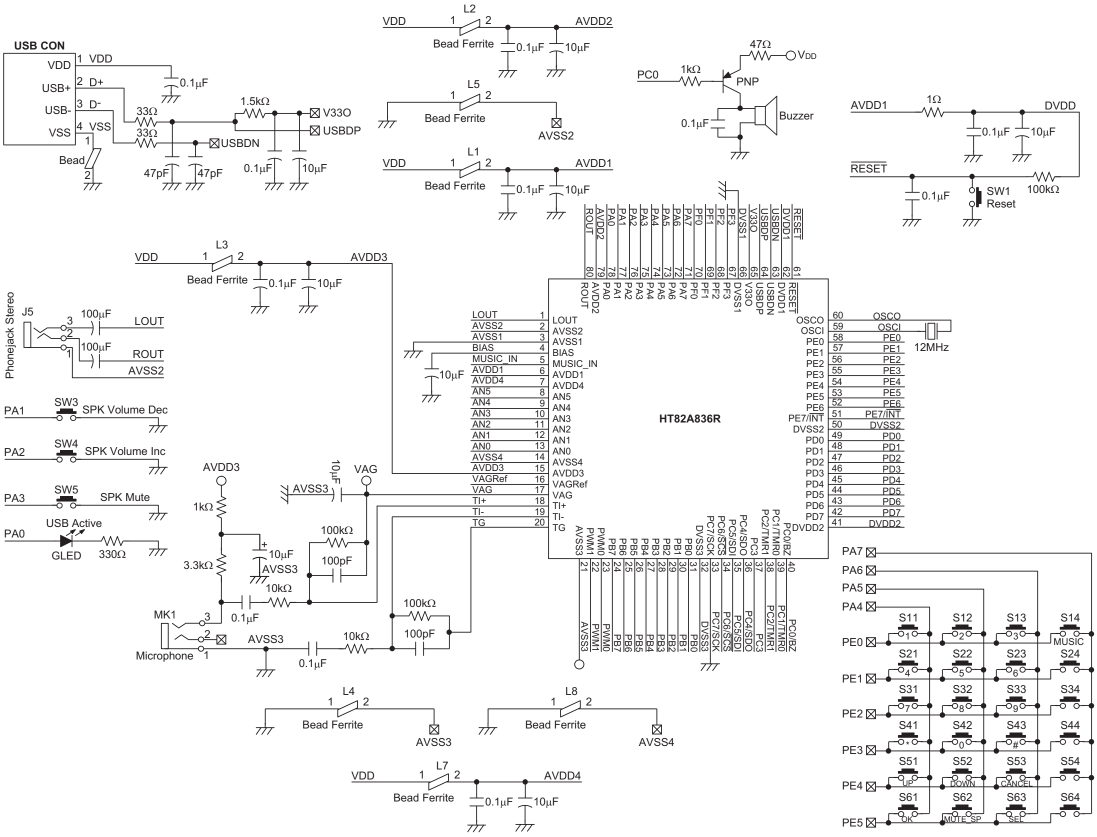
Ideal A/D Transfer Function

Configuration Options

Configuration options refer to certain options within the MCU that are programmed into the OTP Program Memory device during the programming process. During the development process, these options are selected using the HT-IDE software development tools. As these options are programmed into the device using the hardware programming tools, once they are selected they cannot be changed later by the application software. All options must be defined for proper system function, the details of which are shown in the table.

No.	Options
I/O Options	
1	PA0~PA7: pull-high enable or disable (bit option)
2	PB0~PB7: pull-high enable or disable (bit option)
3	PC0~PC7: pull-high enable or disable (nibble option)
4	PD0~PD7: pull-high enable or disable (bit option)
5	PE0~PE7: pull-high enable or disable (bit option)
6	PF0~PF3: pull-high enable or disable (bit option)
7	PA0~PA7: wake-up enable or disable (bit option)
8	PB0~PB7: wake-up enable or disable (bit option)
9	PC0~PC7: wake-up enable or disable (nibble option)
10	PD0~PD7: wake-up enable or disable (bit option)
11	PE0~PE7: wake-up enable or disable (bit option)
12	PF0~PF3: wake-up enable or disable (bit option)
13	PA0~PA7: CMOS or NMOS output type (bit option)
Watchdog Options	
14	WDT: enable or disable
15	CLRWDT instructions: one or two instructions
16	WDT Clock Source: $f_{SYS}/4$ or WDT oscillator
LVR Options	
17	LVR function: enable or disable
TBHP Options	
18	TBHP enable or disable

