I/O USB OTP MCU

HT82B45R

Revision: V1.00 Date: March 24, 2025

www.holtek.com

Features

CPU Features

- Operating voltage
  - V_{DD} (MCU):
    f_{SYS}=12MHz: 2.5V~5.5V
  - V_{DD} (USB mode):
    f_{SYS}=6MHz: 2.2V~5.5V
- Up to 0.33μs instruction cycle with 12MHz system clock at V_{DD}=5V
- Power down and wake-up functions to reduce power consumption
- Oscillator types
  - Internal High Speed 12MHz RC – HIRC
  - Internal Low Speed 32kHz RC – LIRC
- Fully integrated internal oscillators require no external components
- Multi-mode operation: FAST, SLOW, IDLE and SLEEP
- All instructions executed in 1~3 instruction cycles
- Table read instructions
- 115 powerful instructions
- 8-level subroutine nesting
- Bit manipulation instruction

Peripheral Features

- OTP Program Memory: 4K×16
- Data Memory: 256×8
- Watchdog Timer function
- 35 bidirectional I/O lines
- Single 8-bit programmable timer/event counter with overflow interrupt
- Single 16-bit programmable timer/event counter with overflow interrupt and prescaler
- USB interface
  - USB 2.0 low speed function
  - Supports PS2 and USB modes
  - Supports 3 endpoints including endpoint 0
  - Supports a 3.3V LDO and an internal UDN 1.5kΩ pull-up resistor
  - Internal 12MHz RC oscillator with ±1.5% accuracy for all USB modes
- Low voltage reset function
- Package types: Dice, 48-pin LQFP

General Description

The device is an OTP type 8-bit high performance RISC architecture microcontroller designed for multiple I/O control product applications.

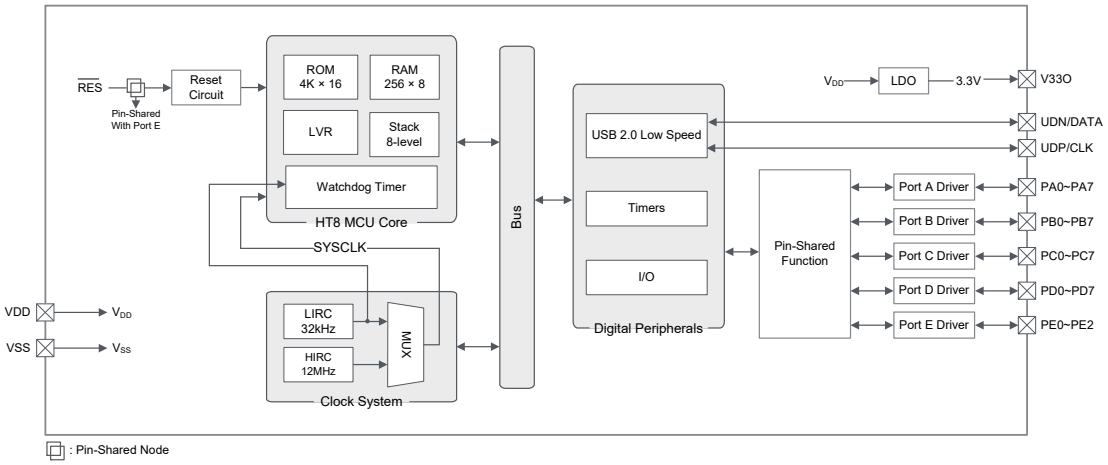
For memory features, the device is supplied with One-Time Programmable, OTP memory. Other memory includes an area of RAM Data Memory.

The device includes two extremely flexible Timer/Event Counters providing functions for timing, event counting and pulse width measurement functions. Protective features such as an internal Watchdog Timer and Low Voltage Reset coupled with excellent noise immunity and ESD protection ensure that reliable operation is maintained in hostile electrical environments.

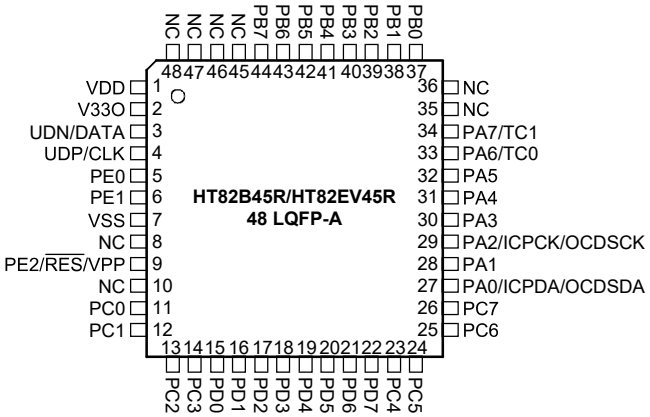
A full choice of internal high and low speed oscillators are provided and the two fully integrated system oscillators require no external components for their implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources gives users the ability to optimize microcontroller operation and minimize power consumption.

The inclusion of flexible I/O programming features, integrated USB interface along with many other features ensure that the device will find excellent use in computer peripheral product applications as well as many other applications such as industrial control, consumer products, subsystem controllers, etc.

Block Diagram



Pin Assignment



Note: 1. If the pin-shared pin functions have multiple outputs, the desired pin-shared function is determined by the corresponding software control bits.

2. The OCDSDA and OCDSCK pins are supplied for the OCDS dedicated pins and as such only available for the HT82EV45R device (Flash type) which is the OCDS EV chip for the HT82B45R device (OTP type).

3. The VPP pin is the High Voltage input for OTP programming and only available for the HT82B45R device.

Pin Description

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.

Pin Name	Function	OPT	I/T	O/T	Description
PA0/ICPDA/ OCDSDA	PA0	PAWUG PAWU PAPU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	ICPDA	—	ST	CMOS	ICP data/address pin
	OCDSDA	—	ST	CMOS	OCDS data/address pin, for EV chip only
PA1	PA1	PAWUG PAWU PAPU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
PA2/ICPCK/ OCDSCK	PA2	PAWUG PAWU PAPU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	ICPCK	—	ST	—	ICP clock pin
	OCDSCK	—	ST	—	OCDS clock pin, for EV chip only
PA3~PA5	PA3~PA5	PAWUG PAWU PAPU PAS0 PAS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
PA6/TC0	PA6	PAWUG PAWU PAPU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	TC0	PAS1	ST	—	Timer/Event Counter 0 clock input

Pin Name	Function	OPT	I/T	O/T	Description
PA7/TC1	PA7	PAWUG PAWU PAPU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	TC1	PAS1	ST	—	Timer/Event Counter 1 clock input
PB0~PB7	PB0~PB7	PBWUG PBWU PBPU PBS0 PBS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
PC0~PC7	PC0~PC7	PCWUG PCWU PCPU PCS0 PCS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
PD0~PD7	PD0~PD7	PDWUG PDWU PDPU PDS0 PDS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
PE0~PE1	PE0~PE1	PEWUG PEWU PEPU PES0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
PE2/$\overline{\text{RES}}$/VPP	PE2	PEWUG PEWU PEPU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up. This I/O is pin-shared with VPP and could result in increased current consumption if it is set to output high.
	$\overline{\text{RES}}$	RSTC	ST	—	External reset input pin
	VPP	—	PWR	—	High Voltage input for OTP programming, not available for EV chip
V33O	V33O	—	—	PWR	USB 3.3V regulator output
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 clock line
VDD	VDD	—	PWR	—	Digital positive power supply
VSS	VSS	—	PWR	—	Digital negative power supply
NC	NC	—	—	—	Not connected

Legend: I/T: Input type; O/T: Output type;
PWR: Power; OPT: Optional by register option;
ST: Schmitt Trigger input; CMOS: CMOS output; NMOS: NMOS output.

Absolute Maximum Ratings

Supply Voltage ..... V_{SS}-0.3V to 6.0V

Input Voltage ..... V_{SS}-0.3V to V_{DD}+0.3V

Storage Temperature..... -60°C to 150°C

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

I_{OH} Total..... -200mA

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 the 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

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency, pin load conditions, temperature and program instruction type, etc., can all exert an influence on the measured values.

Operating Voltage Characteristics

Ta=-40°C~85°C

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{DD}	Operating Voltage – HIRC	$f_{SYS}=f_{HIRC}$=12MHz	2.5	—	5.5	V
		$f_{SYS}=f_H/2$=6MHz	2.2	—	5.5	
	Operating Voltage – LIRC	$f_{SYS}=f_{LIRC}$=32kHz	2.2	—	5.5	

Operating Current Characteristics

Ta=-40°C~85°C

Symbol	Operating Mode	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{DD}	FAST Mode – HIRC	3V	No load, all peripherals off, $f_{SYS}=f_{HIRC}$=12MHz	—	1.6	3.0	mA
		5V		—	2.8	6.0	
		3V	No load, all peripherals off, $f_{SYS}=f_H/2$=6MHz, SYSCLK=0, CKS[1:0]=01B	—	1.7	2.5	mA
		5V		—	2.3	3.2	
		3V	No load, all peripherals off, $f_{SYS}=f_H$=6MHz, SYSCLK=1, CKS[1:0]=00B	—	1.7	2.5	mA
		5V		—	2.3	3.2	
	SLOW Mode – LIRC	3V	No load, all peripherals off, $f_{SYS}=f_{LIRC}$=32kHz	—	50	90	μA
		5V		—	100	150	

Note: When using the characteristic table data, the following notes should be taken into consideration:

1. Any digital inputs are setup in a non-floating condition and the I/O pin-shared with VPP is not setup in an output high condition.
2. All measurements are taken under conditions of no load and with all peripherals in an off state.
3. There are no DC current paths.
4. All Operating Current values are measured using a continuous NOP instruction program loop.

Standby Current Characteristics

Ta = -40°C~85°C

Symbol	Standby Mode	Test Conditions		Min.	Typ.	Max.	Max. @ 85°C	Unit
		V_{DD}	Conditions					
I_{STB}	SLEEP Mode	3V	WDT off	—	0.45	0.90	2.40	μA
		5V		—	0.5	2.0	3.3	μA
		3V	No load, all peripherals off, WDT on, f_{SUB} =f_{LIRC}	—	1.3	3.0	3.0	μA
		5V		—	2.5	5.0	5.0	μA
	IDLE0 Mode – LIRC	3V	No load, all peripherals off, f_{SUB} on, f_{SYS} = f_H	—	1.5	4.0	4.0	μA
		5V		—	2.8	6.0	6.0	μA
		3V	No load, all peripherals off, f_{SUB} on, f_{SYS} = f_{SUB}	—	3	5	5	μA
		5V		—	5	10	10	μA
	IDLE1 Mode – HIRC	3V	No load, all peripherals off, f_{SUB} on, f_{SYS} = f_{HIRC} = 12MHz	—	1.6	2.4	2.5	mA
		5V		—	2.0	3.0	3.1	mA

Note: When using the characteristic table data, the following notes should be taken into consideration:

1. Any digital inputs are setup in a non-floating condition and the I/O pin-shared with VPP is not setup in an output high condition.
2. All measurements are taken under conditions of no load and with all peripherals in an off state.
3. There are no DC current paths.
4. All Standby Current values are taken after a HALT instruction execution thus stopping all instruction execution.

A.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency and temperature etc., can all exert an influence on the measured values.

Internal High Speed Oscillator – HIRC – Frequency Accuracy

During the program writing operation the writer will trim the HIRC oscillator at a user selected HIRC frequency and user selected voltage of 5V.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{HIRC}	12MHz Writer Trimmed HIRC Frequency	4.4V~5.25V	USB mode and CLKADJ=1, UDP/UDN plug in the host Ta=-40°C~85°C	-1.5%	12	+1.5%	MHz
		5V	Ta=25°C	-3%	12	+3%	MHz
		2.5V~5.5V	Ta=-40°C~85°C	-10%	12	+10%	MHz

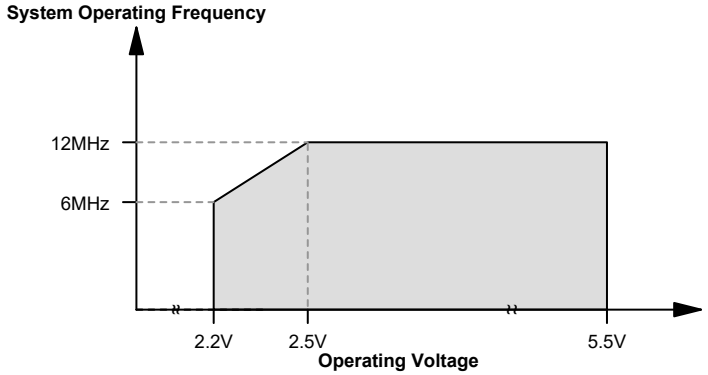
Note: 1. The 5V value for V_{DD} is provided as this is the fixed voltage at which the HIRC frequency is trimmed by the writer.

2. The row below the 5V trim voltage row is provided to show the values for the full V_{DD} range operating voltage.

Internal Low Speed Oscillator Characteristics – LIRC

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Temp.				
f_{LIRC}	LIRC Frequency	2.2V~5.5V	25°C	-20%	32	+20%	kHz
			-40°C~85°C	-50%	32	+60%	
t_{START}	LIRC Start Up Time	—	—	—	—	500	μs

Operating Frequency Characteristic Curves



System Start Up Time Characteristics

Ta=-40°C~85°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
t_{SST}	System Start-up Time (Wake-up from Condition where f_{SYS} is off or $\overline{RES}$ Pin Reset)	—	$f_{SYS}=f_H\sim f_H/4$, $f_H=f_{HIRC}$	—	16	—	t_{SYS}
		—	$f_{SYS}=f_{SUB}=f_{LIRC}$	—	2	—	t_{SYS}
	System Start-up Time (Wake-up from Condition where f_{SYS} is on)	—	$f_{SYS}=f_H\sim f_H/4$, $f_H=f_{HIRC}$	—	2	—	t_{SYS}
		—	$f_{SYS}=f_{SUB}=f_{LIRC}$	—	2	—	t_{SYS}
	System Speed Switch Time (FAST to SLOW Mode or SLOW to FAST Mode)	—	f_{HIRC} switches from off → on (HIRCF=1)	—	16	—	t_{HIRC}
t_{RSTD}	System Reset Delay Time (Reset Source from Power-on Reset or LVR Hardware Reset)	—	RR_{POR}=5V/ms	10	16	24	ms
	System Reset Delay Time (LVRC/WDTC/RSTC Software Reset)	—	—				
	System Reset Delay Time (Reset Source from WDT Overflow Reset or $\overline{RES}$ Pin Reset)	—	—	10	16	24	ms
t_{SRESET}	Minimum Software Reset Width to Reset	—	—	45	90	120	μs

Note: 1. For the System Start-up time values, whether f_{SYS} is on or off depends upon the mode type and the chosen f_{SYS} system oscillator. Details are provided in the System Operating Modes section.

2. The time units, shown by the symbols t_{HIRC}, t_{SYS} etc. are the inverse of the corresponding frequency values as provided in the frequency tables. For example, $t_{HIRC}=1/f_{HIRC}$, $t_{SYS}=1/f_{SYS}$ etc.

3. If the LIRC is used as the system clock and if it is off when in the SLEEP Mode, then an additional LIRC start up time, t_{START}, as provided in the LIRC frequency table, must be added to the t_{SST} time in the table above.

4. The System Speed Switch Time is effectively the time taken for the newly activated oscillator to start up.

Input/Output Characteristics

Ta=-40°C~85°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{IL}	Input Low Voltage for I/O Ports	5V	—	0	—	1.5	V
		—		0	—	$0.2V_{DD}$	
	Input Low Voltage for $\overline{RES}$ Pin	—	$V_{DD}\geq 2.7$	0	—	$0.4V_{DD}$	V
		—	$2.2\leq V_{DD}<2.7$	0	—	$0.3V_{DD}$	
V_{IH}	Input High Voltage for I/O Ports	5V	—	3.5	—	5	V
		—		$0.8V_{DD}$	—	V_{DD}	
	Input High Voltage for $\overline{RES}$ Pin	—	—	$0.9V_{DD}$	—	V_{DD}	V
I_{OL}	Sink Current for I/O Pins (PB7~PB4)	3V	$V_{OL}=0.1V_{DD}$	2	4	—	mA
		5V		4	8	—	
	Sink Current for I/O Pins (except PB7~PB4)	3V	$V_{OL}=0.1V_{DD}$	1	2	—	
		5V		2	4	—	
I_{OH}	Source Current for I/O Pins (PB7~PB4)	3V	$V_{OH}=0.9V_{DD}$	-1	-2	—	mA
		5V		-2	-4	—	
	Source Current for I/O Pins (except PB7~PB4)	3V	$V_{OH}=0.9V_{DD}$	-0.5	-1	—	mA
		5V		-1	-2	—	
R_{PH}	Pull-high Resistance for I/O Ports (Note)	3V	—	20	60	100	kΩ
		5V	—	10	30	50	
SR_{RISE}	Output Rising Edge Slew Rate for I/O Ports	3V	SLEWC0=1, $0.1V_{DD}$ to $0.9V_{DD}$, C_{LOAD}=20pF	30	80	160	V/μs
		5V		60	120	240	
SR_{FALL}	Output Falling Edge Slew Rate for I/O Ports	3V	SLEWC0=1, $0.9V_{DD}$ to $0.1V_{DD}$, C_{LOAD}=20pF	30	80	160	V/μs
		5V		60	120	240	
I_{LEAK}	Input Leakage Current for I/O Ports	5V	$V_{IN}=V_{DD}$ or $V_{IN}=V_{SS}$	—	—	±1	μA
t_{RES}	External Reset Minimum Low Pulse Width	—	—	10	—	—	μs
t_{TC}	TCn Input Pin Minimum Pulse Width	—	—	25	—	—	ns

Note: The R_{PH} internal pull-high resistance value is calculated by connecting to ground and enabling the input pin with a pull-high resistor and then measuring the pin current at the specified supply voltage level. Dividing the voltage by this measured current provides the R_{PH} value.

Memory Electrical Characteristics

Ta=-40°C~85°C, unless otherwise specified

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
OTP Program Memory							
t_{RETD}	ROM Data Retention Time	—	Ta=25°C	—	40	—	Year
Flash Program Memory – for HT82EV45R only							
t_{FWR}	Write Time	—	—	1.364	1.500	1.667	ms
t_{ACTV}	ROM Activation Time (Note)	—	Wake-up from Power Down Mode	1	—	2	t_{LIRC}
RAM Data Memory							
V_{DR}	RAM Data Retention Voltage	—	—	1	—	—	V

Note: The ROM activation time t_{ACTV} should be added when calculating the total system start-up time of a wake-up from the power down mode.

