12V High Current Driver Touch MCU

BS45F5930

Revision: V1.00 Date: February 22, 2018

www.holtek.com

Features

CPU Features

- Operating voltage
  - f_{SYS} = 8MHz: 4.75V~5.25V
  - V_{CC} = 6V~12V@±10%
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD}=5V
- Power down and wake-up functions to reduce power consumption
- Oscillator types:
  - Internal High Speed 8MHz RC – HIRC
  - Internal 32kHz RC – LIRC
- Multi-mode operation: FAST, SLOW, IDLE and SLEEP
- Fully integrated internal oscillators require no external components
- All instructions executed in one or two instruction cycles
- Table read instructions
- 63 powerful instructions
- 4-level subroutine nesting
- Bit manipulation instruction

Peripheral Features

- Flash Program Memory: 2K×16
- Data Memory: 128×8
- Watchdog Timer function
- Up to 5 bidirectional I/O lines
- Single external interrupt line shared with I/O pin
- Single 8-bit programmable Timer/Event Counter
- Single Time Base function for generation of fixed time interrupt signals
- Over Voltage Protection function
- Two high voltage driver output lines
- 4 touch key functions
- 2-channel 8-bit PWM output function
- Flash program memory can be re-programmed up to 100,000 times
- Flash program memory data retention > 10 years
- Package types: 8 /10-pin SOP

General Description

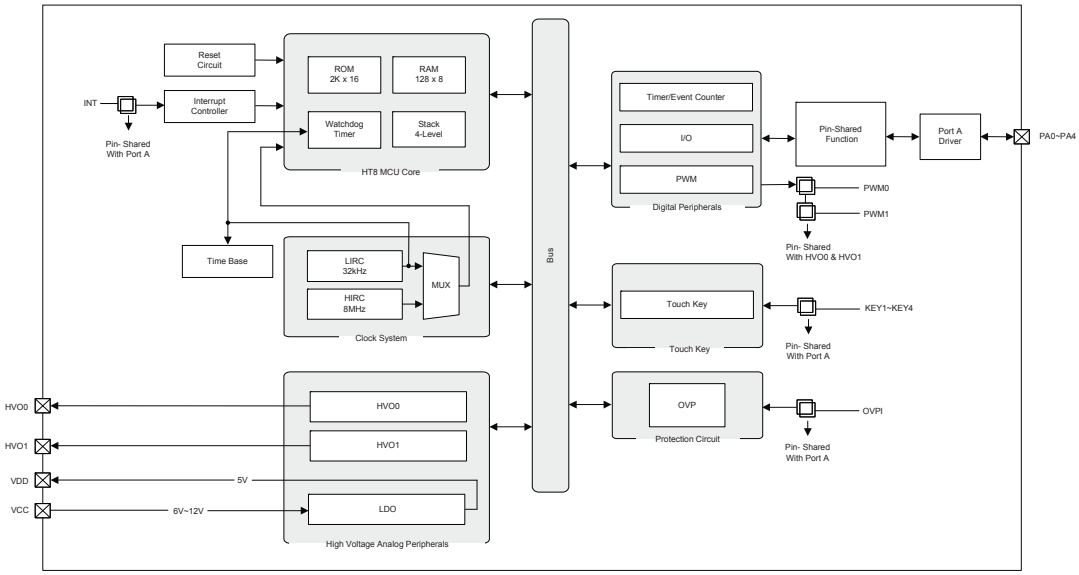
The device is a Flash Memory type 8-bit high performance RISC architecture microcontroller with a high voltage driver of up to 12V. With the touch key functions provided internally and with the convenience of Flash Memory multi-programming features, this device range has all the features to offer designers a reliable and easy means of implementing Touch Keys within their products applications.

The touch key functions are fully integrated completely eliminating the need for external components. In addition to the flash program memory, other memory includes an area of Data Memory. Protective features such as an internal Watchdog Timer and over voltage protection function coupled with excellent noise immunity and ESD protection ensure that reliable operation is maintained in hostile electrical environments.

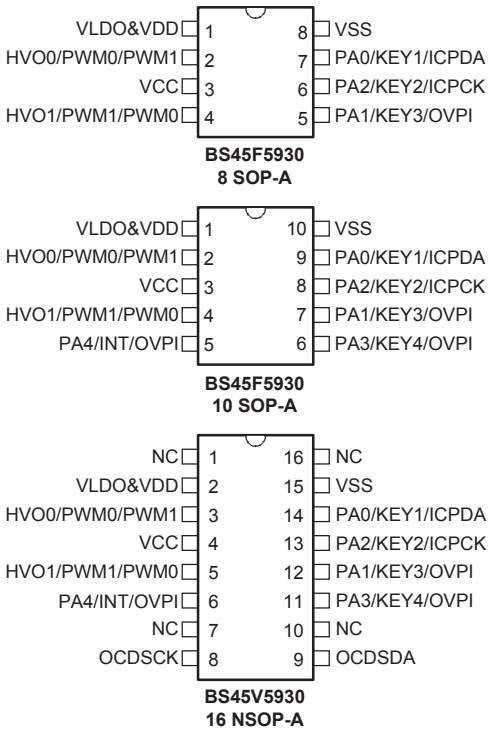
This device includes fully integrated low and high speed oscillators which require no external components for their implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources gives users the ability to optimise microcontroller operation and minimise power consumption. The inclusion of flexible I/O programming features, Time Base, Timer/Event Counter, Pulse Width Modulator and many other features further enhance device functionality and flexibility.

The touch key device will find excellent use in a huge range of modern Touch Key product applications such as touch LED dimming table lamps, touch nail phototherapy machines and various touch controlled products that require 12V high current and so on.

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 16-pin NSOP package type is only for OCDS EV chip. The OCDSDA and OCDSCK pins are the OCDS dedicated pins.

Pin Description

Pin Name	Function	OPT	I/T	O/T	Descriptions
PA0/KEY1/ICPDA	PA0	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	KEY1	PAS0 TKMC1	ST	CMOS	Touch key input
	ICPDA	—	ST	CMOS	ICP address/data
PA1/KEY3/OVPI	PA1	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	KEY3	PAS0 TKMC1	ST	CMOS	Touch key input
	OVPI	PAS0	ST	—	OVP input
PA2/KEY2/ICPCK	PA2	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	KEY2	PAS0 TKMC1	ST	CMOS	Touch key input
	ICPCK	—	ST	CMOS	ICP clock

Pin Name	Function	OPT	I/T	O/T	Descriptions
PA3/KEY4/OVPI	PA3	PAPU PAWU PAPS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	KEY4	PAS0 TKMC1	ST	CMOS	Touch key input
	OVPI	PAS0	ST	—	OVP input
PA4/INT/OVPI	PA4	PAPU PAWU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up
	INT	INTEG	ST	—	External Interrupt input
	OVPI	PAS1	ST	—	OVP input
HVO0/PWM0/ PWM1	HVO0	HVOC1	—	CMOS	High voltage driver output
	PWM0	HVOC1	—	CMOS	PWM channel 0 output
	PWM1	HVOC1	—	CMOS	PWM channel 1 output
HVO1/PWM1/ PWM0	HVO1	HVOC1	—	CMOS	High voltage driver output
	PWM1	HVOC1	—	CMOS	PWM channel 1 output
	PWM0	HVOC1	—	CMOS	PWM channel 0 output
VCC	VCC	—	PWR	—	LDO input power supply, High voltage driver output and Level Shift input
VLDO/VDD	VLDO	—	PWR	—	LDO output digital positive power supply
	VDD	—	PWR	—	Digital positive power supply
VSS	VSS	—	PWR	—	Digital negative power supply
The following pins are only for the BS45V5930					
NC	NC	—	—	—	No connection
OCDSDA	OCDSDA	—	ST	CMOS	OCDS Address/Data, for EV chip only
OCDSCK	OCDSCK	—	ST	—	OCDS Clock pin, for EV chip only

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

Absolute Maximum Ratings

Supply Voltage ..... V_{SS}−0.3V to V_{SS}+6.0V
Input Voltage ..... V_{SS}−0.3V to V_{DD}+0.3V
Storage Temperature ..... -50˚C to 125˚C
Operating Temperature ..... -40˚C to 85˚C
I_{OH} Total ..... -80mA
I_{OL} Total 80mA
Total Power Dissipation 500mW

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

D.C. Characteristics

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}	Conditions				
V_{DD}	Operating Voltage – HIRC	—	f_{SYS} = f_{HIRC} = 8MHz	4.75	—	5.25	V
	Operating Voltage – LIRC	—	f_{SYS} = f_{LIRC} = 32kHz	4.75	—	5.25	V

Standby Current Characteristics

Ta=25°C

Symbol	Standby Mode	Test Conditions		Min.	Typ.	Max.	Max.	Unit
		V_{DD}	Conditions				85°C	
I_{STB}	SLEEP Mode	5V	WDT on	—	330	560	570	μA
	IDLE0 Mode – LIRC	5V	f_{SUB} on	—	320	555	560	μA
	IDLE1 Mode – HIRC	5V	f_{SUB} on, f_{SYS} = 8MHz	—	600	800	960	μA

Notes: 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.
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.

Operating Current Characteristics

Ta=25°C

Symbol	Operating Mode	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{DD}	SLOW Mode – LIRC	5V	f_{SYS} = 32kHz	—	350	600	μA
	FAST Mode – HIRC	5V	f_{SYS} = 8MHz	—	2	3	mA

Notes: 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.
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.

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.

High Speed Internal 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}	Temp.				
f_{HIRC}	8MHz Writer Trimmed HIRC Frequency	5V	25°C	-1%	8	+1%	MHz
			-40°C ~ 85°C	-2%	8	+2%	

Notes: 1. The 5V value for VDD is provided as this is the selectable fixed voltage at which the HIRC frequency is trimmed by the writer.
2. It is recommended that the trim voltage is fixed at 5V for application voltage ranges from 4.75V to 5.25V.
3. The minimum and maximum tolerance values provided in the table are only for the frequency at which the writer trims the HIRC oscillator. After trimming at this chosen specific frequency any change in HIRC oscillator frequency using the oscillator register control bits by the application program will give a frequency tolerance to within ±20%.

