I/O MCU with USB Interface

# HT82B42R/HT82B42RE

Revision: V.1.10 Date: November 05, 2014

www.holtek.com

# Table of Contents

**Features** ..... **5**
**General Description** ..... **5**
**Selection Table** ..... **6**
**Block Diagram** ..... **6**
HT82B42R ..... 6
HT82B42RE ..... 7
**Pin Assignment** ..... **8**
**Pin Description** ..... **9**
**Absolute Maximum Ratings** ..... **11**
**D.C. Characteristics** ..... **11**
**EEPROM Memory D.C. Characteristics** ..... **12**
**A.C. Characteristics** ..... **12**
**Power-on Reset Characteristics** ..... **13**
**System Architecture** ..... **13**
Clocking and Pipelining ..... 13
Program Counter ..... 14
Stack ..... 15
Arithmetic and Logic Unit – ALU ..... 15
**Program Memory** ..... **16**
Structure ..... 16
Special Vectors ..... 17
Look-up Table ..... 18
Table Program Example ..... 19
**Data Memory** ..... **20**
Structure ..... 20
**General Purpose Data Memory** ..... **20**
**Special Function Registers** ..... **21**
Indirect Addressing Register – IAR0, IAR1 ..... 21
Memory Pointer – MP0, MP1 ..... 22
Accumulator – ACC ..... 22
Program Counter Low Register – PCL ..... 22
Look-up Table Registers – TBLP, TBLH, TBHP ..... 23
Status Register – STATUS ..... 23
Bank Pointer – BP ..... 24

**Input/Output Ports ..... 25**
Pull-high Resistors ..... 25
Port A CMOS/NMOS/PMOS Structure ..... 25
Port A VDD/V33O Option Structure ..... 25
Port Pin Wake-up ..... 25
I/O Port Control Registers ..... 26
Pin-shared Functions ..... 26
Programming Considerations ..... 28
**Timer/Event Counters ..... 28**
Configuring the Timer/Event Counter Input Clock Source ..... 29
Timer Register – TMR0, TMR1L/TMR1H ..... 29
Timer Control Register – TMR0C/TMR1C ..... 30
Configuring the Timer Mode ..... 32
Configuring the Event Counter Mode ..... 32
Configuring the Pulse Width Measurement Mode ..... 33
I/O Interfacing ..... 34
Programming Considerations ..... 34
Timer Program Example ..... 35
**Interrupts ..... 36**
Interrupt Registers ..... 36
Interrupt Operation ..... 37
Interrupt Priority ..... 38
Timer/Event Counter Interrupt ..... 38
Programming Considerations ..... 38
USB Interrupt ..... 39
Serial Interface Interrupt ..... 39
**Reset and Initialisation ..... 39**
Reset Functions ..... 40
Reset Initial Conditions ..... 42
**Oscillator ..... 44**
Watchdog Timer Oscillator ..... 44
**Power Down Mode and Wake-up ..... 44**
Power Down Mode ..... 44
Entering the Power Down Mode ..... 44
Standby Current Considerations ..... 45
Wake-up ..... 45
**Watchdog Timer ..... 46**
**USB Interface ..... 47**
Suspend Wake-Up and Remote Wake-Up ..... 47
To Configure as PS2 Device ..... 48
USB Control Registers ..... 48
STALL and PIPE, PIPE_CTRL, Endpt_EN Registers ..... 51

**Serial Interface – SPI ..... 57**
SPI Interface Operation ..... 57
SPI Registers ..... 58
SPI Communication ..... 61
SPI Bus Enable/Disable ..... 63
SPI Operation ..... 64
Error Detection ..... 65
**Configuration Options ..... 66**
**Application Circuit ..... 66**
**Instruction Set ..... 67**
Introduction ..... 67
Instruction Timing ..... 67
Moving and Transferring Data ..... 67
Arithmetic Operations ..... 67
Logical and Rotate Operations ..... 68
Branches and Control Transfer ..... 68
Bit Operations ..... 68
Table Read Operations ..... 68
Other Operations ..... 68
**Instruction Set Summary ..... 69**
Table Conventions ..... 69
**Instruction Definition ..... 71**
**Package Information ..... 80**
16-pin NSOP (150mil) Outline Dimensions ..... 81
20-pin SSOP (150mil) Outline Dimensions ..... 82
SAW Type 20-pin (4mm×4mm) QFN Outline Dimensions ..... 83

## Features

- Operating voltage:
  $f_{SYS}$=6M/12MHz: 3.3V~5.5V
- Fully integrated 6MHz or 12MHz oscillator
- 4096×15 program memory
- 160×8 data memory RAM
- EEPROM Memory: 128×8 (For HT82B42RE only)
- USB 2.0 low speed function
- PS2 and USB modes supported
- 3 endpoints supported – endpoint 0 included
- Integrated 1.5kΩ resistor between V33O and UDN pins for USB applications
- Internal Regulator with 3.3V output
- 8-level subroutine nesting
- 15 bidirectional I/O lines (max.)
- 8-bit programmable timer/event counter with overflow interrupt
- 16-bit programmable timer/event counter with overflow interrupt
- Watchdog Timer
- Serial SPI Interface
- All I/O pins have wake-up functions
- Power-down function and wake-up feature reduce power consumption
- Up to 0.33μs instruction cycle with 12MHz system clock at $V_{DD}$=5V
- Bit manipulation instruction
- 15-bit table read instruction
- 63 powerful instructions
- All instructions in one or two machine cycles
- Low voltage reset function
- 16-pin NSOP, 20-pin SSOP and 20-pin QFN packages

## General Description

The HT82B42R is 8-bit high performance, RISC architecture microcontroller devices specifically designed for multiple I/O control product applications. In addition, the HT82B42RE is embedded with an EEPROM device.

The advantages of low power consumption, I/O flexibility, timer functions, integrated USB interface, serial SPI interface, Power Down and wake-up functions, Watchdog timer etc., make the devices extremely suitable for use in computer peripheral product applications as well as many other applications such as industrial control, consumer products, subsystem controllers, etc..

EEPROM memory is incorporated into the HT82B42RE, which is useful for applications that require an area of non-volatile memory, perhaps to store information such as calibration parameters, part numbers etc.. Most features are common to the HT82B42R, however, they differ in the provision of an EEPROM.

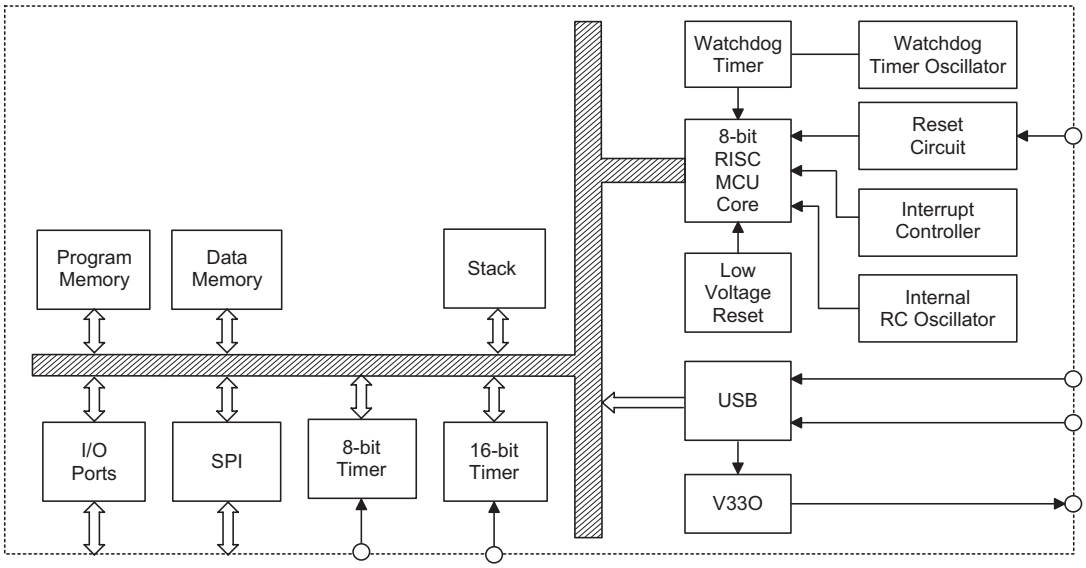
## Selection Table

Most features are common to all devices, the main feature distinguishing them is EEPROM. The following table summarizes the main features of each device.

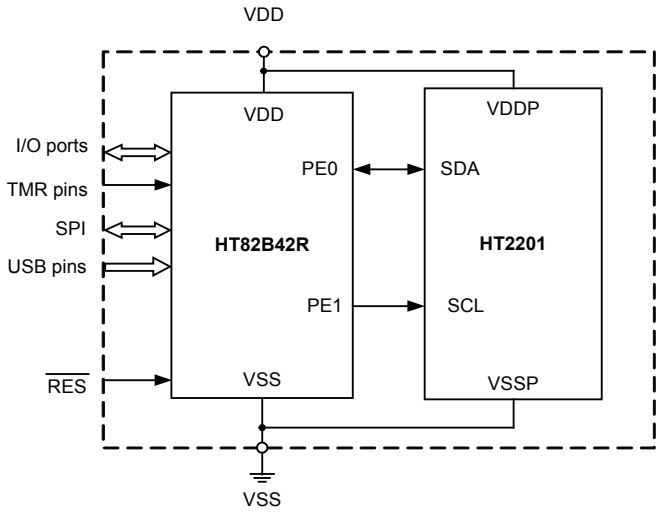
| Part No. | VDD | Program Memory | Data Memory | Data EEPROM | I/O | Timer | | End-Points | HIRC (MHz) | LDO 70mA | I/O VDD Option | SPI | Stack | Package |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | 8-bit | 16-bit | | | | | | | |
| HT82B42R | 3.3V~5.5V | 4k×15 | 160×8 | — | 15 | 1 | 1 | 3 | 6/12 | √ | √ | √ | 8 | 16NSOP<br>20SSOP<br>20QFN |
| HT82B42RE | 3.3V~5.5V | 4k×15 | 160×8 | 128x8 | 13 | 1 | 1 | 3 | 6/12 | √ | √ | √ | 8 | 20QFN |

## Block Diagram

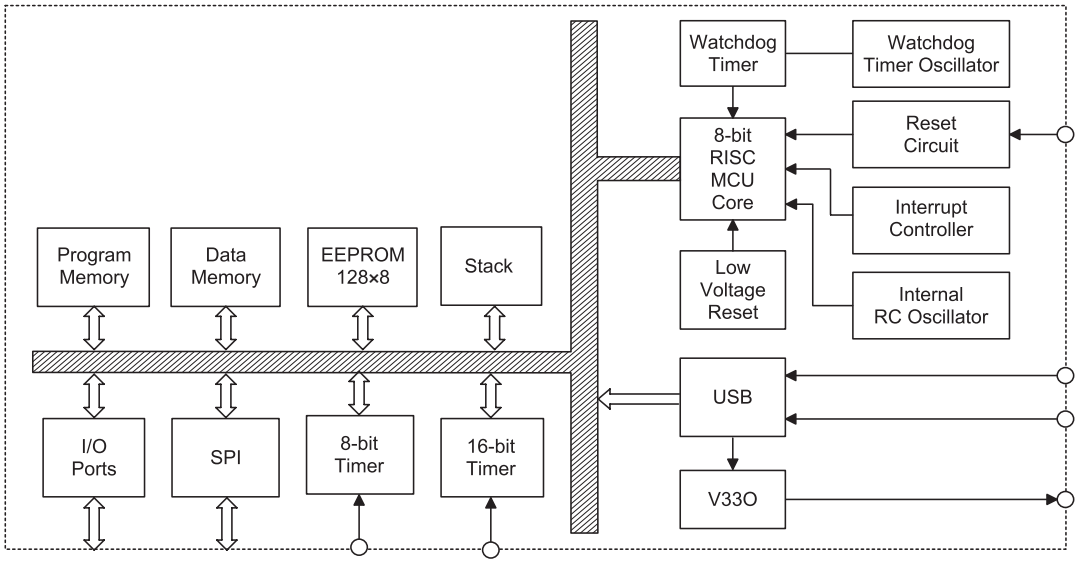
### HT82B42R



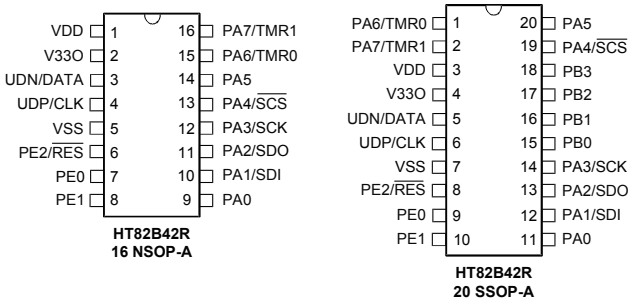
The following block diagrams illustrate the dual-chip structure of the devices, where an individual MCU with EEPROM devices are combined into a single package.

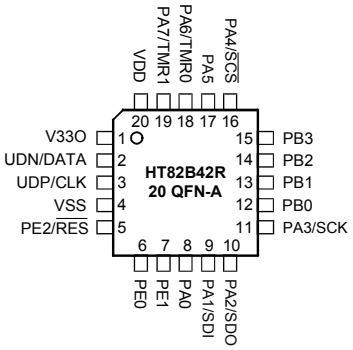


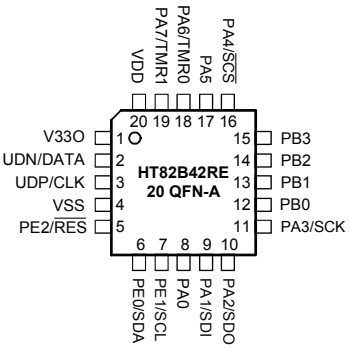

## HT82B42RE



## Pin Assignment



## Pin Description

The pins on this device can be referenced by their Port name, e.g. PA.0, PA.1 etc., which refer to the digital I/O function of the pins. However these Port pins are also shared with other function such as the Timers, Serial Port pins etc.. The function of each pin is listed in the following table, however the details behind how each pin is configured is contained in other sections of the datasheet.