LVR Electrical Characteristics

Ta=-40°C~85°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{LVR}	Low Voltage Reset Voltage	—	LVR enable, voltage select 2.1V	-5%	2.1	+5%	V
			LVR enable, voltage select 2.55V		2.55		
			LVR enable, voltage select 3.15V		3.15		
			LVR enable, voltage select 3.8V		3.8		
I_{LVR}	Operating Current	3V	LVR enable, V_{LVR}=2.1V	—	—	20	μA
		5V	LVR enable, V_{LVR}=2.1V	—	25	35	
t_{LVR}	Minimum Low Voltage Width to Reset	—	—	120	240	480	μs

USB Electrical Characteristics

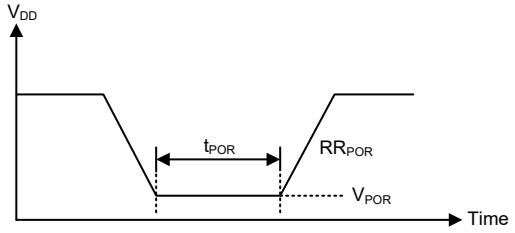
Ta=-40°C~85°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	USB Operating Voltage	—	—	4.2	—	5.5	V
I_{DD}	USB Operating Current	5V	No load, USB on, other peripherals off	—	9.5	15.0	mA
I_{SUS}	USB Suspend Current (IDLE0 Mode)	5V	f_H off, $f_{SUB}=f_{LIRC}$, no load, CPU off, USB transceiver and 3.3V Regulator on, other peripherals off, MODE_CTRL[1:0]=10B	—	360	450	μA
V_{V33O}	3.3V Regulator Output Voltage	5V	I_{V33O}=70mA	3.0	3.3	3.6	V
R_{UDN}	Pull-high Resistance of UDN to V33O	5V	—	-5%	1.5	+5%	kΩ
R_{UDPN}	Pull-high Resistance of UDP and UDN to VDD	5V	PU=1, MODE_CTRL[1:0]=00B	300	650	1000	kΩ
R_{OD1}	Pull-high Resistance of DATA and CLK to VDD	5V	MODE_CTRL[1:0]=01B or 11B	2	4.7	8	kΩ
V_{IH}	Input High Voltage for DATA and CLK Pins	5V	MODE_CTRL[1:0]=01B or 11B	2	—	5	V
V_{IL}	Input Low Voltage for DATA and CLK Pins	5V	MODE_CTRL[1:0]=01B or 11B	0	—	0.8	V
I_{OL_DACK}	Sink Current for DATA and CLK Pins	5V	MODE_CTRL[1:0]=01B or 11B $V_{OL}=0.1V_{DD}$	6	12	—	mA

Power-on Reset Characteristics

Ta=-40°C~85°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_{POR}	V_{DD} Rising 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



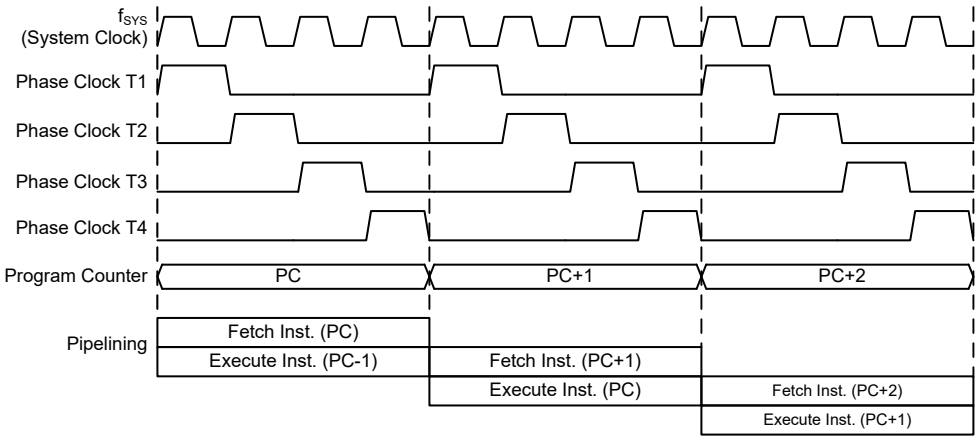
System Architecture

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

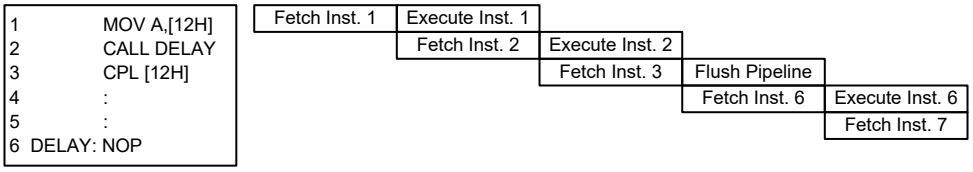
Clocking and Pipelining

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

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



System Clocking and Pipelining



Instruction Fetching

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. Only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by the application program.

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.

Program Counter	
High Byte	**Low Byte (PCL)**
PC11~PC8	PCL7~PCL0

Program Counter

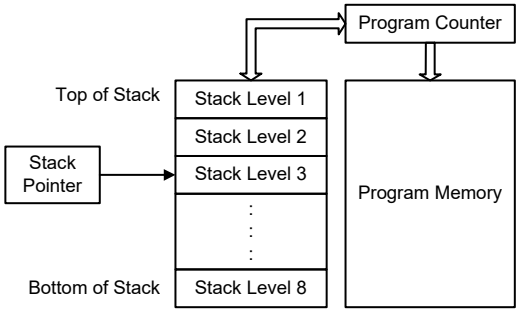
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly; however, as only this low byte is available 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. Manipulating the PCL register may cause program branching, so an extra cycle is needed to pre-fetch.

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organized into 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, 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.

If the stack is overflow, the first Program Counter save in the stack will be lost.



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,

  LADD, LADDM, LADC, LADCM, LSUB, LSUBM, LSBC, LSBCM, LDAA
- Logic operations:

  AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA,

  LAND, LANDM, LOR, LORM, LXOR, LXORM, LCPL, LCPLA
- Rotation:

  RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC,

  LRR, LRRA, LRRCA, LRRC, LRLA, LRL, LRLCA, LRLC
- Increment and Decrement:

  INCA, INC, DECA, DEC,

  LINCA, LINC, LDECA, LDEC

- Branch decision:

  JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET, RETI,

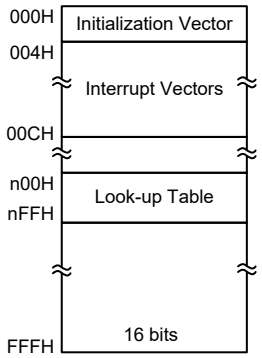
  LSNZ, LSZ, LSZA, LSIZ, LSIZA, LSDZ, LSDZA

OTP Program Memory

The Program Memory is the location where the user code or program is stored. The device is supplied with One-Time Programmable, OTP memory where users can program their application code into the device.

Structure

The Program Memory has a capacity of 4K×16 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be set in any location within the Program Memory, is addressed by a separate table pointer register.



Program Memory Structure

Special Vectors

Within the Program Memory, certain locations are reserved for the reset and interrupts. The location 000H is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.

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 address of the look up data to be retrieved in the table pointer register, TBLP and TBHP. These registers define the total address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the Program Memory using the corresponding table read instruction such as “TABRD [m]” or “TABRDL [m]” respectively when the memory [m] is located in Sector 0. If the memory [m] is located in other sectors except Sector 0, the data can be retrieved from the program memory using the corresponding extended table read instruction such as “LTABRD [m]” or “LTABRDL [m]” respectively. When the instruction is executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register.

The accompanying diagram illustrates the addressing data flow of the look-up table.

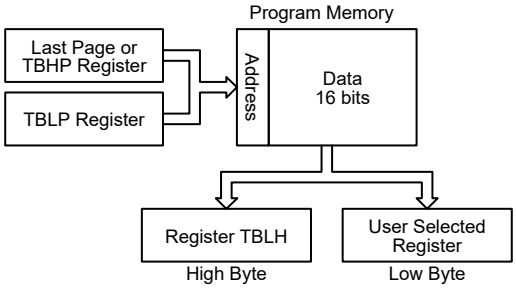


Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the microcontroller. This example uses raw table data located in the Program Memory which is stored there using the ORG statement. The value at this ORG statement is "0F00H" which refers to the start address of the last page within the 4K Program Memory of the microcontroller. The table pointer low byte register 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 "0F06H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the specific address pointed by the TBHP and TBLP registers if the "TABRD [m]" or "LTABRD [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 "TABRD [m]" or "LTABRD [m]" instruction is executed.

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

Table Read Program Example

```
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 the page that tbhp pointed
mov a,0FH         ; initialise high table pointer
mov tbhp,a
:
:
tabrd tempreg1    ; transfers value in table referenced by table pointer
                  ; data at program memory address "0F06H" transferred
                  ; to tempreg1 and TBLH
dec tblp          ; reduce value of table pointer by one
tabrd tempreg2    ; transfers value in table referenced by table pointer
                  ; data at program memory address "0F05H" transferred
                  ; to tempreg2 and TBLH
                  ; in this example the data "1AH" is transferred to tempreg1 and
```

```
                    ; data "0FH" to tempreg2 the value "00H" will be
                    ; transferred to the high byte register TBLH
:
:
org 0F00H           ; sets initial address of last page
dc 00Ah,00Bh,00Ch,00Dh,00Eh,00Fh,01Ah,01Bh
```

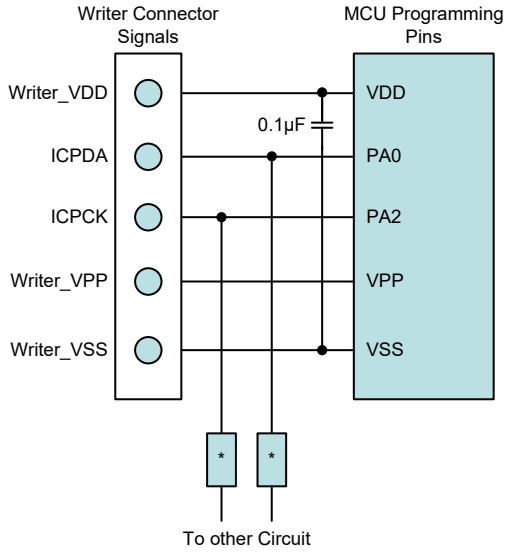
In Circuit Programming – ICP

The OTP type Program Memory is provided for users to program their application code one time into the device. As an additional convenience, Holtek has provided a means of programming the microcontroller in-circuit using a 5-pin interface. This provides manufacturers with the possibility of manufacturing their circuit boards complete with an un-programmed microcontroller, and then programming the program at a later stage.

Holtek Writer Pins	MCU Programming Pins	Pin Description
ICPDA	PA0	Programming Serial Data/Address
ICPCK	PA2	Programming Clock
VPP	VPP	Programming OTP ROM power supply (8.5V)
VDD	VDD	Power Supply. A 0.1μF capacitor is required to be connected between VDD and VSS for programming.
VSS	VSS	Ground

The Program Memory can be programmed serially in-circuit using this 5-wire interface. Data is downloaded and uploaded serially on a single pin with an additional line for the clock. Three additional lines are required for the power supply. The technical details regarding the in-circuit programming of the device is beyond the scope of this document and will be supplied in supplementary literature.

During the programming process, the user must take care of the ICPDA and ICPCK pins for data and clock programming purposes to ensure that no other outputs are connected to these two pins.



Note: 1. A 0.1μF capacitor is required to be connected between VDD and VSS for ICP programming, and as close to these pins as possible.

2. * may be resistor or capacitor. The resistance of * must be greater than 1kΩ or the capacitance of * must be less than 1nF.

On-Chip Debug Support – OCDS

There is an EV chip named HT82EV45R which is used to emulate the HT82B45R device. The EV chip device also provides an "On-Chip Debug" function to debug the real MCU device during the development process. The EV chip and the real MCU device are almost functionally compatible except for "On-Chip Debug" function. Users can use the EV chip device to emulate the real chip device behavior by connecting the OCDSDA and OCDSCK pins to the Holtek HT-IDE development tools. The OCDSDA pin is the OCDS Data/Address input/output pin while the OCDSCK pin is the OCDS clock input pin. When users use the EV chip device for debugging, other pin functions which are shared with the OCDSDA and OCDSCK pins in the device will have no effect in the EV chip. However, the two OCDS pins which are pin-shared with the ICP programming pins are still used as the Program Memory programming pins for ICP. For more detailed OCDS information, refer to the corresponding document named "Holtek e-Link for 8-bit MCU OCDS User's Guide".

Holtek e-Link Pins	EV Chip OCDS Pins	Pin Description
OCDSDA	OCDSDA	On-Chip Debug Support Data/Address input/output
OCDSCK	OCDSCK	On-Chip Debug Support Clock input
VDD	VDD	Power Supply
VSS	VSS	Ground

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.

Categorized into two types, the first of these is an area of RAM where special function registers are located. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is reserved for general purpose use. All locations within this area are read and write accessible under program control.

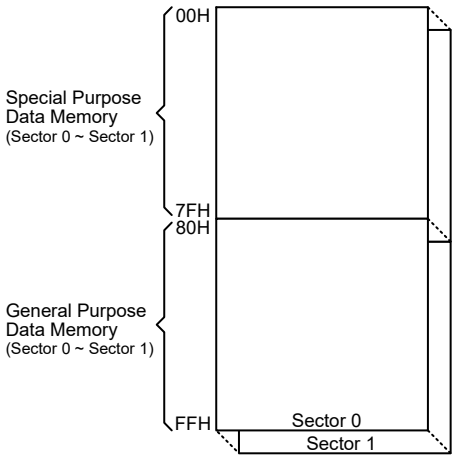
Switching between the different Data Memory sectors is achieved by properly setting the Memory Pointers to correct value if using the indirect addressing method.

Structure

The Data Memory is subdivided into several sectors, all of which are implemented in 8-bit wide RAM. The address range of the Special Purpose Data Memory for the device is from 00H to 7FH while the General Purpose Data Memory address range is from 80H to FFH.

Special Purpose Data Memory	General Purpose Data Memory	
Located Sectors	Capacity	Sector: Address
Sector 0: 00H~7FH	256×8	0: 80H~FFH 1: 80H~FFH

Data Memory Summary



Data Memory Structure

Data Memory Addressing

For device that supports the extended instructions, there is no Bank Pointer for Data Memory. For Data Memory the desired Sector is pointed by the MP1H or MP2H register and the certain Data Memory address in the selected sector is specified by the MP1L or MP2L register when using indirect addressing access.

Direct Addressing can be used in all sectors using the corresponding instruction which can address all available data memory space. For the accessed data memory which is located in any data memory sectors except sector 0, the extended instructions can be used to access the data memory instead of using the indirect addressing access. The main difference between standard instructions and extended instructions is that the data memory address “m” in the extended instructions has 9 valid bits for this device, the high byte indicates a sector and the low byte indicates a specific address.

General Purpose Data Memory

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

Special Purpose Data Memory

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

	Sector 0
00H	IAR0
01H	MP0
02H	IAR1
03H	MP1L
04H	MP1H
05H	ACC
06H	PCL
07H	TBLP
08H	TBLH
09H	TBHP
0AH	STATUS
0BH	
0CH	IAR2
0DH	MP2L
0EH	MP2H
0FH	RSTFC
10H	INTC
11H	WDTC
12H	LVRC
13H	
14H	PA
15H	PAC
16H	PAPU
17H	PAWU
18H	PAWUG
19H	PAS0
1AH	PAS1
1BH	PB
1CH	PBC
1DH	PBPU
1EH	PBWU
1FH	PBWUG
20H	PBS0
21H	PBS1
22H	PC
23H	PCC
24H	PCPU
25H	PCWU
26H	PCWUG
27H	PCS0
28H	PCS1
29H	PD
2AH	PDC
2BH	PDPU
2CH	PDWU
2DH	PDWUG
2EH	PDS0
2FH	PDS1
30H	PE
31H	PEC
32H	PEPU
33H	PEWU
34H	PEWUG
35H	PES0
36H	SLEWC
37H	
38H	
39H	
3AH	
3BH	SCC
3CH	HIRCC
3DH	PSCR
3EH	
3FH	RSTC

	Sector 0
40H	USC
41H	UISR
42H	UCC
43H	USB_STAT
44H	PIPE_CTRL
45H	AWR
46H	STALL
47H	PIPE
48H	SIES
49H	MISC
4AH	ENDPT_EN
4BH	FIFO0
4CH	FIFO1
4DH	FIFO2
4EH	TMR0
4FH	TMR0C
50H	TMR1C
51H	TMR1L
52H	TMR1H
53H	
54H	
55H	
56H	
57H	
58H	
59H	
5AH	
5BH	
5CH	
5DH	
5EH	
5FH	
60H	
61H	
62H	
63H	
64H	
65H	
66H	
67H	
68H	
69H	
6AH	
6BH	
6CH	
6DH	
6EH	
6FH	
70H	
71H	
72H	
73H	
74H	
75H	
76H	
77H	
78H	
79H	
7AH	
7BH	
7CH	
7DH	
7EH	
7FH	

: Unused, read as 00H

: Reserved, cannot be changed

Special Purpose Data Memory Structure

Special Function Register Description

Most of the Special Function Register details will be described in the relevant functional section. However, several registers require a separate description in this section.

Indirect Addressing Registers – IAR0, IAR1, IAR2

The Indirect Addressing Registers, IAR0, IAR1 and IAR2, 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, IAR1 and IAR2 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointers, MP0, MP1L/MP1H or MP2L/MP2H. Acting as a pair, IAR0 and MP0 can together access data only from Sector 0 while the IAR1 register together with MP1L/MP1H register pair and IAR2 register together with MP2L/MP2H register pair can access data from any Data Memory sector. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointers – MP0, MP1H/MP1L, MP2H/MP2L

Five Memory Pointers, known as MP0, MP1L, MP1H, MP2L and MP2H, are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Sector 0, while MP1L/MP1H together with IAR1 and MP2L/MP2H together with IAR2 are used to access data from all data sectors according to the corresponding MP1H or MP2H register. Direct Addressing can be used in all data sectors using the extended instructions which can address all available data memory space.

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

Indirect Addressing Program Example

Example 1

```
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:
```

Example 2

```
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,01h             ; setup the memory sector
     mov mp1h,a
     mov a,offset adres1   ; Accumulator loaded with first RAM address
     mov mp1l,a            ; setup memory pointer with first RAM address
loop:
     clr IAR1              ; clear the data at address defined by MP1L
     inc mp1l              ; increment memory pointer MP1L
     sdz block             ; check if last memory location has been cleared
     jmp loop
continue:
:
```

The important point to note here is that in the example shown above, no reference is made to specific RAM addresses.

Direct Addressing Program Example using extended instructions

```
data .section 'data'
temp db ?
code .section at 0 code
org 00h
start:
     lmov a,[m]            ; move [m] data to acc
     lsub a,[m+1]          ; compare [m] and [m+1] data
     snz  c                ; [m]>[m+1]?
     jmp  continue         ; no
     lmov a,[m]            ; yes, exchange [m] and [m+1] data
     mov  temp,a
     lmov a,[m+1]
     lmov [m],a
     mov  a,temp
     lmov [m+1],a
continue:
:
```

Note: Here “m” is a data memory address located in any data memory sectors. For example, m=1F0H, it indicates address 0F0H in Sector 1.

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 Byte 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, TBHP, TBLH

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

Status Register – STATUS

This 8-bit register contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), SC flag, CZ flag, 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, C, SC and CZ 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.
- CZ is the operational result of different flags for different instructions. Refer to register definitions for more details.
- SC is the result of the “XOR” operation which is performed by the OV flag and the MSB of the current instruction operation result.

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

• STATUS Register

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

“x”: unknown

Bit 7 **SC**: The result of the “XOR” operation which is performed by the OV flag and the MSB of the instruction operation result

Bit 6 **CZ**: The operational result of different flags for different instructions

For SUB/SUBM/LSUB/LSUBM instructions, the CZ flag is equal to the Z flag.

For SBC/SBCM/LSBC/LSBCM instructions, the CZ flag is the “AND” operation result which is performed by the previous operation CZ flag and current operation Z flag. For other instructions, the CZ flag will not be affected.

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

The “C” flag is also affected by a rotate through carry instruction.

Oscillators

Various oscillator types offer the user a wide range of functions according to their various application requirements. The flexible features of the oscillator functions ensure that the best optimisation can be achieved in terms of speed and power saving. Oscillator selections and operation are selected through the relevant control registers.

Oscillator Overview

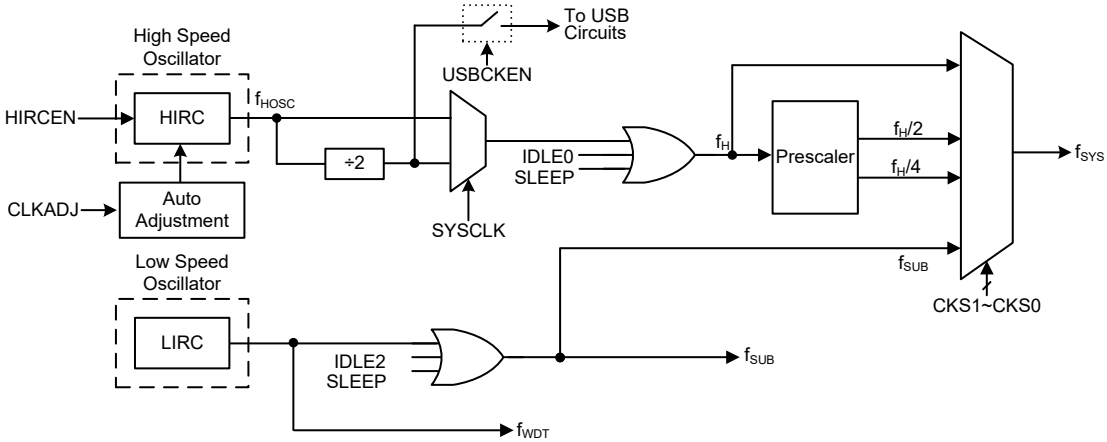
In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer. Two fully integrated internal oscillators, requiring no external components, are provided to form a wide range of both fast and slow system oscillators. The higher frequency oscillator provides higher performance but carry with it the disadvantage of higher power requirements, while the opposite is of course true for the lower frequency oscillator. With the capability of dynamically switching between fast and slow system clock, the device has the flexibility to optimize the performance/power ratio, a feature especially important in power sensitive portable applications.