Low Speed Internal Oscillator Characteristics – LIRC

Ta=25°C, unless otherwise specified

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Temp.				
f_{LIRC}	LIRC Frequency	5V~5.5V	25°C	-10%	32	+10%	kHz
			-40°C ~ 85°C	-50%	32	+60%	
t_{START}	LIRC Start Up Time	—	—	—	—	500	μs

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	—	$f_{SYS} = f_H \sim f_H/64$, $f_H = f_{HIRC}$	—	16	—	t_{HIRC}
		—	$f_{SYS} = f_{SUB} = f_{LIRC}$	—	2	—	t_{LIRC}
	System Start-up Time Wake-up from condition where f_{SYS} is on	—	$f_{SYS} = f_H \sim f_H/64$, $f_H = f_{HIRC}$	—	2	—	t_H
		—	$f_{SYS} = f_{SUB} = f_{LIRC}$	—	2	—	t_{SUB}
	System Speed Switch Time FAST to SLOW Mode or SLOW to FAST Mode	—	f_{HIRC} switches from off → on	—	16	—	t_{HIRC}
t_{RSTD}	System Reset Delay Time Reset Source from Power-on Reset	—	RR_{POR} = 5 V/ms	42	48	54	ms
	System Reset Delay Time WDTC Software Reset	—	—				
	System Reset Delay Time Reset Source from WDT Overflow	—	—	14	16	18	ms
t_{SRESET}	Minimum Software Reset Width to Reset	—	—	45	90	120	μs

Notes: 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} 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=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{IL}	Input Low Voltage for I/O Ports or Input Pins	5V	—	0	—	1.5	V
		—	—	0	—	$0.2V_{DD}$	
V_{IH}	Input High Voltage for I/O Ports or Input Pins	5V	—	3.5	—	5	V
		—	—	$0.8V_{DD}$	—	V_{DD}	
I_{OH}	Source Current for I/O Pins	5V	$V_{OH} = 0.9V_{DD}$	-8	-16	—	mA
I_{OL}	I/O Port Sink Current	5V	$V_{OL}=0.1V_{DD}$	32	65	—	mA
R_{PH}	Pull-high Resistance for I/O Ports	5V	—	10	30	50	kΩ
R_{PH}	Pull-high Resistance for I/O Ports (OCDSCK and OCDSDA for BS45V5930)	5V	—	10	30	50	kΩ
I_{LEAK}	Input Leakage Current	5V	$V_{IN} = V_{DD}$ or $V_{IN} = V_{SS}$	—	—	±1	μA

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 input sink current at the specified supply voltage level. Dividing the voltage by this measured current provides the R_{PH} value.

LDO Electrical Characteristics

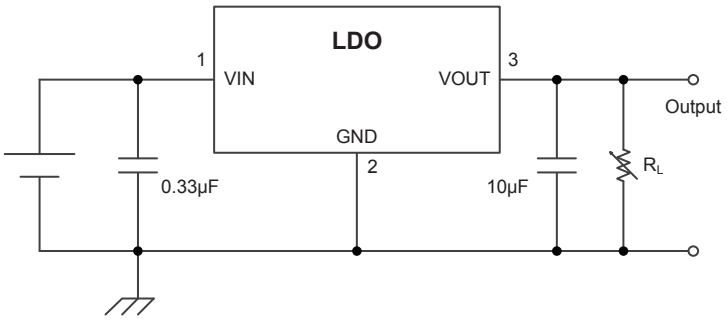
V_{IN} = V_{OUT} +1V, C_{LOAD} = 1μF, Ta = 25°C, unless otherwise specify

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{IN}	Conditions				
V_{IN}	Input Voltage	—	—	5.1	6	12	V
V_{OUT}	Output Voltage	—	I_{LOAD} = 1mA, V_{OUT} = 5.0V	-3%	5.0	3%	V
		—	Ta = -40°C ~ 85°C, I_{LOAD} = 1mA, V_{OUT} = 5.0V	-5%	5.0	5%	V
ΔV_{LOAD}	Load Regulation(1)	—	1mA ≤ I_{LOAD} ≤ 40mA,	—	0.015	0.033	%/mA
V_{DROP}	Dropout Voltage(2)	—	ΔV_{OUT} = 2%, I_{LOAD} = 1mA	—	20	—	mV
		—	ΔV_{OUT} = 2%, I_{LOAD} = 10mA	—	100	—	mV
		—	V_{IN} = V_{OUT} + 1.5V, ΔV_{OUT} = 2%, I_{LOAD} = 40mA	—	400	600	mV
I_{OUT}	Output Current	—	V_{IN} = V_{OUT} + 1V, V_{OUT} = 5V ΔV_{OUT} = -3%	25	—	—	mA
		—	V_{IN} = V_{OUT} + 2V, V_{OUT} = 5V ΔV_{OUT} = -3%	40	—	—	mA
I_Q	Quiescent Current	12V	No load	—	320	—	μA
ΔV_{LINE}	Line Regulation	—	V_{OUT} + 1V ≤ V_{IN} ≤ 12V, I_{LOAD} = 1mA	—	—	0.2	%/V
TC	Temperature Coefficient	—	Ta = -40°C ~ 85°C, I_{LOAD} = 10mA	—	±1.5	±2	mV/°C
RR	Ripple Rejection(3)	—	V_{IN} = $10V_{DC}$+$2V_{P-P(AC)}$, I_{LOAD} ≤ 40mA, f = 120Hz	35	—	—	dB
$t_{LDOSTART}$	LDO Startup Time	6V	I_{LOAD} = 1mA, V_{OUT} settle to ± 5%	—	—	10	ms
VCCO	VCCO Voltage	—	V_{IN} = 5.1V ~ 12V	- 5%	0.2×V_{IN}	+ 5%	V

Notes: 1. Load regulation is measured at a constant junction temperature, using pulse testing with a low ON time and is guaranteed up to the maximum power dissipation. Power dissipation is determined by the input/output differential voltage and the output current. Guaranteed maximum power dissipation will not be available over the full input/output range. The maximum allowable power dissipation at any ambient temperature is $P_D = (T_{J(MAX)}-Ta)/\theta_{JA}$

2. Dropout voltage is defined as the input voltage minus the output voltage that produces a 2% change in the output voltage from the value at appointed V_{IN}.

3. Ripple rejection ratio measurement circuit. $RR=20\times\log(\Delta V_{IN}/\Delta V_{OUT})$.



High Voltage Driver Output Electrical Characteristics

Ta= 25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{IN}	Input Voltage	—	—	V_{DD}	—	12	V
I_{OH}	Source Current for PAn Pins	—	V_{OH} = 0.9 × V_{IN}, V_{IN} = 6V	-250	-300	—	mA
I_{OL}	Sink Current for PAn Pins	—	V_{OL} = 0.1 × V_{IN}, V_{IN} = 6V	40	50	—	mA

OVP Electrical Characteristics

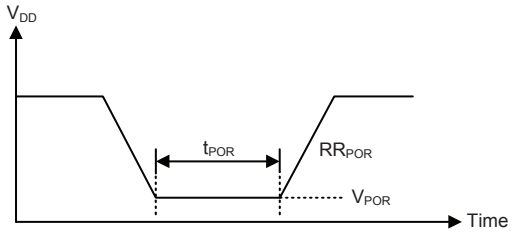
Ta= 25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
VDD	Operating Voltage	—	—	4.75	—	5.25	V
I_{OVP}	Operating Current	5V	OVPEN = 1, {OVPDAH[3:0], OVPDAL[7:0]} = 1000 0000 0000	—	145	180	μA
V_{OS}	Input Offset Voltage	5V	With calibration	-2	—	2	mV
V_{HYS}	Hysteresis	5V	—	20	40	60	mV
V_{CM}	Common Mode Voltage Range	5V	—	V_{SS}	—	V_{DD} - 1.4	V
DNL	Differential Nonlinearity	5V	DAC V_{REF} = V_{DD}	—	—	±3	LSB
INL	Integral Nonlinearity	5V	DAC V_{REF} = V_{DD}	—	—	±4	LSB

Power-on Reset Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{POR}	V_{DD} Start Voltage to Ensure Power-on Reset	—	—	—	—	100	mV
RR_{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

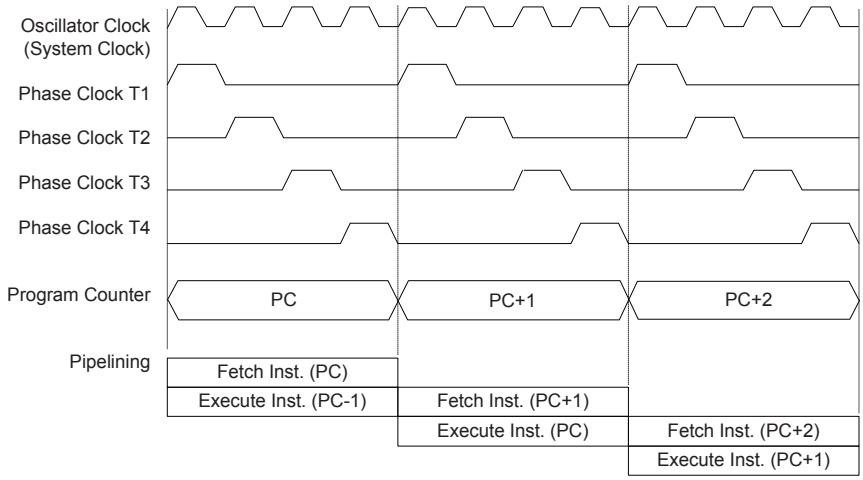


System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to their internal system architecture. The range of the device take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one cycle, with the exception of branch or call instructions. An 8-bit wide ALU is used in practically all 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 low-cost, high-volume production for controller applications.

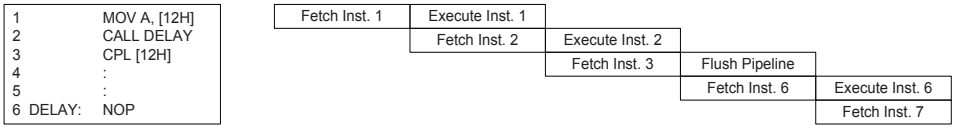
Clocking and Pipelining

The main system clock, derived from either a 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.



System Clocking and Pipelining

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.



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 demands 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)**
PC10~PC8	PCL7~PCL0

Program Counter

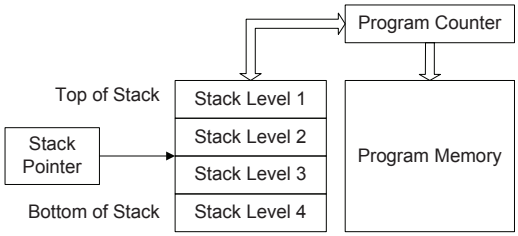
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly; however, as only this low byte is available 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 4 levels and neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack. After a device reset, the Stack Pointer will point to the top of the stack.

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

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:

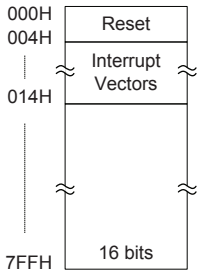
- Arithmetic operations: ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA
- Logic operations: AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA
- Rotation RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC
- Increment and Decrement INCA, INC, DECA, DEC
- Branch decision, JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET,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

Flash Program Memory

The Program Memory is the location where the user code or program is stored. For the device the Program Memory is Flash type, which means it can be programmed and re-programmed a large number of times, allowing the user the convenience of code modification on the same device. By using the appropriate programming tools, the Flash device offers users the flexibility to conveniently debug and develop their applications while also offering a means of field programming and updating.

Structure

The Program Memory has a capacity of 2K×16 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by 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 registers, 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 "TABRD [m]" or "TABRDL [m]" instructions, respectively. When the instruction is executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register. Any unused bits in this transferred higher order byte will be read as 0.