**HT82B42R**

| Pin Name | Function | OPT | I/T | O/T | Description |
|---|---|---|---|---|---|
| PA0 | PA0 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| PA1/SDI | PA1 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | SDI | — | ST | — | SPI Data input |
| PA2/SDO | PA2 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | SDO | — | — | CMOS | SPI Data output |
| PA3/SCK | PA3 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | SCK | — | ST | CMOS | SPI Serial Clock |
| PA4/$\overline{SCS}$ | PA4 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | $\overline{SCS}$ | — | ST | — | SPI Slave select |
| PA5 | PA5 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, output structure and VDD or 3.3V voltage output. The output structure can be selected as CMOS, NMOS or PMOS. |
| PA6/TMR0 | PA6 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, output structure and VDD or 3.3V voltage output. The output structure can be selected as CMOS, NMOS or PMOS. |
| | TMR0 | — | ST | — | Timer 0 External input |
| PA7/TMR1 | PA7 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, output structure and VDD or 3.3V voltage output. The output structure can be selected as CMOS, NMOS or PMOS. |
| | TMR1 | — | ST | — | Timer 1 External input |
| PB0~PB3 | PB0~PB3 | CO | ST | CMOS | General purpose I/O. Configuration option enabled pull-up and wake-up. |
| PE0~PE1 | PE0~PE1 | CO | ST | CMOS | General purpose I/O. Configuration option enabled pull-up and wake-up. |
| PE2/$\overline{RES}$ | PE2 | CO | ST | NMOS | General purpose I/O. Configuration option wake-up. |
| | $\overline{RES}$ | CO | ST | — | Reset input |
| UDN/DATA | UDN | — | ST | CMOS | USB D- line |
| | DATA | — | ST | NMOS | PS2 Data line |
| UDP/CLK | UDP | — | ST | CMOS | USB D+ line |
| | CLK | — | ST | NMOS | PS2 CLK line |
| VDD | VDD | — | PWR | — | Power supply |
| VSS | VSS | — | PWR | — | Ground |
| V33O | V33O | — | — | PWR | 3.3V regulator output |

**HT82B42RE**

| Pin Name | Function | OPT | I/T | O/T | Description |
|---|---|---|---|---|---|
| PA0 | PA0 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| PA1/SDI | PA1 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | SDI | — | ST | — | SPI Data input |
| PA2/SDO | PA2 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | SDO | — | — | CMOS | SPI Data output |
| PA3/SCK | PA3 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | SCK | — | ST | CMOS | SPI Serial Clock |
| PA4/$\overline{SCS}$ | PA4 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | $\overline{SCS}$ | — | ST | — | SPI Slave select |
| PA5 | PA5 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| PA6/TMR0 | PA6 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | TMR0 | — | ST | — | Timer 0 External input |
| PA7/TMR1 | PA7 | CO | ST | NMOS /CMOS /PMOS | General purpose I/O. Configuration option enabled pull-up, wake-up, VDD or 3.3V voltage output and output structure, which can be selected as CMOS, NMOS or PMOS. |
| | TMR1 | — | ST | — | Timer 1 External input |
| PB0~PB3 | PB0~PB3 | CO | ST | CMOS | General purpose I/O. Configuration option enabled pull-up and wake-up. |
| PE0/SDA | PE0 | CO | ST | CMOS | General purpose I/O. Configuration option enabled pull-up and wake-up. |
| | SDA | — | — | — | Internal serial data input/output signal |
| PE1/SCL | PE1 | CO | ST | CMOS | General purpose I/O. Configuration option enabled pull-up and wake-up. |
| | SCL | — | — | — | Serial clock input signal |
| PE2/$\overline{RES}$ | PE2 | CO | ST | NMOS | General purpose I/O. Configuration option wake-up. |
| | $\overline{RES}$ | CO | ST | — | Reset input |
| UDN/DATA | UDN | — | ST | CMOS | USB D- line |
| | DATA | — | ST | NMOS | PS2 Data line |
| UDP/CLK | UDP | — | ST | CMOS | USB D+ line |
| | CLK | — | ST | NMOS | PS2 CLK line |
| VDD | VDD | — | PWR | — | Power supply |
| VSS | VSS | — | PWR | — | Ground |
| V33O | V33O | — | — | PWR | 3.3V regulator output |

Note: I/T: Input type; O/T: Output type
OPT: Optional by configuration option (CO) or register option
PWR: Power; CO: Configuration option
ST: Schmitt Trigger input; CMOS: CMOS output;

Where devices exist in more than one package type the table reflects the situation for the package with the largest number of pins. For this reason not all pins described in the table may exist on all package types.

## Absolute Maximum Ratings

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

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

## D.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage (Integrated oscillator) | — | $f_{SYS}$=6MHz or 12MHz | 3.3 | — | 5.5 | V |
| $I_{DD}$ | Operating Current | 5V | No load, $f_{SYS}$=6MHz | — | 6.5 | 12 | mA |
| | | | No load, $f_{SYS}$=12MHz | — | 7.5 | 16 | mA |
| $I_{STB}$ | Standby Current | 5V | No load, system USB suspend, set CLK_adj [22H] "*" | — | — | 400 | μA |
| | Standby Current (WDT Enabled) | | No load, system HALT, input/output mode, set SUSP2 & CLK_adj [22H] | — | — | 15 | μA |
| $V_{IL}$ | Input Low Voltage for PA | 5V | where $V_{DDIO}$=$V_{DD}$ or V33O by option for port A | 0 | — | 0.2$V_{DDIO}$ | V |
| | Input Low Voltage for PB, PE | | | 0 | — | 0.2$V_{DD}$ | V |
| | Input Low Voltage for $\overline{RES}$ pin | | | 0 | — | 0.4$V_{DD}$ | V |
| $V_{IH}$ | Input High Voltage for PA | 5V | where $V_{DDIO}$=$V_{DD}$ or V33O by option for Port A | 0.8$V_{DDIO}$ | — | 5 | V |
| | Input High Voltage for PB, PE | | | 0.8$V_{DD}$ | — | 5 | V |
| | Input High Voltage for $\overline{RES}$ pin | | | 0.9$V_{DD}$ | — | $V_{DD}$ | V |
| $V_{LVR}$ | Low Voltage Reset | 5V | — | 2.0 | 2.6 | 3.2 | V |
| $V_{V33O}$ | 3.3V Regulator Output for USB SIE | 5V | $I_{V33O}$=70mA | 3.0 | 3.3 | 3.6 | V |
| $I_{OH}$ | Output Source Current for I/O Pin PA, PB, PE0~1 | 5V | $V_{OH}$=3.4V | -2 | -4 | — | mA |
| $I_{OL1}$ | Output Sink Current for I/O Pin PA, PB, PE0~1 | 5V | $V_{OL}$=0.4V | 2 | 4 | — | mA |
| $I_{OL2}$ | Output Sink Current for PE2 | 5V | $V_{OL}$=0.1$V_{DD}$ | 2 | 3 | — | mA |
| $R_{PH}$ | Pull-high Resistance for CLK, DATA | 5V | — | — | 4.7 | — | kΩ |
| | Pull-high Resistance for PA, PB, PE0~1 | | | 20 | 50 | 70 | kΩ |

Note: "*" include 15kΩ loading on the UDP, UDN lines at the host terminal.

## EEPROM Memory D.C. Characteristics

Ta=40°C~85°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DDP}$ | Condition | | | | |
| $V_{CC}$ | Operating Voltage | — | — | 2.2 | — | 5.5 | V |
| $I_{CC1}$* | Operating Current | 5V | Read at 100kHz | — | — | 2 | mA |
| $I_{CC2}$* | Operating Current | 5V | Write at 100kHz | — | — | 5 | mA |
| $I_{STB1}$* | Standby Current | 5V | $V_{IN}$=0 or $V_{DDP}$ | — | — | 4 | μA |
| $I_{STB2}$* | Standby Current | 2.4V | $V_{IN}$=0 or $V_{DDP}$ | — | — | 3 | μA |
| $V_{IL}$ | Input Low Voltage | — | — | -1 | — | $0.3V_{DDP}$ | V |
| $V_{IH}$ | Input High Voltage | — | — | $0.7V_{DDP}$ | — | $V_{DDP}$+0.5 | V |
| $V_{OL}$ | Output Low Voltage | 2.4V | $I_{OL}$=2.1mA | — | — | 0.4 | V |
| $I_{LI}$ | Input Leakage Current | 5V | $V_{IN}$=0 or $V_{DDP}$ | — | — | 1 | μA |
| $I_{LO}$ | Output Leakage Current | 5V | $V_{OUT}$=0 or $V_{DDP}$ | — | — | 1 | μA |

Note: *: The operating current $I_{CC1}$ and $I_{CC2}$ listed here are the additional currents consumed when the EEPROM Memory operates in Read Operation and Write Operation respectively. If the EEPROM is operating, the $I_{CC1}$ or $I_{CC2}$ should be added to calculated the relevant operating current of the device for defferent conditions. To calculate the standby current for the whole device, the standby current shown above should also be taken into account.

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{RCSYS}$ | RC Clock with 8-bit Prescaler Register | 5V | — | — | 32 | — | kHz |
| $t_{WDT}$ | Watchdog Time-out Period (System Clock) | — | — | 1024 | — | — | $1/f_{RCSYS}$ |
| $t_{USB}$ | UDP, UDN Rising & Falling Time | — | — | 75 | — | 300 | ns |
| $t_{OST}$ | Oscillation Start-up Timer Period | — | — | — | 1024 | — | $t_{SYS}$ |
| $t_{OSCsetup}$ | Crystal Setup | — | — | — | 5 | — | ms |
| $f_{INO125V}$ | Internal Oscillator Frequency for 12MHz | 4.0V~5.5V | — | 10.80 | 12.00 | 13.20 | MHz |
| $f_{INO123V}$ | Internal Oscillator Frequency for 12MHz | 3.0V~4.0V | — | 10.56 | 12.00 | 13.44 | MHz |
| $f_{INOUSB}$ | Internal Oscillator Frequency with USB Mode | 4.2V~5.5V | — | 11.82 | 12.00 | 12.18 | MHz |

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

Power_on period=$t_{WDT}+t_{OST}+t_{OSCsetup}$

WDT Time_out in Normal Mode=$1/f_{RCSYS}\times256\times WDTS+t_{WDT}$

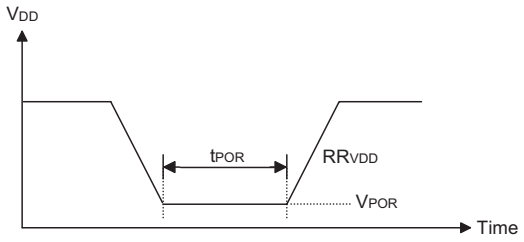
WDT Time_out in Power Down Mode=$1/f_{RCSYS}\times256\times WDTS+t_{OST}+t_{OSC}$setup

Trimmed for 5V operation using factory trim values. Frequency Trim to 12MHz±3%

## Power-on Reset Characteristics

Ta=25°C

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




## System Architecture

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

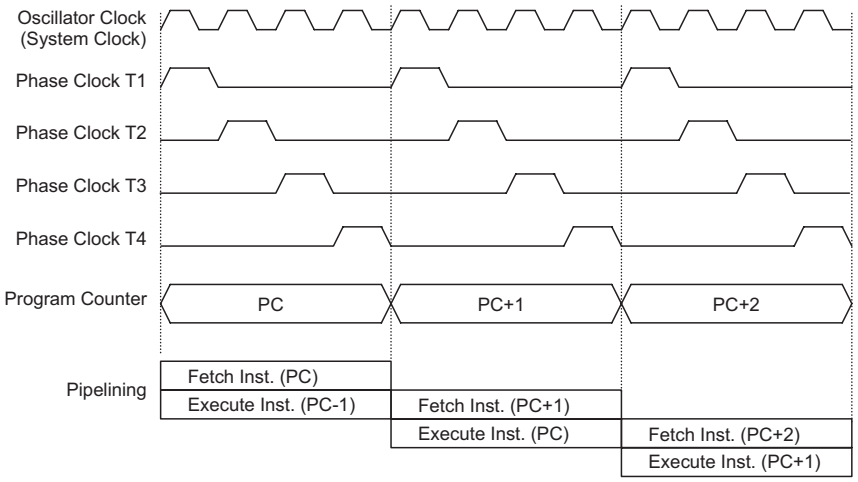
### Clocking and Pipelining

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

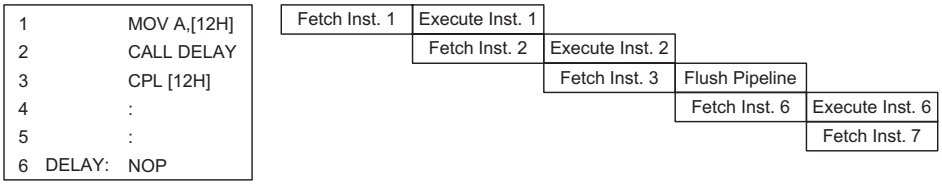
For instructions involving branches, such as jump or call instructions, two machine cycles are required to complete instruction execution. An extra cycle is required as the program takes one cycle to first obtain the actual jump or call address and then another cycle to actually execute the branch. The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

## Program Counter

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as "JMP" or "CALL" that demand a jump to a non-consecutive Program Memory address. It must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by user.




**System Clocking and Pipelining**




**Instruction Fetching**

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.

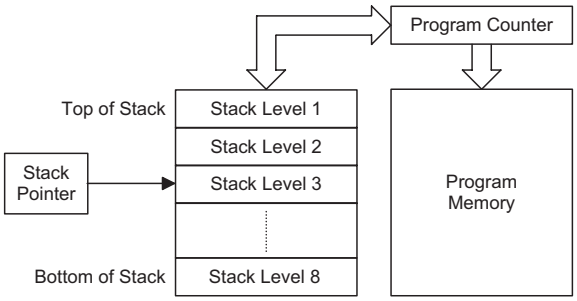
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly, however, as only this low byte is available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

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

## Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has 8 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, SP, and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack. After a device reset, the Stack Pointer will point to the top of the stack.

If the stack is full and an enabled interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal will be inhibited. When the Stack Pointer is decremented, by RET or RETI, the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. However, when the stack is full, a CALL subroutine instruction can still be executed which will result in a stack overflow. Precautions should be taken to avoid such cases which might cause unpredictable program branching.




## Arithmetic and Logic Unit – ALU

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

| Mode | Program Counter Bits | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | *11~*8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| Initial reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| USB interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Timer/Event 0 Counter overflow | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Timer/Event 1 Counter overflow | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| SPI interrupt | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Skip | Program Counter+2 | | | | | | | | |
| Loading PCL | @11~@8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, call branch | #11~#8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return (RET, RETI) | S11~S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

**Program Counter**

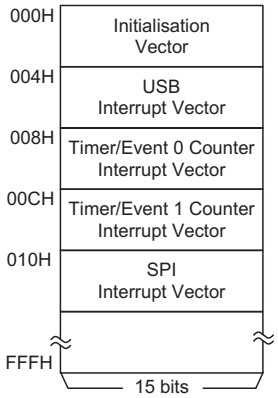
Note: PC11~PC8: Current Program Counter bits
@7~@0: PCL bits
#11~#0: Instruction code address bits
S11~S0: Stack register bits

## Program Memory

The Program Memory is the location where the user code or program is stored. The HT82B42R/HT82B42RE are One-Time Programmable, OTP, memory type devices where users can program their application code into the devices. 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.

### Structure

The Program Memory has a capacity of 4K by 15 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by separate table pointer registers.




**Program Memory Structure**

### Special Vectors

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

- Location 000H
  This area is reserved for program initialization. After chip reset, the program always begins execution at location 000H.
- Location 004H
  This area is reserved for the USB interrupt service program. If the USB interrupt is activated, the interrupt is enabled and the stack is not full, the program jumps to this location and begins execution.
- Location 008H
  This area is reserved for the Timer/Event Counter 0 interrupt service program. If a timer interrupt results from a Timer/Event Counter 0 overflow, and if the interrupt is enabled and the stack is not full, the program jumps to this location and begins execution.
- Location 00CH
  This area is reserved for the Timer/Event Counter 1 interrupt service program. If a timer interrupt results from a Timer/Event Counter 1 overflow, and the interrupt is enabled and the stack is not full, the program jumps to this location and begins execution.
- Location 010H
  This area is reserved for the SPI interrupt service program. If a SPI interrupt results from a byte of data has been transmitted or received by the SPI interface, and the interrupt is enabled and the stack is not full, the program jumps to this location and begins execution.
- Table location
  Any location in the program memory can be used as look-up tables. There are three methods to read the Program Memory data using two table read instructions: "TABRDC" and "TABRDL", transfer the contents of the lower-order byte to the specified data memory, and the higher-order byte to TBLH (08H).
  The three methods are shown as follows:
  - Using the instruction "TABRDC [m]" for the current Program Memory page, where one page= 256words, where the table location is defined by TBLP in the current page. This is where the configuration option has disabled the TBHP register.
  - Using the instruction "TABRDC [m]", where the table location is defined by registers TBLP and TBHP. Here the configuration option has enabled the TBHP register.
  - Using the instruction "TABRDL [m]", where the table location is defined by registers TBLP in the last page which has the address range 0F00H~0FFFH.

Only the destination of the lower-order byte in the table is well-defined, the other bits of the table word are transferred to the lower portion of TBLH, and the remaining 1-bit words are read as "0". The Table Higher-order byte register (TBLH) is read only. The table pointers, TBLP and TBHP, are read/write registers, which indicate the table location. Before accessing the the table, the locations must be placed in the TBLP and TBHP registers (if the configuration option has disabled TBHP then the value in TBHP has no effect). TBLH is read only and cannot be restored. If the main routine and the ISR (Interrupt Service Routine) both employ the table read instruction, the contents of the TBLH in the main routine are likely to be changed by the table read instruction used in the ISR and errors can occur. Using the table read instruction in the main routine and the ISR simultaneously should be avoided. However, if the table read instruction has to be applied in both the main routine and the ISR, the interrupt should be disabled prior to the table read instruction. It will not be enabled until the TBLH has been backed up. All table related instructions require two cycles to complete the

operation. These areas may function as normal program memory depending on the requirements.