Type	Name	Frequency
Internal High Speed RC	HIRC	12MHz
Internal Low Speed RC	LIRC	32kHz

Oscillator Types

System Clock Configurations

There are two methods of generating the system clock, a high speed oscillator and a low speed oscillator. The high speed oscillator is the internal 12MHz RC oscillator, HIRC. The low speed oscillator is the internal 32kHz RC oscillator, LIRC. Selecting whether the low or high speed oscillator is used as the system oscillator is implemented using the CKS1~CKS0 bits in the SCC register and as the system clock can be dynamically selected.



System Clock Configurations

Internal High Speed RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has a fixed frequency of 12MHz. Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuits are used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimised.

Internal 32kHz Oscillator – LIRC

The Internal 32kHz System Oscillator is a low frequency oscillator. It is a fully integrated RC oscillator with a typical frequency of 32kHz, requiring no external components for its implementation.

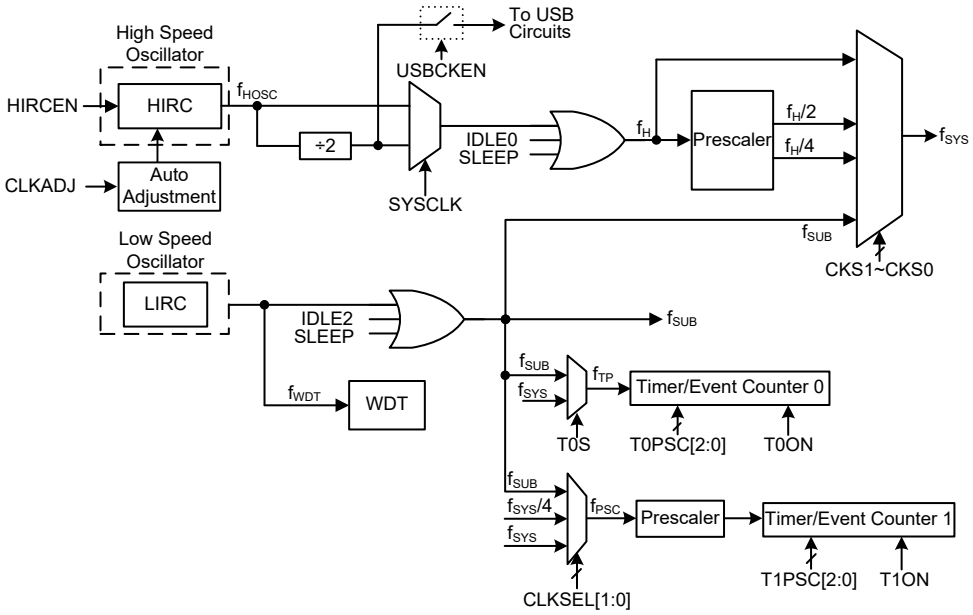
Operating Modes and System Clocks

Present day applications require that their microcontrollers have high performance but often still demand that they consume as little power as possible, conflicting requirements that are especially true in battery powered portable applications. The fast clocks required for high performance will by their nature increase current consumption and of course vice versa, lower speed clocks reduce current consumption. As Holtek has provided the device with both high and low speed clock sources and the means to switch between them dynamically, the user can optimise the operation of their microcontroller to achieve the best performance/power ratio.

System Clocks

The device has many different clock sources for both the CPU and peripheral function operation. By providing the user with a wide range of clock options using register programming, a clock system can be configured to obtain maximum application performance.

The main system clock can come from either a high frequency, f_H, or low frequency, f_{SUB}, source, and is selected using the CKS1~CKS0 bits in the SCC register. The high speed system clock is sourced from the HIRC oscillator. The low speed system clock source is sourced from the LIRC oscillator. The other choice, which is a divided version of the high speed system oscillator has a range of $f_H/2$~$f_H/4$.



Device Clock Configurations

Note: When the system clock source f_{SYS} is switched to f_{SUB} from f_H, the high speed oscillator can be stopped to conserve the power or continue to oscillate to provide the clock source, f_H~$f_H/4$, for peripheral circuit to use, which is determined by configuring the corresponding high speed oscillator enable control bit.

System Operation Modes

There are six different modes of operation for the microcontroller, each one with its own special characteristics and which can be chosen according to the specific performance and power requirements of the application. There are two modes allowing normal operation of the microcontroller, the FAST Mode and SLOW Mode. The remaining four modes, the SLEEP, IDLE0, IDLE1 and IDLE2 Mode are used when the microcontroller CPU is switched off to conserve power.

Operation Mode	CPU	Register Setting				f_{SYS}	f_H	f_{SUB}	f_{WDT}
		FHIDEN	FSIDEN	CKS1~CKS0	SYSCLK				
FAST	On	x	x	00~10	x	f_H~f_H/4	On	On	On
SLOW	On	x	x	11	x	f_{SUB}	On/Off (1)	On	On
IDLE0	Off	0	1	00~10	x	Off	Off	On	On
				11	x	On			
IDLE1	Off	1	1	xx	x	On	On	On	On
IDLE2	Off	1	0	00~10	x	On	On	Off	On/Off (2)
				11	x	Off			
SLEEP	Off	0	0	xx	x	Off	Off	Off	On/Off (2)

"x": don't care

Note: 1. The f_H clock will be switched on or off by configuring the corresponding oscillator enable bit in the SLOW mode.
2. The f_{WDT} clock can be switched on or off which is controlled by the WDT function being enabled or disabled in the IDLE2 and SLEEP mode.
3. The USB function is inactive in the IDLE0 and SLEEP modes.

FAST Mode

This is one of the main operating modes where the microcontroller has all of its functions operational and where the system clock is provided by one of the high speed oscillators. This mode allows the microcontroller to operate normally with a clock source coming from the HIRC oscillator. The high speed oscillator will however first be divided by a ratio ranging from 1 to 4, the actual ratio being selected by the CKS1~CKS0 bits in the SCC register. Although a high speed oscillator is used, running the microcontroller at a divided clock ratio reduces the operating current.

SLOW Mode

This is also a mode where the microcontroller operates normally although now with a slower speed clock source. The clock source used will be from f_{SUB}. The f_{SUB} clock is derived from the LIRC oscillator.

SLEEP Mode

The SLEEP Mode is entered when a HALT instruction is executed and when the FHIDEN and FSIDEN bits are low. In the SLEEP mode the CPU will be stopped, and the f_{SUB} clock to peripheral will be stopped too. However the f_{WDT} clock will continue to operate if the WDT function is enabled by the WDTC register.

IDLE0 Mode

The IDLE0 Mode is entered when a HALT instruction is executed and when the FHIDEN bit in the SCC register is low and the FSIDEN bit in the SCC register is high. In the IDLE0 Mode the CPU will be switched off but the low speed oscillator will be turned on to drive some peripheral functions.

IDLE1 Mode

The IDLE1 Mode is entered when a HALT instruction is executed and when the FHIDEN bit in the SCC register is high and the FSIDEN bit in the SCC register is high. In the IDLE1 Mode the CPU will be switched off but both the high and low speed oscillators will be turned on to provide a clock source to keep some peripheral functions operational.

IDLE2 Mode

The IDLE2 Mode is entered when a HALT instruction is executed and when the FHIDEN bit in the SCC register is high and the FSIDEN bit in the SCC register is low. In the IDLE2 Mode the CPU will be switched off but the high speed oscillator will be turned on to provide a clock source to keep some peripheral functions operational.

Control Registers

The registers, SCC and HIRCC, are used to control the system clock and the corresponding oscillator configurations.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SCC	CKS1	CKS0	SYSCLK	—	—	—	FHIDEN	FSIDEN
HIRCC	CLKADJ	CLKADJF	—	—	—	—	HIRCF	HIRCEN

System Operating Mode Control Register List

- **SCC Register**

Bit	7	6	5	4	3	2	1	0
Name	CKS1	CKS0	SYSCLK	—	—	—	FHIDEN	FSIDEN
R/W	R/W	R/W	R/W	—	—	—	R/W	R/W
POR	0	1	0	—	—	—	0	0

Bit 7~6 **CKS1~CKS0**: System clock selection
00: f_H
01: $f_H/2$
10: $f_H/4$
11: f_{SUB}
These two bits are used to select which clock is used as the system clock source. In addition to the system clock source directly derived from f_H or f_{SUB}, a divided version of the high speed system oscillator can also be chosen as the system clock source.

Bit 5 **SYSCLK**: System oscillator frequency selection
0: 12MHz
1: 6MHz
This bit is used to specify the system oscillator frequency used by the MCU.

Bit 4~2 Unimplemented, read as "0"

Bit 1 **FHIDEN**: High Frequency oscillator control when CPU is switched off
0: Disable
1: Enable
This bit is used to control whether the high speed oscillator is activated or stopped when the CPU is switched off by executing a "HALT" instruction.

Bit 0 **FSIDEN**: Low Frequency oscillator control when CPU is switched off
0: Disable
1: Enable
This bit is used to control whether the low speed oscillator is activated or stopped when the CPU is switched off by executing a "HALT" instruction.

Note: A certain delay is required before the relevant clock is successfully switched to the target clock source after any clock switching setup using the CKS1~CKS0 bits. A proper delay time must be arranged before executing the following operations which require immediate reaction with the target clock source.

Clock switching delay time = $4\times t_{SYS}+[0\sim(1.5\times t_{Curr.}+0.5\times t_{Tar.})]$, where $t_{Curr.}$ indicates the current clock period, $t_{Tar.}$ indicates the target clock period and t_{SYS} indicates the current system clock period.

• HIRCC Register

Bit	7	6	5	4	3	2	1	0
Name	CLKADJ	CLKADJF	—	—	—	—	HIRCF	HIRCEN
R/W	R/W	R/W	—	—	—	—	R	R/W
POR	0	0	—	—	—	—	0	1

Bit 7 **CLKADJ**: HIRC clock automatic adjustment function control in the USB mode
0: Disable
1: Enable

Bit 6 **CLKADJF**: HIRC clock automatic adjustment stable flag
0: Unstable
1: Stable

The CLKADJF bit indicates whether the HIRC frequency adjusting operation is completed or not when the CLKADJ bit is set to 1. Users can continuously monitor the CLKADJF bit by application programs to make sure that the HIRC frequency accuracy is stably adjusted in the range of ±1.5%. The CLKADJF bit can be cleared by the application program.

Bit 5~2 Unimplemented, read as "0"

Bit 1 **HIRCF**: HIRC oscillator stable flag
0: HIRC unstable
1: HIRC stable

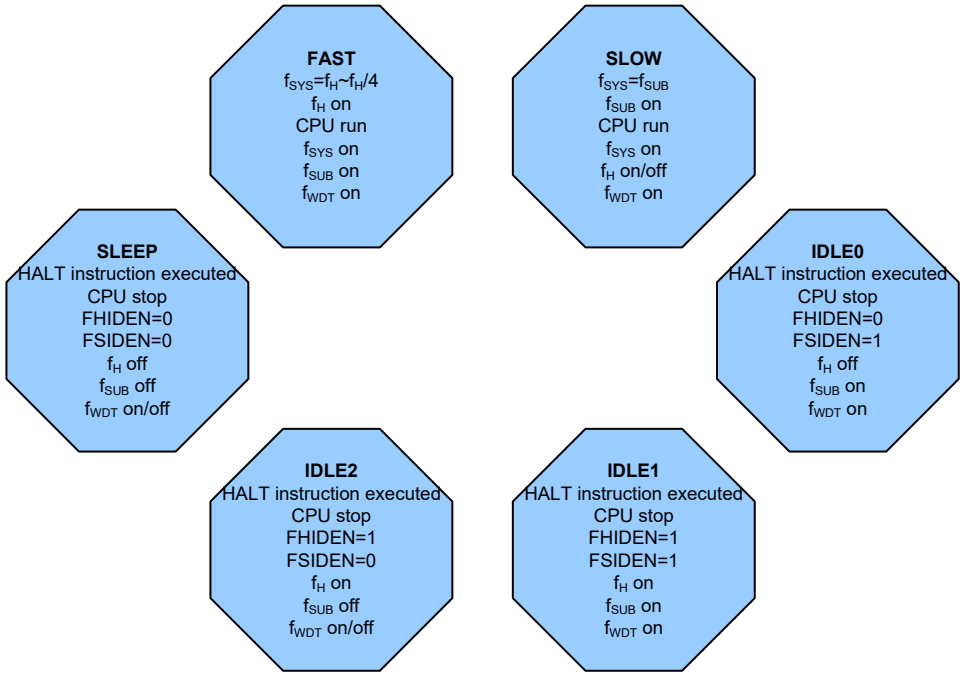
This bit is used to indicate whether the HIRC oscillator is stable or not. When the HIRCEN bit is set to 1 to enable the HIRC oscillator, the HIRCF bit will first be cleared to 0 and then set to 1 after the HIRC oscillator is stable.

Bit 0 **HIRCEN**: HIRC oscillator enable control
0: Disable
1: Enable

Operating Mode Switching

The device can switch between operating modes dynamically allowing the user to select the best performance/power ratio for the present task in hand. In this way microcontroller operations that do not require high performance can be executed using slower clocks thus requiring less operating current and prolonging battery life in portable applications.

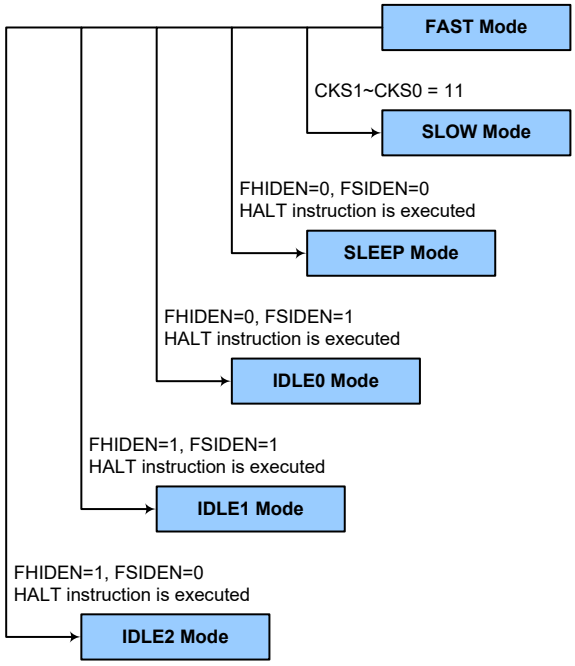
In simple terms, Mode Switching between the FAST Mode and SLOW Mode is executed using the CKS1~CKS0 bits in the SCC register while Mode Switching from the FAST/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When a HALT instruction is executed, whether the device enters the IDLE Mode or the SLEEP Mode is determined by the condition of the FHIDEN and FSIDEN bits in the SCC register.



FAST Mode to SLOW Mode Switching

When running in the FAST Mode, which uses the high speed system oscillator, and therefore consumes more power, the system clock can switch to run in the SLOW Mode by setting the CKS1~CKS0 bits to "11" in the SCC register. This will then use the low speed system oscillator which will consume less power. Users may decide to do this for certain operations which do not require high performance and can subsequently reduce power consumption.

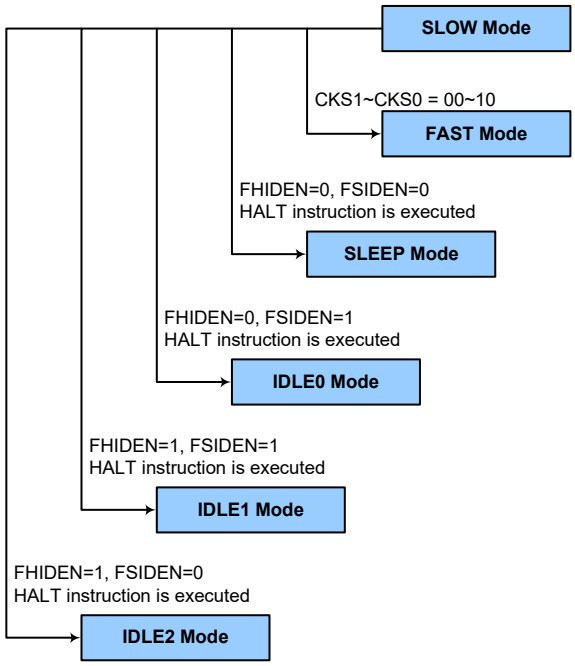
The SLOW Mode is sourced from the LIRC oscillator and therefore requires this oscillator to be stable before full mode switching occurs.



SLOW Mode to FAST Mode Switching

In SLOW mode the system clock is derived from f_{SUB}. When system clock is switched back to the FAST mode from f_{SUB}, the CKS1~CKS0 bits should be set to "00"~"10" and then the system clock will respectively be switched to f_H~f_H/4.

However, if f_H is not used in SLOW mode and thus switched off, it will take some time to re-oscillate and stabilise when switching to the FAST mode from the SLOW Mode. This is monitored using the HIRCF bit in the HIRCC register. The time duration required for the high speed system oscillator stabilization is specified in the System Start Up Time Characteristics.



Entering the SLEEP Mode

There is only one way for the device to enter the SLEEP Mode and that is to execute the "HALT" instruction in the application program with both the FHIDEN and FSIDEN bits in the SCC register equal to "0". In this mode all the clocks and functions will be switched off except the WDT function. When this instruction is executed under the conditions described above, the following will occur:

- The system clock will be stopped and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.
- The USB will enter the suspend mode if the USB function is enabled.

Entering the IDLE0 Mode

There is only one way for the device to enter the IDLE0 Mode and that is to execute the "HALT" instruction in the application program with the FHIDEN bit in the SCC register equal to "0" and the FSIDEN bit in the SCC register equal to "1". When this instruction is executed under the conditions described above, the following will occur:

- The f_H clock will be stopped and the application program will stop at the "HALT" instruction, but the f_{SUB} clock will be on.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.
- The USB will enter the suspend mode if the USB function is enabled.

Entering the IDLE1 Mode

There is only one way for the device to enter the IDLE1 Mode and that is to execute the "HALT" instruction in the application program with both the FHIDEN and FSIDEN bits in the SCC register equal to "1". When this instruction is executed under the conditions described above, the following will occur:

- The f_H and f_{SUB} clocks will be on but the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

Entering the IDLE2 Mode

There is only one way for the device to enter the IDLE2 Mode and that is to execute the "HALT" instruction in the application program with the FHIDEN bit in the SCC register equal to "1" and the FSIDEN bit in the SCC register equal to "0". When this instruction is executed under the conditions described above, the following will occur:

- The f_H clock will be on but the f_{SUB} clock will be off and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

Standby Current Considerations

As the main reason for entering the SLEEP or IDLE Mode is to keep the current consumption of the device to as low a value as possible, perhaps only in the order of several micro-amps except in the IDLE1 and IDLE2 Mode, 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. This also applies to the device which has different package types, as there may be unbonded pins. These must either be setup as outputs or if setup as inputs must have pull-high resistors connected. In addition, the I/O pin-shared with VPP must not be set to output high, as this could 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. Also note that additional standby current will also be required if the LIRC oscillator has enabled.

In the IDLE1 and IDLE2 Mode the high speed oscillator is on, if the peripheral function clock source is derived from the high speed oscillator, the additional standby current will also be perhaps in the order of several hundred micro-amps.

Wake-up

To minimise power consumption the device can enter the SLEEP or any IDLE Mode, where the CPU will be switched off. However, when the device is woken up again, it will take a considerable time for the original system oscillator to restart, stabilise and allow normal operation to resume.

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

- An external $\overline{\text{RES}}$ pin reset
- A USB reset signal reset
- An external falling or rising edge on Port A~Port E
- A system interrupt
- A WDT overflow

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

Each pin on Port A~E can be setup using the PxWU and PxWUG registers to permit a negative or positive 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 high before entering the SLEEP or IDLE Mode, the wake-up function of the related interrupt will be disabled.