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

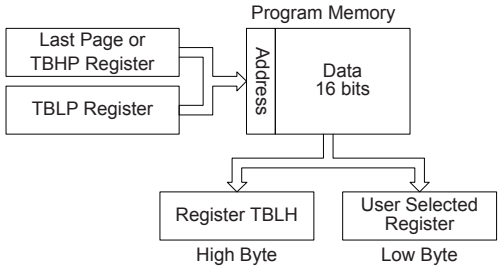


Table Program Example

The 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 "0700H" which refers to the start address of the last page within the 2K 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 "0706H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address specified by TBLP and TBHP if the "TABRD [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRDL [m]" instruction is executed.

Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use 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 low table pointer - note that this address is referenced
mov tblp,a      ; to the last page or the page that tbhp pointed
mov a,07h       ; initialise high table pointer
mov tbhp,a
:
:
tabrd tempreg1  ; transfers value in table referenced by table pointer data at program
                ; memory address "0706H" 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 "0705H" transferred to
                ; tempreg2 and TBLH in this example the data "1AH" is
                ; transferred to tempreg1 and data "0FH" to register tempreg2
:
:
org 0700h       ; sets initial address of program memory
dc 00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
```

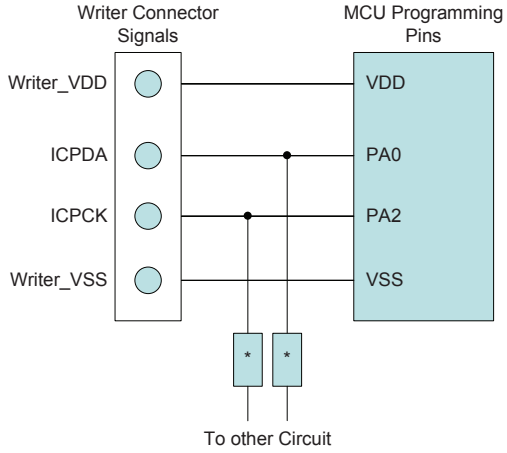
In Circuit Programming – ICP

The provision of Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device. As an additional convenience, Holtek has provided a means of programming the microcontroller in-circuit using a 4-pin interface. This provides manufacturers with the possibility of manufacturing their circuit boards complete with a programmed or un-programmed microcontroller, and then programming or upgrading the program at a later stage. This enables product manufacturers to easily keep their manufactured products supplied with the latest program releases without removal and re-insertion of the device.

Holtek Writer Pins	MCU Programming Pins	Pin Description
ICPDA	PA0	Programming Serial Data/Address
ICPCK	PA2	Programming Clock
VDD	VDD	Power Supply
VSS	VSS	Ground

The Program Memory can be programmed serially in-circuit using this 4-wire interface. Data is downloaded and uploaded serially on a single pin with an additional line for the clock. Two additional lines are required for the power supply. The technical details regarding the in-circuit programming of the device are beyond the scope of this document and will be supplied in supplementary literature.

During the programming process, the user must take control 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: * 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 BS45V5930 which is used to emulate the BS45F5930 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 and package type. 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. 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 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.

Structure

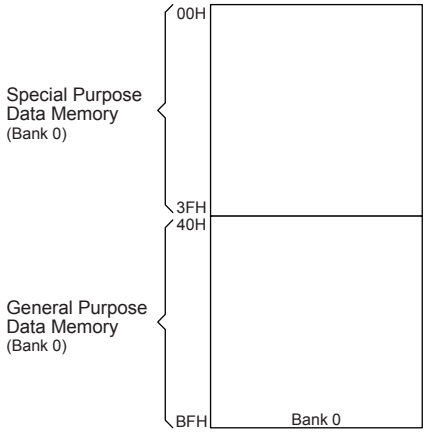
Categorized into two types, the first of these is an area of RAM, known as the Special Function Data Memory. Here are located registers which are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is known as the General Purpose Data Memory, which is reserved for general purpose use. All locations within this area are read and write accessible under program control.

The Data Memory has a bank, which is implemented in 8-bit wide Memory. The Data Memory Bank is categorized into two types, the special Purpose Data Memory and the General Purpose Data Memory.

The address range of the Special Purpose Data Memory for the device is from 00H to 3FH while the General Purpose Data Memory address range is from 40H to BFH.

Special Purpose Data Memory		General Purpose Data Memory	
Available Banks	Bank: Address	Capacity	Bank: Address
0	Bank 0: 00H~3FH	128×8	0: 40H~BFH

Data Memory Summary



Data Memory Structure

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 programing 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".

	Bank 0		Bank0
00H	IAR0	20H	HVOC1
01H	MP0	21H	DIVOC
02H	IAR1	22H	
03H	MP1	23H	
04H		24H	
05H	ACC	25H	
06H	PCL	26H	
07H	TBLP	27H	
08H	TBLH	28H	
09H	TBHP	29H	
0AH	STATUS	2AH	
0BH	SCC	2BH	PWMC
0CH	HIRCC	2CH	PWM0DATA
0DH	INTEG	2DH	
0EH	INTC0	2EH	PWM1DATA
0FH	RSTFC	2FH	TMRC
10H		30H	TMR
11H	INTC1	31H	OVPC0
12H		32H	OVPC1
13H		33H	OVPDAL
14H	PA	34H	OVPDAH
15H	PAC	35H	TKTMR
16H	PAPU	36H	TKC0
17H	PAWU	37H	TK16DL
18H		38H	TK16DH
19H		39H	TKC1
1AH	WDTC	3AH	TKM16DL
1BH	TBC	3BH	TKM16DH
1CH	PSCR	3CH	TKMROL
1DH	PAS0	3DH	TKMROH
1EH	PAS1	3EH	TKMC0
1FH	HVOC	3FH	TKMC1

: Unused, read as "00"

Special Purpose Data Memory

Special Function Register Description

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

Indirect Addressing Register – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 and IAR1 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointers, MP0 or MP1. Acting as a pair, IAR0 and MP0 can together access data from Bank 0 while the IAR1 and MP1 register pair can access data from any bank. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointers – MP0, MP1

Two Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to, is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Bank 0, while IAR1 and MP1 register pair are used to access data from any bank. Direct Addressing can only be used with Bank 0, all other Banks must be addressed indirectly using MP1 and IAR1.

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

Indirect Addressing Program Example