Once TBHP is enabled, the instruction "TABRDC [m]" reads the Program Memory data as defined by the TBLP and TBHP values. If the Program Memory code option has disabled TBHP, the instruction "TABRDC [m]" reads the Program Memory data as defined by TBLP only in the current Program Memory page.

## Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, the table pointer must first be setup by placing the lower order address of the look up data to be retrieved in the TBLP register and the higher order address in the TBHP register. These two registers define the full address of the look-up table. Using the TBHP must be selected by configuration option, if not used table data can still be accessed but only the lower byte address in the current page or last page can be defined.

After setting up the table pointers, the table data can be retrieved from the current Program Memory page or last Program Memory page using the "TABRDC [m]" or "TABRDL [m]" instructions, respectively. When these instructions are executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register. Any unused bits in this transferred higher order byte will be read as "0".

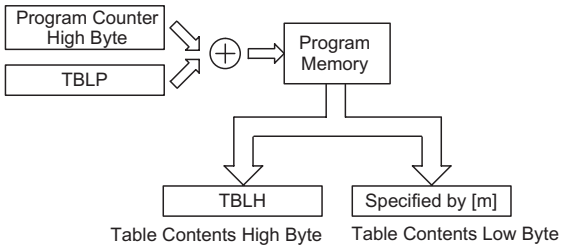



**Table Read – TBLP only**

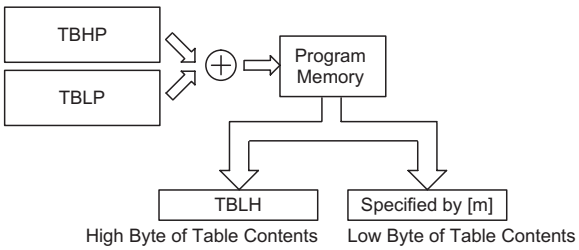



**Table Read – TBLP/TBHP**

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

**Table Location**

Note: PC11~PC8: Current Program Counter bits. TBHP register Bit3~0 when TBHP is enabled.
@7~@0: Table Pointer TBLP bits

Table High Byte Pointer for Current Table Read TBHP (Address 0X1F)

| Register | Bits | Read/Write | Functions |
|---|---|---|---|
| TBHP(0X1F) | 3~0 | R/W | Store current table read bit11~bit8 data |

### Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the microcontroller. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "F00H" which refers to the start address of the last page within the 4K Program Memory of device. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "F06H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address of the present page if the "TABRDC [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRDL [m]" instruction is executed.

```
tempreg1    db      ?       ; temporary register #1
tempreg2    db      ?       ; temporary register #2
:
:
mov a,06h                   ; initialise table pointer-note that this
                            ; address is referenced
mov tblp,a                  ; to the last page or present page
:
:
tabrdl      tempreg1        ; transfers value in table referenced by table
                            ; pointer to tempregl data at prog. memory
                            ; address "F06H" transferred to tempreg1 and TBLH
dec tblp                    ; reduce value of table pointer by one
tabrdl      tempreg2        ; transfers value in table referenced by table
                            ; pointer to tempreg2 data at prog. memory
                            ; address "F05H" transferred to tempreg2 and TBLH
                            ; in this example the data "1AH" is transferred
                            ; to tempreg1 and data "0FH" to register tempreg2
                            ; the value "00H" will be transferred to the high
                            ; byte register TBLH
:
:
org F00h                    ; sets initial address of last page
dc 00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
:
:
```

Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use the table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of TBLH and subsequently cause errors if used again by the main routine. As a rule it is recommended that simultaneous use of the table read instructions should be avoided. However, in situations where simultaneous use cannot be avoided, the interrupts should be disabled prior to the execution of any main routine table-read instructions. Note that all table related instructions require two instruction cycles to complete their operation.

## Data Memory

The Data Memory is a volatile area of 8-bit wide RAM internal memory and is the location where temporary information is stored. The data memory is divided into two banks, Bank0 and Bank1. The Bank0 is subdivided 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. In addition, the Bank1 is dedicated for the USB related registers. All locations within this Data Memory are read and write accessible under program control.

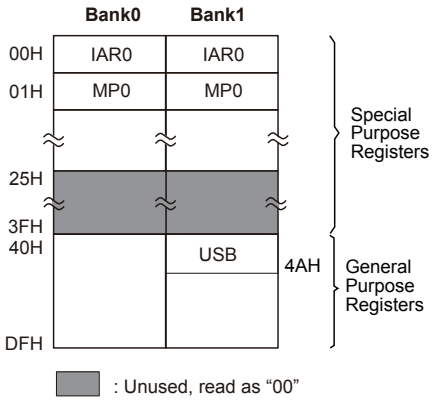
### Structure

The Data Memory is subdivided into two banks, all of which are implemented in 8-bit wide RAM. The Data memory located in Bank0 is subdivided into two sections, the Special Purpose and General Purpose Data Memory.

The start address of the Data Memory for all devices is the address “00H”. Registers which are common to all microcontrollers, such as ACC, PCL, etc., have the same Data Memory address. The USB control registers is mapped into Bank1.

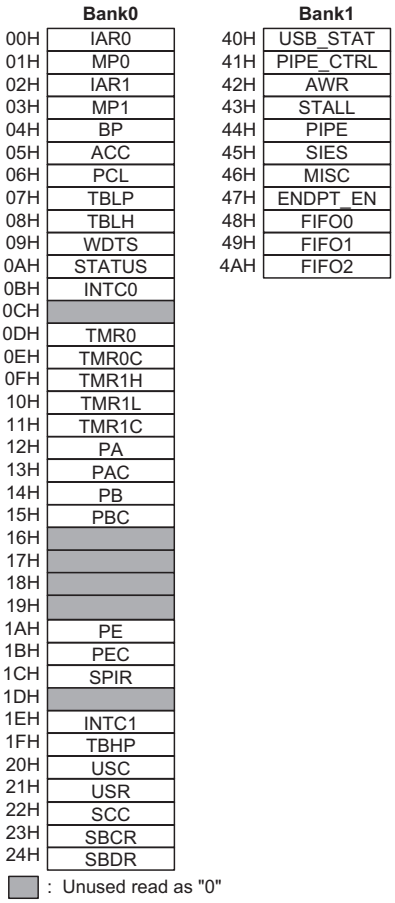
## General Purpose Data Memory

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.




**Data Memory Structure**

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




**Special Purpose Data Memory**

## Special Function Registers

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

### Indirect Addressing Register – 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 can only access data in Bank 0 while MP1 can access both banks.

```
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.

### 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 two special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP and TBHP are the table pointers and indicate the location where the table data is located. Their value must be setup before any table read commands are executed. Their values can be changed, for example using the "INC" or "DEC" instructions, allowing for easy table data pointing and reading. TBLH is the location where the high order byte of the table data is stored after a table read data instruction has been executed.

### Status Register – STATUS

This 8-bit register contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), power down flag (PDF), and watchdog time-out flag (TO). These arithmetic/logical operation and system management flags are used to record the status and operation of the microcontroller.

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

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

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

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

**STATUS Register**

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

"x" unknown

Bit 7, 6 Unimplemented, read as "0"

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

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

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

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

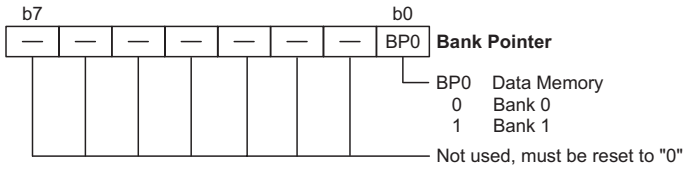
Bit 1 **AC:** Auxiliary flag
0: no auxiliary carry
1: an operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction

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

C is also affected by a rotate through carry instruction.

## Bank Pointer – BP

The Special Purpose Data Memory is divided into two Banks, Bank 0 and Bank 1. The USB control registers are located in Bank 1, while all other registers are located in Bank 0. The Bank Pointer selects which bank data is to be accessed from. If Bank 0 is to be accessed then BP must be set to a value of 00H, while if Bank 1 is to be accessed then BP must be set to a value of 01H.




**Bank Pointer**

## Input/Output Ports

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

Depending upon which package is chosen, the microcontroller provides up to 15 bidirectional input/output lines labeled with port names PA, PB and PE.

These I/O ports are mapped to the Data Memory with addresses as shown in the Special Purpose Data Memory table. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction "MOV A,[m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

### Pull-high Resistors

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

Note that the PE2 is pin shared with reset pin, the I/O structure is NMOS open drain, and there is no pull-high resistor for this pin.

### Port A CMOS/NMOS/PMOS Structure

The pins on Port A can be setup via configuration option to be either CMOS, NMOS or PMOS types.

### Port A VDD/V33O Option Structure

The power supply for the Port A pins can be setup via configuration option to be either VDD or V33O.

### Port Pin Wake-up

If the HALT instruction is executed, the device will enter the Power Down Mode, where the system clock will stop resulting in power being conserved, a feature that is important for battery and other low-power applications. Various methods exist to wake-up the microcontroller, one of which is to change the logic condition on one of the port pins from high to low. After a HALT instruction forces the microcontroller into entering the Power Down Mode, the processor will remain in a low-power state until the logic condition of the selected wake-up pin on the port pin changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Each pin on PA, PB and PE has the capability to wake-up the device on an external falling edge. Note that some pins can only be setup nibble wide whereas other can be bit selected to have a wake-up function.

## I/O Port Control Registers

Each I/O port has its own control register PAC, PBC and PEC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each of the I/O ports is directly mapped to a bit in its associated port control register. Note that several pins can be setup to have NMOS outputs using configuration options.

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

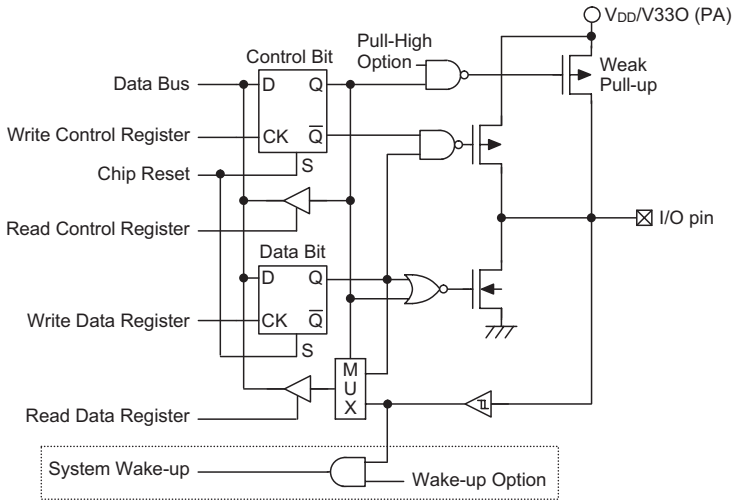
## Pin-shared Functions

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

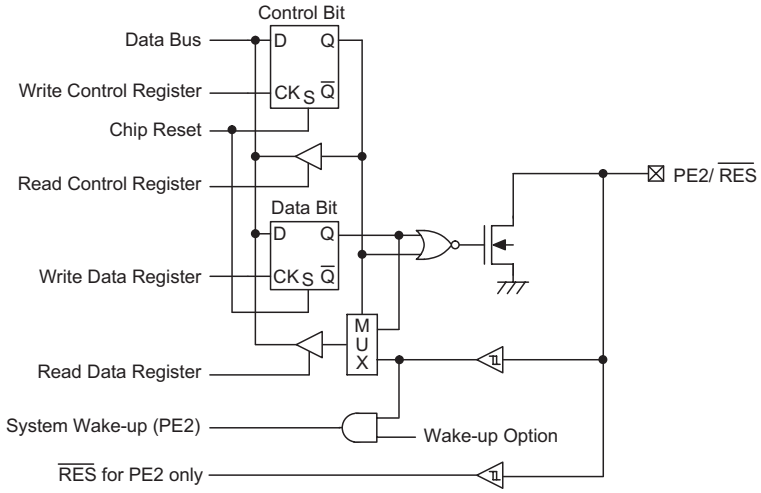
- External Timer0 Clock Input
  The external timer pin TMR0 is pin-shared with the I/O pin PA6. To configure this pin to operate as timer input, the corresponding control bits in the timer control register must be correctly set. For applications that do not require an external timer input, this pin can be used as a normal I/O pin. Note that if used as a normal I/O pin the timer mode control bits in the timer control register must select the timer mode, which has an internal clock source, to prevent the input pin from interfering with the timer operation.
- External Timer1 Clock Input
  The external timer pin TMR1 is pin-shared with the I/O pin PA7. To configure this pin to operate as timer input, the corresponding control bits in the timer control register must be correctly set. For applications that do not require an external timer input, this pin can be used as a normal I/O pin. Note that if used as a normal I/O pin the timer mode control bits in the timer control register must select the timer mode, which has an internal clock source, to prevent the input pin from interfering with the timer operation.

### I/O Pin Structures

The diagram illustrates a generic I/O pin internal structures. As the exact logical construction of the I/O pin will differ and as the pin-shared structures are not illustrated this diagram is supplied as a guide only to assist with the functional understanding of the I/O pins.



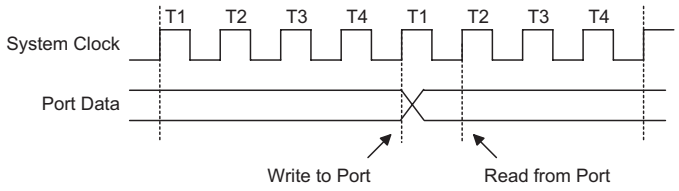