Watchdog Timer

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

Watchdog Timer Clock Source

The Watchdog Timer clock source is provided by the internal clock, f_{WDT}, which is sourced from the LIRC oscillator. The LIRC internal oscillator has an approximate frequency of 32kHz and this specified internal clock period can vary with V_{DD}, temperature and process variations. The Watchdog Timer source clock is then subdivided by a ratio of 2^8 to 2^{18} to give longer timeouts, the actual value being chosen using the WS2~WS0 bits in the WDTC register.

Watchdog Timer Control Register

A single register, WDTC, controls the required time-out period as well as the Watchdog Timer enable/disable and the MCU reset operation.

• WDTC Register

Bit	7	6	5	4	3	2	1	0
Name	WE4	WE3	WE2	WE1	WE0	WS2	WS1	WS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	0	1	0	0	1	1

Bit 7~3 **WE4~WE0**: WDT function enable control
10101: Disable
01010: Enable
Other values: Reset MCU
When these bits are changed to any other values due to environmental noise the microcontroller will be reset; this reset operation will be activated after a delay time, t_{SRESET}, and the WRF bit in the RSTFC register will be set high.

Bit 2~0 **WS2~WS0**: WDT time-out period selection
000: $2^8/f_{WDT}$
001: $2^{10}/f_{WDT}$
010: $2^{12}/f_{WDT}$
011: $2^{14}/f_{WDT}$
100: $2^{15}/f_{WDT}$
101: $2^{16}/f_{WDT}$
110: $2^{17}/f_{WDT}$
111: $2^{18}/f_{WDT}$
These three bits determine the division ratio of the watchdog timer source clock, which in turn determines the time-out period.

• RSTFC Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

"x": unknown

Bit 7~4 Unimplemented, read as "0"

Bit 3 **RSTF**: Reset control register software reset flag
Refer to the $\overline{RES}$ Pin Reset section.

Bit 2 **LVRF**: LVR function reset flag
Refer to the Low Voltage Reset section.

Bit 1 **LRF**: LVRC register software reset flag
Refer to the Low Voltage Reset section.

Bit 0 **WRF**: WDT control register software reset flag
0: Not occurred
1: Occurred
This bit is set to 1 by the WDT control register software reset and cleared by the application program. This bit can only be cleared to zero by application program.

Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog instruction. If the program malfunctions for whatever reason, jumps to an unknown location, or enters an endless loop, the clear instruction will not be executed in the correct manner, in which case the Watchdog Timer will overflow and reset the device. There are five bits, WE4~WE0, in the WDTC register to offer the Watchdog Timer enable/disable control and the MCU reset. The WDT function will be enabled when the WE4~WE0 bits are set to a value of 01010B while the WDT function will be disabled if the WE4~WE0 bits are equal to 10101B. If the WE4~WE0 bits are set to any other values rather than 01010B and 10101B, it will reset the device after a delay time, t_{SRESET}. After power on these bits will have a value of 01010B.

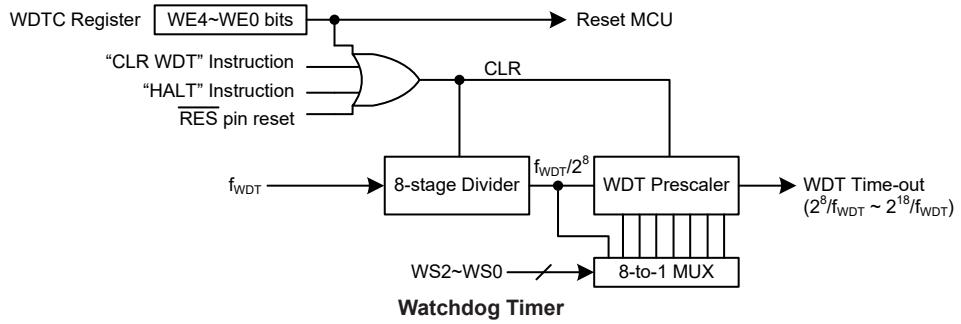
WE4~WE0 Bits	WDT Function
10101B	Disable
01010B	Enable
Any other value	Reset MCU

Watchdog Timer Function Control

Under normal program operation, a Watchdog Timer time-out will initialise a device reset and set the status bit TO. However, if the system is in the SLEEP or IDLE Mode, when a Watchdog Timer time-out occurs, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Four methods can be adopted to clear the contents of the Watchdog Timer. The first is a WDTC register software reset, which means a certain value except 01010B and 10101B written into the WE4~WE0 bits, the second is using the Watchdog Timer software clear instruction, the third is via a HALT instruction and the fourth is an external hardware reset, which means a low level on the external reset pin if the external reset pin is selected by the RSTC register.

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

The maximum time out period is when the 2^{18} division ratio is selected. As an example, with a 32kHz LIRC oscillator as its source clock, this will give a maximum watchdog period of around 8 seconds for the 2^{18} division ratio and a minimum timeout of 8ms for the 2^8 division ration.



Watchdog Timer

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 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 preceed with normal operation after the reset line is allowed to return high.

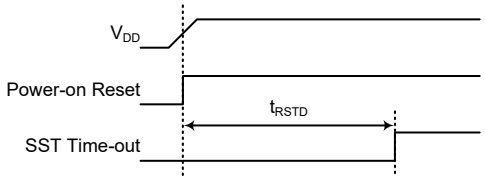
Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{\text{RES}}$ reset is implemented in situations where the power supply voltage falls below a certain threshold. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

Reset Functions

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

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.



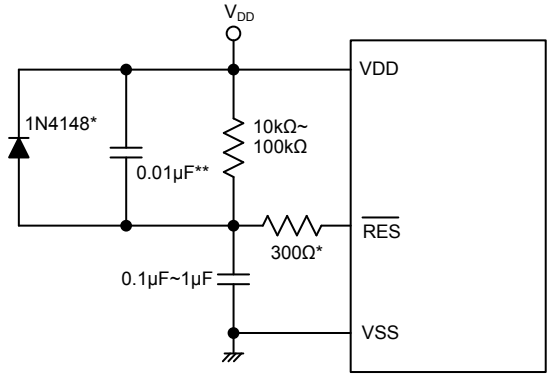
Power-On Reset Timing Chart

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

As the reset pin is shared with I/O pins, the reset function must be selected using the control register, RSTC. 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 proper reset operation. For this reason it is recommended that an external RC network is connected to the $\overline{\text{RES}}$ pin, whose additional time delay will ensure that the $\overline{\text{RES}}$ pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{\text{RES}}$ line reaches a certain voltage value, the reset delay time, t_{RSTD}, is invoked to provide an extea delay time after which the microcontroller will begin normal operation. The abbreviation SST in the figures stands for System Start-up Time.

For most applications a resistor connected between VDD and the $\overline{RES}$ line 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.

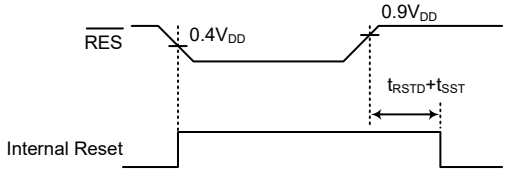
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



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

External $\overline{RES}$ Circuit

Pulling the $\overline{RES}$ pin low using external hardware will also execute a device reset. In this case, as in the case of other resets, the Program Counter will reset to zero and program execution initiated from this point.



$\overline{RES}$ Reset Timing Chart

There is an internal reset control register, RSTC, which is used to select the external $\overline{RES}$ pin function and provide a reset when the device operates abnormally due to the environmental noise interference. If the content of the RSTC register is set to any value other than 01010101B or 10101010B, it will reset the device after a delay time, t_{SRESET}. After power on the register will have a value of 01010101B.

RSTC7~RSTC0 Bits	Reset Function
01010101B	I/O pin or other pin-shared functions
10101010B	$\overline{RES}$ pin
Any other value	Reset MCU

Internal Reset Function Control

• **RSTC Register**

Bit	7	6	5	4	3	2	1	0
Name	RSTC7	RSTC6	RSTC5	RSTC4	RSTC3	RSTC2	RSTC1	RSTC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	0	1	0	1	0	1

Bit 7~0 **RSTC7~RSTC0**: Reset function control
01010101: I/O pin or other pin-shared functions
10101010: $\overline{\text{RES}}$ pin
Other values: Reset MCU

If these bits are changed due to adverse environmental conditions, the microcontroller will be reset. The reset operation will be activated after a delay time, t_{SRESET} and the RSTF bit in the RSTFC register will be set to 1.

All resets will reset this register to POR value except the WDT time-out hardware warm reset. Note that if the register is set to 10101010 to select the $\overline{\text{RES}}$ pin function, select this configuration has higher priority than other related pin-shared controls.

• **RSTFC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

"x": unknown

Bit 7~4 Unimplemented, read as "0"

Bit 3 **RSTF**: Reset control register software reset flag
0: Not occurred
1: Occurred
This bit is set to 1 by the RSTC control register software reset and cleared by the application program. This bit can only be cleared to zero by application program.

Bit 2 **LVRF**: LVR function reset flag
Refer to the Low Voltage Reset section.

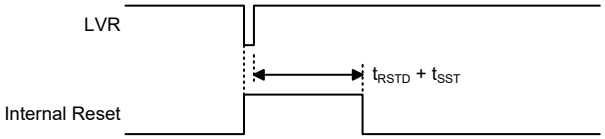
Bit 1 **LRF**: LVRC register software reset flag
Refer to the Low Voltage Reset section.

Bit 0 **WRF**: WDT control register software reset flag
Refer to the Watchdog Timer Control Register section.

Low Voltage Reset – LVR

The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device and provides an MCU reset when the value falls below a certain predefined level.

This LVR function is enabled or disabled by the LVRC control register. 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 in battery powered applications, the LVR will automatically reset the device internally and the LVRF bit in the RSTFC register will also be set to 1. 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 LVR Electrical 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 by the LVS7~LVS0 bits in the LVRC register. If the LVS7~LVS0 bits are changed to any other values by environmental noise, the LVR will reset the device after a delay time, t_{SRESET}. When this happens, the LRF bit in the RSTFC register will be set to 1. After power on the register will have the value of 01010101B. Note that the LVR function will be automatically disabled when the device enters the SLEEP or IDLE mode.



Low Voltage Reset Timing Chart

- **LVRC Register**

Bit	7	6	5	4	3	2	1	0
Name	LVS7	LVS6	LVS5	LVS4	LVS3	LVS2	LVS1	LVS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	1	0	0	1	1	0

Bit 7~0 **LVS7~LVS0**: LVR voltage select
01100110B: 2.1V (default)
01010101B: 2.55V
00110011B: 3.15V
10011001B: 3.8V
11110000B: LVR disable
Other values: Generates an MCU reset – register is reset to POR value

When an actual low voltage condition as specified above occurs, an MCU reset will be generated. The reset operation will be activated after the low voltage condition keeps more than a t_{LVR} time. In this situation the register contents will remain the same after such a reset occurs.

Any register value, other than the five defined register values above, will also result in the generation of an MCU reset. The reset operation will be activated after a delay time, t_{SRESET}. However in this situation the register contents will be reset to the POR value.

- **RSTFC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

"x": unknown

Bit 7~4 Unimplemented, read as "0"

Bit 3 **RSTF**: Reset control register software reset flag
Refer to the $\overline{RES}$ Pin Reset section.

Bit 2 **LVRF**: LVR function reset flag
0: Not occurred
1: Occurred
This bit is set to 1 when a specific low voltage reset condition occurs. This bit can only be cleared to zero by application program.

Bit 1 **LRF**: LVRC register software reset flag
0: Not occurred
1: Occurred
This bit is set high if the LVRC register contains any non-defined register values. This in effect acts like a software reset function. This bit can only be cleared to zero by application program.

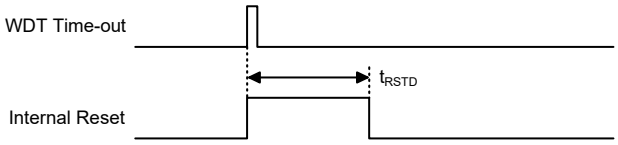
Bit 0 **WRF**: WDT control register software reset flag
Refer to the Watchdog Timer Control Register section.

USB Reset

The device contains a USB circuit, a reset signal can be generated by properly configuring the USB control register to reset the whole device. Refer to the "USB Interface" section for more associated details.

Watchdog Time-out Reset during Normal Operation

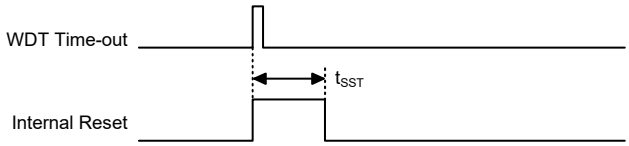
The Watchdog time-out Reset during normal operation in the FAST or SLOW Mode is the same as the LVR Reset except that the Watchdog time-out flag TO will be set to "1".



WDT Time-out Reset during Normal Operation Timing Chart

Watchdog Time-out Reset during SLEEP or IDLE Mode

The Watchdog time-out Reset during SLEEP or IDLE Mode is a little different from other kinds of reset. Most of the conditions remain unchanged except that the Program Counter and the Stack Pointer will be cleared to "0" and the TO flag will be set to "1". Refer to the System Start Up Time Characteristics for t_{SST} details.



WDT Time-out Reset during SLEEP or IDLE Mode Timing Chart

Reset Initial Conditions

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

TO	PDF	Reset Conditions
0	0	Power-on reset
u	u	$\overline{RES}$, LVR or USB reset during FAST or SLOW Mode operation
1	u	WDT time-out reset during FAST or SLOW Mode operation
1	1	WDT time-out reset during IDLE or SLEEP Mode operation

"u": 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	Cleared after reset, WDT begins counting
Timer/Event Counters	Timer/Event Counters will be turned off
Input/Output Ports	I/O ports will be setup as inputs
Stack Pointer	Stack Pointer will point to the top of the stack

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

Register	Power-On Reset	$\overline{\text{RES}}$ Reset (Normal Operation)	USB Reset (Normal Operation)	USB Reset (IDLE/SLEEP)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
IAR0	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP0	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IAR1	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP1L	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP1H	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBHP	---- xxxx	---- uuuu	---- uuuu	---- uuuu	---- uuuu	---- uuuu
STATUS	xx00 xxxx	uuuu uuuu	uuuu uuuu	uu01 uuuu	uu1u uuuu	uu11 uuuu
IAR2	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP2L	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP2H	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
RSTFC	---- 0x00	---- uuuu	---- uuuu	---- uuuu	---- uuuu	---- uuuu
INTC	-000 0000	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
WDTC	0101 0011	0101 0011	0101 0011	0101 0011	0101 0011	uuuu uuuu
LVRC	0110 0110	0110 0110	0110 0110	0110 0110	0110 0110	uuuu uuuu
PA	1111 1111	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAPU	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PAWU	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PAWUG	---- ---0	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
PAS0	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PAS1	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBPU	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PBWU	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PBWUG	---- ---0	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
PBS0	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PBS1	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCPU	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PCWU	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PCWUG	---- ---0	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
PCS0	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PCS1	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PD	1111 1111	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDC	1111 1111	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDPU	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu

Register	Power-On Reset	$\overline{\text{RES}}$ Reset (Normal Operation)	USB Reset (Normal Operation)	USB Reset (IDLE/SLEEP)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
PDWU	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PDWUG	---- ---0	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
PDS0	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PDS1	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PE	---- -111	---- -111	---- -111	---- -111	---- -111	---- -uuu
PEC	---- -111	---- -111	---- -111	---- -111	---- -111	---- -uuu
PEPU	---- -000	---- -000	---- -000	---- -000	---- -000	---- -uuu
PEWU	---- -000	---- -000	---- -000	---- -000	---- -000	---- -uuu
PEWUG	---- ---0	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
PES0	---- 0000	------ 0000	------ 0000	------ 0000	---- 0000	------ uuuu
SLEWC	---- ---0	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
SCC	010- --00	010- --00	010- --00	010- --00	010- --00	uuu- --uu
HIRCC	00-- --01	uu-- --01	uu-- --01	uu-- --01	uu-- --01	uu-- --uu
PSCR	---- --00	---- --00	---- --00	---- --00	---- --00	---- --uu
RSTC	0101 0101	0101 0101	0101 0101	0101 0101	0101 0101	uuuu uuuu
USC	11xx 0000	11xx 0000	uu00 0u00	uu00 0u00	uuxx uuuu	uuxx uuuu
UISR	0100 0000	u1uu 0000	uuuu u000	uuuu u000	uuuu uuuu	u1uu uuuu
UCC	--0- 0-00	--0- 0-00	--u- u-00	--u- u-00	--u- u-uu	--u- u-uu
USB_STAT	---x xxxx	---x xxxx	---x xxx0	---x xxx0	---x xxxx	---x xxxx
PIPE_CTRL	---- -110	---- -110	---- -110	---- -110	---- -uuu	---- -uuu
AWR	xxx xxx0	xxx xxxu	0000 0000	0000 0000	xxx xxxu	xxx xxxu
STALL	---- -11x	---- -uux	---- -110	---- -110	---- -uux	---- -uux
PIPE	---- -xxx	---- -xxx	---- -000	---- -000	---- -xxx	---- -xxx
SIES	0xxx xxxx	uxxx xxxx	0xxx xxxx	0xxx xxxx	uxxx xxxx	uxxx xxxx
MISC	xxx0 000x	xxxu uuux	xxx0 000x	xxx0 000x	xxxu uuux	xxxu uuux
ENDPT_EN	---- -111	---- -111	---- -111	---- -111	---- -uuu	---- -uuu
FIFO0	xxxx xxxx	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu
FIFO1	xxxx xxxx	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu
FIFO2	xxxx xxxx	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu
TMR0	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TMR0C	0000 1000	0000 1000	0000 1000	0000 1000	0000 1000	uuuu uuuu
TMR1C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
TMR1L	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TMR1H	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu

Note: “u” stands for unchanged
“x” stands for unknown
“-” stands for unimplemented

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 selections for all ports and wake-up selections on certain pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

The device provides bidirectional input/output lines labeled with port names PA~PE. These I/O ports are mapped to the RAM Data Memory with specific addresses as shown in the Special Purpose Data Memory table. The I/O port can be used for input and output operations. For input operation, the port is 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.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PAC	PAC7	PAC6	PAC5	PAC4	PAC3	PAC2	PAC1	PAC0
PAPU	PAPU7	PAPU6	PAPU5	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
PAWU	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	PAWU2	PAWU1	PAWU0
PAWUG	—	—	—	—	—	—	—	PAWUGS
PB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
PBC	PBC7	PBC6	PBC5	PBC4	PBC3	PBC2	PBC1	PBC0
PBPU	PBPU7	PBPU6	PBPU5	PBPU4	PBPU3	PBPU2	PBPU1	PBPU0
PBWU	PBWU7	PBWU6	PBWU5	PBWU4	PBWU3	PBWU2	PBWU1	PBWU0
PBWUG	—	—	—	—	—	—	—	PBWUGS
PC	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
PCC	PCC7	PCC6	PCC5	PCC4	PCC3	PCC2	PCC1	PCC0
PCPU	PCPU7	PCPU6	PCPU5	PCPU4	PCPU3	PCPU2	PCPU1	PCPU0
PCWU	PCWU7	PCWU6	PCWU5	PCWU4	PCWU3	PCWU2	PCWU1	PCWU0
PCWUG	—	—	—	—	—	—	—	PCWUGS
PD	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
PDC	PDC7	PDC6	PDC5	PDC4	PDC3	PDC2	PDC1	PDC0
PDPU	PDPU7	PDPU6	PDPU5	PDPU4	PDPU3	PDPU2	PDPU1	PDPU0
PDWU	PDWU7	PDWU6	PDWU5	PDWU4	PDWU3	PDWU2	PDWU1	PDWU0
PDWUG	—	—	—	—	—	—	—	PDWUGS
PE	—	—	—	—	—	PE2	PE1	PE0
PEC	—	—	—	—	—	PEC2	PEC1	PEC0
PEPU	—	—	—	—	—	PEPU2	PEPU1	PEPU0
PEWU	—	—	—	—	—	PEWU2	PEWU1	PEWU0
PEWUG	—	—	—	—	—	—	—	PEWUGS

"—": Unimplemented, read as "0"

I/O Logic Function Register List

Pull-high Resistors

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

Note that the pull-high resistor can be controlled by the relevant pull-high control register only when the pin-shared functional pin is selected as a digital input or NMOS output. Otherwise, the pull-high resistors cannot be enabled.