```
data .section ´data´
adres1   db ?
adres2   db ?
adres3   db ?
adres4   db ?
block    db ?
code .section at 0 ´code´
org 00h
start:
     mov a,04h            ; setup size of block
     mov block,a
     mov a,offset adres1  ; Accumulator loaded with first RAM address
     mov mp0,a            ; setup memory pointer with first RAM address
loop:
     clr IAR0             ; clear the data at address defined by mp0
     inc mp0              ; increment memory pointer
     sdz block            ; check if last memory location has been cleared
     jmp loop
continue:
```

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

Accumulator – ACC

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

Program Counter Low Register – PCL

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

Look-up Table Registers – TBLP, TBHP, TBLH

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

Status Register – STATUS

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

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

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

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

- TO is cleared by a system power-up or executing the "CLR WDT" or "HALT" instruction. TO is set by a WDT time-out.

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

- **STATUS Register**

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

"x": unknown

Bit 7~6 Unimplemented, read as "0"

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

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

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

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

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

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

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

Oscillators

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

Oscillator Overview

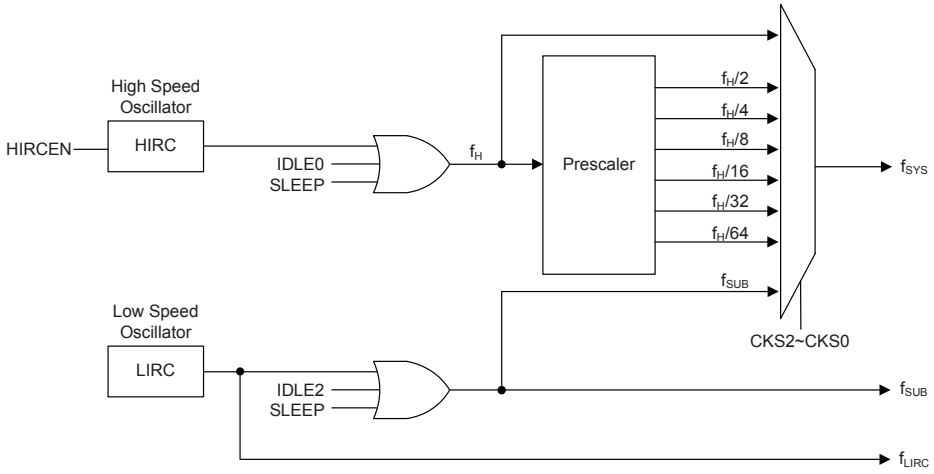
In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer and Time Base Interrupts. Two fully integrated internal oscillators, requiring no external components, are provided to form a range of both fast and slow system oscillators. The 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	8MHz
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 8MHz RC oscillator, HIRC. The low speed oscillator is the internal 32kHz RC oscillator, LIRC. The frequency of the slow speed or high speed system clock is also determined using the CKS2~CKS0 bits in the SCC register.



System Clock Configurations

High Speed Internal RC Oscillator – HIRC

The high speed internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has a fixed frequency of 8MHz. 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 the low frequency oscillator. It is a fully integrated RC oscillator with a typical frequency of 32kHz, requiring no external components for its implementation. 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.

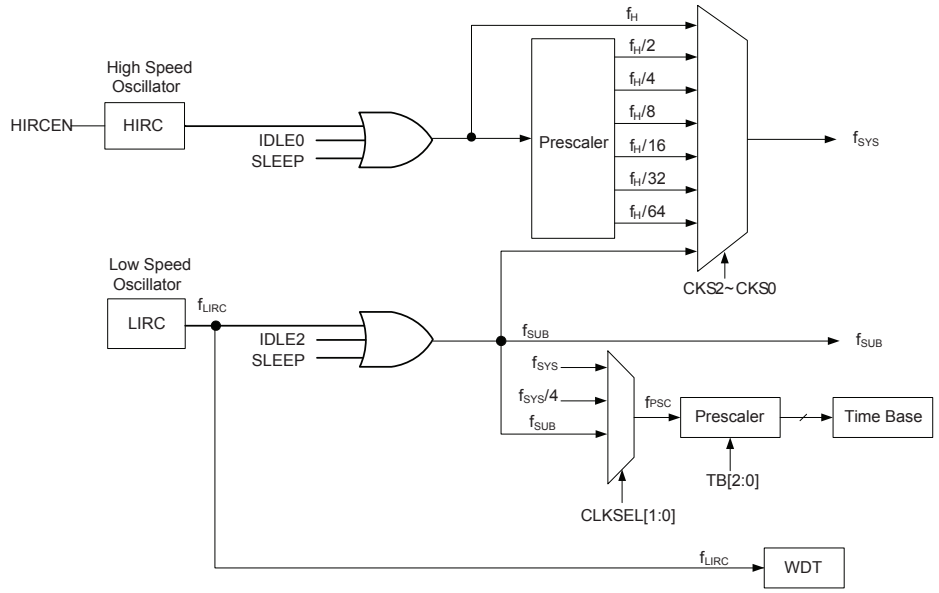
Operating Modes and System Clocks

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 selections 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 CKS2~CKS0 bits in the SCC register. The high speed system clock is sourced from HIRC oscillator. The low speed system clock source can be sourced from the LIRC oscillator. The other choice, which is a divided version of the high speed system oscillator has a range of $f_H/2$~$f_H/64$.



Device Clock Configuration

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~$f_H/64$, 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_{LIRC}
		FHIDEN	FSIDEN	CKS[2:0]				
FAST Mode	On	x	x	000~110	On	On	On	On
SLOW Mode	On	x	x	111	On	On/Off(1)	On	On
IDLE0 Mode	Off	0	1	000~110	Off	Off	On	On
				111	On			
IDLE1 Mode	Off	1	1	xxx	On	On	On	On
IDLE2 Mode	Off	1	0	000~110	On	On	Off	On
				111	Off			
SLEEP Mode	Off	0	0	xxx	Off	Off	Off	On (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_{LIRC} clock will be switched on since the WDT function is always enabled in the SLEEP mode.

FAST Mode

As the name suggests 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 oscillator. This mode operates allowing the microcontroller to operate normally with a clock source will come from the high speed oscillator, HIRC. The high speed oscillator will however first be divided by a ratio ranging from 1 to 64, the actual ratio being selected by the CKS2~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 an HALT instruction is executed and when the FHIDEN and FSIDEN bit 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_{LIRC} clock still continues to operate since the WDT function is always enabled.

IDLE0 Mode

The IDLE0 Mode is entered when an 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 an 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 an 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 Register

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	CKS2	CKS1	CKS0	—	—	—	FHIDEN	FSIDEN
HIRCC	—	—	—	—	—	—	HIRCF	HIRCEN

System Operating Mode Control Registers List

- **SCC Register**

Bit	7	6	5	4	3	2	1	0
Name	CKS2	CKS1	CKS0	—	—	—	FHIDEN	FSIDEN
R/W	R/W	R/W	R/W	—	—	—	R/W	R/W
POR	0	0	0	—	—	—	0	0

Bit 7~5 **CKS2~CKS0**: System clock selection
000: f_H
001: $f_H/2$
010: $f_H/4$
011: $f_H/8$
100: $f_H/16$
101: $f_H/32$
110: $f_H/64$
111: f_{SUB}
These three 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 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 an "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 an "HALT" instruction. If this bit is cleared to 0 but the WDT function is always enabled, the LIRC oscillator will also be enabled.

- **HIRCC Register**

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

Bit 7~2 Unimplemented, read as "0"

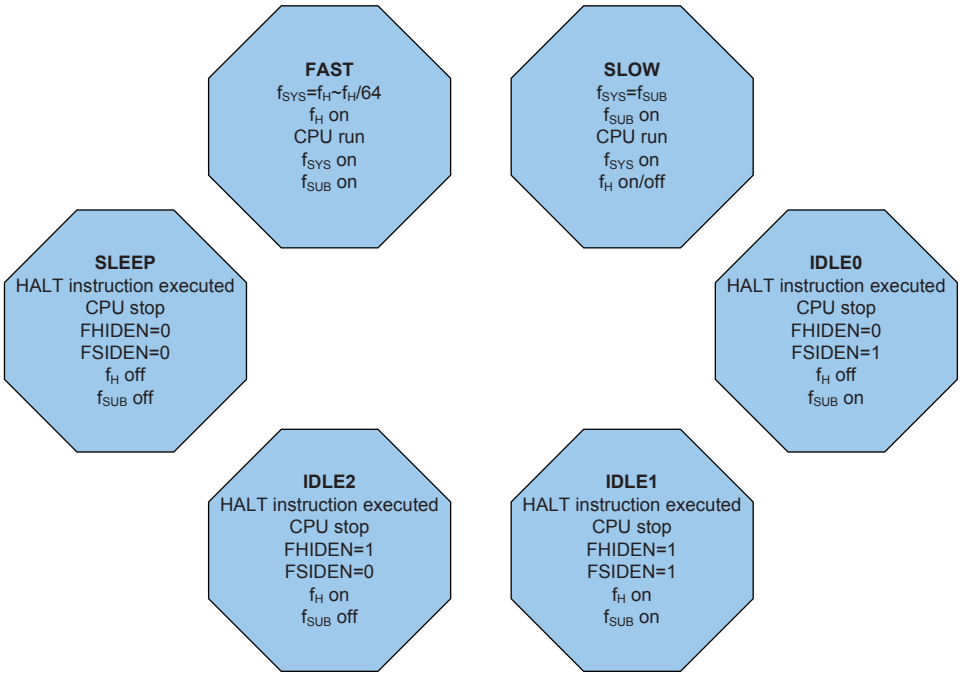
Bit 1 **HIRCF**: HIRC oscillator stable flag
0: Unstable
1: Stable
This bit is used to indicate whether the HIRC oscillator is stable or not. When the HIRCEN bit is set to 1 to enable the HIRC oscillator, the HIRCF bit will first be cleared to 0 and then set to 1 after the HIRC oscillator is stable.

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.

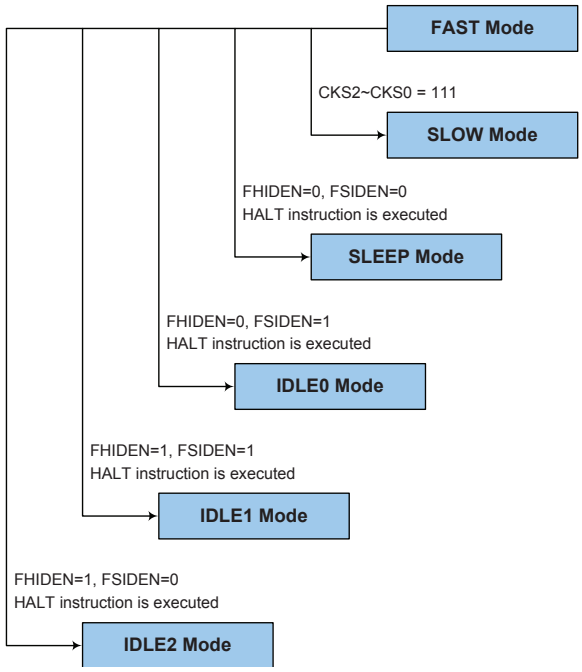
In simple terms, mode switching between the FAST Mode and SLOW Mode is executed using the CKS2~CKS0 bits in the SCC register while mode switching from the FAST/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When an HALT instruction is executed, whether the device enters the 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 set the CKS2~CKS0 bits to "111" 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.

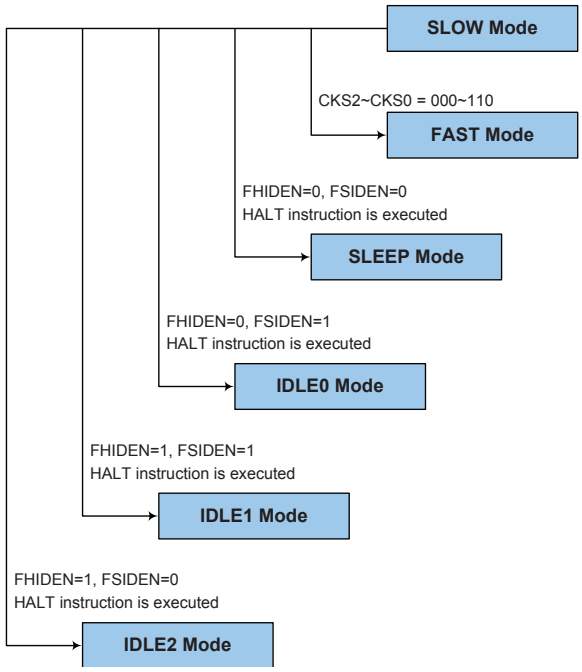
The SLOW Mode is sourced from the LIRC oscillator and therefore requires this oscillator to be stable before full mode switching occurs.



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 CKS2~CKS0 bits should be set to "000"~"110" and then the system clock will respectively be switched to f_H~f_H/64.

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 A.C. 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 as the WDT function is always 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 as the WDT function is always 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 as the WDT function is always enabled.

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 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 as the WDT function is always enabled.

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. These must either be setup as outputs or if setup as inputs must have pull-high resistors connected.

Care must also be taken with the loads, which are connected to I/O pins, which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs. Also note that additional standby current will also be required if the 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, stablise 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 falling edge on Port A
- A system interrupt
- A WDT overflow

When the device executes the "HALT" instruction, the PDF flag will be set to 1. The PDF flag will be cleared to 0 if the device experiences a system power-up or executes the clear Watchdog Timer instruction. If the system is woken up by a WDT overflow, a Watchdog Timer reset will be initiated and the TO flag will be set to 1. 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, other flags remain in their original status.