**Input/Output Ports**




**Input/output port (PE2)**

### 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 and PEC port control register, are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated PA, PB and PE 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**

All pins have the additional capability of providing wake-up functions. When the device is in the Power Down Mode, various methods are available to wake the device up. One of these is a high to low transition of any of the Port pins. Single or multiple pins can be setup to have this function.

## Timer/Event Counters

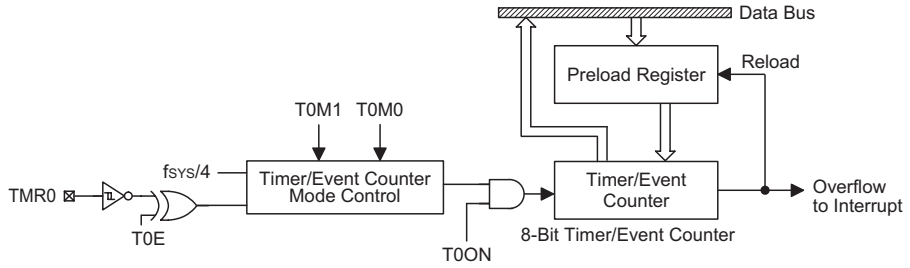
The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. This device contains two count-up timers of 8-bit and 16-bit capacities respectively. As each timer has three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width measurement device.

There are two types of registers related to the Timer/Event Counters. The first is the register that contains the actual value of the Timer/Event Counter and into which an initial value can be preloaded, and is known as TMR0, TMR1H or TMR1L. Reading from this register retrieves the contents of the Timer/Event Counter. The second type of associated register is the Timer Control Register, which defines the timer options and determines how the Timer/Event Counter is to be used, and has the name TMR0C or TMR1C. This device can have the timer clocks configured to come from the internal clock sources. In addition, the timer clock source can also be configured to come from the external timer pins.

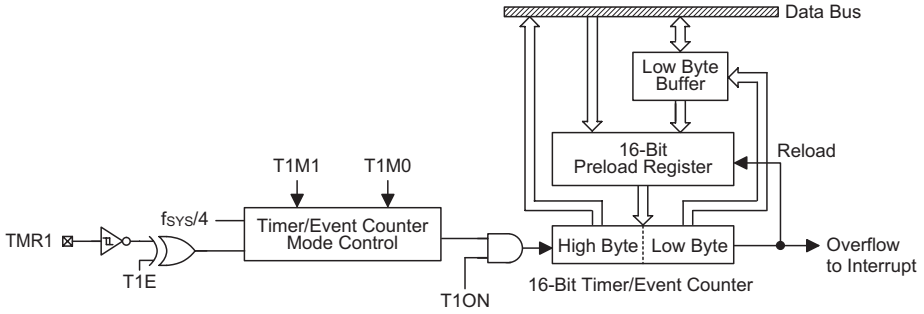
The external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on the external timer pin. The pin has the name TMR0 or TMR1 and is pin-shared with an I/O pin. Depending upon the condition of the T0E or T1E bit in the Timer Control Register, each high to low, or low to high transition on the external timer input pin will increment the Timer/Event Counter by one.

### 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 0 is in the timer mode or in the pulse width measurement mode. The instruction clock source (system clock source divided by 4) is used when the Timer/Event Counter 1 is in the timer mode or in the pulse width measurement mode. The 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.




**8-bit Timer/Event Counter 0 Structure**




**16-bit Timer/Event Counter 1 Structure**

### Timer Register – TMR0, TMR1L/TMR1H

The timer registers are special function registers located in the Special Purpose RAM Data Memory and are the places where the actual timer values are stored. For 8-bit Timer/Event Counter 0, this register is known as TMR0. For 16-bit Timer/Event Counter 1, the timer registers are known as TMR1L and TMR1H. The value in the timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFH for the 8-bit timer or FFFFH for the 16-bit timer at which point the timer overflows and an internal 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 FFH for the 8-bit timer or FFFFH for the 16-bit timer, the preload registers 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 registers, this data will be immediately written into the actual timer registers. However, if the Timer/Event Counter is enabled and counting, any new data written into the preload data registers during this period will remain in the preload registers and will only be written into the timer registers the next time an overflow occurs.

For the 16-bit Timer/Event Counter which has both low byte and high byte timer registers, accessing these registers is carried out in a specific way. It must be note when using instructions to preload data into the low byte timer register, namely TMR1L, the data will only be placed in a low byte buffer and not directly into the low byte timer register. The actual transfer of the data into the low byte timer register is only carried out when a write to its associated high byte timer register, namely TMR1H, is executed. On the other hand, using instructions to preload data into the high byte timer register will result in the data being directly written to the high byte timer register. At the same time the data in the low byte buffer will be transferred into its associated low byte timer register. For this reason, the low byte timer register should be written first when preloading data into the 16-bit timer registers. It must also be noted that to read the contents of the low byte timer register, a read to the high byte timer register must be executed first to latch the contents of the low byte timer register into its associated low byte buffer. After this has been done, the low byte timer register can be read in the normal way. Note that reading the low byte timer register will result in reading the previously latched contents of the low byte buffer and not the actual contents of the low byte timer register.

### Timer Control 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 respective control register. For devices are two timer control registers known as TMR0C, TMR1C. It is the timer control register together with its corresponding timer registers that control the full operation of the Timer/Event Counters. Before the timers can be used, it is essential that the appropriate timer control register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialization.

To choose which of the three modes the timer is to operate in, either in the timer mode, the event counting mode or the pulse width measurement mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair T0M1/T0M0 or T1M1/T1M0 respectively, depending upon which timer is used, must be set to the required logic levels. The timer-on bit, which is bit 4 of the Timer Control Register and known as T0ON or T1ON, depending upon which timer is used, provides the basic on/off control of the respective timer. Setting the bit high allows the counter to run, clearing the bit stops the counter. If the timer is in the event count or pulse width measurement mode, the active transition edge level type is selected by the logic level of bit 3 of the Timer Control Register which is known as T0E or T1E, depending upon which timer is used.

### TMR0C Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | T0M1 | T0M0 | — | T0ON | T0E | — | — | — |
| R/W | R/W | R/W | — | R/W | R/W | — | — | — |
| POR | 0 | 0 | — | 0 | 1 | — | — | — |

Bit 7, 6 **T0M1, T0M0:** Timer 0 operation mode selection
00: no mode available
01: event counter mode
10: timer mode
11: pulse width capture mode

Bit 5 Unimplemented

Bit 4 **T0ON:** Timer/Event Counter counting enable
0: disable
1: enable

Bit 3 **T0E:**
Event Counter active edge selection
0: count on rising edge
1: count on falling edge
Pulse Width Capture active edge selection
0: start counting on falling edge, stop on rising edge
1: start counting on rising edge, stop on falling edge

Bit 2~0 Unimplemented

### TMR1C Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | T1M1 | T1M0 | — | T1ON | T1E | — | — | — |
| R/W | R/W | R/W | — | R/W | R/W | — | — | — |
| POR | 0 | 0 | — | 0 | 1 | — | — | — |

Bit 7, 6 **T1M1, T1M0:** Timer 1 operation mode selection
00: no mode available
01: event counter mode
10: timer mode
11: pulse width capture mode

Bit 5 Unimplemented

Bit 4 **T1ON:** Timer/Event Counter counting enable
0: disable
1: enable

Bit 3 **T1E:**
Event Counter active edge selection
0: count on rising edge
1: count on falling edge
Pulse Width Capture active edge selection
0: start counting on falling edge, stop on rising edge
1: start counting on rising edge, stop on falling edge

Bit 2~0 Unimplemented

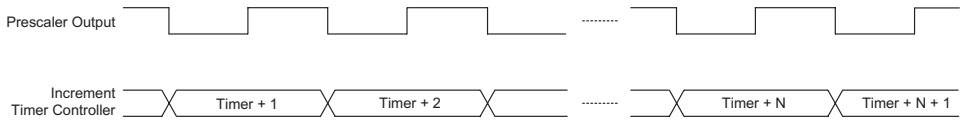
## Configuring the Timer Mode

In this mode, the Timer/Event Counter can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0 or T1M1/T1M0, in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Timer Mode

| Bit7 | Bit6 |
|---|---|
| 1 | 0 |

In this mode the internal clock, $f_{SYS}/4$ is used as the internal clock for the Timer/Event Counters. After the other bits in the Timer Control Register have been setup, the enable bit T0ON or T1ON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. Each time an internal clock cycle occurs, the Timer/Event Counter increments by one. 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.




**Timer Mode Timing Chart**

## Configuring the Event Counter Mode

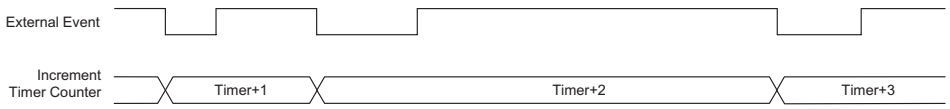
In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the Timer/Event Counter. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0 or T1M1/T1M0, in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Event Counter Mode

| Bit7 | Bit6 |
|---|---|
| 0 | 1 |

In this mode, the external timer pin, TMR0 or TMR1, is used as the Timer/Event Counter clock source, however it is not divided by the internal prescaler. After the other bits in the Timer Control Register have been setup, the enable bit T0ON or T1ON, 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 T0E or T1E, 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.




**Event Counter Mode Timing Chart**

## 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, T0M1/T0M0 or T1M1/T1M0, in the Timer Control Register must be set to the correct valueas shown.

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

| Bit7 | Bit6 |
|---|---|
| 1 | 1 |

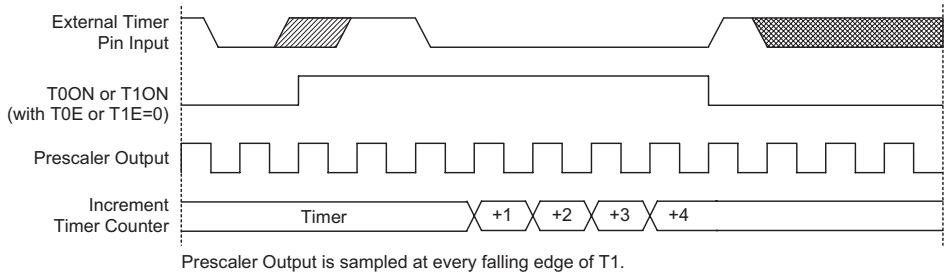
In this mode the internal clock, $f_{SYS}/4$ is used as the internal clock for the Timer/Event Counters. After the other bits in the Timer Control Register have been setup, the enable bit T0ON or T1ON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter, however it will not actually start counting until an active edge is received on the external timer pin.

If the Active Edge Select bit T0E or T1E, 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, TMR0 or TMR1, 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 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.




**Pulse Width Capture Mode Timing Chart**

## I/O Interfacing

The Timer/Event Counter, when configured to run in the event counter or pulse width measurement mode, require the use of the external TMR0 and TMR1 pins for correct operation. As these pins are shared pins they must be configured correctly to ensure they are setup for use as Timer/Event Counter inputs and not as a normal I/O pins. 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 bits for these pins must be set high to ensure that the pin is setup as an input. Any pull-high resistor configuration option on these 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 internal system clock is used as the timer clock source and is therefore synchronised with the overall operation of the microcontroller. In this mode when the appropriate timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the pulse width measurement mode, the internal system clock is also used as the timer clock source but the timer will only run when the correct logic condition appears on the external timer input pin. As this is an external event and not synchronised with the internal timer clock, the microcontroller will only see this external event when the next timer clock pulse arrives. As a result, there may be small differences in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to be in the event counting mode, which again is an external event and not synchronised with the internal system or timer clock.

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

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

### Timer Program Example

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