• **PxPU Register**

Bit	7	6	5	4	3	2	1	0
Name	PxPU7	PxPU6	PxPU5	PxPU4	PxPU3	PxPU2	PxPU1	PxPU0
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

PxPUn: I/O Port x pin pull-high function control
0: Disable
1: Enable

The PxPUn bit is used to control the pin pull-high function. Here the "x" is the Port name which can be A, B, C, D or E. However, the actual available bits for each I/O Port may be different.

I/O Pin Wake-up and Wake-up Polarity Control Registers

The HALT instruction forces the microcontroller into the SLEEP or IDLE Mode which preserves power, a feature that is important for battery and other low-power applications. Various methods exist to wake up the microcontroller, one of which is to change the logic condition on one of the port pins from high to low or from low to high. This function is especially suitable for applications that can be woken up via external switches. Each pin on Port A~E can be selected individually to have this wake-up feature using the PxWU register. The Port A~E also can be setup to have a choice of wake-up polarity using specific registers. Each pin on Port A~E can be selected individually to have this Wake-up polarity feature using the PxWUG register.

Note that the wake-up function can be controlled by the wake-up control registers only when the pin is selected as a general purpose input and the MCU enters the IDLE or SLEEP mode.

• **PxWU Register**

Bit	7	6	5	4	3	2	1	0
Name	PxWU7	PxWU6	PxWU5	PxWU4	PxWU3	PxWU2	PxWU1	PxWU0
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

PxWUn: I/O Port x.n Pin wake-up function control
0: Disable
1: Enable

The PxWUn bit is used to control the Px.n pin wake-up function. Here the "x" can be A, B, C, D or E. However, the actual available bits for each I/O Port may be different.

• **PxWUG Register**

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

Bit 7~1 Unimplemented, read as "0"

Bit 0 **PxWUGS**: I/O Port x wake-up edge control
0: Rising edge trigger
1: Falling edge trigger

The PxWUGS bit is used to control the Px port wake-up polarity function. Here the "x" can be A, B, C, D or E.

I/O Port Control Register

Each I/O Port has its own control register known as PAC~PEC, to control the input/output configuration. With this control register, each CMOS output or input can be reconfigured dynamically under software control. Each pin of the I/O ports is directly mapped to a bit in its associated port control register. For the I/O pin to function as an input, the corresponding bit of the control register must be written as a "1". This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a "0", the I/O pin will be setup as a CMOS output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

• **PxC Register**

Bit	7	6	5	4	3	2	1	0
Name	PxC7	PxC6	PxC5	PxC4	PxC3	PxC2	PxC1	PxC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

PxCn: I/O Port x Pin type selection
0: Output
1: Input
The PxCn bit is used to control the pin type selection. Here the "x" is the Port name which can be A, B, C, D or E. However, the actual available bits for each I/O Port may be different.

I/O Port Output Slew Rate Control Register

The I/O Ports have a control register known as SLEWC, to control the output slew rate configuration. The slew rate control bit is available when the corresponding pin is configured as a CMOS output. Otherwise, the control bit has no effect.

• **SLEWC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	—	SLEWC0
R/W	—	—	—	—	—	—	—	R/W
POR	—	—	—	—	—	—	—	0

Bit 7~1 Unimplemented, read as "0"
Bit 0 **SLEWC0**: I/O port output slew rate control
0: Disable
1: Enable

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 these pins, the desired function of the multi-function I/O pins is selected by a series of registers via the application program control.

Pin-shared Function Selection Registers

The limited number of supplied pins in a package can impose restrictions on the amount of functions a certain device can contain. However by allowing the same pins to share several different functions and providing a means of function selection, a wide range of different functions can be incorporated into even relatively small package sizes. The device includes Port "x" output function Selection register "n", labeled as PxSn, which can select the desired functions of the multi-function pin-shared pins.

The most important point to note is to make sure that the desired pin-shared function is properly selected and also deselected. For most pin-shared functions, to select the desired pin-shared function, the pin-shared function should first be correctly selected using the corresponding pin-shared control register. After that the corresponding peripheral functional setting should be configured and then the peripheral function can be enabled. However, a special point must be noted for some digital input pins, such as TCn, etc., which share the same pin-shared control configuration with their corresponding general purpose I/O functions when setting the relevant pin-shared control bit fields. To select these pin functions, in addition to the necessary pin-shared control and peripheral functional setup aforementioned, they must also be setup as input by setting the corresponding bit in the I/O port control register. To correctly deselect the pin-shared function, the peripheral function should first be disabled and then the corresponding pin-shared function control register can be modified to select other pin-shared functions.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PAS0	PAS07	PAS06	PAS05	PAS04	PAS03	PAS02	PAS01	PAS00
PAS1	PAS17	PAS16	PAS15	PAS14	PAS13	PAS12	PAS11	PAS10
PBS0	PBS07	PBS06	PBS05	PBS04	PBS03	PBS02	PBS01	PBS00
PBS1	PBS17	PBS16	PBS15	PBS14	PBS13	PBS12	PBS11	PBS10
PCS0	PCS07	PCS06	PCS05	PCS04	PCS03	PCS02	PCS01	PCS00
PCS1	PCS17	PCS16	PCS15	PCS14	PCS13	PCS12	PCS11	PCS10
PDS0	PDS07	PDS06	PDS05	PDS04	PDS03	PDS02	PDS01	PDS00
PDS1	PDS17	PDS16	PDS15	PDS14	PDS13	PDS12	PDS11	PDS10
PES0	—	—	—	—	PES03	PES02	PES01	PES00

Pin-shared Function Selection Register List

- **PAS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PAS07	PAS06	PAS05	PAS04	PAS03	PAS02	PAS01	PAS00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PAS07~PAS06**: PA3 pin-shared function selection
00: PA3
01: PA3
10: PA3
11: Reserved, cannot be used

Bit 5~4 **PAS05~PAS04**: PA2 pin-shared function selection
00: PA2
01: PA2
10: PA2
11: Reserved, cannot be used

Bit 3~2 **PAS03~PAS02**: PA1 pin-shared function selection
00: PA1
01: PA1
10: PA1
11: Reserved, cannot be used

Bit 1~0 **PAS01~PAS00**: PA0 pin-shared function selection
00: PA0
01: PA0
10: PA0
11: Reserved, cannot be used

• **PAS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PAS17	PAS16	PAS15	PAS14	PAS13	PAS12	PAS11	PAS10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PAS17~PAS16**: PA7 pin-shared function selection
00: PA7/TC1
01: PA7/TC1
10: PA7/TC1
11: Reserved, cannot be used

Bit 5~4 **PAS15~PAS14**: PA6 pin-shared function selection
00: PA6/TC0
01: PA6/TC0
10: PA6/TC0
11: Reserved, cannot be used

Bit 3~2 **PAS13~PAS12**: PA5 pin-shared function selection
00: PA5
01: PA5
10: PA5
11: Reserved, cannot be used

Bit 1~0 **PAS11~PAS10**: PA4 pin-shared function selection
00: PA4
01: PA4
10: PA4
11: Reserved, cannot be used

• **PBS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PBS07	PBS06	PBS05	PBS04	PBS03	PBS02	PBS01	PBS00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PBS07~PBS06**: PB3 pin-shared function selection
00: PB3
01: PB3
10: PB3
11: Reserved, cannot be used

Bit 5~4 **PBS05~PBS04**: PB2 pin-shared function selection
00: PB2
01: PB2
10: PB2
11: Reserved, cannot be used

Bit 3~2 **PBS03~PBS02**: PB1 pin-shared function selection
00: PB1
01: PB1
10: PB1
11: Reserved, cannot be used

Bit 1~0 **PBS01~PBS00**: PB0 pin-shared function selection
00: PB0
01: PB0
10: PB0
11: Reserved, cannot be used

• **PBS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PBS17	PBS16	PBS15	PBS14	PBS13	PBS12	PBS11	PBS10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PBS17~PBS16**: PB7 pin-shared function selection
00: PB7
01: PB7
10: PB7
11: Reserved, cannot be used

Bit 5~4 **PBS15~PBS14**: PB6 pin-shared function selection
00: PB6
01: PB6
10: PB6
11: Reserved, cannot be used

Bit 3~2 **PBS13~PBS12**: PB5 pin-shared function selection
00: PB5
01: PB5
10: PB5
11: Reserved, cannot be used

Bit 1~0 **PBS11~PBS10**: PB4 pin-shared function selection
00: PB4
01: PB4
10: PB4
11: Reserved, cannot be used

• **PCS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PCS07	PCS06	PCS05	PCS04	PCS03	PCS02	PCS01	PCS00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PCS07~PCS06**: PC3 pin-shared function selection
00: PC3
01: PC3
10: PC3
11: Reserved, cannot be used

Bit 5~4 **PCS05~PCS04**: PC2 pin-shared function selection
00: PC2
01: PC2
10: PC2
11: Reserved, cannot be used

Bit 3~2 **PCS03~PCS02**: PC1 pin-shared function selection
00: PC1
01: PC1
10: PC1
11: Reserved, cannot be used

Bit 1~0 **PCS01~PCS00**: PC0 pin-shared function selection
00: PC0
01: PC0
10: PC0
11: Reserved, cannot be used

• **PCS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PCS17	PCS16	PCS15	PCS14	PCS13	PCS12	PCS11	PCS10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PCS17~PCS16**: PC7 pin-shared function selection
00: PC7
01: PC7
10: PC7
11: Reserved, cannot be used

Bit 5~4 **PCS15~PCS14**: PC6 pin-shared function selection
00: PC6
01: PC6
10: PC6
11: Reserved, cannot be used

Bit 3~2 **PCS13~PCS12**: PC5 pin-shared function selection
00: PC5
01: PC5
10: PC5
11: Reserved, cannot be used

Bit 1~0 **PCS11~PCS10**: PC4 pin-shared function selection
00: PC4
01: PC4
10: PC4
11: Reserved, cannot be used

• **PDS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PDS07	PDS06	PDS05	PDS04	PDS03	PDS02	PDS01	PDS00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PDS07~PDS06**: PD3 pin-shared function selection
00: PD3
01: PD3
10: PD3
11: Reserved, cannot be used

Bit 5~4 **PDS05~PDS04**: PD2 pin-shared function selection
00: PD2
01: PD2
10: PD2
11: Reserved, cannot be used

Bit 3~2 **PDS03~PDS02**: PD1 pin-shared function selection
00: PD1
01: PD1
10: PD1
11: Reserved, cannot be used

Bit 1~0 **PDS01~PDS00**: PD0 pin-shared function selection
00: PD0
01: PD0
10: PD0
11: Reserved, cannot be used

• **PDS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PDS17	PDS16	PDS15	PDS14	PDS13	PDS12	PDS11	PDS10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PDS17~PDS16**: PD7 pin-shared function selection
00: PD7
01: PD7
10: PD7
11: Reserved, cannot be used

Bit 5~4 **PDS15~PDS14**: PD6 pin-shared function selection
00: PD6
01: PD6
10: PD6
11: Reserved, cannot be used

Bit 3~2 **PDS13~PDS12**: PD5 pin-shared function selection
00: PD5
01: PD5
10: PD5
11: Reserved, cannot be used

Bit 1~0 **PDS11~PDS10**: PD4 pin-shared function selection
00: PD4
01: PD4
10: PD4
11: Reserved, cannot be used

• **PES0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	PES03	PES02	PES01	PES00
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

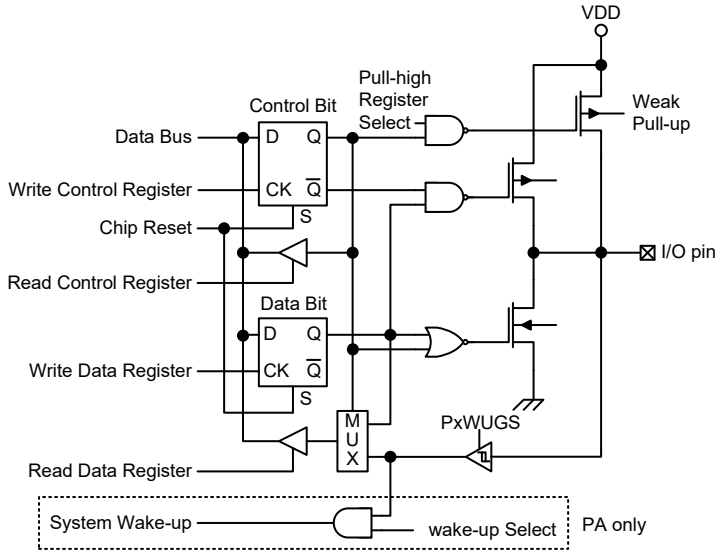
Bit 7~4 Unimplemented, read as "0"

Bit 3~2 **PES03~PES02**: PE1 pin-shared function selection
00: PE1
01: PE1
10: PE1
11: Reserved, cannot be used

Bit 1~0 **PES01~PES00**: PE0 pin-shared function selection
00: PE0
01: PE0
10: Reserved, cannot be used
11: Reserved, cannot be used

I/O Pin Structures

The accompanying diagram illustrates the internal structure of the I/O logic function. As the exact logical construction of the I/O pin will differ from this drawing, it is supplied as a guide only to assist with the functional understanding of the I/O logic function. The wide range of pin-shared structures does not permit all types to be shown.



Logic Function Input/Output Structure

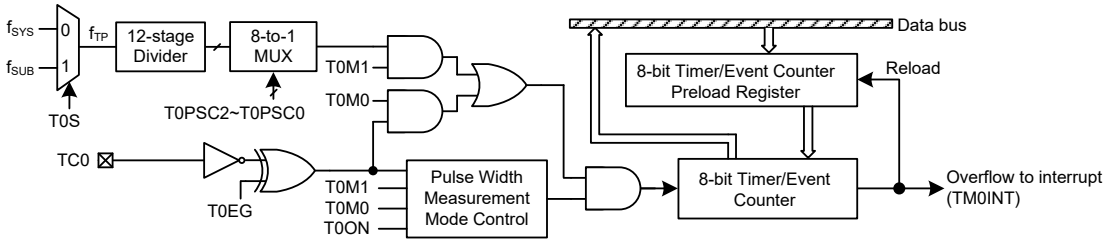
Programming Considerations

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

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

8-bit Timer/Event Counter

The provision of the Timer/Event Counter forms an important part of any microcontroller, giving the designer a means of carrying out time related functions. The device contains an 8-bit Timer/Event Counter, which contains an 8-bit programmable count-up counter and the clock may come from an external or internal clock source. As the timer has three different operating modes, it can be configured to operate as a general timer, an external event counter or a pulse width measurement device.



8-bit Timer/Event Counter 0 Structure

Timer/Event Counter Input Clock Source

The Timer/Event Counter clock source can originate from various sources, an internal clock or an external pin. The internal clock source is used when the timer is in the Timer Mode and Pulse Width Measurement Mode. For the Timer/Event Counter, this internal clock source can be configured by the T0S bit in the TMR0C Timer Control Register to be derived from the f_{SYS} or f_{SUB} clock, the division ratio of which is selected by the T0PSC2~T0PSC0 bits in the TMR0C Timer/Event Control Register.

An external clock source is used when the Timer/Event Counter is in the Event Counter Mode, the clock source is provided on the external TC0 pin. Depending upon the condition of the T0EG bit, each high to low or low to high transition on the external timer pin will increase the counter by one.

Timer/Event Counter Registers

There are two registers related to the Timer/Event Counter. The first is the TMR0 register that contains the actual value of the timer and into which an initial value can be preloaded. Writing to the TMR0 register will transfer the specified data to the Timer/Event Counter. Reading the TMR0 register will read the contents of the Timer/Event Counter. The second is the TMR0C control register, which is used to define the operating mode, select the internal clock source, control the counting enable or disable and select the active edge.

Register Name	Bit							
	7	6	5	4	3	2	1	0
TMR0	D7	D6	D5	D4	D3	D2	D1	D0
TMR0C	T0M1	T0M0	T0S	T0ON	T0EG	T0PSC2	T0PSC1	T0PSC0

Timer/Event Counter Register List

Timer Register – TMR0

The timer register TMR0 is the place where the actual timer value is stored. The value in the timer register increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFH for the 8-bit Timer/Event Counter, at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be loaded with the preload register value and continue counting.

Note that to achieve a maximum full range count of FFH, the preload register must first be cleared.

Note that if the Timer/Event Counter is in an off condition and data is written to its preload register, this data will be immediately written into the actual counter. However, if the counter is enabled and counting, any new data written into the preload data register during this period will remain in the preload register and will only be written into the actual counter until an overflow occurs.

• **TMR0 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	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: Timer preload register byte

Timer Control Register – TMR0C

The flexible features of the Holtek microcontroller Timer/Event Counter are implemented by operating in three different modes, the options of which are determined by the contents of control register bits.

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

To select which of the three modes the timer is to operate in, namely the Timer Mode, the Event Counter Mode or the Pulse Width Measurement Mode, the T0M1~T0M0 bits in the Timer Control Register must be set to the required logic levels. The timer-on bit T0ON provides the basic on/off control of the respective timer. Setting the bit to high allows the counter to run. Clearing the bit stops the counter. When the internal clock source is used, it can be sourced from the f_{SYS} or f_{SUB} clock selected by setting the T0S bit. Bits T0PSC2~T0PSC0 determine the division ratio of the selected clock source. The internal clock selection will have no effect if an external clock source is used. If the timer is in the Event Counter or Pulse Width Measurement Mode, the active transition edge type is selected by the logic level of the T0EG bit in the Timer Control Register.

• **TMR0C Register**

Bit	7	6	5	4	3	2	1	0
Name	T0M1	T0M0	T0S	T0ON	T0EG	T0PSC2	T0PSC1	T0PSC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	1	0	0	0

Bit 7~6 **T0M1~T0M0**: Timer/Event Counter operating mode selection
- 00: Unused
- 01: Event Counter Mode
- 10: Timer Mode
- 11: Pulse Width Measurement Mode

Bit 5 **T0S**: Timer f_{TP} clock source selection
- 0: f_{SYS}
- 1: f_{SUB}

Bit 4 **T0ON**: Timer/Event Counter counting enable
- 0: Disable
- 1: Enable

Bit 3 **T0EG**: Timer/Event Counter active edge selection

Event Counter Mode
- 0: Count on rising edge
- 1: Count on falling edge

Pulse Width Measurement Mode
0: Start counting on falling edge, stop on rising edge
1: Start counting on rising edge, stop on falling edge

Bit 2~0 **T0PSC2~T0PSC0**: Timer/Event Counter 0 prescaler rate selection
000: f_{TP}
001: $f_{TP}/2$
010: $f_{TP}/4$
011: $f_{TP}/8$
100: $f_{TP}/16$
101: $f_{TP}/32$
110: $f_{TP}/64$
111: $f_{TP}/128$

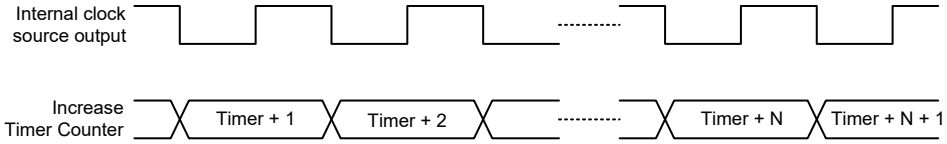
Timer/Event Counter Operating Modes

The Timer/Event Counter can operate in one of three operating modes, Timer Mode, Event Counter Mode or Pulse Width Measurement Mode. The operating mode is selected using the T0M1 and T0M0 bits in the TMR0C register.

Timer Mode

To select this mode, bits T0M1 and T0M0 in the TMR0C register should be set to "10" respectively. 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.