Each pin on Port A can be setup using the PAWU register to permit a negative transition on the pin to wake up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction. If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt which woke up the device will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set 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_{LIRC}, 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 enable and reset MCU 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 software control
01010/10101: Enabled
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: $2^8/f_{LIRC}$
001: $2^{10}/f_{LIRC}$
010: $2^{12}/f_{LIRC}$
011: $2^{14}/f_{LIRC}$
100: $2^{15}/f_{LIRC}$
101: $2^{16}/f_{LIRC}$
110: $2^{17}/f_{LIRC}$
111: $2^{18}/f_{LIRC}$
These three bits determine the division ratio of the Watchdog Timer source clock, which in turn determines the timeout period.

- **RSTFC Register**

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

Bit 7~1 Unimplemented, read as "0"

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. Note that this bit can only be cleared to 0 by the 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 instructions. If the program malfunctions for whatever reason, jumps to an unknown location, or enters an endless loop, these clear instructions 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 enable and reset control of the Watchdog Timer. The WDT function will be enabled when the WE4~WE0 bits are set to a value of 01010B or 10101B. If the WE4~WE0 bits are set to any other values other than 01010B and 10101B, it will reset the device after a delay time, t_{SRESET}. After power on these bits will have the value of 01010B.

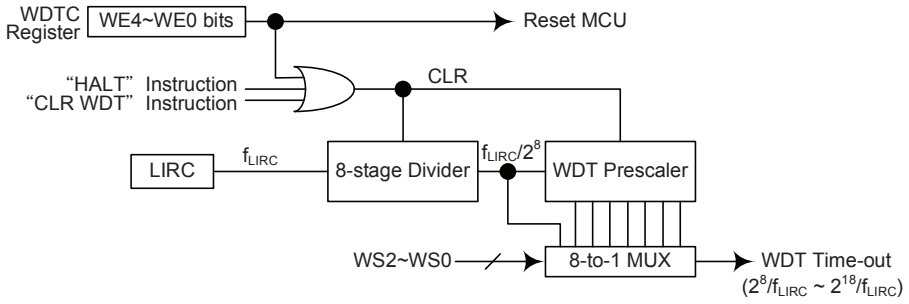
WE4~WE0 Bits	WDT Function
01010B/10101B	Enable
Any other values	Reset MCU

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. Three methods can be adopted to clear the contents of the Watchdog Timer. The first is a WDT reset, which means a certain value except 01010B and 10101B written into the WE4~WE0 bit filed, the second is using the Watchdog Timer software clear instructions and the third is via a HALT instruction.

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

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.

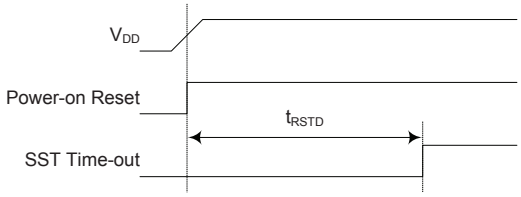
A type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

Reset Functions

There are three ways in which a microcontroller reset can occur, through events occurring internally.

Power-on Reset

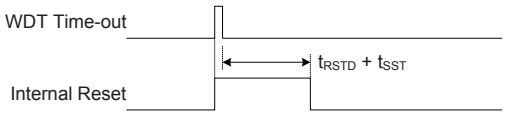
The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions. 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

Watchdog Time-out Reset during Normal Operation

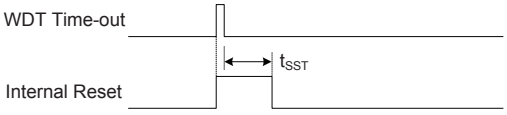
The Watchdog time-out Reset during normal operations in the FAST or SLOW mode is the same as a Power On reset except that the Watchdog time-out flag TO will be set to "1".



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 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
1	u	WDT time-out reset during FAST or SLOW Mode operation
1	1	WDT time-out reset during IDLE or SLEEP Mode operation

Note: "u" stands for unchanged

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

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

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

Register Name	Power On Reset	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
IAR0	xxxx xxxx	xxxx xxxx	uuuu uuuu
MP0	xxxx xxxx	xxxx xxxx	uuuu uuuu
IAR1	xxxx xxxx	xxxx xxxx	uuuu uuuu
MP1	xxxx xxxx	xxxx xxxx	uuuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu
TBHP	---- -xxx	---- -uuu	---- -uuu
STATUS	--00 xxxx	--1u uuuu	--11 uuuu
SCC	000- --00	000- --00	uuu- --uu
HIRCC	---- --01	---- --01	---- --uu
INTEG	---- --00	---- --00	---- --uu
INTC0	-000 0000	-000 0000	-uuu uuuu
RSTFC	---- ---0	---- ---u	---- ---u
INTC1	--00 --00	--00 --00	--uu --uu
PA	---1 1111	---1 1111	---u uuuu
PAC	---1 1111	---1 1111	---u uuuu
PAPU	---0 0000	---0 0000	---u uuuu
PAWU	---0 0000	---0 0000	---u uuuu
WDTC	0101 0011	0101 0011	uuuu uuuu
TBC	0--- -000	0--- -000	u--- -uuu

Register Name	Power On Reset	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
PSCR	---- --00	---- --00	---- --uu
PAS0	0000 0000	0000 0000	uuuu uuuu
PAS1	---- --00	---- --00	---- --uu
HVOC	---- 0000	---- 0000	---- uuuu
HVOC1	0000 --00	0000 --00	uuuu --uu
DIVOC	---- ---0	---- ---0	---- ---u
PWMC	000- --00	000- --00	uuu- --uu
PWM0DATA	0000 0000	0000 0000	uuuu uuuu
PWM1DATA	0000 0000	0000 0000	uuuu uuuu
TMRC	--00 -000	--00 -000	--uu -uuu
TMR	0000 0000	0000 0000	uuuu uuuu
OVPC0	0000 -000	0000 -000	uuuu -uuu
OVPC1	0001 0000	0001 0000	uuuu uuuu
OVPDAL	0000 0000	0000 0000	uuuu uuuu
OVPDAH	---- 0000	---- 0000	---- uuuu
TKTMR	0000 0000	0000 0000	uuuu uuuu
TKC0	-000 0000	-000 0000	-uuu uuuu
TK16DL	0000 0000	0000 0000	uuuu uuuu
TK16DH	0000 0000	0000 0000	uuuu uuuu
TKC1	---- --11	---- --11	---- --uu
TKM16DL	0000 0000	0000 0000	uuuu uuuu
TKM16DH	0000 0000	0000 0000	uuuu uuuu
TKMROL	0000 0000	0000 0000	uuuu uuuu
TKMROH	---- --00	---- --00	---- --uu
TKMC0	0000 0000	0000 0000	uuuu uuuu
TKMC1	0-00 0000	0-00 0000	u-uu 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 name PA. These I/O ports are mapped to the RAM Data Memory with specific addresses as shown in the Special Purpose Data Memory table. All of these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction "MOV A, [m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PA	—	—	—	PA4	PA3	PA2	PA1	PA0
PAC	—	—	—	PAC4	PAC3	PAC2	PAC1	PAC0
PAPU	—	—	—	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
PAWU	—	—	—	PAWU4	PAWU3	PAWU2	PAWU1	PAWU0

"—": 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 an input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selected using the relevant pull-high control registers PAPU, and are implemented using weak PMOS transistors.

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

- **PAPU Register**

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

Bit 7~5 Unimplemented, read as "0"

Bit 4~0 **PAPU4~PAPU0**: Port A bit 4 ~ bit 0 Pull-high Control
0: Disable
1: Enable

Port A Wake-up

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 A pins from high to low. This function is especially suitable for applications that can be woken up via external switches. Each pin on Port A can be selected individually to have this wake-up feature using the PAWU register.

Note that the wake-up function can be controlled by the wake-up control registers only when the pin-shared functional pin is selected as general purpose input/output and the MCU enters the Power down mode.

- **PAWU Register**

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

Bit 7~5 Unimplemented, read as "0"

Bit 4~0 **PAWU4~PAWU0**: Port A bit 4 ~ bit 0 Wake-up Control
0: Disable
1: Enable

I/O Port Control Registers

Each I/O port has its own control register known as PAC, 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.

- **PAC Register**

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

Bit 7~5 Unimplemented, read as "0"

Bit 4~0 **PAC4~PAC0**: Port A bit 4 ~ bit 0 Input/Output Control
0: Output
1: Input

Pin-shared Functions

The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For 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 "A" pin shared function selection registers, labeled as PASn, 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 INT, OVPI and KEYn 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 an 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	—	—	—	—	—	—	PAS11	PAS10

Pin-shared Function Selection Registers 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: OVPI
11: KEY4

Bit 5~4 **PAS05~PAS04**: PA2 Pin-shared function selection
00: PA2
01: PA2
10: PA2
11: KEY2

Bit 3~2 **PAS03~PAS02**: PA1 Pin-shared function selection
00: PA1
01: PA1
10: OVPI
11: KEY3

Bit 1~0 **PAS01~PAS00**: PA0 Pin-shared function selection
00: PA0
01: PA0
10: PA0
11: KEY1

- **PAS1 Register**

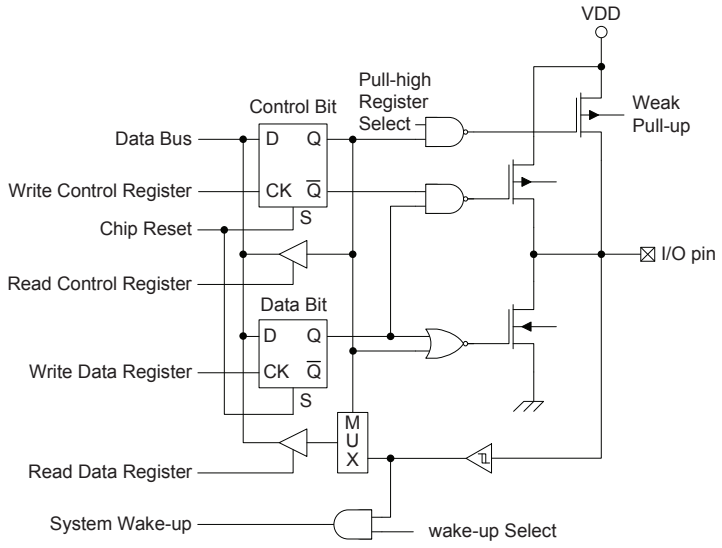
Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	PAS11	PAS10
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **PAS11~PAS10**: PA4 Pin-Shared function selection
00: PA4
01: PA4
10: PA4
11: OVPI

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