```
org 04h             ; USB interrupt vector
reti
org 08h             ; Timer/Event Counter interrupt vector
jmp tmr0int         ; jump here when Timer0 overflows
:
org 20h             ; main program
                    ;internal Timer/Event Counter 0 interrupt routine
Tmr0int:
:
                    ; Timer/Event Counter 0 main program placed here
:
reti
:
:
begin:
                    ;setup Timer registers
mov a,09bh          ; setup Timer preload value
mov tmr0,a;
mov a,080h          ; setup Timer 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/Event Counter - note mode bits must be
                    ; previously setup
```

## Interrupts

Interrupts are an important part of any microcontroller system. When an internal function such as a Timer/Event Counter overflow, a USB interrupt occur, or a SPI interrupt takes place, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs while the internal interrupts are controlled by the Timer/Event Counter overflow, USB interrupt or reception and the SPI one byte reception or transmission.

### Interrupt Registers

Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by the interrupt control registers, INTC0 and INTC1. By controlling the appropriate enable bits in the registers each individual interrupt can be enabled or disabled. Also when an interrupt occurs, the corresponding request flag will be set by the microcontroller. The global enable flag if cleared to zero will disable all interrupts.

#### INTC0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | T1F | T0F | USBF | ET1I | ET0I | EUI | EMI |
| R/W | — | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 Unimplemented, read as "0"

Bit 6 **T1F:** Timer/Event Counter 1 interrupt request flag
0: inactive
1: active

Bit 5 **T0F:** Timer/Event Counter 0 interrupt request flag
0: inactive
1: active

Bit 4 **USBF:** USB interrupt request flag
0: inactive
1: active

Bit 3 **ET1I:** Timer/Event Counter 1 interrupt enable
0: disable
1: enable

Bit 2 **ET0I:** Timer/Event Counter 0 interrupt enable
0: disable
1: enable

Bit 1 **EUI:** USB interrupt enable
0: disable
1: enable

Bit 0 **EMI:** Master interrupt global enable
0: disable
1: enable

**INTC1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | SIF | — | — | — | ESII |
| R/W | — | — | — | R/W | — | — | — | R/W |
| POR | — | — | — | 0 | — | — | — | 0 |

Bit 7~5 Unimplemented, read as "0"

Bit 4 **SIF:** SPI interrupt request flag
0: inactive
1: active

Bit 3~1 Unimplemented, read as "0"

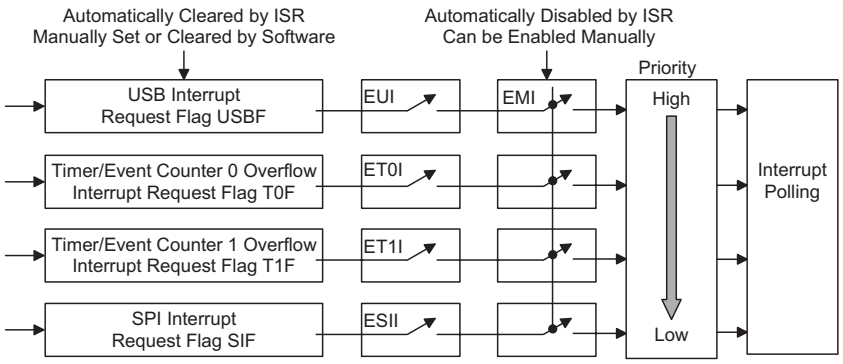
Bit 0 **ESII:** SPI interrupt enable
0: disable
1: enable

## Interrupt Operation

When a USB interrupt occurs, a SPI insterrupt takes place, or one of the Timer/Event Counters overflow, if their appropriate interrupt enable bit is set, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a JMP statement which will jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI statement, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

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

Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately serviced, the request flag will still be recorded. If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full.




**Interrupt Structure**

## Interrupt Priority

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

| Interrupt Source | Priority | Vector |
|---|---|---|
| USB Interrupt | 1 | 0004H |
| Timer/Event Counter 0 Overflow Interrupt | 2 | 0008H |
| Timer/Event Counter 1 Overflow Interrupt | 3 | 000CH |
| SPI Interrupt | 4 | 0010H |

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

## 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/ET1I, must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter interrupt request flag, T0F/T1F, is set, a situation that will occur when the Timer/Event Counter overflows. When the interrupt is enabled, the stack is not full and a Timer/Event Counter overflow occurs, a subroutine call to the timer interrupt vector at location 08H/0CH, will take place. When the interrupt is serviced, the timer interrupt request flag, T0F/T1F, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

## Programming Considerations

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

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

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

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

### USB Interrupt

The USB interrupts are triggered by the following USB events causing the related interrupt request flag, USBF, to be set.

- Access of the corresponding USB FIFO from PC
- 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 USB interrupt is active, a subroutine call to location 04H will occur. The interrupt request flag, USBF, and the EMI bit will be cleared to disable other interrupts.

When the PC Host accesses the FIFO of the device, the corresponding request bit, USR, is set, and a USB interrupt is triggered. So the user can easy determine which FIFO has been accessed. When the interrupt has been served, the corresponding bit should be cleared by firmware. When the device receive a USB Suspend signal from Host PC, the suspend line (bit 0 of USC) is set and a USB interrupt is also triggered.

Also when device receive a Resume signal from Host PC, the resume line (bit 3 of USC) is set and a USB interrupt is triggered.

### Serial Interface Interrupt

The Serial Interface Interrupt, also known as the SPI interrupt. A SPI Interrupt request will take place when the SPI Interrupt request flag, SIF, is set, which occurs when a byte of data has been received or transmitted by the SPI interface. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and the Serial Interface Interrupt enable bit, ESII, must first be set. When the interrupt is enabled, the stack is not full and a byte of data has been transmitted or received by the SPI interface, a subroutine call to the respective Interrupt vector, will take place. When the Serial Interface Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, and the interrupt request flag, SIF, will be also automatically cleared.

## 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 $\overline{\text{RES}}$ line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to proceed with normal operation after the reset line is allowed to return high. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

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

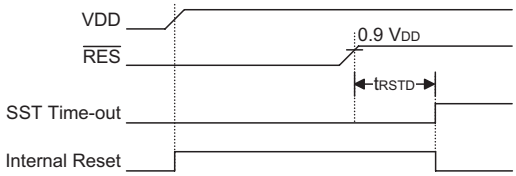
## Reset Functions

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

- Power-on Reset

  The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions. All the I/O port and port control registers will power up in a high condition ensuring that all pins will be first set to inputs.

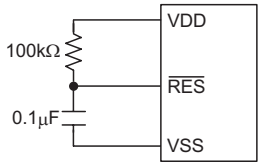
  Although the microcontroller has an internal RC reset function, if the $V_{DD}$ power supply rise time is not fast enough or does not stabilise quickly at power-on, the internal reset function may be incapable of providing a proper reset operation. In such cases it is recommended that an external RC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




**Power-On Reset Timing Chart**

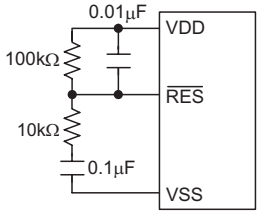
- $\overline{RES}$ Pin Reset

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




**Basic Reset Circuit**

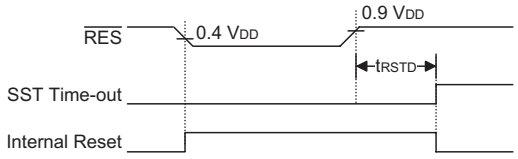
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.




**Enhanced Reset Circuit**

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

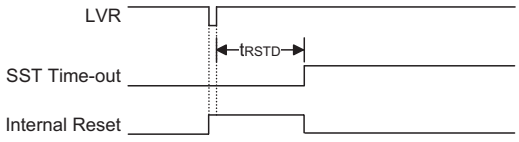
This type of reset occurs when the microcontroller is already running and the $\overline{RES}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point. Note that as the external reset pin is also pin-shared with PE2 if it is to be used as a reset pin, the correct reset configuration option must be selected.




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

- Low Voltage Reset – LVR

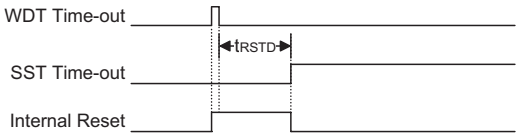
  The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. The LVR function is selected via a configuration option. If the supply voltage of the device drops to within a range of 0.9V~$V_{LVR}$ such as might occur when changing the battery, the LVR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




**Low Voltage Reset Timing Chart**

- Watchdog Time-out Reset during Normal Operation

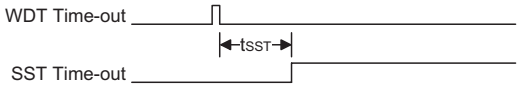
  The Watchdog time-out Reset during normal operation is the same as a hardware $\overline{RES}$ pin reset except that the Watchdog time-out flag TO will be set to "1".




**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 Char**

## 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 |
| 0 | 0 | $\overline{\text{RES}}$ wake-up during Power Down |
| 0 | 0 | $\overline{\text{RES}}$ 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 |
| Prescaler | The Timer Counter Prescaler will be cleared |
| Input/Output Ports | I/O ports will be setup as inputs |
| Stack Pointer | Stack Pointer will point to the top of the stack |

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

| Register | Reset (Power On) | WDT Time-out (Normal Operation) | $\overline{RES}$ Reset (Normal Operation) | $\overline{RES}$ Reset (HALT) | WDT Time-Out (HALT)* | USB-Reset (Normal) | USB-Reset (HALT) |
|---|---|---|---|---|---|---|---|
| TMR0 | xxxx xxxx | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR0C | 00-0 1--- | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- | 00-0 1--- | 00-0 1--- |
| TMR1H | xxxx xxxx | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR1L | xxxx xxxx | 0000 0000 | 0000 0000 | 0000 0000 | 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--- |
| Program Counter | 0000H | 0000H | 0000H | 0000H | 0000H | 0000H | 0000H |
| 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 |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBHP | ---- xxxx | ---- uuuu | ---- uuuu | ---- uuuu | ---- uuuu | ---- uuuu | ---- uuuu |
| TBLH | -xxx xxxx | -uuu uuuu | -uuu uuuu | -uuu uuuu | -uuu uuuu | -uuu uuuu | -uuu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --00 uuuu | --00 uuuu | --11 uuuu | --uu uuuu | --01 uuuu |
| INTC0 | -000 0000 | -000 0000 | -000 0000 | -000 0000 | -uuu uuuu | -000 0000 | -000 0000 |
| INTC1 | ---0 ---0 | ---0 ---0 | ---0 ---0 | ---0 ---0 | ---u ---u | ---0 ---0 | ---0 ---0 |
| WDTS | 1000 0111 | 1000 0111 | 1000 0111 | 1000 0111 | uuuu uuuu | 1000 0111 | 1000 0111 |
| 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 |
| PB | ---- 1111 | ---- 1111 | ---- 1111 | ---- 1111 | ---- uuuu | ---- 1111 | ---- 1111 |
| PBC | ---- 1111 | ---- 1111 | ---- 1111 | ---- 1111 | ---- uuuu | ---- 1111 | ---- 1111 |
| PE | ---- -111 | ---- -111 | ---- -111 | ---- -111 | ---- -uuu | ---- -111 | ---- -111 |
| PEC | ---- -111 | ---- -111 | ---- -111 | ---- -111 | ---- -uuu | ---- -111 | ---- -111 |
| USB_STAT | ---x xxxx | ---u uuuu | ---x xxxx | ---x xxxx | ---u uuuu | ---x xxxx | ---x xxxx |
| PIPE_CTRL | 0000 0110 | uuuu uuuu | 0000 0110 | 0000 0110 | uuuu uuuu | 0000 0110 | 0000 0110 |
| AWR | 0000 0000 | uuuu uuuu | 0000 0000 | 0000 0000 | uuuu uuuu | 0000 0000 | 0000 0000 |
| PIPE | 0000 0000 | uuuu uuuu | 0000 0000 | 0000 0000 | uuuu uuuu | 0000 0000 | 0000 0000 |
| STALL | 0000 0110 | uuuu uuuu | 0000 0110 | 0000 0110 | uuuu uuuu | 0000 0110 | 0000 0110 |
| SIES | 0x0x x000 | uuuu uuuu | 0x0x x000 | 0x0x x000 | uuuu uuuu | 0x0x x000 | 0x0x x000 |
| MISC | 0000 0000 | uuuu uuuu | 0000 0000 | 0000 0000 | uuuu uuuu | 0000 0000 | 0000 0000 |
| Endpt_EN | 0000 0111 | uuuu uuuu | 0000 0111 | 0000 0111 | uuuu uuuu | 0000 0111 | 0000 0111 |
| 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 |
| USC | 11xx 0000 | uuxx uuuu | 11xx 0000 | 11xx 0000 | uuxx uuuu | uu00 0u00 | uu00 0u00 |
| USR | 0000 0000 | uuuu uuuu | 0000 0000 | 0000 0000 | uuuu uuuu | u1uu 0000 | u1uu 0000 |
| SCC | 0000 0000 | uuuu uuuu | 0000 0000 | 0000 0000 | uuuu uuuu | 0uu0 u000 | 0uu0 u000 |
| SPIR | ---- 0000 | ---- 0000 | ---- 0000 | ---- 0000 | ---- uuuu | ---- 0000 | ---- 0000 |
| SBCR | 0110 0000 | 0110 0000 | 0110 0000 | 0110 0000 | uuuu uuuu | 0110 0000 | 0110 0000 |
| SBDR | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu | xxxx xxxx | xxxx xxxx |

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

## Oscillator

The clock source for these devices is provided by an integrated oscillator requiring no external components. This oscillator has two fixed frequencies of either 6MHz or 12MHz, the selection of which is made by the SYSCLK bit in the SCC register.

### Watchdog Timer Oscillator

The WDT oscillator is a fully self-contained free running on-chip RC oscillator with a typical period of 32μs at 5V requiring no external components. When the device enters the Power Down Mode, the system clock will stop running but the WDT oscillator continues to free-run and to keep the watchdog active. However, to preserve power in certain applications the WDT oscillator can be disabled via a configuration option.

## Power Down Mode and Wake-up

### Power Down Mode

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

### Entering the Power Down Mode

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

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

### Standby Current Considerations

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

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

If the configuration options have enabled the Watchdog Timer internal oscillator then this will continue to run when in the Power Down Mode and will thus consume some power. For power sensitive applications it may be therefore preferable to use the system clock source for the Watchdog Timer.

### Wake-up

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

- An external reset
- An external falling edge Wake-up
- 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 can be setup via an individual configuration option to permit a negative transition on the pin to wake-up the system. When a pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction.

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

No matter what the source of the wake-up event is, once a wake-up situation occurs, a time period equal to 1024 system clock periods will be required before normal system operation resumes. However, if the wake-up has originated due to an interrupt, the actual interrupt 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.

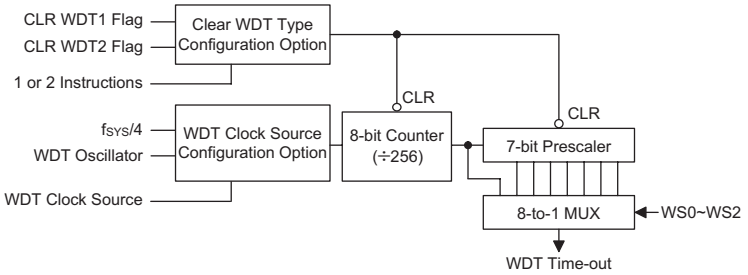
## Watchdog Timer

The WDT clock source is implemented by a dedicated RC oscillator (WDT oscillator) or instruction clock (system clock divided by 4), enabled using a configuration option. This timer is designed to prevent a software malfunction or sequence jumping to an unknown location with unpredictable results. If the Watchdog Timer is disabled, all the executions related to the WDT results in no operation.

Once the internal WDT oscillator (RC oscillator normally with a period of 32μs) is selected, it is first divided by 256 (8-stages) to get the nominal time-out period of approximately 8.19ms. This time-out period may vary with temperature, $V_{DD}$ and process variations. By using the WDT prescaler, longer time-out periods can be realized. Writing data to WS2, WS1, WS0 (bit 2, 1, 0 of the WDTS) can give different time-out periods. If WS2, WS1, WS0 are all equal to "1", the division ratio is up to 1:128, and the maximum time-out period is 1.048s.