When operating in this mode the internal clock f_{TP} is used as the timer clock, which can be selected to be devirved from f_{SYS} or f_{SUB} by setting the T0S bit in the TMR0C register. The division of the f_{TP} clock is selected by the T0PSC2~T0PSC0 bits in the same register. The timer-on bit T0ON must be set high to enable the timer to run. Each time an internal clock high to low transition occurs, the timer increases by one. It should be noted that in the Timer mode, even if the device is in the IDLE/SLEEP mode, if the selected internal clock is still activated the timer will continue to count. When the timer reaches its maximum 8-bit, FFH Hex, value and overflows, an interrupt request is generated and the timer will reload the value already loaded into the preload register and continue counting. A timer overflow condition will trigger the corresponding internal interrupt request, with the T0F flag being set, which is one of wake-up sources.



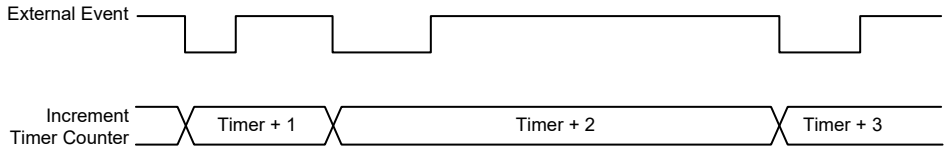
Timer Mode Timing Chart

Event Counter Mode

To select this mode, bits T0M1 and T0M0 in the TMR0C register should be set to "01" respectively. In this mode, a number of externally changing logic events, occurring on the external timer TC0 pin, can be recorded by the Timer/Event Counter.

When operating in this mode, the external timer pin, TC0, is used as the Timer/Event Counter clock source. After the other bits in the Timer Control Register have been properly configured, the enable bit T0ON, can be set high to enable the Timer/Event Counter. If the Active Edge Selection bit, T0EG, is low, the Timer/Event Counter will increase each time the TC0 pin receives a low to high transition. If the T0EG bit is high, the counter will increase each time the TC0 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.

As the external timer pin TC0 is pin-shared with other pin functions, the TC0 pin function must first be selected using relevant pin-shared function selection bits. The pin must also be set as an input by setting the corresponding bit in the port control register. It should be noted that in the Event Counter mode, even if the device is in the IDLE/SLEEP Mode, the Timer/Event Counter will continue to record externally changing logic events on the TC0 pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.



Event Counter Mode Timing Chart (T0EG=1)

Pulse Width Measurement Mode

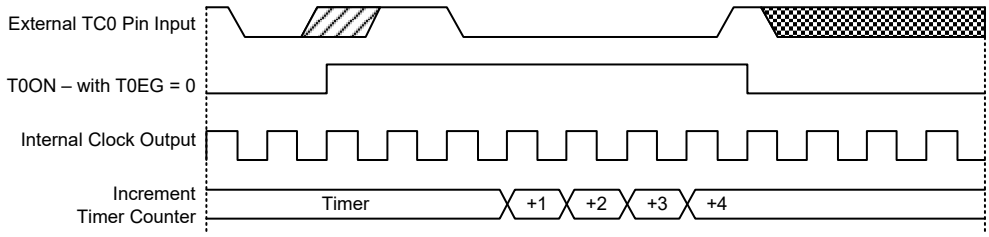
To select this mode, bits T0M1 and T0M0 in the TMR0C register should be set to "11" respectively. In this mode, the Timer/Event Counter can be utilised to measure the width of external pulses applied to the external timer pin.

When operating in this mode the internal clock f_{TP} is used as the timer clock, which can be selected to be devirved from f_{SYS} or f_{SUB} by setting the T0S bit in the TMR0C register. The division of the f_{TP} clock is selected by the T0PSC2~T0PSC0 bits in the same register. After the other bits in the Timer Control Register have been properly configured, the enable bit T0ON, 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 TC0 pin.

If the active Edge Selection bit T0EG is low, once a high to low transition has been received on the TC0 pin, the Timer/Event Counter will start counting based on the selected internal clock source until the TC0 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 Selection bit is high, the Timer/Event Counter will begin counting once a low to high transition has been received on the TC0 pin and stop counting when the external timer pin returns to its original low level. As before, the enable bit will then be automatically reset to zero. 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 TC0 pin returns to its original level, whereas in the other two modes the enable bit can only be reset to zero under application 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 TC0 pin. As the enable bit has now been reset, any further transitions on the external timer pin will be ignored. The timer cannot begin further pulse width measurement until the enable bit is set high again by the application program. In this way, single shot pulse measurements can be easily made. It should be noted that in this mode the Timer/Event Counter is controlled by logical transitions on the external timer pin and not by the logic level. When the Timer/Event Counter is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting.

The external timer pin TC0 is pin-shared with other functions, the TC0 pin function must first be setup using relevant pin-shared function selection register and should also be set as an input by setting the corresponding bit in the port control register. It should be noted that in the Pulse Width Measurement mode, even if the device is in the IDLE/SLEEP Mode, the Timer/Event Counter will continue to increase if the selected internal clock source is still activated and the required logical transitions occur on the external TC0 pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.



Pulse Width Measurement Mode Timing Chart (T0EG=0)

Programming Considerations

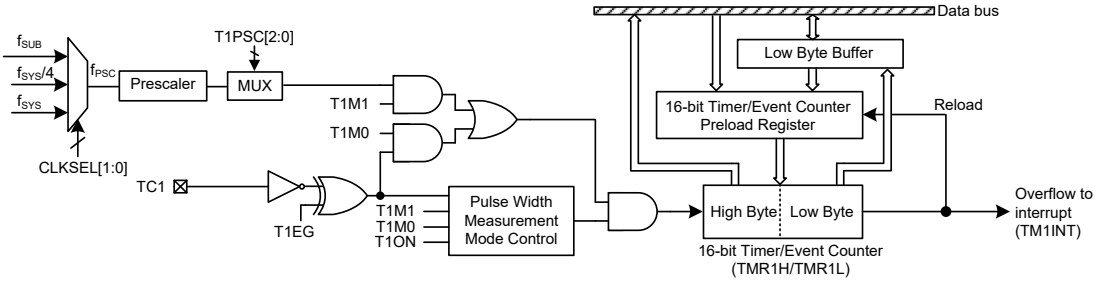
When running in the Timer Mode, if the internal system clock is used as the timer clock source, the timer can be synchronised with the overall operation of the microcontroller. In this mode when the timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the Pulse Width Measurment Mode, the internal 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 errors in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to operate in the Event Counter Mode, which again is an external event and not synchronised with the internal 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, it 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 interrupt enable bit in the interrupt control register must be properly set otherwise the internal interrupt associated with the timer will remain inactive. The active edge selection, timer operating mode selection and clock source control bits in timer control register must also be correctly issued to ensure the timer is properly configured for the required applications. It is also important to ensure that a desired initial value is first loaded into the timer register before the timer is switched on. After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the timer control register.

When the Timer/Event Counter overflows, its corresponding interrupt request flag in the interrupt control register will be set to generate an interrupt signal. If the Timer/Event Counter interrupt is enabled this will in turn allow program branch to its interrupt vector. However irrespective of whether the interrupt is enabled or not, a Timer/Event Counter overflow will also generate a wake-up signal if the device is in the IDLE/SLEEP mode. This situation may occur if the Timer/Event Counter internal clock source is still activated or if the external signal continues to change state. In such cases, the Timer/Event Counter will continue to count and if an overflow occurs the device will be woken up. 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 IDLE/SLEEP mode.

16-bit Timer/Event Counter

The provision of the Timer/Event Counter forms an important part of any microcontroller, giving the designer a means of carrying out time related functions. The device contains a 16-bit Timer/Event Counter, which contains a 16-bit programmable count-up counter and the clock may come from an external or internal clock source. As the Timer/Event Counter has three different operating modes, it can be configured to operate as a general timer, an external event counter or a pulse width measurement device. The provision of an internal prescaler to the clock circuitry gives added range to the timer.



16-bit Timer/Event Counter 1 Structure

Timer/Event Counter Input Clock Source

The Timer/Event Counter clock source can originate from various sources, an internal clock or an external pin. The internal clock source is used when the timer is in the Timer Mode and Pulse Width Measurement Mode. For the Timer/Event Counter, this internal clock source is sourced from a prescaler, the division ratio of which is configured by the T1PSC2~T1PSC0 bits in the TMR1C Timer/Event Control Register.

An external clock source is used when the Timer/Event Counter is in the Event Counter Mode, the clock source is provided on the external TC1 pin. Depending upon the condition of the T1EG bit, each high to low or low to high transition on the external timer pin will increase the counter by one.

Timer/Event Counter Registers

There are two types of registers related to the Timer/Event Counter. The first is the TMR1L/TMR1H register pair that contains the actual value of the timer and into which an initial value can be preloaded. Writing to the TMR1L/TMR1H register will transfer the specified data to the Timer/Event Counter. Reading the TMR1L/TMR1H register will read the contents of the Timer/Event Counter. The second is the TMR1C control register, which is used to define the operating mode, control the counting enable or disable and select the active edge.

Register Name	Bit							
	7	6	5	4	3	2	1	0
TMR1C	T1M1	T1M0	—	T1ON	T1EG	T1PSC2	T1PSC1	T1PSC0
TMR1L	D7	D6	D5	D4	D3	D2	D1	D0
TMR1H	D15	D14	D13	D12	D11	D10	D9	D8

Timer/Event Counter Register List

Timer Register – TMR1L, TMR1H

As the timer/event counter contains an internal 16-bit timer, it requires two data registers to store the value. There are a high byte register, known as TMR1H, and a low byte register, known as TMR1L. The value in the timer register pair increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value

loaded by the preload register to the full count of FFFFH, at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reloaded with the initial preload register value and continue counting.

Note that to achieve a maximum full counting range of FFFFH, the preload register must first be cleared. If the timer/event counter is in an OFF condition and data is written to its preload register, this data will be immediately written into the actual counter. However, if the counter is enabled and counting, any new data written into the preload data register during this period will be remained in the preload register and will not be written into the actual counter until an overflow occurs.

Writing to the TMR1L register will only write the data into an internal lower byte buffer while writing to the TMR1H register will transfer the high byte data and the contents of the lower byte buffer into the TMR1H and TMR1L registers respectively. Therefore the timer/event counter preload register is changed by each write operation to the TMR1H register. Reading from the TMR1H register will latch the contents of the TMR1H and TMR1L counters to the destination and the lower byte buffer respectively, while reading from TMR1L will read the contents of the lower byte buffer.

- **TMR1L 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	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: Timer preload register low byte

- **TMR1H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: Timer preload register high byte

Timer Control Register – TMR1C

The Timer Control Register is known as TMR1C. It is the Timer Control Register together with its corresponding timer register that controls the full operation of the Timer/Event Counter. Before the timer can be used, it is essential that the Timer Control Register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialisation.

To select which of the three modes the timer is to operate in, namely the Timer Mode, the Event Counter Mode or the Pulse Width Measurement Mode, the T1M1~T1M0 bits in the Timer Control Register must be set to the required logic levels. The timer-on bit T1ON provides the basic on/off control of the respective timer. Setting the bit to high allows the counter to run. Clearing the bit stops the counter. When the internal clock f_{PSC} is used, its division ratio can be selected by properly setting the T1PSC2~T1PSC0 bits. The internal clock selection will have no effect if an external clock source is used. If the timer is in the Event Counter or Pulse Width Measurement Mode, the active transition edge type is selected by the logic level of the T1EG bit in the Timer Control Register.

• **TMR1C Register**

Bit	7	6	5	4	3	2	1	0
Name	T1M1	T1M0	—	T1ON	T1EG	T1PSC2	T1PSC1	T1PSC0
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	0	0	—	0	1	0	0	0

Bit 7~6 **T1M1~T1M0**: Timer/Event Counter operating mode selection
00: Unused
01: Event Counter Mode
10: Timer Mode
11: Pulse Width Measurement Mode

Bit 5 Unimplemented, read as "0"

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

Bit 3 **T1EG**: Timer/Event Counter active edge selection
Event Counter Mode
0: Count on rising edge
1: Count on falling edge
Pulse Width Measurement Mode
0: Start counting on falling edge, stop on rising edge
1: Start counting on rising edge, stop on falling edge

Bit 2~0 **T1PSC2~T1PSC0**: Timer/Event Counter 1 prescaler rate selection
000: f_{PSC}
001: $f_{PSC}/2$
010: $f_{PSC}/4$
011: $f_{PSC}/8$
100: $f_{PSC}/16$
101: $f_{PSC}/32$
110: $f_{PSC}/64$
111: $f_{PSC}/128$

• **PSCR Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	CLKSEL1	CLKSEL0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **CLKSEL1~CLKSEL0**: Prescaler clock source f_{PSC} selection
00: f_{SYS}
01: $f_{SYS}/4$
1x: f_{SUB}

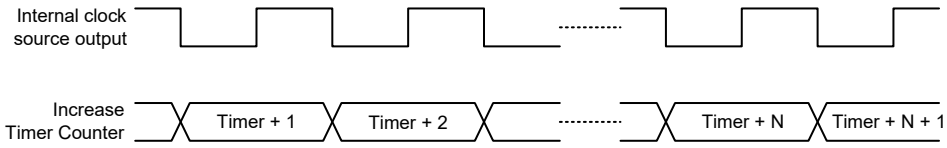
Timer/Event Counter Operating Modes

The Timer/Event Counter can operate in one of three operating modes, Timer Mode, Event Counter Mode or Pulse Width Measurement Mode. The operating mode is selected using the T1M1 and T1M0 bits in the TMR1C register.

Timer Mode

To select this mode, bits T1M1 and T1M0 in the TMR1C register should be set to "10" respectively. 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.

When operating in this mode the internal clock f_{PSC} is used as the timer clock. The clock source is from the Prescaler, and the clock division ratio is selected by the T1PSC2~T1PSC0 bits in the TMR1C register. The timer-on bit T1ON must be set high to enable the timer to run. Each time an internal clock high to low transition occurs, the timer increases by one. It should be noted that in the Timer mode, even if the device is in the IDLE/SLEEP mode, if the selected internal clock is still activated the timer will continue to count. When the timer reaches its maximum 16-bit, FFFF Hex, value and overflows, an interrupt request is generated and the timer will reload the value already loaded into the preload register and continue counting. A timer overflow condition will trigger the corresponding internal interrupt request, with the T1F flag being set, which is one of wake-up sources.



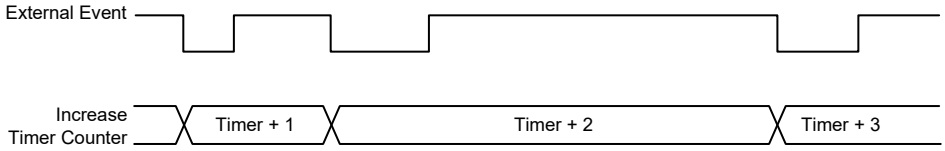
Timer Mode Timing Chart

Event Counter Mode

To select this mode, bits T1M1 and T1M0 in the TMR1C register should be set to "01" respectively. In this mode, a number of externally changing logic events, occurring on the external timer TC1 pin, can be recorded by the Timer/Event Counter.

When operating in this mode, the external timer pin, TC1, is used as the Timer/Event Counter clock source. After the other bits in the Timer Control Register have been properly configured, the enable bit T1ON, can be set high to enable the Timer/Event Counter. If the Active Edge Selection bit, T1EG, is low, the Timer/Event Counter will increase each time the TC1 pin receives a low to high transition. If the T1EG bit is high, the counter will increase each time the TC1 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.

As the external timer pin TC1 is pin-shared with other pin functions, the TC1 pin function must first be selected using relevant pin-shared function selection bits. The pin must also be set as an input by setting the corresponding bit in the port control register. It should be noted that in the Event Counter mode, even if the device is in the IDLE/SLEEP Mode, the Timer/Event Counter will continue to record externally changing logic events on the TC1 pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.



Event Counter Mode Timing Chart (T1EG=1)

Pulse Width Measurement Mode

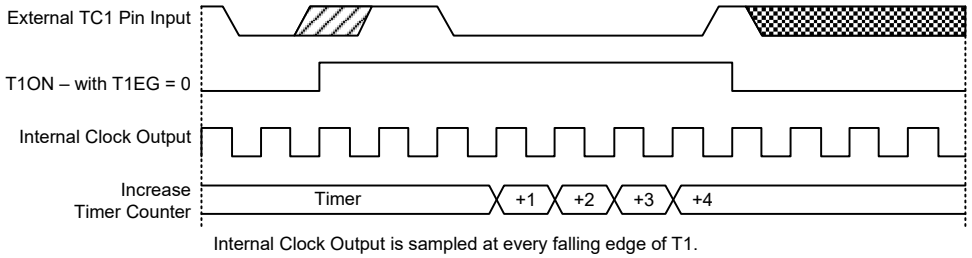
To select this mode, bits T1M1 and T1M0 in the TMR1C register should be set to "11" respectively. In this mode, the Timer/Event Counter can be utilised to measure the width of external pulses applied to the external timer pin.

When operating in this mode the internal clock f_{PSC} is used as the timer clock. The clock source is from the Prescaler, the division ratio is determined by the T1PSC2~T1PSC0 bits in the TMR1C register. After the other bits in the Timer Control Register have been properly configured, the enable bit T1ON, 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 TC1 pin.

If the active Edge Selection bit T1EG is low, once a high to low transition has been received on the TC1 pin, the Timer/Event Counter will start counting based on the internal selected clock source until the TC1 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 Selection bit is high, the Timer/Event Counter will begin counting once a low to high transition has been received on the TC1 pin and stop counting when the external timer pin returns to its original low level. As before, the enable bit will then be automatically reset to zero. 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 TC1 pin returns to its original level, whereas in the other two modes the enable bit can only be reset to zero under application 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 TC1 pin. As the enable bit has now been reset, any further transitions on the external timer pin will be ignored. The timer cannot begin further pulse width measurement until the enable bit is set high again by the application program. In this way, single shot pulse measurements can be easily made. It should be noted that in this mode the Timer/Event Counter is controlled by logical transitions on the external timer pin and not by the logic level. When the Timer/Event Counter is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting.

The external timer pin TC1 is pin-shared with other functions, the TC1 pin function must first be setup using relevant pin-shared function selection register and should also be set as an input by setting the corresponding bit in the port control register. It should be noted that in the Pulse Width Measurement mode, even if the device is in the IDLE/SLEEP Mode, the Timer/Event Counter will continue to increase if the selected internal clock source is still activated and the required logical transitions occur on the external TC1 pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.



Pulse Width Measurement Mode Timing Chart (T1EG=0)

Programming Considerations

When running in the Timer Mode, if the internal system clock is used as the timer clock source, the timer can be synchronised with the overall operation of the microcontroller. In this mode when the 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 timer 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 errors in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to operate in the Event Counter Mode, which again is an external event and not synchronised with the internal timer clock.

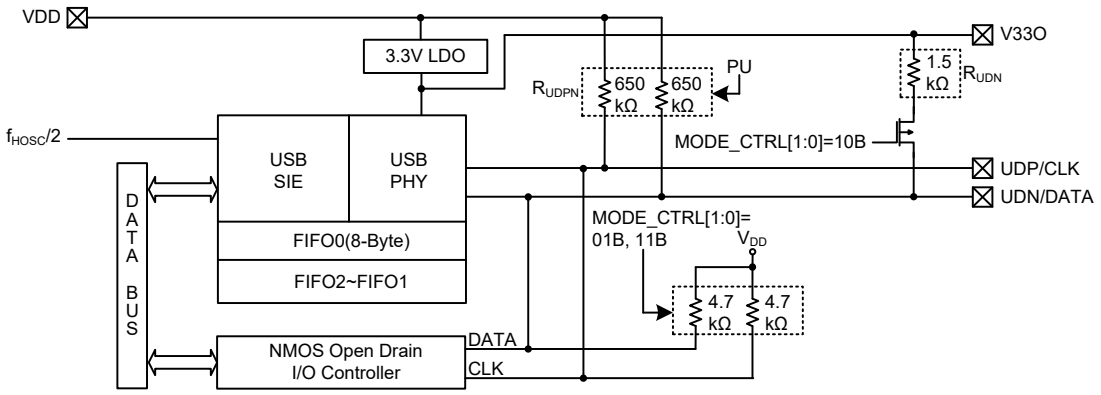
When the Timer/Event Counter is read, or if data is written to the preload register, the clock is inhibited to avoid errors, however as this may result in a counting error, it should be taken into account by the programmer. Care must be taken to ensure that the timers are properly initialised before using them for the first time. The associated timer enable bit in the interrupt control register must be properly set otherwise the internal interrupt associated with the timer will remain inactive. The active edge selection, timer operating mode selection and clock source control bits in timer control register must also be correctly issued to ensure the timer is properly configured for the required applications. It is also important to ensure that a desired initial value is first loaded into the timer register before the timer is switched on. After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the timer control register.