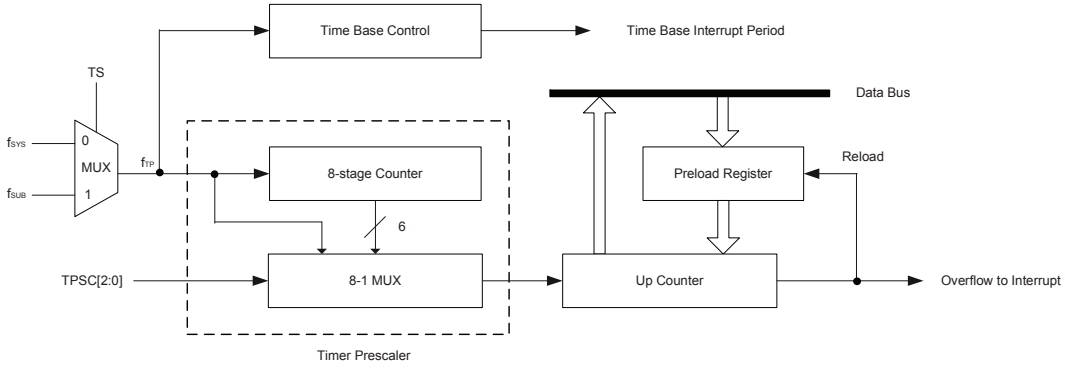
Within the user program, one of the first things to consider is port initialisation. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high selections have been chosen. If the port control register, PAC, are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated port data register, PA, 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 has the additional capability of providing wake-up functions. 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 transition of any of the Port A pins. Single or multiple pins on Port A can be setup to have this function.

Timer/Event Counter

The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. The device contains an 8-bit Timer/Event Counter. And the provision of an internal prescaler to the clock circuitry on gives added range to the timer.

There are two types of registers related to the Timer/Event Counter. The first is the register that contains the actual value of the timer and into which an initial value can be preloaded. Reading from this register retrieves the contents of the Timer/Event Counter. The second type of associated register is the Timer Control Register which defines the timer options and determines how the timer is to be used.



Timer/Event Counter Structure

Configuring the Timer/Event Counter Input Clock Source

The Timer/Event Counter clock source can originate from either f_{SYS} or the f_{SUB} oscillator, the choice of which is determined by the TS bit in the TMRC register. This internal clock source is first divided by a prescaler, the division ratio of which is conditioned by the Timer Control Register bits TPSC2~TPSC0.

Timer Register – TMR

The timer register TMR is a special function register located in the Special Purpose Data Memory and 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. 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 reset with the initial preload register value and continue counting.

Note that to achieve a maximum full range count of FFH, the preload register must first be cleared to all zeros. Note that if the Timer/Event Counter is in an OFF condition and data is written to its preload register, this data will be immediately written into the actual counter. However, if the counter is enabled and counting, any new data written into the preload data register during this period will remain in the preload register and will only be written into the actual counter the next time an overflow occurs.

Timer Control Register – TMRC

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.

The timer-on bit, which is bit 4 of the Timer Control Register and known as TON bit, provides the basic on/off control of the timer. Setting the bit high allows the counter to run. Clearing the bit stops the counter. Bits 0~2 of the TMRC Register determine the division ratio of the input clock prescaler. In addition, the bit TS is used to select the internal clock source.

- **TMRC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	TS	TON	—	TPSC2	TPSC1	TPSC0
R/W	—	—	R/W	R/W	—	R/W	R/W	R/W
POR	—	—	0	0	—	0	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **TS**: Timer/Event Counter Clock Source
0: f_{SYS}
1: f_{SUB}

Bit 4 **TON**: Timer/Event Counter Counting Enable
0: Disable
1: Enable

Bit 3 Unimplemented, read as "0"

Bit 2 ~ 0 **TPSC2~TPSC0**: Timer prescaler rate selection
Timer internal clock=
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 Operation

The Timer/Event Counter can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. The internal clock is used as the timer clock. The timer input clock is either f_{SYS} or the f_{SUB} oscillator. However, this timer clock source is further divided by a prescaler, the value of which is determined by the bits TPSC0~TPSC2 in the Timer Control Register. The timer-on bit, TON must be set high to enable the timer to run. Each time when an internal clock high to low transition occurs, the timer will reload the value already loaded into the preload register and continues counting. A timer overflow condition and corresponding internal interrupt is one of the wake-up sources, however, the internal interrupts can be disabled by ensuring that the TE bit of the INTC0 register are reset to zero.

Prescaler

Bits TPSC0~TPSC2 of the TMRC register can be used to define a division ratio for the internal clock source of the Timer/Event Counter enabling longer time out periods to be setup.

Programming Consideration

When the Timer/Event Counter is read, or if data is written to the preload register, the clock is inhibited to avoid errors, however as this may result in a counting error, this should be taken into account by the programmer. Care must be taken to ensure that the timer is properly initialized before using them for the first time. The associated timer enable bits in the interrupt control register must be properly set otherwise the internal interrupt associated with the timer will remain inactive. The edge select, timer mode and clock source control bits in timer control register must also be correctly set to ensure the timer is properly configured for the required application. It is also important to ensure that an initial value is first loaded into the timer registers before the timer is switched on; this is because after power-on the initial values of the timer registers are unknown.

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

Touch Key Function

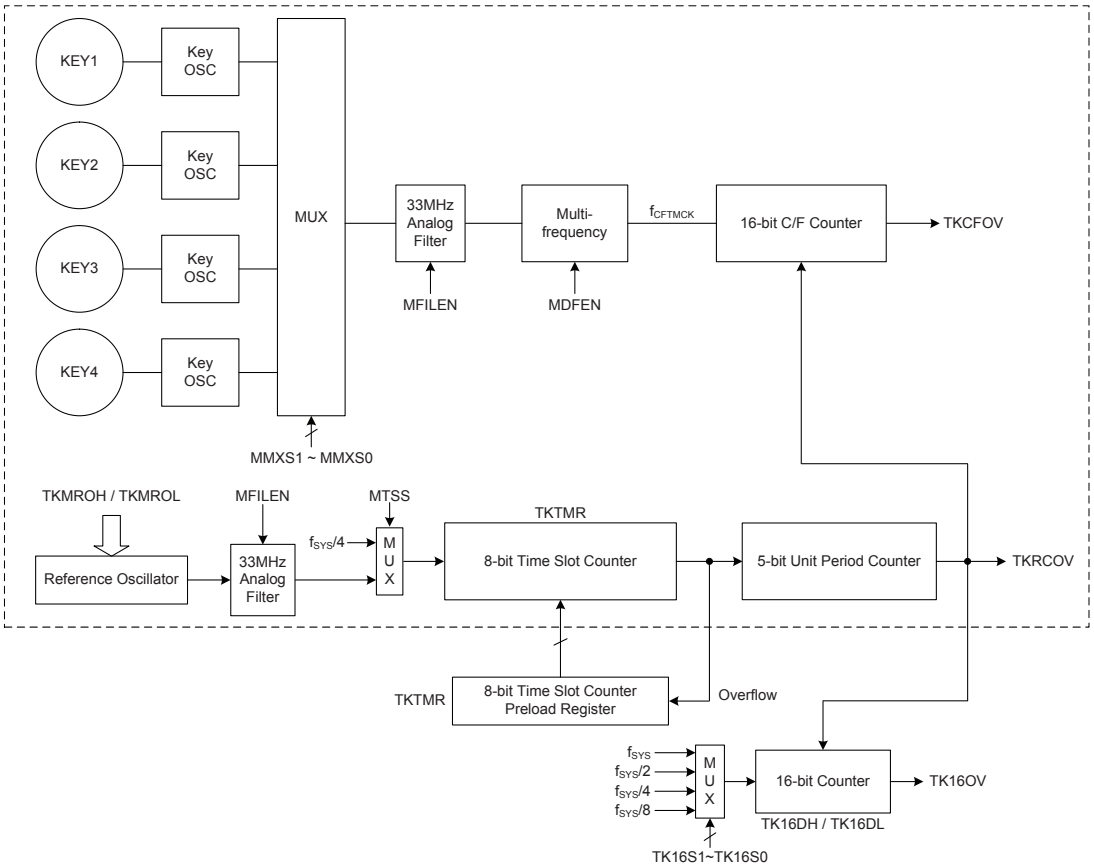
The device provides 4 touch key functions. The touch key function is fully integrated and requires no external components, allowing touch key functions to be implemented by the simple manipulation of internal registers.

Touch Key Structure

The touch keys are pin shared with the I/O pins, with the desired function chosen via the pin-shared selection register bit. Keys are organised into one group, known as a module. The module is a fully independent set of four Touch Keys and has its own oscillator. The module contains its own control logic circuits and register set.

Total Key Number	Touch Key	Shared I/O Pin
4	KEY1~KEY4	PA0, PA2, PA1, PA3

Touch Key Structure



Note: 1. The structure contained in the dash line is for the touch key module which contains four touch keys.
2. When MTSS=0 and MROEN=1 or when MTSS=1, the touch key function 16-bit counter can operate normally.

Touch Key Function Block Diagram

Touch Key Registers Description

The touch key module, which contains four touch key functions, has its own suite registers. The following table shows the register set for the touch key module.

Register Name	Description
TKTMR	Touch key time slot 8-bit counter preload register
TKC0	Touch key function control register 0
TKC1	Touch key function control register 1
TK16DL	Touch key function 16-bit counter low byte
TK16DH	Touch key function 16-bit counter high byte
TKM16DL	Touch key module 16-bit C/F counter low byte
TKM16DH	Touch key module 16-bit C/F counter high byte
TKMROL	Touch key module reference oscillator capacitor select low byte
TKMROH	Touch key module reference oscillator capacitor select high byte
TKMC0	Touch key module control register 0
TKMC1	Touch key module control register 1

Touch Key Function Register Definition

Register Name	Bit							
	7	6	5	4	3	2	1	0
TKTMR	D7	D6	D5	D4	D3	D2	D1	D0
TKC0	—	TKRCOV	TKST	TKCFOV	TK16OV	—	TK16S1	TK16S0
TKC1	—	—	—	—	—	—	TKFS1	TKFS0
TK16DL	D7	D6	D5	D4	D3	D2	D1	D0
TK16DH	D15	D14	D13	D12	D11	D10	D9	D8
TKM16DL	D7	D6	D5	D4	D3	D2	D1	D0
TKM16DH	D15	D14	D13	D12	D11	D10	D9	D8
TKMROL	D7	D6	D5	D4	D3	D2	D1	D0
TKMROH	—	—	—	—	—	—	D9	D8
TKMC0	MMXS1	MMXS0	MDFEN	MFILEN	MSOFC	MSOF2	MSOF1	MSOF0
TKMC1	MTSS	—	MROEN	MKOEN	MK4EN	MK3EN	MK2EN	MK1EN

Touch Key Function Register List

- **TKTMR 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**: Touch key time slot 8-bit counter preload register

The touch key time slot counter preload register is used to determine the touch key time slot overflow time. The time slot unit period is obtained by a 5-bit counter and equal to 32 time slot clock cycles. Therefore, the time slot counter overflow time is equal to the following equation shown.

Time slot counter overflow time = (256 - TKTMR[7:0]) × $32t_{TSC}$, where the t_{TSC} is the time slot counter clock period.

- **TKC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	TKRCOV	TKST	TKCFOV	TK16OV	—	TK16S1	TK16S0