If the WDT oscillator is disabled, the WDT clock source may still come from the instruction clock and operate in the same manner except that in the Power down Mode state the WDT may stop counting and lose its protecting purpose. In this situation the WDT logic can be restarted by external logic. The high nibble and bit 3 of the WDTS are reserved for user defined flags, which can be used to indicate some specified status.

If the device operates in a noisy environment, using the on-chip RC oscillator (WDT OSC) is strongly recommended, since the HALT will stop the system clock.




**Watchdog Timer**

### WDTS Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | WS7 | — | — | — | WS3 | WS2 | WS1 | WS0 |
| R/W | R/W | — | — | — | R/W | R/W | R/W | R/W |
| POR | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

Bit 7 **WS7:** USB reset enable control bit
Described elsewhere

Bit 6~4 Unimplemented, read as "0"

Bit 3 **WS3:** D+, and D- have a 300K ± 50% ohm pull-high control bit
Described elsewhere

Bit 2~0 **WS2, WS1, WS0:** WDT Time-out period selection
000: 1:1
001: 1:2
010: 1:4
011: 1:8
100: 1:16
101: 1:32
110: 1:64
111: 1:128

## USB Interface

### Suspend Wake-Up and Remote Wake-Up

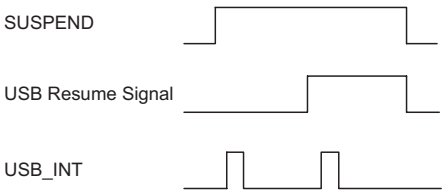
If there is no signal on the USB bus for over 3ms, the device will go into a suspend mode. The Suspend line (bit 0 of the USC register) will be set to "1" and a USB interrupt is triggered to indicate that the devices should jump to the suspend state to meet the 500μA USB suspend current spec.

In order to meet the 500μA suspend current, the firmware should disable the USB clock by clearing the USBCKEN bit which is bit 3 of the SCC register to "0". The suspend current is 400μA.

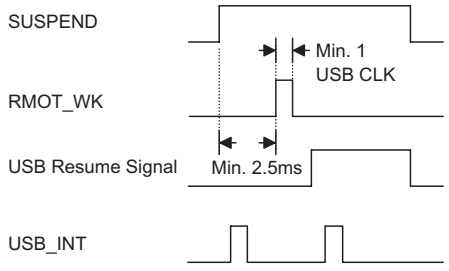
The user can further decrease the suspend current to 250μA by setting the SUSP2 bit which is bit 4 of the SCC register. If in the USB mode set this bit LVR OPT must disable.

When the resume signal is sent out by the host, the devices will wake up the MCU with a USB interrupt and the Resume line (bit 3 of the USC register) is set. In order to make the device function properly, the firmware must set the USBCKEN (bit 3 of the SCC register) to "1" and clear the SUSP2 (bit 4 of the SCC register). Since the Resume signal will be cleared before the Idle signal is sent out by the host, the Suspend line (bit 0 of the USC register) will be set to "0". So when the MCU is detecting the Suspend line (bit 0 of USC register), the Resume line condition should be noted and taken into consideration.

After finishing the resume signal, the suspend line will go inactive and a USB interrupt will be triggered. The following is the timing diagram.




As the device has a remote wake up function it can wake-up the USB Host by sending a wake-up pulse through RMOT_WK (bit 1 of the USC register). Once the USB Host receives a wake-up signal from the devices, it will send a Resume signal to the device. The timing is as follows:



### To Configure as PS2 Device

The devices can also be configured as a USB interface or PS2 interface device, by configuring MODE_CTRL0 (bit 4 of the USR register) and MODE_CTRL1 (bit 5 of the USR register). If MODE_CTRL0=1, and MODE_CTRL1=0, the device will be configured as a PS2 interface, pin UDN is configured as a PS2 Data pin and UDP is configured as a PS2 Clk pin. The user can read or write to the PS2 Data or PS2 Clk pin by accessing the corresponding bit PS2_DAI (bit 4 of the USC register), PS2_CKI (bit 5 of the USC register), PS2_DAO (bit 6 of the USC register) and PS2_CKO (bit 7 of the USC register) respectively.

The user should make sure that in order to read the data properly, the corresponding output bit must be set to “1”. For example, if it is desired to read the PS2 Data by reading PS2_DAI, then PS2_DAO should set to “1”. Otherwise it is always read as “0”.

If MODE_CTRL0=0, and MODE_CTRL1=1, the device is configured as a USB interface. Both the UDN and UDP are driven by the SIE of the devices. The user can only write or read the USB data through the corresponding FIFO. Both the MODE_CTRL0 and MODE_CTRL1 default is “0”.

### USB Control Registers

There are twelve registers used for the USB function. The AWR register contains the current address and a remote wake up function control bit. The initial value of AWR is "00H". The address value extracted from the USB command is not to be loaded into this register until the SETUP stage is completed.

#### AWR Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | AD6 | AD5 | AD4 | AD3 | AD2 | AD1 | AD0 | WKEN |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7~1 **AD6~AD0:** USB device address

Bit 0 **WKEN:** USB remote-wake-up control bit
0: disable
1: enable

The AWR register contains the current address and a remote wake up function control bit. The initial value of AWR is “00H”. The address value extracted from the USB command has not to be loaded into this register until the SETUP stage has finished.

#### WDTS Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | WS7 | — | — | — | WS3 | WS2 | WS1 | WS0 |
| R/W | R/W | — | — | — | R/W | R/W | R/W | R/W |
| POR | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

Bit 7 **WS7:** USB reset enable control bit
0: USB reset signal cannot reset MCU
1: USB reset signal can reset MCU and set URST_FLAG (bit 2 of the USC register)
(default on at MCU reset)

Bit 6~4 unimplemented, read as “0”

Bit 3 **WS3:** D+, and D- have a 300K ± 50% ohm pull-high control bit
0: Non-pull-high (Default)
1: Pull-high

Bit 2~0 **WS2, WS1, WS0:** WDT Time-out period selection
Described elsewhere

### USC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | PS2_CKO | PS2_DAO | PS2_CKI | PS2_DAI | RESUME_O | URST_FLAG | RMOT_WK | SUSPEND |
| R/W | W | W | R | R | R | R/W | W | R |
| POR | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 **PS2_CKO:** Output for driving UDP/CLK pin, when the device works under 3D PS2 mouse function. The default value is "1".

Bit 6 **PS2_DAO:** Output for driving UDN/DATA pin, when the device works under 3D PS2 mouse function. The default value is "1".

Bit 5 **PS2_CKI:** UDP/CLK input detect bit
0: input "0"
1: input "1"

Bit 4 **PS2_DAI:** UDN/DATA input detect bit
0: input "0"
1: input "1"

Bit 3 **RESUME_O:** USB resume indication bit
0: SUSPEND bit goes to "0"
1: leave the suspend mode

When the USB leaves the suspend mode, this bit is set to "1" (set by SIE). When the RESUME 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 USBCKEN and clear SUSP2 (in the SCC register) to enable the SIE detect function. RESUME will be cleared when the SUSPEND goes to "0". When the MCU is detecting the SUSPEND, the condition of RESUME (causes the MCU to wake-up) should be noted and taken into consideration.

Bit 2 **URST_FLAG:** USB reset indication bit
0: no USB reset
1: USB reset occurred

This bit is set/cleared by the USB SIE. This bit is used to detect 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 initialized.

Bit 1 **RMOT_WK:** USB remote wake-up command
0: no remote wake-up
1: remote wake-up

It is set by MCU to leave the USB host leaving the suspend mode. Indicate that the USB host leaves the suspend mode.

Bit 0 **SUSPEND:** USB suspend indication
0: not in the suspend mode
1: enter the suspend mode

When this bit is set to 1 (set by 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.

The USR (USB endpoint interrupt status register) register is used to indicate which endpoint is accessed and to select the serial bus, PS2 or USB. The endpoint request flags, EP0_INT, EP1_INT and EP2_INT, 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 occur, if the USB interrupt is enabled and the stack is not full. When the active endpoint request flag is served, the endpoint request flag has to be cleared to "0".

### USR Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | USB_flag | — | MODE_CTRL1 | MODE_CTRL0 | — | EP2_INT | EP1_INT | EP0_INT |
| R/W | R/W | — | R/W | R/W | — | R/W | R/W | R/W |
| POR | 0 | — | 0 | 0 | — | 0 | 0 | 0 |

Bit 7 **USB_flag:** USB mode indication flag
0: not in USB mode
1: in USB mode
This flag is used to indicate the MCU is in USB mode or not. This bit will be cleared to "0" after power-on reset. The default value is "0".

Bit 6 Unimplemented

Bit 5~4 **MODE_CTRL1, MODE_CTRL0:** USB mode control bits
00: Non-USB mode, turn-off V33O, both UDP and UDN can be read and write (default)
01: Non-USB mode, has 200Ω between VDD and V33O, both UDP and UDN can be read and write
10: USB mode, 1.5kΩ between UDN and V33O, V33O output 3.3V, both UDP and UDN are read only
11: Non-USB mode, V33O output 3.3V, both UDP and UDN can be read and write

Bit 3 Unimplemented

Bit 2 **EP2_INT:** Endpoint 2 accessed detection
0: not accessed
1: accessed

Bit 1 **EP1_INT:** Endpoint 1 accessed detection
0: not accessed
1: accessed

Bit 0 **EP0_INT:** Endpoint 0 accessed detection
0: not accessed
1: accessed
When the EP0_INT, EP1_INT, or EP2_INT bit is set to "1" (set by the SIE), it indicates that endpoint 0, 1, or 2, is accessed and a USB interrupt will occur. When the interrupt has been served, this bit should be cleared by firmware.

There is a system clock control register implement to select the clock used in the MCU. This register consisters of the USB clock sontrol bit, USBCKEN, second suspend mode control bit, SUSP2, and a system colck selection bit, SYSCLK. The PS2 mode indicate bit, PS2_flag, and a system clock adjust control bit, CLK_adj.

### SCC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | CLK_adj | SYSCLK | PS2_flag | SUSP2 | USBCKEN | — | — | — |
| R/W | R/W | R/W | R/W | R/W | R/W | — | — | — |
| POR | 0 | 0 | 0 | 0 | 0 | — | — | — |

Bit 7 **CLK_adj:** USB mode system clock adjustment
0: enable (default)
1: disable
This bit is used to adjust the system clock for the USB mode for temperature changes. In the Power-down Mode this bit should be set high to reduce power consumption.

Bit 6 **SYSCLK:** Specify MCU oscillator frequency indication bit
0: 12MHz crystal oscillator or resonator, clear this bit to “0”
1: 6MHz crystal oscillator or resonator, set this bit to “1”
This bit is used to specify the system oscillator frequency used by the MCU. If an Integrated 6MHz oscillator is used, this bit should be set to “1”. If an Integrated 12MHz oscillator is used, this bit should be cleared to “0” (default).

Bit 5 **PS2_flag:** PS2 mode indication bit
0: not PS2 mode
1: PS2 mode
This flag is used to indicate that the MCU is in the PS2 mode. (Bit=1)
This bit is R/W by FW and will be cleared to “0” after power-on reset. (Default="0")

Bit 4 **SUSP2:** Reduce power consumption in suspend mode control bit
0: in normal mode
1: in halt mode, set this bit to “1” for reducing power consumption

Bit 3 **USBCKEN:** USB clock control bit
0: disable
1: enable

Bit 2~0 Unimplemented

## STALL and PIPE, PIPE_CTRL, Endpt_EN Registers

The PIPE register represents whether the corresponding endpoint is accessed by the host or not. After an ACT_EN signal has been sent out, the MCU can check which endpoint had been accessed. This register is set only after the a time when the host is accessing the corresponding endpoint.

The STALL register shows whether the corresponding endpoint works or not. As soon as the endpoint works improperly, the corresponding bit must be set.

The PIPE_CTRL Register is used for configuring the IN (Bit=1) or OUT (Bit=0) Pipe. The default is define IN pipe. Bit 0 (DATA0) of the PIPE_CTRL Register is used to set the data toggle of any endpoint (except endpoint 0) using data toggles to the value DATA0. Once the user wants any endpoint (except endpoint 0) using data toggles to the value DATA0. the user can output a LOW pulse to this bit. The LOW pulse period must at least 10 instruction cycles.

The Endpt_EN Register is used to enable or disable the corresponding endpoint (except endpoint 0) Enable Endpoint (Bit=1) or disable Endpoint (Bit=0)

### PIPE_CTRL Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | SETIO2 | SETIO1 | DATA0 |
| R/W | — | — | — | — | — | R/W | R/W | R/W |
| POR | — | — | — | — | — | 1 | 1 | 1 |

Bit 7~3 Unimplemented

Bit 2 **SETIO2:** USB PIPE 2 IN or OUT control bit
0: out
1: in (default)

Bit 1 **SETIO1:** USB PIPE 1 IN or OUT control bit
0: out
1: in (default)

Bit 0 **DATA0:** USB Endpoint data control bit
0: low
1: high

### STALL Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | STL2 | STL1 | STL0 |
| R/W | — | — | — | — | — | R/W | R/W | R/W |
| POR | — | — | — | — | — | 1 | 1 | 1 |

Bit 7~3 Unimplemented

Bit 2 **STL2:** USB Endpoint 2 Stall indication bit
0: not stall
1: stall

Bit 1 **STL1:** USB Endpoint 1 Stall indication bit
0: not stall
1: stall

Bit 0 **STL0:** USB Endpoint 0 Stall indication bit
0: not stall
1: stall

### PIPE Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | Pipe2 | Pipe1 | Pipe0 |
| R/W | — | — | — | — | — | R | R | R |
| POR | — | — | — | — | — | 0 | 0 | 0 |

Bit 7~3 Unimplemented

Bit 2 **Pipe2:** USB PIPE 2 in use indication bit
0: not in use
1: in use

Bit 1 **Pipe1:** USB PIPE 1 in use indication bit
0: not in use
1: in use

Bit 0 **Pipe0:** USB PIPE 0 in use indication bit
0: not in use
1: in use

### Endpt_EN Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | EP2EN | EP1EN | EP0EN |
| R/W | — | — | — | — | — | R/W | R/W | R/W |
| POR | — | — | — | — | — | 1 | 1 | 1 |

Bit 7~3 Unimplemented

Bit 2 **EP2EN:** USB Endpoint 2 control bit
0: disable
1: enable

Bit 1 **EP1EN:** USB Endpoint 1 control bit
0: disable
1: enable

Bit 0 **EP0EN:** USB Endpoint 0 control bit
0: disable
1: enable

### USB_STAT Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | SE1 | SE0 | K_state | J_state | EOP |
| R/W | — | — | — | R/W | R/W | R/W | R/W | R/W |
| POR | — | — | — | x | x | x | x | x |

Bit 7~5 Unimplemented

Bit 4 **SE1:** USB SE1 noise indication bit
0: no noise
1: noise
This bit is used to indicate the SIE has detected a SE1 noise in the USB Bus. This bit is set by SIE and cleared by F/W.

Bit 3 **SE0:** USB SE0 noise indication bit
0: no noise
1: noise
This bit is used to indicate the SIE has detected a SE0 noise in the USB Bus. This bit is set by SIE and cleared by F/W.

Bit 2 **K_state:** USB SIE K_state indication bit
0: not K_state
1: K_state
This bit is used to indicate the SIE has detected a K_state USB signal in the USB Bus. This bit is set by SIE and cleared by F/W.

Bit 1 **J_state:** USB SIE J_state indication bit
0: not J_state
1: J_state
This bit is used to indicate the SIE has detected a J_state USB signal in the USB Bus. This bit is set by SIE and cleared by F/W.

Bit 0 **EOP:** USB EOP indication bit
0: not EOP
1: EOP
This bit is used to indicate the SIE has detected an EOP USB signal in the USB Bus. This bit is set by SIE and cleared by F/W.

The USB_STAT Register is used to indicate the present USB signal state.

### SIES Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | NMI | — | — | — | — | — | F0_ERR | ADR_SET |
| R/W | R/W | — | — | — | — | — | R/W | R/W |
| POR | 0 | — | — | — | — | — | 0 | 0 |

Bit 7 **NMI:** NAK token interrupt mask flag
0: always has USB interrupt if the USB accesses FIFO0
1: has only USB interrupt, data is transmitted to the PC host or data is received from the PC Host