Application Circuits



Instruction Set

Introduction

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

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

Instruction Timing

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

Moving and Transferring Data

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

Arithmetic Operations

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

Logical and Rotate Operations

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

Branches and Control Transfer

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

Bit Operations

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

Table Read Operations

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

Other Operations

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

Instruction Set Summary

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

Table conventions:

x: Bits immediate data

m: Data Memory address

A: Accumulator

i: 0~7 number of bits

addr: Program memory address

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
ADD A,[m]	Add Data Memory to ACC	1	Z, C, AC, OV
ADDM A,[m]	Add ACC to Data Memory	1^{Note}	Z, C, AC, OV
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV
ADCM A,[m]	Add ACC to Data memory with Carry	1^{Note}	Z, C, AC, OV
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1^{Note}	Z, C, AC, OV
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1^{Note}	Z, C, AC, OV
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1^{Note}	C
Logic Operation			
AND A,[m]	Logical AND Data Memory to ACC	1	Z
OR A,[m]	Logical OR Data Memory to ACC	1	Z
XOR A,[m]	Logical XOR Data Memory to ACC	1	Z
ANDM A,[m]	Logical AND ACC to Data Memory	1^{Note}	Z
ORM A,[m]	Logical OR ACC to Data Memory	1^{Note}	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1^{Note}	Z
AND A,x	Logical AND immediate Data to ACC	1	Z
OR A,x	Logical OR immediate Data to ACC	1	Z
XOR A,x	Logical XOR immediate Data to ACC	1	Z
CPL [m]	Complement Data Memory	1^{Note}	Z
CPLA [m]	Complement Data Memory with result in ACC	1	Z
Increment & Decrement			
INCA [m]	Increment Data Memory with result in ACC	1	Z
INC [m]	Increment Data Memory	1^{Note}	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1^{Note}	Z

Mnemonic	Description	Cycles	Flag Affected
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	1^{Note}	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	1^{Note}	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	1^{Note}	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	1^{Note}	C
Data Move			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	1^{Note}	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	1^{Note}	None
SET [m].i	Set bit of Data Memory	1^{Note}	None
Branch			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	1^{Note}	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	1^{note}	None
SZ [m].i	Skip if bit i of Data Memory is zero	1^{Note}	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	1^{Note}	None
SIZ [m]	Skip if increment Data Memory is zero	1^{Note}	None
SDZ [m]	Skip if decrement Data Memory is zero	1^{Note}	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	1^{Note}	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	1^{Note}	None
CALL addr	Subroutine call	2	None
RET	Return from subroutine	2	None
RET A,x	Return from subroutine and load immediate data to ACC	2	None
RETI	Return from interrupt	2	None
Table Read			
TABRDC [m]	Read table (current page) to TBLH and Data Memory	2^{Note}	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2^{Note}	None
Miscellaneous			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	1^{Note}	None
SET [m]	Set Data Memory	1^{Note}	None
CLR WDT	Clear Watchdog Timer	1	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer	1	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer	1	TO, PDF
SWAP [m]	Swap nibbles of Data Memory	1^{Note}	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC	1	None
HALT	Enter power down mode	1	TO, PDF

Note:

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

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

ADCM A,[m]	Add ACC to Data Memory with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C

ADD A,[m]	Add Data Memory to ACC
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m]
Affected flag(s)	OV, Z, AC, C

ADD A,x	Add immediate data to ACC
Description	The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + x
Affected flag(s)	OV, Z, AC, C

ADDM A,[m]	Add ACC to Data Memory
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m]
Affected flag(s)	OV, Z, AC, C

AND A,[m]	Logical AND Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ [m]
Affected flag(s)	Z

AND A,x	Logical AND immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ x
Affected flag(s)	Z

ANDM A,[m]	Logical AND ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″AND″ [m]
Affected flag(s)	Z

CALL addr	Subroutine call
Description	Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.
Operation	Stack ← Program Counter + 1 Program Counter ← addr
Affected flag(s)	None
CLR [m]	Clear Data Memory
Description	Each bit of the specified Data Memory is cleared to 0.
Operation	[m] ← 00H
Affected flag(s)	None
CLR [m].i	Clear bit of Data Memory
Description	Bit i of the specified Data Memory is cleared to 0.
Operation	[m].i ← 0
Affected flag(s)	None
CLR WDT	Clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF

CPL [m]	Complement Data Memory
Description	Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa.
Operation	[m] ← $\overline{[m]}$
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 and the carry flag are rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation	ACC.i ← [m].(i+1); (i = 0~6) ACC.7 ← [m].0
Affected flag(s)	None
RRC [m]	Rotate Data Memory right through Carry
Description	The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.
Operation	[m].i ← [m].(i+1); (i = 0~6) [m].7 ← C C ← [m].0
Affected flag(s)	C
RRCA [m]	Rotate Data Memory right through Carry with result in ACC
Description	Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation	ACC.i ← [m].(i+1); (i = 0~6) ACC.7 ← C C ← [m].0
Affected flag(s)	C

SBC A,[m]	Subtract Data Memory from ACC with Carry
Description	The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC − [m] − $\overline{C}$
Affected flag(s)	OV, Z, AC, C

SBCM A,[m]	Subtract Data Memory from ACC with Carry and result in Data Memory
Description	The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	[m] ← ACC − [m] − $\overline{C}$
Affected flag(s)	OV, Z, AC, C