When the Timer/Event Counter overflows, its corresponding interrupt request flag in the interrupt control register will be set to generate an interrupt signal. If the Timer/Event Counter interrupt is enabled this will in turn allow program branch to its interrupt vector. However irrespective of whether the interrupt is enabled or not, a Timer/Event Counter overflow will also generate a wake-up signal if the device is in the IDLE/SLEEP mode. This situation may occur if the Timer/Event Counter internal clock source is still activated or if the external signal continues to change state. In such cases, the Timer/Event Counter will continue to count and if an overflow occurs the device will be woken up. 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 IDLE/SLEEP mode.

USB Interface

The USB interface is a 4-wire serial bus that allows communication between a host device and up to 127 max peripheral devices on the same bus. A token based protocol method is used by the host device for communication control. Other advantages of the USB bus include live plugging and unplugging and dynamic device configuration. As the complexity of USB data protocol does not permit comprehensive USB operation information to be provided in this datasheet, the reader should therefore consult other external information for a detailed USB understanding.

The device includes a USB interface function allowing for the convenient design of USB peripheral products.



USB Interface Block Diagram

USB Interface Operation

To communicate with an external USB host, the internal USB module has external pins known as UDP and UDN along with the 3.3V regulator output V33O. A Serial Interface Engine (SIE) decodes the incoming USB data stream and transfers it to the correct endpoint buffer memory known as the FIFO. The USB module has 3 endpoints, EP0~EP2. The endpoint 0, endpoint 1 and endpoint 2 each has 8-byte size.

USB Interface Registers

The USB function control is implemented using a series of registers. A series of status registers provide the user with the USB data transfer situation as well as any error conditions. The USB contains its own independent interrupt which can be used to indicate when the USB FIFOs are accessed by the host device or a change of the USB operating conditions including the USB suspend mode, resume event or USB reset occurs.

Register Name	Bit							
	7	6	5	4	3	2	1	0
USC	PS2_CKO	PS2_DAO	PS2_CKI	PS2_DAI	RESUME_O	URST_FLAG	RMOT_WK	SUSPEND
UISR	USB_flag	URD	MODE_CTRL1	MODE_CTRL0	PU	EP2_INT	EP1_INT	EP0_INT
UCC	—	—	PS2_FLAG	—	USBCKEN	—	D1	D0
USB_STAT	—	—	—	SE1	SE0	K_state	J_state	EOP
PIPE_CTRL	—	—	—	—	—	SETIO2	SETIO1	DATAG
AWR	AD6	AD5	AD4	AD3	AD2	AD1	AD0	WKEN
STALL	—	—	—	—	—	STL2	STL1	STL0
PIPE	—	—	—	—	—	PIPE2	PIPE1	PIPE0
SIES	NMI	D6	D5	D4	D3	D2	F0_ERR	ADR_SET
MISC	LEN0	READY	SCMD	SELP1	SELP0	CLEAR	TX	REQ
ENDPT_EN	—	—	—	—	—	EP2EN	EP1EN	EP0EN
FIFO0	D7	D6	D5	D4	D3	D2	D1	D0
FIFO1	D7	D6	D5	D4	D3	D2	D1	D0
FIFO2	D7	D6	D5	D4	D3	D2	D1	D0

USB Interface Register List

- **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	R/W	R/W	R	R	R	R/W	R/W	R
POR	1	1	x	x	0	0	0	0

"x": unknown

Bit 7 **PS2_CKO**: Output data for driving the UDP/CLK pin when the device operates in the PS2 mode

Bit 6 **PS2_DAO**: Output data for driving the UDN/DATA pin when the device operates in the PS2 mode

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

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

Bit 3 **RESUME_O**: USB resume indication
0: Resume signal is not asserted or USB device has left the suspend mode
1: Resume signal is asserted and USB device is going to leave the suspend mode
This bit is read only. When the resume event occurs, this bit will be set high by SIE and then an interrupt will also be generated to wake up the MCU. In order to detect the suspend state, the MCU should set the USBCKEN bit to 1. When the USB device leaves the suspend mode, the SUSPEND bit will be cleared to 0 and then the RESUME_O bit will also be cleared to 0. The resume signal which causes the MCU to wake up should be noted and taken into consideration when the MCU is detecting the suspend mode.

Bit 2 **URST_FLAG**: USB reset indication
0: No USB reset event occurs
1: USB reset event occurs
This bit is set and cleared by the USB SIE. This bit is used to detect which bus (PS2 or USB) is attached. When this bit is set to 1, this indicates that a USB reset has occurred (the attached bus is USB) and a USB interrupt will be initialed.

Bit 1 **RMOT_WK**: USB remote wake-up command
0: No USB remote wake-up command initiated
1: Initiate USB remote wake-up command
This bit is set to 1 by the MCU to force the USB host to leave the suspend mode. When this bit is set to 1, a 2μs delay for clearing this bit to "0" is needed to insure the remote wake-up command is accepted by SIE.

Bit 0 **SUSPEND**: USB suspend indication
0: USB leaves the suspend mode
1: USB enters the suspend mode
When this bit is set to 1 by the SIE, it indicates that the USB bus has entered the suspend mode. The USB interrupt is also triggered on any change of this bit.

• UISR Register

Bit	7	6	5	4	3	2	1	0
Name	USB_flag	URD	MODE_ CTRL1	MODE_ CTRL0	PU	EP2_INT	EP1_INT	EP0_INT
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	0	0	0	0	0	0

Bit 7 **USB_flag**: USB mode indication
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 **URD**: USB reset signal reset function control
0: USB reset signal cannot reset MCU
1: USB reset signal will reset MCU

Bit 5~4 **MODE_CTRL1~ MODE_CTRL0**: USB and PS2 mode control
00: Non-USB mode, turn off V33O, both UDP and UDN can be read and written
01: PS2 mode, V33O has no output, both UDP and UDN can be read and written
10: USB mode, there is a 1.5kΩ resistor between UDN and V33O, V33O outputs 3.3V, both UDP and UDN can be read and written
11: PS2 mode, V33O outputs 3.3V, both UDP and UDN can be read and written

Bit 3 **PU**: UDP and UDN pins pull-high function control
0: Disable – no internal pull-high resistor
1: Enable – internal 650kΩ pull-high resistor for UDP and UDN pins

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 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 software.

- **UCC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	PS2_flag	—	USBCKEN	—	D1	D0
R/W	—	—	R/W	—	R/W	—	R/W	R/W
POR	—	—	0	—	0	—	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **PS2_flag**: PS2 mode indication
0: Not PS2 mode
1: PS2 mode
This flag is used to indicate that the MCU is in the PS2 mode when it is set to 1. This bit will be cleared to 0 after power-on reset.

Bit 4 Unimplemented, read as "0"

Bit 3 **USBCKEN**: USB clock control
0: Disable
1: Enable

Bit 2 Unimplemented, read as "0"

Bit 1~0 **D1~D0**: Reserved bits, cannot be used and must be fixed at "00"

- **USB_STAT Register**

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

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

"x": unknown

Bit 7~5 Unimplemented, read as "0"

Bit 4 **SE1**: USB bus SE1 noise indication
0: No noise
1: Noise
This bit is used to indicate that the SIE has detected an SE1 noise on the USB bus. This bit is set by SIE and cleared by software.

Bit 3 **SE0**: USB bus SE0 noise indication
0: No noise
1: Noise
This bit is used to indicate that the SIE has detected an SE0 noise on the USB bus. This bit is set by SIE and cleared by software.

Bit 2 **K_state**: USB SIE K_state indication
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 software.

Bit 1 **J_state**: USB SIE J_state indication
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 software.

Bit 0 **EOP**: USB EOP indication
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 software.

• PIPE_CTRL Register

The PIPE_CTRL Register is used for configuring endpoint FIFO as an input pipe or output pipe. The default setting is the input pipe. The DATAG bit is used to set the data toggle of any endpoint (except endpoint 0) using data toggles to the DATAG value. Once the user wants any endpoint (except endpoint 0) using data toggles to the DATAG value, the user can output a LOW pulse to this bit. The LOW pulse period must be at least 10 instruction cycles long.

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

Bit 7~3 Unimplemented, read as "0"

Bit 2 **SETIO2**: Endpoint 2 FIFO control
0: Output pipe
1: Input pipe

Bit 1 **SETIO1**: Endpoint 1 FIFO control
0: Output pipe
1: Input pipe

Bit 0 **DATAG**: Data token toggle control bit
0: DATA0 will be sent first
1: DATA1 will be sent first

• AWR Register

The AWR register contains the current address and a remote wake-up function control bit.

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

"x": unknown

Bit 7~1 **AD6~AD0**: USB device address bit 6 ~ bit 0

Bit 0 **WKEN**: USB remote wake-up control
0: Disable
1: Enable

• STALL Register

The STALL register shows whether the corresponding endpoint works properly or not. As soon as an endpoint improper operation occurs, the related bit in the STALL register has to be set high by application program. The STALL register content will be cleared by a USB reset signal and a setup token event.

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

"x": unknown

Bit 7~3 Unimplemented, read as "0"

Bit 2 **STL2**: USB endpoint 2 FIFO stall indication
0: Endpoint 2 FIFO is not stalled
1: Endpoint 2 FIFO is stalled

Bit 1 **STL1**: USB endpoint 1 FIFO stall indication
0: Endpoint 1 FIFO is not stalled
1: Endpoint 1 FIFO is stalled

Bit 0 **STL0**: USB endpoint 0 FIFO stall indication
0: Endpoint 0 FIFO is not stalled
1: Endpoint 0 FIFO is stalled

• PIPE Register

The PIPE register represents whether the corresponding endpoint is accessed by the host or not. 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.

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	PIPE2	PIPE1	PIPE0
R/W	—	—	—	—	—	R	R	R
POR	—	—	—	—	—	x	x	x

"x": unknown

Bit 7 ~ 3 Unimplemented, read as "0"

Bit 2 **PIPE2**: Endpoint 2 accessed indication
0: Not accessed
1: be accessed
This register is set high only after the time when the host access the endpoint 2. It will be cleared by a USB reset.

Bit 1 **PIPE1**: Endpoint 1 accessed indication
0: Not accessed
1: be accessed
This register is set high only after the time when the host access the endpoint 1. It will be cleared by a USB reset.

Bit 0 **PIPE0**: Endpoint 0 accessed indication
0: Not accessed
1: be accessed
This register is set high only after the time when the host access the endpoint 0. It will be cleared by a USB reset.

• **SIES Register**

The SIES register is used to indicate the present signal state which the SIE receives and also control whether the SIE changes the device address automatically or not.

Bit	7	6	5	4	3	2	1	0
Name	NMI	D6	D5	D4	D3	D2	F0_ERR	ADR_SET
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	x	x	x	x	x	x	x

"x": unknown

Bit 7 **NMI**: NAK token interrupt mask control
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 **D6~D2**: Reserved bits

Bit 1 **F0_ERR**: FIFO0 accessed error indication
0: No error occurs
1: Error occurs
This bit is used to indicate that some errors have occurred when the FIFO0 is accessed. This bit is set by SIE and should be cleared by software.

Bit 0 **ADR_SET**: Device address update method control
0: Device address is immediately updated when an address is written into the AWR register
1: Device address is updated after the device IN token data have completely been read by the USB host
This bit is used to configure the SIE to automatically change the device address by the value stored in the AWR register. When this bit is set to "1" by software, the SIE will update the device address by the value stored in the AWR register after the USB host has successfully read the data from the device by an IN operation. Otherwise, when this bit is cleared to "0", the SIE will update the device address immediately after an address is written to the AWR register.

• **MISC Register**

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

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

"x": unknown

Bit 7 **LEN0**: Zero-length packet indication
0: The received packet is not zero-length packet
1: The received packet is zero-length packet
This bit is used to show whether the USB host sends a zero-length packet or not. This bit should be cleared by software.

Bit 6 **READY**: Endpoint FIFO ready indication
0: The desired endpoint FIFO is not ready
1: The desired endpoint FIFO is ready

Bit 5 **SCMD**: SETUP command indication
0: The data in the FIFO is not a SETUP token
1: The data in the FIFO is a SETUP token
This bit has to be cleared by software. 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
00: Endpoint FIFO0
01: Endpoint FIFO1
10: Endpoint FIFO2
11: Reserved

Bit 2 **CLEAR**: FIFO clear function enable control
0: Disable
1: Enable
This bit is used to clear the requested FIFO even if the corresponding FIFO is not ready.

Bit 1 **TX**: Data transfer direction control
0: MCU reads data from the USB FIFO
1: MCU writes data to the USB FIFO
This bit defines the data transfer direction between the MCU and USB endpoint FIFO. When the TX bit is set to 1, it means that the MCU wants to write data to the USB endpoint FIFO. After the MCU write operation has completed, this bit has to be cleared to 0 before terminating the FIFO request to indicate the end of the data transfer. For an MCU read operation this bit has to be cleared to 0 to indicate that the MCU wants to read data from the USB endpoint FIFO. Then this bit has to be set to 1 before terminating the FIFO request to indicate the end of the data transfer after an MCU read operation completion.

Bit 0 **REQ**: FIFO request control
0: No request or request completion
1: Request the desired FIFO
This bit is used to request an operation of the desired endpoint FIFO. After selecting the desired endpoint FIFO, the FIFO can be requested by setting this bit high. Then this bit should be cleared to zero after the operation completion.

- **ENDPT_EN Register**

The ENDPT_EN Register is used to enable or disable the corresponding endpoint.

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, read as “0”

Bit 2 **EP2EN**: USB Endpoint 2 control
0: Disable
1: Enable

Bit 1 **EP1EN**: USB Endpoint 1 control
0: Disable
1: Enable

Bit 0 **EP0EN**: USB Endpoint 0 control
0: Disable
1: Enable

• **FIFOn Registers (n=0~2)**

The FIFOn Register is used for data transaction storage between the USB device and the USB host. The MCU reads data from or writes data to the FIFOn via the specific combination of the corresponding control and selection bits.

Name	Type	POR	Description
FIFO0	R/W	xxxx xxxx	Endpoint 0 Data Pipe
FIFO1	R/W	xxxx xxxx	Endpoint 1 Data Pipe
FIFO2	R/W	xxxx xxxx	Endpoint 2 Data Pipe

"x" : unknown

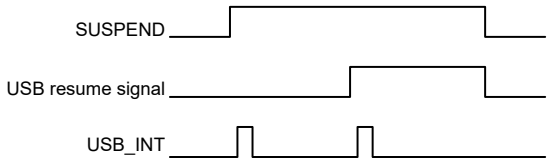
USB Suspend Mode and Wake-Up

USB Suspend Mode

If there is no signal on the USB bus for over 3ms, the USB device will enter the suspend mode. The suspend flag, SUSPEND, in the USC register will then be set high and a USB interrupt will be generated to indicate that the device should jump to the suspend state to meet the requirements of the USB suspend current specification. In order to meet the requirements of the suspend current, the software should disable the USB clock by clearing the USBCKEN bit to "0".

USB Host Wake-up

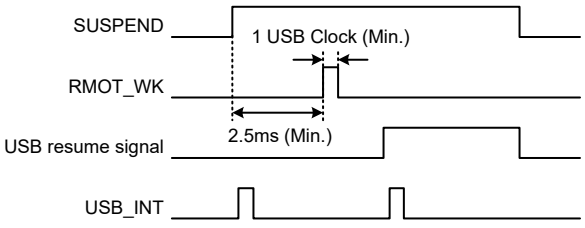
When the resume signal is asserted by the USB host, a pulse will appear on the signal and the device will be woken up by the USB interrupt and the RESUME_O bit in the USC register will be set. To ensure correct device operation, the application program should set the USBCKEN bit high and the USB host will start to communicate with the USB device. Then the SUSPEND bit will be cleared to zero together with the RESUME_O bit when the USB device actually leaves the suspend mode. Therefore, when the device detects the suspend bit, SUSPEND, the resume bit, RESUME_O, should also be monitored and taken into consideration.



Suspend and Host Wake-up

USB Remote Wake-up

As the USB device has a remote wake-up function, the USB device can wake up the USB host by sending a remote wake-up pulse which is generated by setting the RMOT_WK bit high. Once the USB host receives a remote wake-up signal from the USB device, the host will send a resume signal to device.



Suspend and Remote Wake-up

USB Interrupt

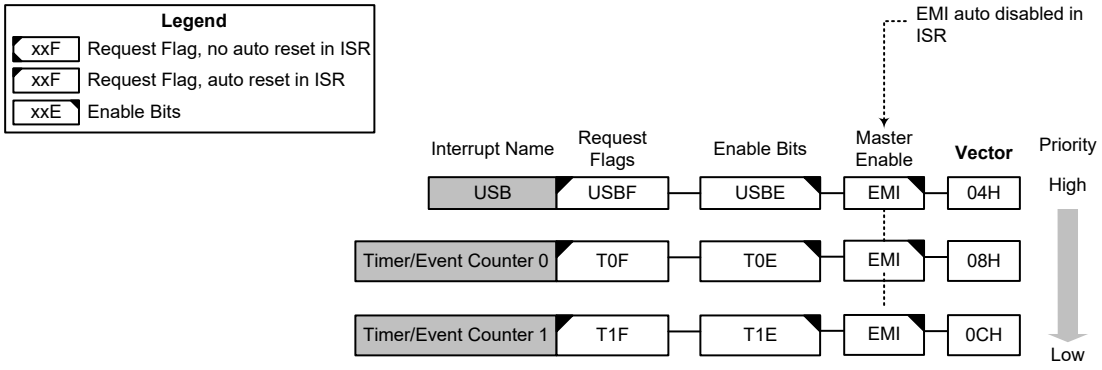
Several USB conditions can generate a USB interrupt. When one of these conditions exists, an interrupt pulse will be generated to get the attention of the microcontroller. These conditions are the USB suspended, USB resumed, USB reset and USB endpoint FIFO access events. When the USB interrupt caused by any of these conditions occurs, if the corresponding interrupt control is enabled and the stack is not full, the program will jump to the corresponding interrupt vector where it can be serviced before returning to the main program.

For the USB Endpoint FIFO access event, there are corresponding indication flags to indicate which endpoint FIFO is accessed. As the Endpoint FIFO access flag is set, it will generate a USB interrupt if the associated Endpoint FIFO pipe and interrupt control are both enabled. The Endpoint FIFO access flags should be cleared by the application program. As the USB suspended or USB resume condition occurs, the corresponding indication flag, known as SUSPEND and RESUME_O bits, will be set and a USB interrupt will directly generate without enabling the associated interrupt control bit. The SUSPEND and RESUME_O bits are read only and set or cleared by the USB SIE. For a USB interrupt occurred to be serviced, in addition to the bits for the corresponding interrupt enable control in USB module being set, the global interrupt enable control and the related interrupt enable control bits in the host MCU must also be set. If these bits are not set, then no interrupt will be serviced.

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer/Event Counter requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. The device contains three internal interrupt functions. The internal interrupts are generated by various internal functions such as the Timer/Event Counter and USB interface, etc.

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



Interrupt Structure

Interrupt Registers

Overall interrupt control, which basically means the setting of request flags when certain microcontroller conditions occur and the setting of interrupt enable bits by the application program, is controlled by a register, located in the Special Purpose Data Memory, as shown in the accompanying table. The INTC register is used to setup the interrupts.

This register contains a number of enable bits to enable or disable individual registers as well as

interrupt flags to indicate the presence of an interrupt request. The naming convention of these follows a specific pattern. First is listed an abbreviated interrupt type, then the (optional) number of that interrupt followed by either an "E" for enable/disable bit or "F" for request flag.