R/W	—	R/W	R/W	R/W	R/W	—	R/W	R/W
POR	—	0	0	0	0	—	0	0

Bit 7 Unimplemented, read as "0"

Bit 6 **TKRCOV**: Touch key time slot counter overflow flag
0: No overflow occurs
1: Overflow occurs
This bit can be accessed by application program. When this bit is set by touch key time slot counter overflow, the corresponding touch key interrupt request flag will be set. However, if this bit is set by application program, the touch key interrupt request flag will not be affected. Therefore, this bit cannot be set by application program but must be cleared to 0 by application program.
If the time slot counter overflows, the TKRCOV bit and the Touch Key Interrupt request flag, TKMF, will be set and all module keys and reference oscillators will automatically stop. The touch key module 16-bit C/F counter, touch key function 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot counter will be automatically switched off.

Bit 5 **TKST**: Touch key detection start control
0: Stopped or no operation
0→1: Start detection
The touch key module 16-bit C/F counter, touch key function 16-bit counter and 5-bit time slot unit period counter will automatically be cleared when this bit is cleared to zero. However, the 8-bit programmable time slot counter will not be cleared. When this bit is changed from low to high, the touch key module 16-bit C/F counter, touch key function 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot counter will be switched on together with the key and reference oscillators to drive the corresponding counters.

Bit 4 **TKCFOV**: Touch key module 16-bit C/F counter overflow flag
0: No overflow occurs
1: Overflow occurs
This bit is set by touch key module 16-bit C/F counter overflow and must be cleared to 0 by application program.

Bit 3 **TK16OV**: Touch key function 16-bit counter overflow flag
0: No overflow occurs
1: Overflow occurs
This bit is set by touch key function 16-bit counter overflow and must be cleared to 0 by application program.

Bit 2 Unimplemented, read as "0"

Bit 1~0 **TK16S1~TK16S0**: Touch key function 16-bit counter clock source selection
00: f_{SYS}
01: $f_{SYS}/2$
10: $f_{SYS}/4$
11: $f_{SYS}/8$

- **TKC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	TKFS1	TKFS0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	1	1

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **TKFS1~TKFS0**: Touch key oscillator and Reference oscillator frequency selection
00: 1MHz
01: 3MHz
10: 7MHz
11: 11MHz

- **TK16DL Register**

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

Bit 7~0 **D7~D0**: Touch key function 16-bit counter low byte contents

- **TK16DH Register**

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

Bit 7~0 **D15~D8**: Touch key function 16-bit counter high byte contents
This register pair is used to store the touch key function 16-bit counter value. This 16-bit counter can be used to calibrate the reference or key oscillator frequency. When the touch key time slot counter overflows, this 16-bit counter will be stopped and the counter content will be unchanged. This register pair will be cleared to zero when the TKST bit is cleared.

- **TKM16DL Register**

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

Bit 7~0 **D7~D0**: Touch key module 16-bit C/F counter low byte contents

- **TKM16DH Register**

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

Bit 7~0 **D15~D8**: Touch key module 16-bit C/F counter high byte contents
This register pair is used to store the touch key module 16-bit C/F counter value. This 16-bit C/F counter will be stopped and the counter content will be kept unchanged when the touch key time slot counter overflows. This register pair will be cleared to zero when the TKST bit is set low.

- **TKMROL 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**: Touch key module reference oscillator internal capacitor selection

- **TKMROH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **D9~D8**: Touch key module reference oscillator internal capacitor selection

This register pair is used to store the touch key module reference oscillator capacitor value.

The reference oscillator internal capacitor value = (TKMRO[9:0] × 50pF) / 1024

- **TKMC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	MMXS1	MMXS0	MDFEN	MFILEN	MSOFC	MSOF2	MSOF1	MSOF0
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 **MMXS1~ MMXS0**: Touch key module multiplexer key selection
00: KEY1
01: KEY2
10: KEY3
11: KEY4

Bit 5 **MDFEN**: Touch key module multi-frequency control
0: Disable
1: Enable

This bit is used to control the touch key oscillator frequency doubling function. When this bit is set to 1, the key oscillator frequency will be doubled.

Bit 4 **MFILEN**: Touch key module filter function control
0: Disable
1: Enable

Bit 3 **MSOFC**: Touch key module C-to-F oscillator frequency hopping function control selection
0: Controlled by the MSOF2~MSOF0
1: Controlled by hardware circuit

This bit is used to select the touch key oscillator frequency hopping function control method. When this bit is set to 1, the key oscillator frequency hopping function is controlled by the hardware circuit regardless of the MSOF2~MSOF0 bits value.

Bit 2~0 **MSOF2~MSOF0**: Touch key module Reference and Key oscillators hopping frequency selection
000: 1.020MHz
001: 1.040MHz
010: 1.059MHz
011: 1.074MHz
100: 1.085MHz
101: 1.099MHz
110: 1.111MHz
111: 1.125MHz

These bits are used to select the touch key oscillator frequency for the hopping function. Note that these bits are only available when the MSOFC bit is cleared to 0.

The frequency mentioned here will be changed when the external or internal capacitor is with different values. If the touch key operates at 1MHz frequency, users can adjust the frequency in scale when any other frequency is selected.

- **TKMC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	MTSS	—	MROEN	MKOEN	MK4EN	MK3EN	MK2EN	MK1EN
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 **MTSS**: Touch key module time slot counter clock source selection
0: Touch key module reference oscillator
1: $f_{SYS}/4$

Bit 6 Unimplemented, read as "0"

Bit 5 **MROEN**: Touch key module Reference oscillator enable control
0: Disable
1: Enable
This bit is used to enable the touch key module reference oscillator.
The reference oscillator should first be enabled before setting the TKST bit from low to high if the reference oscillator is selected to be used.

Bit 4 **MKOEN**: Touch key module Key oscillator enable control
0: Disable
1: Enable
This bit is used to enable the touch key module key oscillator.
The key oscillator should first be enabled before setting the TKST bit from low to high if the relevant key is enabled to be scanned.

Bit 3 **MK4EN**: Touch key module KEY4 enable control
0: Disable
1: Enable

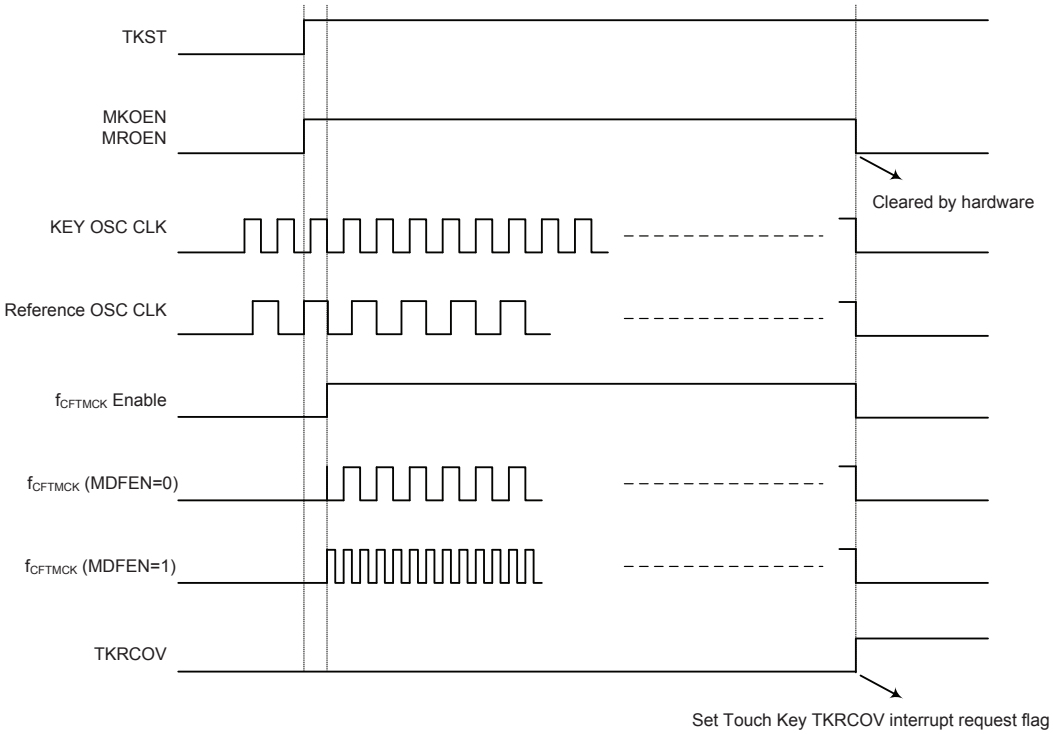
Bit 2 **MK3EN**: Touch key module KEY3 enable control
0: Disable
1: Enable

Bit 1 **MK2EN**: Touch key module KEY2 enable control
0: Disable
1: Enable

Bit 0 **MK1EN**: Touch key module KEY1 enable control
0: Disable
1: Enable

Touch Key Operation

When a finger touches or is in proximity to a touch pad, the capacitance of the pad will increase. By using this capacitance variation to change slightly the frequency of the internal sense oscillator, touch actions can be sensed by measuring these frequency changes. Using an internal programmable divider the reference clock is used to generate a fixed time period. By counting a number of generated clock cycles from the sense oscillator during this fixed time period touch key actions can be determined.



Touch Key Module Timing Diagram

The touch key module contains four touch key inputs, namely KEY1~KEY4, which are shared with logical I/O pins, and the desired function is selected using register bits. The touch key has its own independent sense oscillator. There are therefore four sense oscillators within the touch key module.

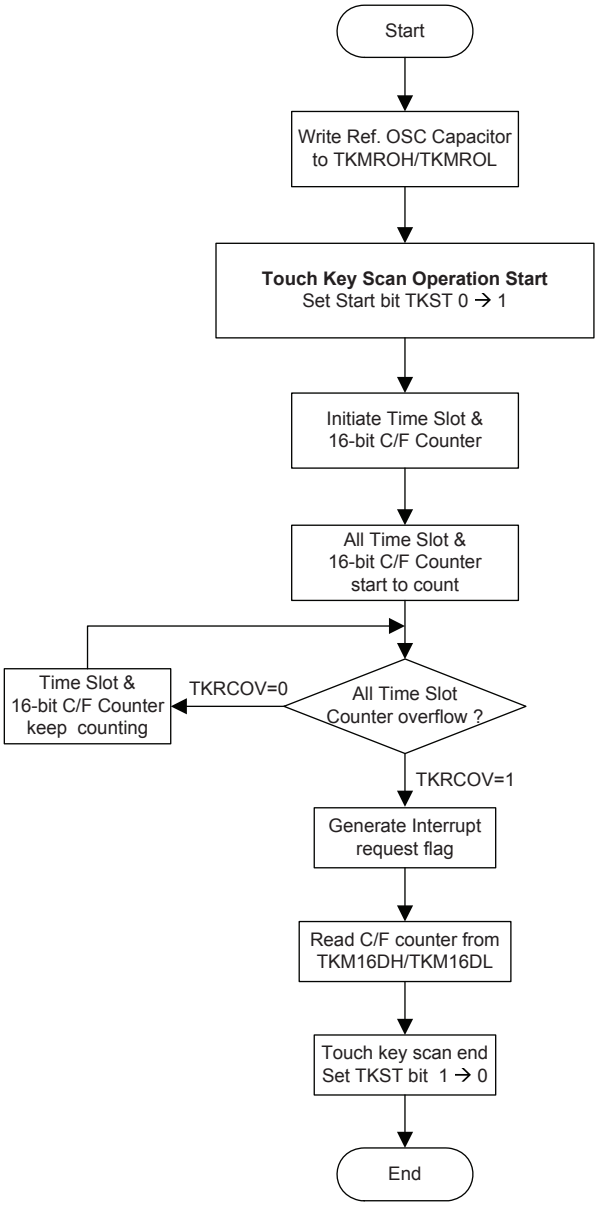
During this reference clock fixed interval, the number of clock cycles generated by the sense oscillator is measured, and it is this value that is used to determine if a touch action has been made or not. At the end of the fixed reference clock time interval a Touch Key interrupt signal will be generated.