This bit is used to control whether the USB interrupt is output to the MCU in a NAK response to the PC Host IN or OUT token, only for Endpoint0.

Bit 6~2 Unimplemented

Bit 1 **F0_ERR:** FIFO accessed error indicator
0: no error
1: error

This bit is used to indicate that some errors have occurred when the FIFO is accessed.

This bit is set by SIE and should be cleared by firmware.

Bit 0 **ADR_SET:** device address updated method control bit
0: update address after an written address to the AWR register
1: update address after PC host read out data

This bit is used to configure the SIE to automatically change the device address with the value of the Address+Remote_Wake-Up Register.

When this bit is set to "1" by F/W, the SIE will update the device address with the value of the Address+Remote_Wake-Up Register after the PC Host has successfully read the data from the device by the IN operation. The SIE will clear the bit after updating the device address.

Otherwise, when this bit is cleared to "0", the SIE will update the device address immediately after an address is written to the Address+Remote_Wake-Up Register. Default 0.

### MISC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | LEN0 | READY | SCMD | SELP1 | SELP0 | CLEAR | TX | REQ |
| R/W | R | R | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 **LEN0:** 0-sized packet indication flag
0: not 0-sized packet
1: 0-sized packet
This bit is used to indicate that a 0-sized packet has been sent from a host to the MCU. This bit should be cleared by firmware.

Bit 6 **Ready:** Desired FIFO ready indication flag
0: not ready
1: ready
This bit is used to indicate that the desired endpoint FIFO is ready for operation.

Bit 5 **SCMD:** Setup command indication flag
0: not setup command
1: setup command
This bit is used to show that the data in the endpoint FIFO is a SETUP command. This bit has to be cleared by firmware. That is to say, even if the MCU is busy, the device will not miss any SETUP commands from the host.

Bit 4~3 **SELP1, SELP0:** endpoint FIFO selection bits
00: endpoint FIFO0
01: endpoint FIFO1
10: endpoint FIFO2
11: reserved

Bit 2 **CLEAR:** Clear FIFO function control bit
0: disable
1: enable
This bit is used to clear the FIFO, even if the FIFO is not ready.

Bit 1 **TX:** data writing to FIFO status indication flag
0: data writing finished
1: data writing to FIFO
This bit defines the direction of data transferring between the MCU and endpoint FIFO. When the TX is set to “1”, this means that the MCU wants to write data to the endpoint FIFO. After the task is completed, this bit must be cleared to “0” before terminating the request to represent the end of transferring. For a read action, this bit has to be cleared to “0” to represent that MCU wants to read data from the endpoint FIFO and has to be set to “1” after completion.

Bit 0 **REQUEST:** Desired FIFO request status indication flag
0: no request
1: request
After setting the other status of the desired one in the MISC, endpoint FIFO can be requested by setting this bit to “1”. After the task is completed, this bit must be cleared to “0”.

The MCU can communicate with the endpoint FIFO by setting the corresponding registers, of which the address is listed in the following table. After reading the current data, the next data will show after 2μs, this is used to check the endpoint FIFO status and response to the MISC register, if read/write action is still going on.

| Registers | R/W | Bank | Address | Bit7~Bit0 |
|---|---|---|---|---|
| FIFO0 | R/W | 1 | 48H | Data7~Data0 |
| FIFO1 | R/W | 1 | 49H | Data7~Data0 |
| FIFO2 | R/W | 1 | 4AH | Data7~Data0 |

There are some timing constrains and usages illustrated here. By setting the MISC register, the MCU can perform reading, writing and clearing actions. There are some examples shown in the following table for endpoin FIFO reading, writing and clearing.

| Actions | MISC Setting Flow and Status |
|---|---|
| Read FIFO0 sequence | 00H→01H→delay 2μs, check 41H→read* from FIFO0 register and check not ready (01H) →03H→02H |
| Write FIFO1 sequence | 0AH→0BH→delay 2μs, check 4BH→write* to FIFO1 register and check not ready (0BH) →09H→08H |
| Check whether FIFO0 can be read or not | 00H→01H→delay 2μs, check 41H (read) or 01H (not ready) →00H |
| Check whether FIFO1 can be written or not | 0AH→0BH→delay 2μs, check 4BH (read) or 0BH (not ready) →0AH |
| Read 0-sized packet sequence from FIFO0 | 00H→01H→delay 2μs, check 81H→read once (01H)→03H→S02H |
| Write 0-sized packet sequence to FIFO1 | 0AH→0BH→delay 2μs, check 4BH→09H→08H |

Note: *: There is a 2μs time between 2 read actions or between 2 write actions.

## Serial Interface – SPI

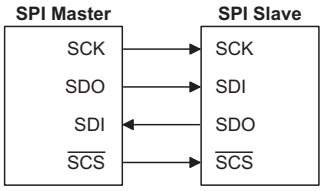
The SPI interface is often used to communicate with external peripheral devices such as sensors, Flash or EEPROM memory devices etc.. Originally developed by Motorola, the four line SPI interface is a synchronous serial data interface that has a relatively simple communication protocol simplifying the programming requirements when communicating with external hardware devices.

The communication is full duplex and operates as a slave/master type, where the device can be either master or slave. Although the SPI interface specification can control multiple slave devices from a single master, however this device is provided with only one $\overline{SCS}$ pin. If the master needs to control multiple slave devices from a single master, the master can use I/O pins to select the slave devices.

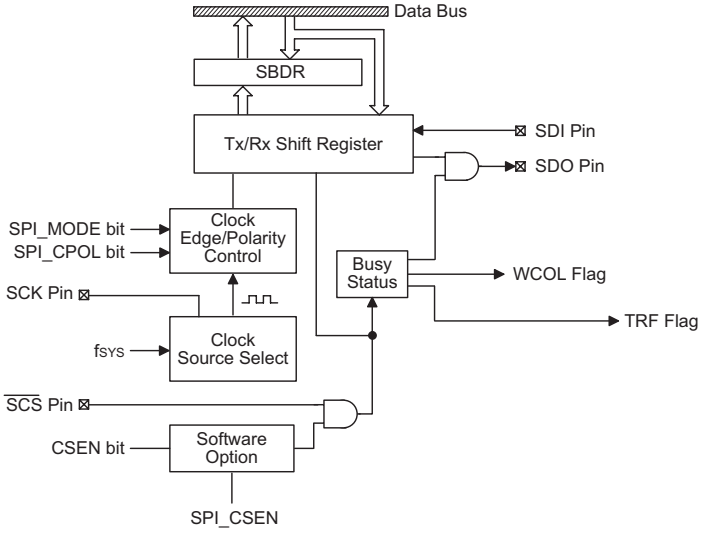
### SPI Interface Operation

The SPI interface is a full duplex synchronous serial data link. It is a four line interface with pin names SDI, SDO, SCK and $\overline{SCS}$. Pins SDI and SDO are the Serial Data Input and Serial Data Output lines, SCK is the Serial Clock line and $\overline{SCS}$ is the Slave Select line. As the SPI interface pins are pin-shared with normal I/O pins, the SPI interface must first be enabled by setting the correct bits in the SBCR and SPIR registers. The SPI can be disabled or enabled using the SPI_EN bit in the SPIR register. Communication between devices connected to the SPI interface is carried out in a slave/master mode with all data transfer initiations being implemented by the master. The Master also controls the clock signal. As the device only contains a single $\overline{SCS}$ pin only one slave device can be utilized.

The $\overline{SCS}$ pin is controlled by the application program, set the CSEN bit to "1" to enable the $\overline{SCS}$ pin function and clear the CSEN bit to "0" to place the $\overline{SCS}$ pin into a floating state.




**SPI Master/Slave Connection**




**SPI Block Diagram**

The SPI function in this device offers the following features:

- Full duplex synchronous data transfer
- Both Master and Slave modes
- LSB first or MSB first data transmission modes
- Transmission complete flag
- Rising or falling active clock edge

The status of the SPI interface pins is determined by a number of factors such as whether the device is in the master or slave mode and upon the condition of certain control bits such as SPI_CSEN and SPI_EN.

## SPI Registers

There are three internal registers which control the overall operation of the SPI interface. These are the SBDR data register and two registers SPIR and SBCR.

| Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SPIR | — | — | — | — | SPI_EN | SPI_CSEN | SPI_MODE | SPI_CPOL |
| SBCR | CKS | M1 | M0 | SBEN | MLS | CSEN | WCOL | TRF |
| SBDR | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

The SBDR register is used to store the data being transmitted and received. Before the device writes data to the SPI bus, the actual data to be transmitted must be placed in the SBDR register. After the data is received from the SPI bus, the device can read it from the SBDR register. Any transmission or reception of data from the SPI bus must be made via the SBDR register.

### SBDR Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | RW |
| POR | X | X | X | X | X | X | X | X |

Bit 7~0 **D7~D0:** SPI data bits

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

There are also two control registers for the SPI interface, SPIR and SBCR. Register SPIR is used to control the enable/disable function and to set the SPI clock active edge type. Register SBCR is used for other control functions such as LSB/MSB selection, write collision flag, data transmission clock frequency selection etc..

### SPIR Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | SPI_EN | SPI_CSEN | SPI_MODE | SPI_CPOL |
| R/W | — | — | — | — | R/W | R/W | R/W | R/W |
| POR | — | — | — | — | 0 | 0 | 0 | 0 |

Bit 7~4 Unimplemented

Bit 3 **SPI_EN:** SPI interface pins control
0: I/O mode (default)
1: SPI mode

Bit 2 **SPI_CSEN:** SPI software bit CSEN function control
0: disable. The CSEN bit has no effect on the $\overline{SCS}$ pin and the $\overline{SCS}$ pin is used as an I/O pin
1: enable. The CSEN bit is used as the enable/disable control for the $\overline{SCS}$ pin

Bit 1 **SPI_MODE:** Determines SPI clock SCK active clock edge type
SPI_CPOL=0
0: SCK is high base level and data capture at SCK rising edge
1: SCK is high base level and data capture at SCK falling edge
SPI_CPOL=1
0: SCK is low base level and data capture at SCK falling edge
1: SCK is low base level and data capture at SCK rising edge
The SPI_MODE and SPI_CPOL bits are used to setup the way that the clock signal outputs and inputs data on the SPI bus. These two bits must be configured before data transfer is executed otherwise an erroneous clock edge may be generated. The SPI_CPOL bit determines the base condition of the clock line. If the bit is high, then the SCK line will be low when the clock is inactive. When the SPI_CPOL bit is low, then the SCK line will be high when the clock is inactive. The SPI_MODE bit determines active clock edge type which depends upon the condition of SPI_CPOL bit.

Bit 0 **SPI_CPOL:** Determines the base condition of the SPI clock line SCK
0: the SCK line will be high when the SPI clock is inactive
1: the SCK line will be low when the SPI clock is inactive
The SPI_CPOL bit determines the base condition of the SPI clock line, if the bit is high, then the SCK line will be low when the clock is inactive. When the SPI_CPOL bit is low, then the SCK line will be high when the clock is inactive.

### SBCR Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | CKS | M1 | M0 | SBEN | MLS | CSEN | WCOL | TRF |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | RW |
| POR | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

Bit 7 **CKS:** SPI clock $f_{SPI}$ source selection
0: $f_{SPI}=f_{SYS}/2$
1: $f_{SPI}=f_{SYS}$

Bit 6~5 **M1~M0:** SPI Operating Mode and baud rate control bits
00: SPI master mode; SPI clock is $f_{SPI}$
01: SPI master mode; SPI clock is $f_{SPI}/4$
10: SPI master mode; SPI clock is $f_{SPI}/16$
11: SPI slave mode
This bit can be read or written by user software program.

Bit 4 **SBEN:** SPI serial bus enable control
0: disable
1: enable
The bit is the overall on/off control for the SPI serial bus. When the SBEN bit is cleared to zero to disable the SPI interface, the SDI, SDO, SCK and $\overline{SCS}$ lines will be in a floating condition and the SPI operating current will be reduced to a minimum value. When the bit is high, the SPI interface is enabled.

Bit 3 **MLS:** SPI Data shift order
0: LSB shift first
1: MSB shift first
This is the data shift select bit and is used to select how the data is transferred, either MSB or LSB first. Setting the bit high will select MSB first and low for LSB first.

Bit 2 **CSEN:** SPI $\overline{SCS}$ pin control
0: disable, other functions
1: enable
The CSEN bit is used as an enable/disable for the $\overline{SCS}$ pin. If this bit is low, then the $\overline{SCS}$ pin will be disabled and placed into a floating condition. If the bit is high the $\overline{SCS}$ pin will be enabled and used as a select pin.
Note that using the CSEN bit can be disabled or enabled by the CSEN control bit named SPI_CSEN in the SPIR register.

Bit 1 **WCOL:** SPI Write Collision flag
0: collision free
1: collision detected
The WCOL flag is used to detect if a data collision has occurred. If this bit is high it means that data has been attempted to be written to the SBDR register during a data transfer operation. This writing operation will be ignored if data is being transferred. The bit can be cleared by the application program.

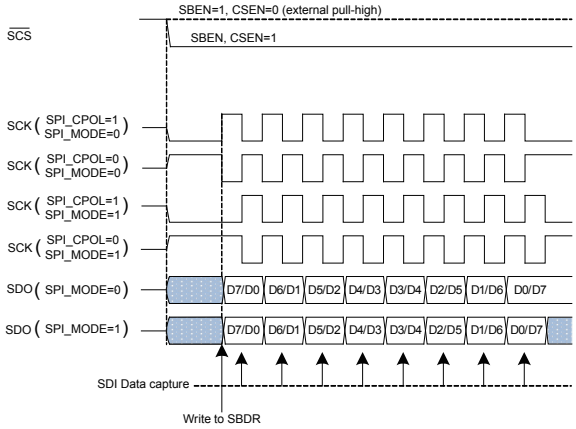
Bit 0 **TRF:** SPI Transmit/Receive Complete flag
0: not complete
1: transmission/reception complete
The TRF bit is the Transmit/Receive Complete flag and is set to 1 automatically when an SPI data transmission is completed, but must be set to 0 by the application program. It can be used to generate an interrupt.

### SPI Communication

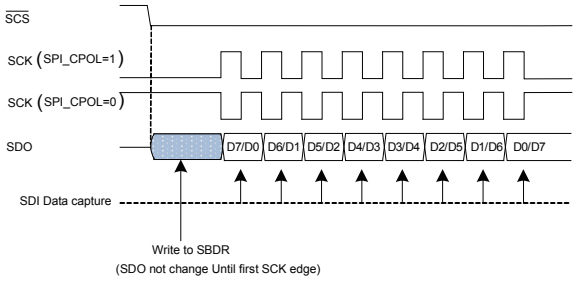
After the SPI interface is enabled by setting the SPI_EN bit high, then in the Master Mode, when data is written to the SBDR register, transmission/reception will begin simultaneously. When the data transfer is complete, the TRF flag will be set automatically, but must be cleared using the application program. In the Slave Mode, when the clock signal from the master has been received, any data in the SBDR register will be transmitted and any data on the SDI pin will be shifted into the SBDR register. The master should output an $\overline{SCS}$ signal to enable the slave device before a clock signal is provided. The slave data to be transferred should be well prepared at the appropriate moment relative to the $\overline{SCS}$ signal depending upon the options of the SPI_MODE bit and SPI_CPOL bit. The accompanying timing diagram shows the relationship between the slave data and $\overline{SCS}$ signal for various configurations of the SPI_MODE and SPI_CPOL bits.

The SPI will continue to function even in the IDLE Mode.

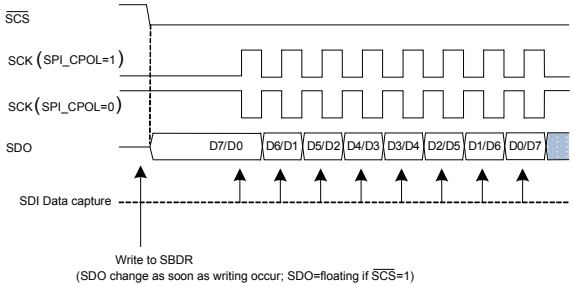
SPI master mode timing



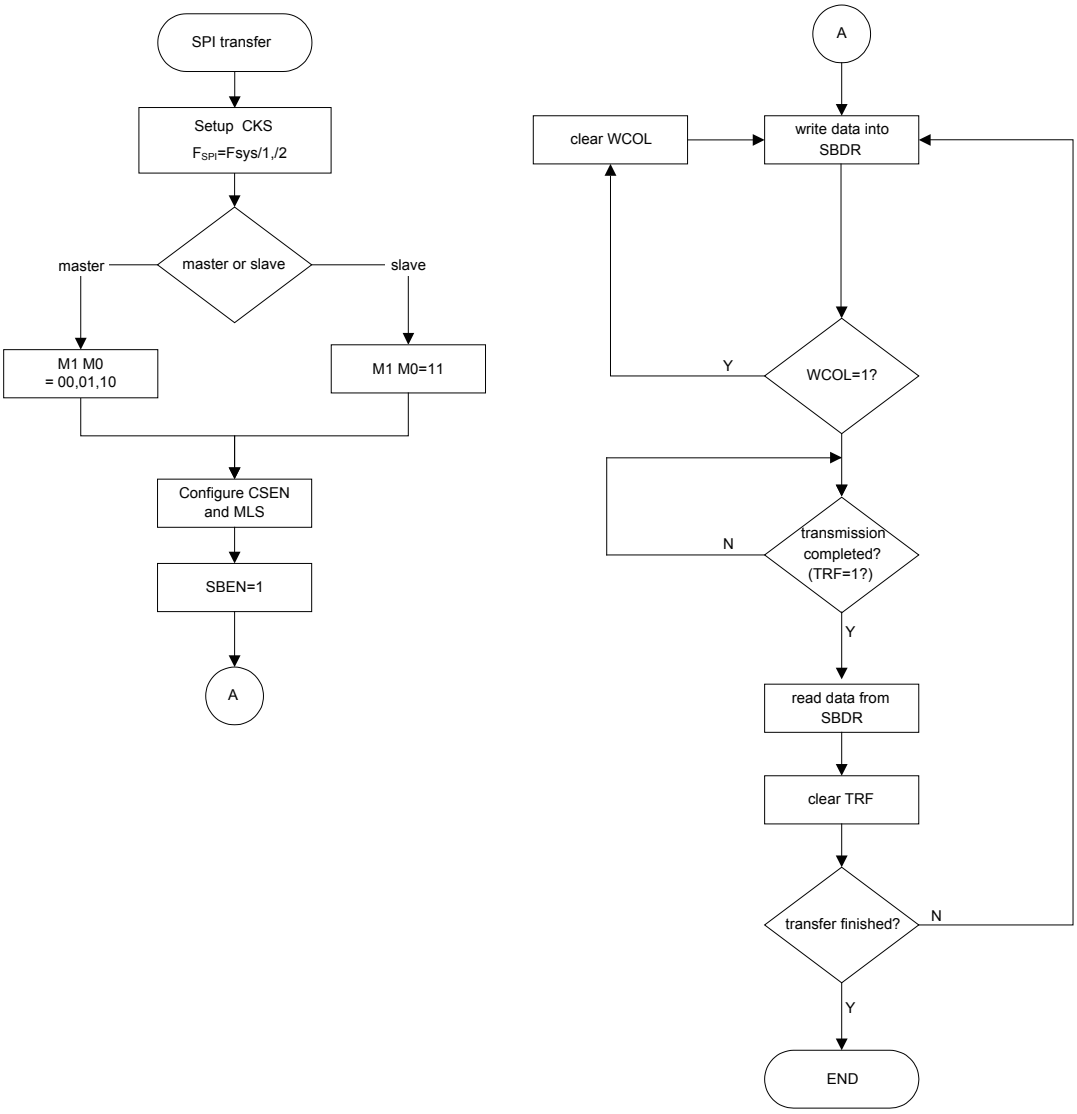

SPI slave mode timing(SPI_MODE=0)




SPI slave mode timing(SPI_MODE=1)




**Note:**
For SPI slave mode, If SBEN=1 and CSEN=0, SPI is always enabled and ignore the SCS level




**SPI Transfer Control Flowchart**

## SPI Bus Enable/Disable

To enable the SPI bus, set SBEN =1, CSEN = 1 and $\overline{SCS}$=0, then wait for data to be written into the SBDR (TXRX buffer) register. For the Master Mode, after data has been written to the SBDR (TXRX buffer) register, then transmission or reception will start automatically. When all the data has been transferred the TRF bit should be set. For the Slave Mode, when clock pulses are received on SCK, data in the TXRX buffer will be shifted out or data on SDI will be shifted in.

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

## SPI Operation

All communication is carried out using the 4-line interface for either Master or Slave Mode.

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 $\overline{SCS}$ 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 $\overline{SCS}$ line will be in a floating condition and can therefore not be used for control of the SPI interface. The SBEN bit in the SBCR register must also be high which will place the SDI line in a floating condition and the SDO line high. If in Master Mode the SCK line will be either high or low depending upon the clock polarity selection bit SPI_CPOL in the SPIR register. If in Slave Mode the SCK line will be in a floating condition. If SBEN is low then the bus will be disabled and $\overline{SCS}$, SDI, SDO and SCK will all be in a floating condition.