SDZ [m]	Skip if decrement Data Memory is 0
Description	The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	[m] ← [m] − 1 Skip if [m] = 0
Affected flag(s)	None

SDZA [m]	Skip if decrement Data Memory is zero with result in ACC
Description	The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.
Operation	ACC ← [m] − 1 Skip if ACC = 0
Affected flag(s)	None

SET [m]	Set Data Memory
Description	Each bit of the specified Data Memory is set to 1.
Operation	[m] ← FFH
Affected flag(s)	None

SET [m].i	Set bit of Data Memory
Description	Bit i of the specified Data Memory is set to 1.
Operation	[m].i ← 1
Affected flag(s)	None

SIZ [m]	Skip if increment Data Memory is 0
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	$[m] \leftarrow [m] + 1$ Skip if $[m] = 0$
Affected flag(s)	None

SIZA [m]	Skip if increment Data Memory is zero with result in ACC
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	$ACC \leftarrow [m] + 1$ Skip if $ACC = 0$
Affected flag(s)	None

SNZ [m].i	Skip if bit i of Data Memory is not 0
Description	If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.
Operation	Skip if $[m].i \neq 0$
Affected flag(s)	None

SUB A,[m]	Subtract Data Memory from ACC
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$ACC \leftarrow ACC - [m]$
Affected flag(s)	OV, Z, AC, C

SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$[m] \leftarrow ACC - [m]$
Affected flag(s)	OV, Z, AC, C

SUB A,x	Subtract immediate data from ACC
Description	The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$ACC \leftarrow ACC - x$
Affected flag(s)	OV, Z, AC, C

SWAP [m]	Swap nibbles of Data Memory
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged.
Operation	[m].3~[m].0 ↔ [m].7 ~ [m].4
Affected flag(s)	None

SWAPA [m]	Swap nibbles of Data Memory with result in ACC
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC.3 ~ ACC.0 ← [m].7 ~ [m].4 ACC.7 ~ ACC.4 ← [m].3 ~ [m].0
Affected flag(s)	None

SZ [m]	Skip if Data Memory is 0
Description	If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	Skip if [m] = 0
Affected flag(s)	None

SZA [m]	Skip if Data Memory is 0 with data movement to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	ACC ← [m] Skip if [m] = 0
Affected flag(s)	None

SZ [m].i	Skip if bit i of Data Memory is 0
Description	If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.
Operation	Skip if [m].i = 0
Affected flag(s)	None

TABRDC [m]	Read table (current page) to TBLH and Data Memory
Description	The low byte of the program code (current page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

TABRDL [m]	Read table (last page) to TBLH and Data Memory
Description	The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

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

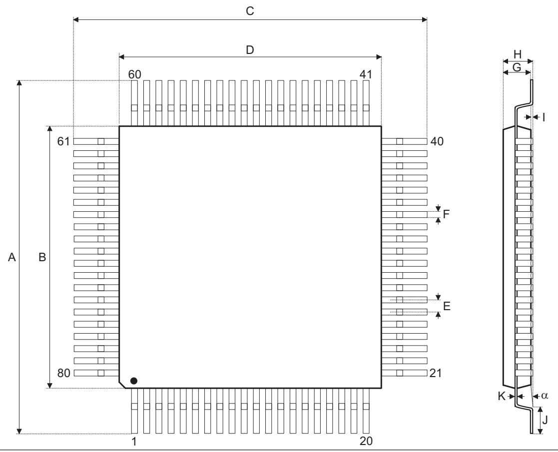
XORM A,[m]	Logical XOR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″XOR″ [m]
Affected flag(s)	Z

XOR A,x	Logical XOR immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ x
Affected flag(s)	Z

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 (http://www.holtek.com.tw/english/literature/package.pdf) for the latest version of the package information.

80-pin LQFP (10mm×10mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.469	—	0.476
B	0.390	—	0.398
C	0.469	—	0.476
D	0.390	—	0.398
E	—	0.016	—
F	—	0.006	—
G	0.053	—	0.057
H	—	—	0.063
I	—	0.004	—
J	0.018	—	0.030
K	0.004	—	0.008
α	0°	—	7°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	11.90	—	12.10
B	9.90	—	10.10
C	11.90	—	12.10
D	9.90	—	10.10
E	—	0.40	—
F	—	0.16	—
G	1.35	—	1.45
H	—	—	1.60
I	—	0.10	—
J	0.45	—	0.75
K	0.10	—	0.20
α	0°	—	7°

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

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

Holtek Semiconductor (China) Inc. (Dongguan Sales Office)
Building No. 10, Xinzhu Court, (No. 1 Headquarters), 4 Cuizhu Road, Songshan Lake, Dongguan, China 523808
Tel: 86-769-2626-1300
Fax: 86-769-2626-1311

Holtek Semiconductor (USA), Inc. (North America Sales Office)
46729 Fremont Blvd., Fremont, CA 94538, USA
Tel: 1-510-252-9880
Fax: 1-510-252-9885
http://www.holtek.com