Function	Enable Bit	Request Flag	Notes
Global	EMI	—	—
Timer/Event Counter	TnE	TnF	n=0~1
USB	USBE	USBF	—

Interrupt Register Bit Naming Conventions

- **INTC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	T1F	T0F	USBF	T1E	T0E	USBE	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: No request
1: Interrupt request

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

Bit 4 **USBF**: USB interrupt request flag
0: No request
1: Interrupt request

Bit 3 **T1E**: Timer/Event Counter 1 interrupt control
0: Disable
1: Enable

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

Bit 1 **USBE**: USB interrupt control
0: Disable
1: Enable

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

Interrupt Operation

When the conditions for an interrupt event occur, such as a timer overflow, etc, the relevant interrupt request flag will be set. Whether the request flag actually generates a program jump to the relevant interrupt vector is determined by the condition of the interrupt enable bit. If the enable bit is set high then the program will jump to its relevant vector; if the enable bit is zero then although the interrupt request flag is set an actual interrupt will not be generated and the program will not jump to the relevant interrupt vector. The global interrupt enable bit, if cleared to zero, will disable all interrupts.

When an interrupt is generated, 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" 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”, 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.

Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the global interrupt enable bit, 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. In case of simultaneous requests, the Interrupt Structure diagram shows the priority that is applied. All of the interrupt request flags when set will wake up the device if it is in SLEEP or IDLE Mode, however to prevent a wake-up from occurring the corresponding flag should be set before the device is in SLEEP or IDLE Mode.

USB Interrupt

Several USB conditions can generate a USB interrupt. When one of specific conditions occurs, an interrupt pulse will be generated to get the attention of the microcontroller. These specific conditions are the USB suspended, USB resumed, USB reset and USB endpoint FIFO access events. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI, and USB interrupt enable bit, USBE, must first be set. When the interrupt is enabled, the stack is not full and any of aforementioned conditions are created, a subroutine call to the USB interrupt vector, will take place. When the interrupt is serviced, the USB interrupt request flag, USBF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Timer/Event Counter Interrupts

An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter request flag, TnF, is set, which occurs when the Timer/Event Counter overflows. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and the Timer/Event Counter Interrupt enable bit, TnE, must first be set. When the interrupt is enabled, the stack is not full and the Timer/Event Counter overflows, a subroutine call to its interrupt vector, will take place. When the interrupt is serviced, the Timer/Event Counter Interrupt flag, TnF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

Interrupt Wake-up Function

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

Programming Considerations

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

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

Every interrupt has the capability of waking up the microcontroller when it is in the SLEEP or IDLE Mode, the wake-up being generated when the interrupt request flag changes from low to high. If it is required to prevent a certain interrupt from waking up the microcontroller then its respective request flag should be first set high before enter SLEEP or IDLE Mode.

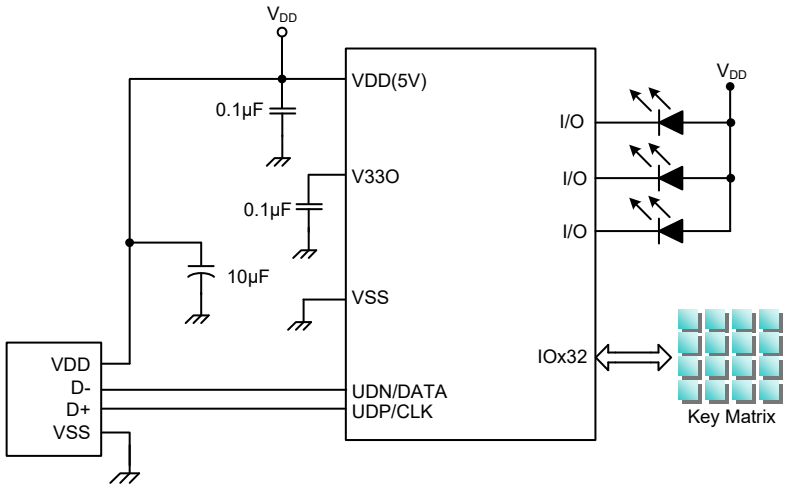
As only the Program Counter is pushed onto the stack, then when the interrupt is serviced, if the contents of the accumulator, status register or other registers are altered by the interrupt service program, their contents should be saved to the memory at the beginning of the interrupt service routine.

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

Application Descriptions

The HT82B45R is mainly used for USB low-speed keyboard applications. Most of the I/O ports are used for key matrix scanning, except for three I/O ports which are used to drive LEDs. The device communicates with the PC via the USB port.

Hardware Circuit



Keyboard Application Circuit

Instruction Set

Introduction

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

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

Instruction Timing

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

Moving and Transferring Data

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

Arithmetic Operations

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

Logical and Rotate Operation

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

Branches and Control Transfer

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

Bit Operations

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

Table Read Operations

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

Other Operations

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

Instruction Set Summary

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

Table Conventions

x: Bits immediate data
m: Data Memory address
A: Accumulator
i: 0~7 number of bits
addr: Program memory address

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
ADD A,[m]	Add Data Memory to ACC	1	Z, C, AC, OV, SC
ADDM A,[m]	Add ACC to Data Memory	1Note	Z, C, AC, OV, SC
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV, SC
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV, SC
ADCM A,[m]	Add ACC to Data memory with Carry	1Note	Z, C, AC, OV, SC
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV, SC, CZ
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV, SC, CZ
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1Note	Z, C, AC, OV, SC, CZ
SBC A,x	Subtract immediate data from ACC with Carry	1	Z, C, AC, OV, SC, CZ
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV, SC, CZ
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1Note	Z, C, AC, OV, SC, CZ
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1Note	C
Logic Operation			
AND A,[m]	Logical AND Data Memory to ACC	1	Z
OR A,[m]	Logical OR Data Memory to ACC	1	Z
XOR A,[m]	Logical XOR Data Memory to ACC	1	Z
ANDM A,[m]	Logical AND ACC to Data Memory	1Note	Z
ORM A,[m]	Logical OR ACC to Data Memory	1Note	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1Note	Z
AND A,x	Logical AND immediate Data to ACC	1	Z
OR A,x	Logical OR immediate Data to ACC	1	Z
XOR A,x	Logical XOR immediate Data to ACC	1	Z
CPL [m]	Complement Data Memory	1Note	Z
CPLA [m]	Complement Data Memory with result in ACC	1	Z
Increment & Decrement			
INCA [m]	Increment Data Memory with result in ACC	1	Z
INC [m]	Increment Data Memory	1Note	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1Note	Z
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	1Note	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	1Note	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	1Note	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	1Note	C

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

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

Extended Instruction Set

The extended instructions are used to support the full range address access for the data memory. When the accessed data memory is located in any data memory sector except sector 0, the extended instruction can be used to directly access the data memory instead of using the indirect addressing access. This can not only reduce the use of Flash memory space but also improve the CPU execution efficiency.

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
LADD A,[m]	Add Data Memory to ACC	2	Z, C, AC, OV, SC
LADDM A,[m]	Add ACC to Data Memory	2Note	Z, C, AC, OV, SC
LADC A,[m]	Add Data Memory to ACC with Carry	2	Z, C, AC, OV, SC
LADCM A,[m]	Add ACC to Data memory with Carry	2Note	Z, C, AC, OV, SC
LSUB A,[m]	Subtract Data Memory from ACC	2	Z, C, AC, OV, SC, CZ
LSUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	2Note	Z, C, AC, OV, SC, CZ
LSBC A,[m]	Subtract Data Memory from ACC with Carry	2	Z, C, AC, OV, SC, CZ
LSBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	2Note	Z, C, AC, OV, SC, CZ
LDAA [m]	Decimal adjust ACC for Addition with result in Data Memory	2Note	C
Logic Operation			
LAND A,[m]	Logical AND Data Memory to ACC	2	Z
LOR A,[m]	Logical OR Data Memory to ACC	2	Z
LXOR A,[m]	Logical XOR Data Memory to ACC	2	Z
LANDM A,[m]	Logical AND ACC to Data Memory	2Note	Z
LORM A,[m]	Logical OR ACC to Data Memory	2Note	Z
LXORM A,[m]	Logical XOR ACC to Data Memory	2Note	Z
LCPL [m]	Complement Data Memory	2Note	Z
LCPLA [m]	Complement Data Memory with result in ACC	2	Z
Increment & Decrement			
LINCA [m]	Increment Data Memory with result in ACC	2	Z
LINC [m]	Increment Data Memory	2Note	Z
LDECA [m]	Decrement Data Memory with result in ACC	2	Z
LDEC [m]	Decrement Data Memory	2Note	Z
Rotate			
LRRA [m]	Rotate Data Memory right with result in ACC	2	None
LRR [m]	Rotate Data Memory right	2Note	None
LRRCA [m]	Rotate Data Memory right through Carry with result in ACC	2	C
LRRC [m]	Rotate Data Memory right through Carry	2Note	C
LRLA [m]	Rotate Data Memory left with result in ACC	2	None
LRL [m]	Rotate Data Memory left	2Note	None
LRLCA [m]	Rotate Data Memory left through Carry with result in ACC	2	C
LRLC [m]	Rotate Data Memory left through Carry	2Note	C
Data Move			
LMOV A,[m]	Move Data Memory to ACC	2	None
LMOV [m],A	Move ACC to Data Memory	2Note	None
Bit Operation			
LCLR [m].i	Clear bit of Data Memory	2Note	None
LSET [m].i	Set bit of Data Memory	2Note	None

Mnemonic	Description	Cycles	Flag Affected
Branch			
LSZ [m]	Skip if Data Memory is zero	2Note	None
LSZA [m]	Skip if Data Memory is zero with data movement to ACC	2Note	None
LSNZ [m]	Skip if Data Memory is not zero	2Note	None
LSZ [m].i	Skip if bit i of Data Memory is zero	2Note	None
LSNZ [m].i	Skip if bit i of Data Memory is not zero	2Note	None
LSIZ [m]	Skip if increment Data Memory is zero	2Note	None
LSDZ [m]	Skip if decrement Data Memory is zero	2Note	None
LSIZA [m]	Skip if increment Data Memory is zero with result in ACC	2Note	None
LSDZA [m]	Skip if decrement Data Memory is zero with result in ACC	2Note	None
Table Read			
LTABRD [m]	Read table (specific page) to TBLH and Data Memory	3Note	None
LTABRDL [m]	Read table (last page) to TBLH and Data Memory	3Note	None
LITABRD [m]	Increment table pointer TBLP first and Read table (specific page) to TBLH and Data Memory	3Note	None
LITABRDL [m]	Increment table pointer TBLP first and Read table (last page) to TBLH and Data Memory	3Note	None
Miscellaneous			
LCLR [m]	Clear Data Memory	2Note	None
LSET [m]	Set Data Memory	2Note	None
LSWAP [m]	Swap nibbles of Data Memory	2Note	None
LSWAPA [m]	Swap nibbles of Data Memory with result in ACC	2	None

Note: 1. For these extended skip instructions, if the result of the comparison involves a skip then three cycles are required, if no skip takes place two cycles is required.

2. Any extended instruction which changes the contents of the PCL register will also require three cycles for execution.

Instruction Definition

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

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

Operation ACC ← ACC + [m] + C

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

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

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

Operation [m] ← ACC + [m] + C

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

ADD A,[m] Add Data Memory to ACC

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

Operation ACC ← ACC + [m]

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

ADD A,x Add immediate data to ACC

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

Operation ACC ← ACC + x

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

ADDM A,[m] Add ACC to Data Memory

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

Operation [m] ← ACC + [m]

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

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

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

Operation ACC ← ACC "AND" [m]

Affected flag(s) Z

AND A,x Logical AND immediate data to ACC

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

Operation ACC ← ACC "AND" x

Affected flag(s) Z

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

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

Operation [m] ← ACC "AND" [m]

Affected flag(s) Z

CALL addr Subroutine call

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

Operation Stack ← Program Counter + 1
Program Counter ← addr

Affected flag(s) None

CLR [m] Clear Data Memory

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

Operation [m] ← 00H

Affected flag(s) None

CLR [m].i Clear bit of Data Memory

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

Operation [m].i ← 0

Affected flag(s) None

CLR WDT Clear Watchdog Timer

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

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

CPL [m] Complement Data Memory

Description Each bit of the specified Data Memory is logically complemented (1′s complement).
Bits which previously contained a 1 are changed to 0 and vice versa.

Operation [m] ← [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 ← [m]

Affected flag(s) Z

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

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

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

Affected flag(s) C

DEC [m] Decrement Data Memory

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

Operation [m] ← [m] − 1

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

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

Operation ACC ← [m] − 1

Affected flag(s) Z

HALT Enter power down mode

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

Operation TO ← 0
PDF ← 1

Affected flag(s) TO, PDF

INC [m] Increment Data Memory

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

Operation [m] ← [m] + 1

Affected flag(s) Z

INCA [m] Increment Data Memory with result in ACC

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

Operation ACC ← [m] + 1

Affected flag(s) Z

JMP addr Jump unconditionally

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

Operation Program Counter ← addr

Affected flag(s) None

MOV A,[m] Move Data Memory to ACC

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

Operation ACC ← [m]

Affected flag(s) None

MOV A,x Move immediate data to ACC

Description The immediate data specified is loaded into the Accumulator.

Operation ACC ← x

Affected flag(s) None

MOV [m],A Move ACC to Data Memory

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

Operation [m] ← ACC

Affected flag(s) None

NOP No operation

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

Operation No operation

Affected flag(s) None

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

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

Operation ACC ← ACC "OR" [m]

Affected flag(s) Z

OR A,x Logical OR immediate data to ACC

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

Operation ACC ← ACC "OR" x

Affected flag(s) Z

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

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

Operation [m] ← ACC "OR" [m]

Affected flag(s) Z

RET Return from subroutine

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

Operation Program Counter ← Stack

Affected flag(s) None

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

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

Operation Program Counter ← Stack
ACC ← x

Affected flag(s) None

RETI Return from interrupt

Description The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. If an interrupt was pending when the RETI instruction is executed, the pending Interrupt routine will be processed before returning to the main program.

Operation Program Counter ← Stack
EMI ← 1

Affected flag(s) None

RL [m] Rotate Data Memory left

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

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

Affected flag(s) None

RLA [m] Rotate Data Memory left with result in ACC

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

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

Affected flag(s) None

RLC [m] Rotate Data Memory left through Carry

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

RR [m] Rotate Data Memory right

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

RRC [m] Rotate Data Memory right through Carry

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

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

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

Operation ACC ← ACC − [m] − C

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

SBC A, x Subtract immediate data from ACC with Carry

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

Operation ACC ← ACC – [m] – C

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

SBCM A,[m] Subtract Data Memory from ACC with Carry and result in Data Memory

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

Operation [m] ← ACC − [m] − C

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

SDZ [m] Skip if decrement Data Memory is 0

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

Operation [m] ← [m] − 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] − 1
Skip if ACC=0

Affected flag(s) None

SET [m] Set Data Memory

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

Operation [m] ← FFH

Affected flag(s) None

SET [m].i Set bit of Data Memory

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

Operation [m].i ← 1

Affected flag(s) None

SIZ [m] Skip if increment Data Memory is 0

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

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

SNZ [m].i Skip if 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

SNZ [m] Skip if Data Memory is not 0

Description The contents of the specified Data Memory are read out and then written back to the specified Data Memory again. If the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m]≠ 0

Affected flag(s) None

SUB A,[m] Subtract Data Memory from ACC

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

Operation ACC ← ACC − [m]

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

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

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

Operation [m] ← ACC − [m]

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

SUB A,x Subtract immediate data from ACC

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

Operation ACC ← ACC − x

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

SWAP [m] Swap nibbles of Data Memory

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

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

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

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

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

Affected flag(s) None

SZ [m] Skip if Data Memory is 0

Description The contents of the specified Data Memory are read out and then written back to the specified Data Memory again. If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

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

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

Operation Skip if [m].i=0

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

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

Operation [m] ← program code (low byte)

TBLH ← program code (high byte)

Affected flag(s) None

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

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

Operation [m] ← program code (low byte)

TBLH ← program code (high byte)

Affected flag(s) None

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

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

Operation ACC ← ACC "XOR" [m]

Affected flag(s) Z

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

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

Operation [m] ← ACC "XOR" [m]

Affected flag(s) Z

XOR A,x Logical XOR immediate data to ACC

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

Operation ACC ← ACC "XOR" x

Affected flag(s) Z

Extended Instruction Definition

The extended instructions are used to directly access the data stored in any data memory sections.

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

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

Operation ACC ← ACC + [m] + C

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

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

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

Operation [m] ← ACC + [m] + C

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

LADD A,[m] Add Data Memory to ACC

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

Operation ACC ← ACC + [m]

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

LADDM A,[m] Add ACC to Data Memory

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

Operation [m] ← ACC + [m]

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

LAND 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

LANDM 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

LCLR [m] Clear Data Memory

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

Operation [m] ← 00H

Affected flag(s) None

LCLR [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

LCPL [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] ← [m]

Affected flag(s) Z

LCPLA [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 ← [m]

Affected flag(s) Z

LDAA [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

LDEC [m] Decrement Data Memory

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

Operation [m] ← [m] − 1

Affected flag(s) Z

LDECA [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

LINC [m] Increment Data Memory

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

Operation [m] ← [m] + 1

Affected flag(s) Z

LINCA [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

LMOV 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

LMOV [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

LOR 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

LORM 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

LRL [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

LRLA [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

LRLC [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

LRLCA [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 theAccumulator 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

LRR [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

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

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

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

Affected flag(s) None

LRRC [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

LRRCA [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

LSBC 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, SC, CZ

LSBCM 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, SC, CZ

LSDZ [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 three 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

LSDZA [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 three 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

LSET [m] Set Data Memory

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

Operation [m] ← FFH

Affected flag(s) None

LSET [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

LSIZ [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 three 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

LSIZA [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 three 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

LSNZ [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 three cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m].i ≠ 0

Affected flag(s) None

LSNZ [m] Skip if Data Memory is not 0

Description The contents of the specified Data Memory are read out and then written to the specified Data Memory again. If the content 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 three cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m] ≠ 0

Affected flag(s) None

LSUB A,[m] Subtract Data Memory from ACC

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

Operation ACC ← ACC − [m]

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

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

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

Operation [m] ← ACC − [m]

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

LSWAP [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

LSWAPA [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

LSZ [m] Skip if Data Memory is 0

Description The contents of the specified Data Memory are read out and then written to the specified Data Memory again. 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 three cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation Skip if [m]=0

Affected flag(s) None

LSZA [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 three 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

LSZ [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 three 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

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

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

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

Affected flag(s) None

LTABRDL [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

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

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

LXOR 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

LXORM 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

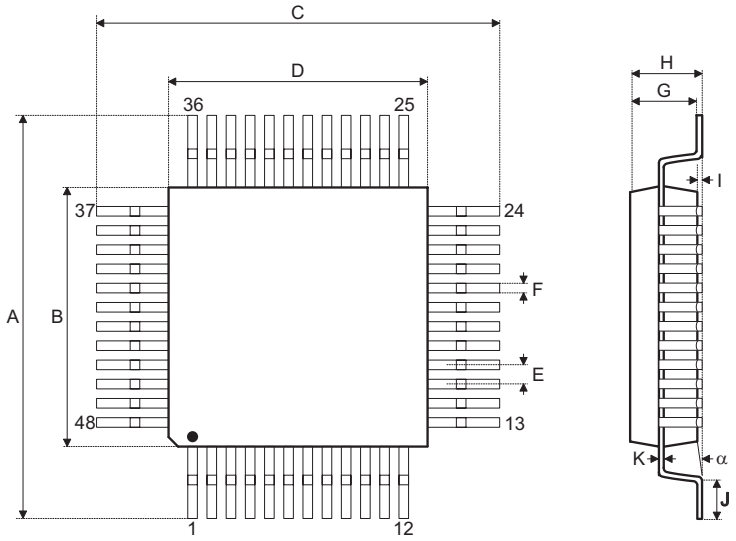
Package Information

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

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

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

48-pin LQFP (7mm×7mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A		0.354 BSC	
B		0.276 BSC	
C		0.354 BSC	
D		0.276 BSC	
E		0.020 BSC	
F	0.007	0.009	0.011
G	0.053	0.055	0.057
H	—	—	0.063
I	0.002	—	0.006
J	0.018	0.024	0.030
K	0.004	—	0.008
α	0°	—	7°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A		9.00 BSC	
B		7.00 BSC	
C		9.00 BSC	
D		7.00 BSC	
E		0.50 BSC	
F	0.17	0.22	0.27
G	1.35	1.40	1.45
H	—	—	1.60
I	0.05	—	0.15
J	0.45	0.60	0.75
K	0.09	—	0.20
α	0°	—	7°