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 into the SBDR register. In the Slave Mode, the clock signal will be received from an external master device for both data transmission and 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 $\overline{SCS}$ lines to output the data. After this go to step6.
  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 a SPI serial bus interrupt.
- Step 8
  Read data from the SBDR register.
- Step 9
  Clear TRF.
- Step10
  Go to 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 in the SBCR control register to 11 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 the data to the SBDR register, which will actually place the data into the TXRX buffer. Then wait for the master clock SCK and $\overline{SCS}$ signal. After this go to step6.
  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 a SPI serial bus interrupt.
- Step 8
  Read data from the SBDR register.
- Step 9
  Clear TRF.
- Step10
  Go to step 5.

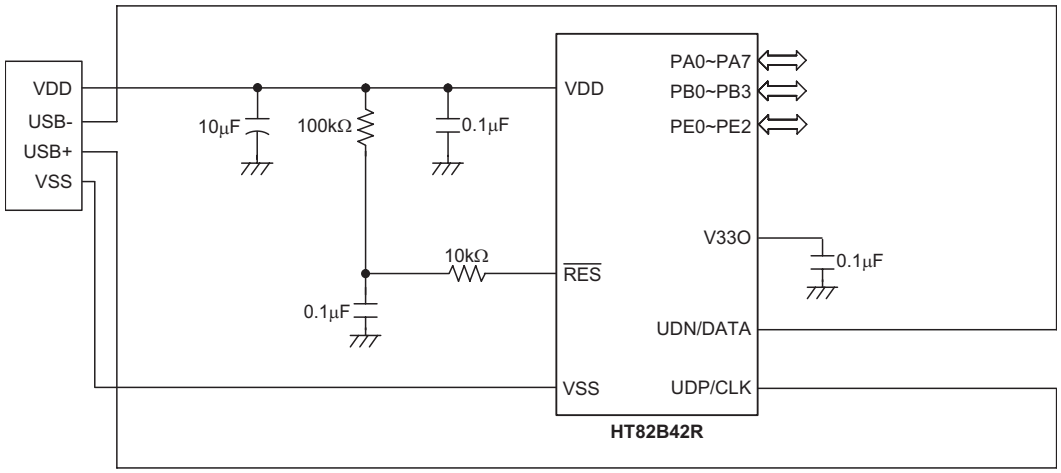
## Error Detection

The WCOL bit in the SBCR register is provided to indicate errors during data transfer. The bit is set by the SPI 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.

## Configuration Options

| No. | Options |
|---|---|
| 1 | PA0~7 Pull-high by bit (default Pull-high) |
| 2 | PB wake-up by nibble (default Pull-high) |
| 3 | PB Pull-high by nibble (default Pull-high) |
| 4 | LVR enable/disable (default enable) |
| 5 | WDT function: enable, disable for normal mode (default enable) |
| 6 | WDT clock source: RC; $f_{SYS}/4$ (default T1) |
| 7 | CLRWDT instruction is by 1 or 2 |
| 8 | PA output mode (CMOS/NMOS/PMOS) by bit (default CMOS) |
| 9 | PA0~7 wake-up by bit (default enable) |
| 10 | TBHP enable/disable (default disable) |
| 11 | PE0, PE1 Pull-high by bit |
| 12 | PE0, PE1, PE2 wake-up by bit |
| 13 | PA0~7 Power source: VDD (default VDD)/V33O regulator output |
| 14 | PE2/$\overline{RES}$ pin option (default $\overline{RES}$ pin) |

## Application Circuit




Note: The resistance and capacitance for the reset circuit should be designed in such a way as to ensure that the VDD is stable and remains within a valid operating voltage range before bringing $\overline{RES}$ high.

# 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 | 1Note | 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 | 1Note | 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 | 1Note | 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 | 1Note | Z, C, AC, OV |
| DAA [m] | Decimal adjust ACC for Addition with result in Data Memory | 1Note | C |
| **Logic Operation** | | | |
| AND A,[m] | Logical AND Data Memory to ACC | 1 | Z |
| OR A,[m] | Logical OR Data Memory to ACC | 1 | Z |
| XOR A,[m] | Logical XOR Data Memory to ACC | 1 | Z |
| ANDM A,[m] | Logical AND ACC to Data Memory | 1Note | Z |
| ORM A,[m] | Logical OR ACC to Data Memory | 1Note | Z |
| XORM A,[m] | Logical XOR ACC to Data Memory | 1Note | Z |
| AND A,x | Logical AND immediate Data to ACC | 1 | Z |
| OR A,x | Logical OR immediate Data to ACC | 1 | Z |
| XOR A,x | Logical XOR immediate Data to ACC | 1 | Z |
| CPL [m] | Complement Data Memory | 1Note | Z |
| CPLA [m] | Complement Data Memory with result in ACC | 1 | Z |
| **Increment & Decrement** | | | |
| INCA [m] | Increment Data Memory with result in ACC | 1 | Z |
| INC [m] | Increment Data Memory | 1Note | Z |
| DECA [m] | Decrement Data Memory with result in ACC | 1 | Z |
| DEC [m] | Decrement Data Memory | 1Note | Z |
| **Rotate** | | | |
| RRA [m] | Rotate Data Memory right with result in ACC | 1 | None |
| RR [m] | Rotate Data Memory right | 1Note | None |
| RRCA [m] | Rotate Data Memory right through Carry with result in ACC | 1 | C |
| RRC [m] | Rotate Data Memory right through Carry | 1Note | C |
| RLA [m] | Rotate Data Memory left with result in ACC | 1 | None |
| RL [m] | Rotate Data Memory left | 1Note | None |
| RLCA [m] | Rotate Data Memory left through Carry with result in ACC | 1 | C |
| RLC [m] | Rotate Data Memory left through Carry | 1Note | C |

| Mnemonic | Description | Cycles | Flag Affected |
|---|---|---|---|
| **Data Move** | | | |
| MOV A,[m] | Move Data Memory to ACC | 1 | None |
| MOV [m],A | Move ACC to Data Memory | 1Note | None |
| MOV A,x | Move immediate data to ACC | 1 | None |
| **Bit Operation** | | | |
| CLR [m].i | Clear bit of Data Memory | 1Note | None |
| SET [m].i | Set bit of Data Memory | 1Note | None |
| **Branch Operation** | | | |
| JMP addr | Jump unconditionally | 2 | None |
| SZ [m] | Skip if Data Memory is zero | 1Note | None |
| SZA [m] | Skip if Data Memory is zero with data movement to ACC | 1Note | None |
| SZ [m].i | Skip if bit i of Data Memory is zero | 1Note | None |
| SNZ [m].i | Skip if bit i of Data Memory is not zero | 1Note | None |
| SIZ [m] | Skip if increment Data Memory is zero | 1Note | None |
| SDZ [m] | Skip if decrement Data Memory is zero | 1Note | None |
| SIZA [m] | Skip if increment Data Memory is zero with result in ACC | 1Note | None |
| SDZA [m] | Skip if decrement Data Memory is zero with result in ACC | 1Note | None |
| CALL addr | Subroutine call | 2 | None |
| RET | Return from subroutine | 2 | None |
| RET A,x | Return from subroutine and load immediate data to ACC | 2 | None |
| RETI | Return from interrupt | 2 | None |
| **Table Read Operation** | | | |
| TABRDC [m](4) | Read ROM code(locate by TBLP and TBHP) to data memory and TBLH | 2Note | None |
| TABRDC [m](5) | Read ROM code(current page) to data memory and TBLH | 2Note | None |
| TABRDL [m] | Read table (last page) to TBLH and Data Memory | 2Note | None |
| **Miscellaneous** | | | |
| NOP | No operation | 1 | None |
| CLR [m] | Clear Data Memory | 1Note | None |
| SET [m] | Set Data Memory | 1Note | None |
| CLR WDT | Clear Watchdog Timer | 1 | TO, PDF |
| 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 | 1Note | None |
| SWAPA [m] | Swap nibbles of Data Memory with result in ACC | 1 | None |
| HALT | Enter power down mode | 1 | TO, PDF |

Note: 1. For skip instructions, if the result of the comparison involves a skip then 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.

4. "TBHP option" is enabled by Configuration Option.

5. "TBHP option" is disabled by Configuration Option.

## 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 bit wise logical AND operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″AND″ x

Affected flag(s) Z

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

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

Operation [m] ← ACC ″AND″ [m]

Affected flag(s) Z

**CALL addr** Subroutine call

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

Operation Stack ← Program Counter + 1
Program Counter ← addr

Affected flag(s) None

**CLR [m]** Clear Data Memory

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

Operation [m] ← 00H

Affected flag(s) None

**CLR [m].i** Clear bit of Data Memory

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

Operation [m].i ← 0

Affected flag(s) None

**CLR WDT** Clear Watchdog Timer

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

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

**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 $ACC \leftarrow \overline{[m]}$

Affected flag(s) Z

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

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

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

Affected flag(s) C

**DEC [m]** Decrement Data Memory

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

Operation [m] ← [m] − 1

Affected flag(s) Z

**DECA [m]** Decrement Data Memory with result in ACC

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

Operation ACC ← [m] − 1

Affected flag(s) Z

**HALT** Enter power down mode

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

Operation TO ← 0
PDF ← 1

Affected flag(s) TO, PDF

**INC [m]** Increment Data Memory

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

Operation [m] ← [m] + 1

Affected flag(s) Z

**INCA [m]** Increment Data Memory with result in ACC

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

Operation ACC ← [m] + 1

Affected flag(s) Z

**JMP addr** Jump unconditionally

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

Operation Program Counter ← addr

Affected flag(s) None

**MOV A,[m]** Move Data Memory to ACC

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

Operation ACC ← [m]

Affected flag(s) None

**MOV A,x** Move immediate data to ACC

Description The immediate data specified is loaded into the Accumulator.

Operation ACC ← x

Affected flag(s) None

**MOV [m],A** Move ACC to Data Memory

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

Operation [m] ← ACC

Affected flag(s) None

**NOP** No operation

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

Operation No operation

Affected flag(s) None

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

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

Operation ACC ← ACC ″OR″ [m]

Affected flag(s) Z

**OR A,x** Logical OR immediate data to ACC

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

Operation ACC ← ACC ″OR″ x

Affected flag(s) Z

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

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

Operation [m] ← ACC ″OR″ [m]

Affected flag(s) Z

**RET** Return from subroutine

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

Operation Program Counter ← Stack

Affected flag(s) None

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

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

Operation Program Counter ← Stack
ACC ← x

Affected flag(s) None

**RETI** Return from interrupt

Description The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the 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] − 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] − C

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

**SDZ [m]** Skip if decrement Data Memory is 0

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

Operation [m] ← [m] − 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] − 1
Skip if ACC=0

Affected flag(s) None

**SET [m]** Set Data Memory

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

Operation [m] ← FFH

Affected flag(s) None

**SET [m].i** Set bit of Data Memory

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

Operation [m].i ← 1

Affected flag(s) None

**SIZ [m]** Skip if increment Data Memory is 0

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

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

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

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

Operation Skip if [m].i ≠ 0

Affected flag(s) None

**SUB A,[m]** Subtract Data Memory from ACC

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

Operation ACC ← ACC − [m]

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

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

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

Operation [m] ← ACC − [m]

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

**SUB A,x** Subtract immediate data from ACC

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

Operation ACC ← ACC − x

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

**SWAP [m]** Swap nibbles of Data Memory

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

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

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

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

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

Affected flag(s) None

**SZ [m]** Skip if Data Memory is 0

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

Operation Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

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

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

Operation Skip if [m].i=0

Affected flag(s) None

**TABRDC [m]** Read table (current page) to TBLH and Data Memory

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

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

Affected flag(s) None

**TABRDC [m]** Move the ROM code (locate by TBLP and TBHP) to TBLH and data memory (ROM code TBHP is enabled)

Description The low byte of ROM code addressed by the table pointers (TBLP and TBHP) is moved to the specified data memory and the high byte transferred to TBLH directly.

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

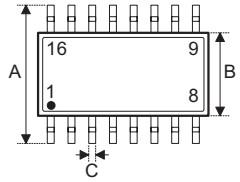
## Package Information

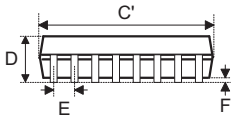
Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the package information.

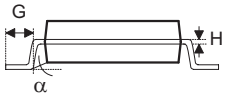
Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

- Further Package Information (include Outline Dimensions, Product Tape and Reel Specifications)
- Packing Meterials Information
- Carton information

## 16-pin NSOP (150mil) Outline Dimensions



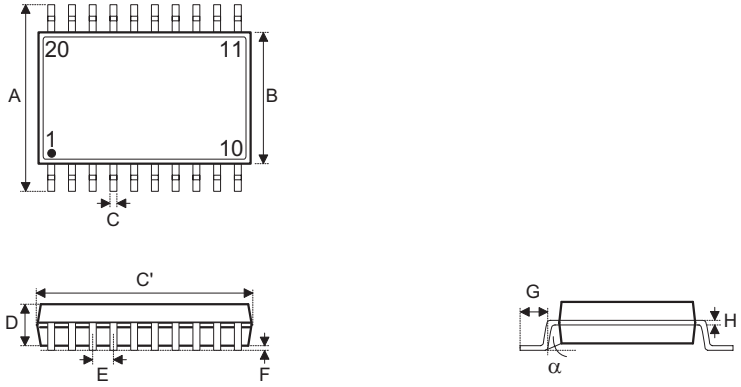




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.236 BSC | — |
| B | — | 0.154 BSC | — |
| C | 0.012 | — | 0.020 |
| C' | — | 0.390 BSC | — |
| D | — | — | 0.069 |
| E | — | 0.050 BSC | — |
| F | 0.004 | — | 0.010 |
| G | 0.016 | — | 0.050 |
| H | 0.004 | — |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

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

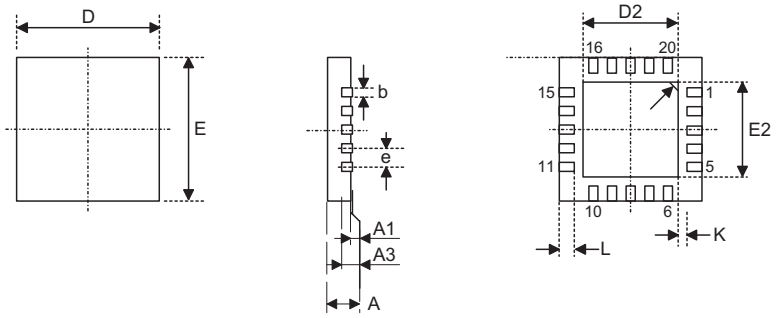
## 20-pin SSOP (150mil) Outline Dimensions




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.236 BSC | — |
| B | — | 0.155 BSC | — |
| C | 0.008 | — | 0.012 |
| C’ | — | 0.341 BSC | — |
| D | — | — | 0.069 |
| E | — | 0.025 BSC | — |
| F | 0.004 | — | 0.0098 |
| G | 0.016 | — | 0.05 |
| H | 0.004 | — | 0.01 |
| α | 0° | — | 8° |

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 6.000 BSC | — |
| B | — | 3.900 BSC | — |
| C | 0.20 | — | 0.30 |
| C’ | — | 8.660 BSC | — |
| D | — | — | 1.75 |
| E | — | 0.635 BSC | — |
| F | 0.10 | — | 0.25 |
| G | 0.41 | — | 1.27 |
| H | 0.10 | — | 0.25 |
| α | 0° | — | 8° |

## SAW Type 20-pin (4mm×4mm) QFN Outline Dimensions




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 0.031 | 0.033 | 0.035 |
| A1 | 0.000 | 0.001 | 0.002 |
| A3 | — | 0.008 BSC | — |
| b | 0.007 | 0.010 | 0.012 |
| D | — | 0.157 BSC | — |
| E | — | 0.157 BSC | — |
| e | — | 0.020 BSC | — |
| D2 | 0.075 | 0.079 | 0.081 |
| E2 | 0.075 | 0.079 | 0.081 |
| L | 0.012 | 0.016 | 0.020 |
| K | 0.008 | — | — |

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 0.800 | 0.850 | 0.900 |
| A1 | 0.000 | 0.020 | 0.050 |
| A3 | — | 0.203 BSC | — |
| b | 0.180 | 0.250 | 0.300 |
| D | — | 4.000 BSC | — |
| E | — | 4.000 BSC | — |
| e | — | 0.50 BSC | — |
| D2 | 1.90 | 2.00 | 2.05 |
| E2 | 1.90 | 2.00 | 2.05 |
| L | 0.30 | 0.40 | 0.50 |
| K | 0.20 | — | — |

Copyright© 2014 by HOLTEK SEMICONDUCTOR INC.
The information appearing in this Data Sheet is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek's products are not authorized for use as critical components in life support devices or systems. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at http://www.holtek.com.