The touch key module 16-bit C/F counter, 16-bit counter, 5-bit time slot unit period counter in the module will be automatically cleared when the TKST bit is cleared to zero, but the 8-Bit programmable time slot counter will not be cleared. The overflow time is setup by user. When the TKST bit changes from low to high, the 16-bit C/F counter, 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot timer counter will be automatically switched on.

The key oscillator and reference oscillator in the module will be automatically stopped and the 16-bit C/F counter, 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot timer counter will be automatically switched off when the time slot counter overflows. The clock source for the time slot counter is sourced from the reference oscillator or $f_{SYS}/4$ which is selected using the MTSS bit in the TKMC1 register. The reference oscillator and key oscillator will be enabled by setting the MROEN bit and MKOEN bits in the TKMC1 register.

When the time slot counter in the touch key module overflows, an actual touch key interrupt will take place. The touch keys mentioned here are the keys which are enabled.

Touch Key Scan Operation Flowchart



Touch Key Scan Operation Flowchart

Touch Key Interrupt

The touch key only has single interrupt, when the touch key module time slot counter overflows, an actual touch key interrupt will take place. The touch keys mentioned here are the keys which are enabled. The 16-bit C/F counter, 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot counter in the module will be automatically cleared.

The TKCFOV flag which is the 16-bit C/F counter overflow flag will go high when the Touch Key Module 16-bit C/F counter overflows. As this flag will not be automatically cleared, it has to be cleared by the application program.

The TK16OV flag which is the 16-bit counter overflow flag will go high when the 16-bit counter overflows. As this flag will not be automatically cleared, it has to be cleared by the application program. More details regarding the touch key interrupts are located in the interrupt section of the datasheet.

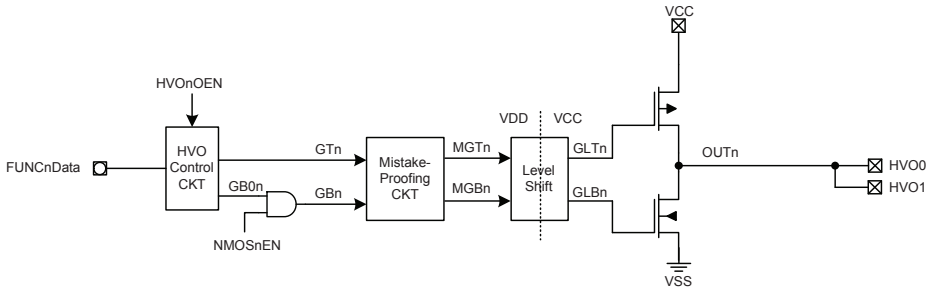
Progrsmming Considerations

After the relevant registers are setup, the touch key detection process is initiated the changing the TKST Bit from low to high. This will enable and synchronise all relevant oscillators. The TKRCOV flag which is the time slot counter flag will go high when the counter overflows. When this happens an interrupt signal will be generated.

When the external touch key size and layout are defined, their related capacitances will then determine the sensor oscillator frequency.

High Voltage Driver Output

The device contains a high voltage driver circuit, which has two groups of high voltage output. Each group is mainly composed of a mistake-proofing circuit and level shift circuit. With the two integrated 12V high voltage process level shift circuits, the high voltage driver provides two output pins, HVO0~HVO1, which can be used for driving LEDs using by PMOS.



Note: 1. n=0 or 1.

2. The FUNC0Data is sourced from PWM0/PWM1/HVO0 pin, while the FUNC1Data is sourced from PWM1/PWM0/HVO1 pin, and acted as the control signal output High/Low data source.

3. It is recommended to clear the NMOSnEN bit to zero when driving LEDs.

High Voltage Driver Output Block Diagram

High Voltage Driver Output Registers

The overall operation of the high voltage driver output is controlled using the HVOC register and HVOC1 register. The HVOC register controls the high voltage driver output enable/disable operation. The HVOC1 register is used to select the HVOn pin source and high voltage driver output data.

- **HVOC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	NMOS1EN	HVO1OEN	NMOS0EN	HVO0OEN
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as "0"

Bit 3 **NMOS1EN**: NMOS1 output enable control
0: NMOS1 is always disabled
1: NMOS1 is controlled by GB01 signal
Note: When the PWM0 or PWM1 pin is selected for the high voltage output, the NMOS1EN bit should be cleared to zero.

Bit 2 **HVO1OEN**: High voltage driver output 1 enable control
0: Disable
1: Enable
When the HVO1OEN bit is cleared to zero, the HVO1 pin will be in a floating status.

Bit 1 **NMOS0EN**: NMOS0 output enable control
0: NMOS0 is always disabled
1: NMOS0 is controlled by GB00 signal
Note: When the PWM0 or PWM1 pin is selected as the high voltage output, the NMOS0EN bit should be cleared to zero.

Bit 0 **HVO0OEN**: High voltage driver output 0 enable control
0: Disable
1: Enable
When the HVO0OEN bit is cleared to zero, the HVO0 pin will be in a floating status.

- **HVOC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	HVO1S1	HVO1S0	HVO0S1	HVO0S0	—	—	HVOD1	HVOD0
R/W	R/W	R/W	R/W	R/W	—	—	R/W	R/W
POR	0	0	0	0	—	—	0	0

Bit 7~6 **HVO1S1~HVO1S0**: High voltage driver output 1 selection
00: HVO1
01: HVO1
10: PWM0
11: PWM1

Bit 5~4 **HVO0S1~HVO0S0**: High voltage driver output 0 selection
00: HVO0
01: HVO0
10: PWM0
11: PWM1

Bit 3~2 Unimplemented, read as "0"

Bit 1 **HVOD1**: High voltage driver output 1 data
1: HVO1=VCC
0: HVO1=VSS

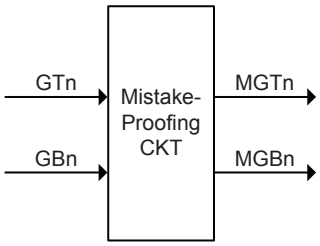
Bit 0 **HVOD0**: High voltage driver output 0 data
1: HVO0=VCC
0: HVO0=VSS

Functional Description

The High voltage driver output on the HVOn pins are used to drive the external power MOS by the HVOnOEN bit or using FUNCnData output to control power MOS loads.

Mistake-Proofing Circuit for High Voltage Driver

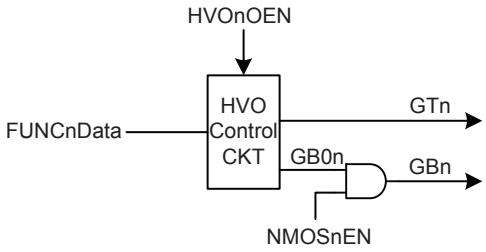
Incorrect write operations or external factors such an ESD condition, may cause incorrect on/off control resulting in the top and bottom sides of external transistor being both turned on simultaneously. A mistake-proof circuit is provided to avoid such situation.



GTn	GBn	MGTn	MGBn	HVOn
0	0	0	0	VCC
0	1	1	1	VSS
1	0	1	0	Floating
1	1	1	1	VSS

Note: 0: MOS off; 1: MOS on.

High Voltage Driver Output Control Circuit

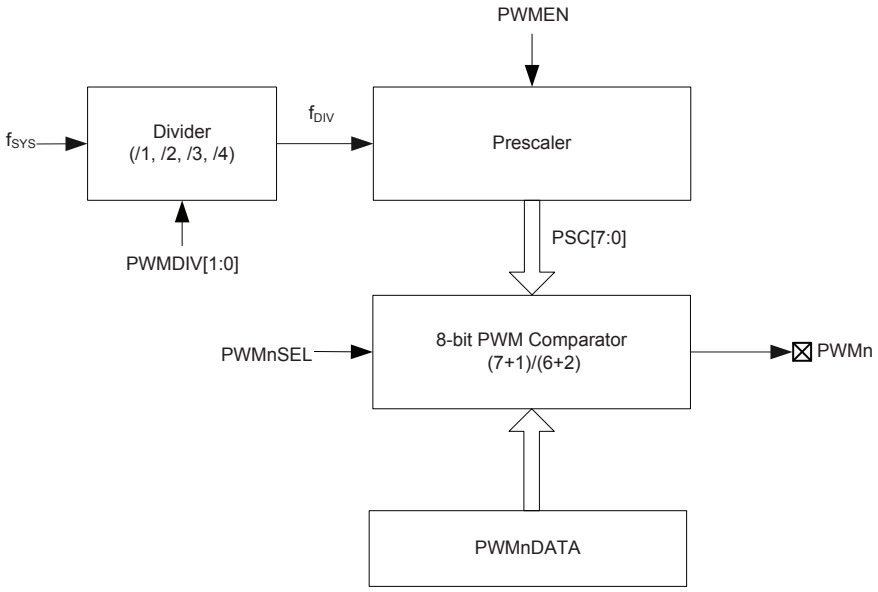


HVOnOEN	HVO Function data	NMOSnEN	GTn	GBn	HVOn
1	0	1	1	1	VSS
1	1	1	0	0	VCC
1	0	0	1	0	Floating
1	1	0	0	0	VCC
0	x	x	1	0	Floating

"x ": Don't care

Pulse Width Modulator

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



PWM Block Diagram

PWM Registers Description

The two registers control the overall operation of the Pulse Width Modulator. These are the data register, PWMnDATA and a control register, PWMC. The frequency of the PWM counter is sourced from the prescaler output signal.

- **PWMC Register**

Bit	7	6	5	4	3	2	1	0
Name	PWMEN	PWMDIV1	PWMDIV0	—	—	—	PWM1SEL	PWM0SEL
R/W	R/W	R/W	R/W	—	—	—	R/W	R/W
POR	0	0	0	—	—	—	0	0

Bit 7 **PWMEN**: PWM enable control
0: Disable
1: Enable

Bit 6~5 **PWMDIV1 ~ PWMDIV0**: f_{DIV} Frequency selection
00: $f_{DIV} = f_{SYS}$
01: $f_{DIV} = f_{SYS}/2$
10: $f_{DIV} = f_{SYS}/3$
11: $f_{DIV} = f_{SYS}/4$

Bit 4~2 Unimplemented, read as "0"

Bit 1 **PWM1SEL**: PWM1 type selection
0: (6+2) mode
1: (7+1) mode

Bit 0 **PWM0SEL**: PWM0 type selection
0: (6+2) mode
1: (7+1) mode

- **PWMnDATA Register (n= 0 or 1)**

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**: PWM data bits

PWM Operation

The PWMC register and PWMnDATA register are assigned to each Pulse Width Modulator channel. The PWMnDATA register represent the overall duty cycle of one modulation cycle of the output waveform, should be placed. To increase the PWM modulation frequency, each modulation cycle is subdivided into two or four individual modulation subsections, known as the 7+1 mode or 6+2 mode respectively. The required mode and the enable/disable control for each PWM channel is selected using the PWMC register. Note that when using the PWM, it is only necessary to write the required value into the PWMnDATA register and select the required mode setup and enable/disable control using the PWMC register, the subdivision of the waveform into its sub-modulation cycles is implemented automatically within the microcontroller hardware. The PWM clock source is the system clock f_{SYS}.

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

The modulation frequency, cycle frequency and cycle duty of the PWM output signal are summarized in the following table.

PWM Modulation Frequency	PWM Cycle Frequency	PWM Cycle Duty
$f_{DIV}/64$ for (6+2) bits mode	$f_{DIV}/256$	(PWMnDATA register value)/256
$f_{DIV}/128$ for (7+1) bits mode		

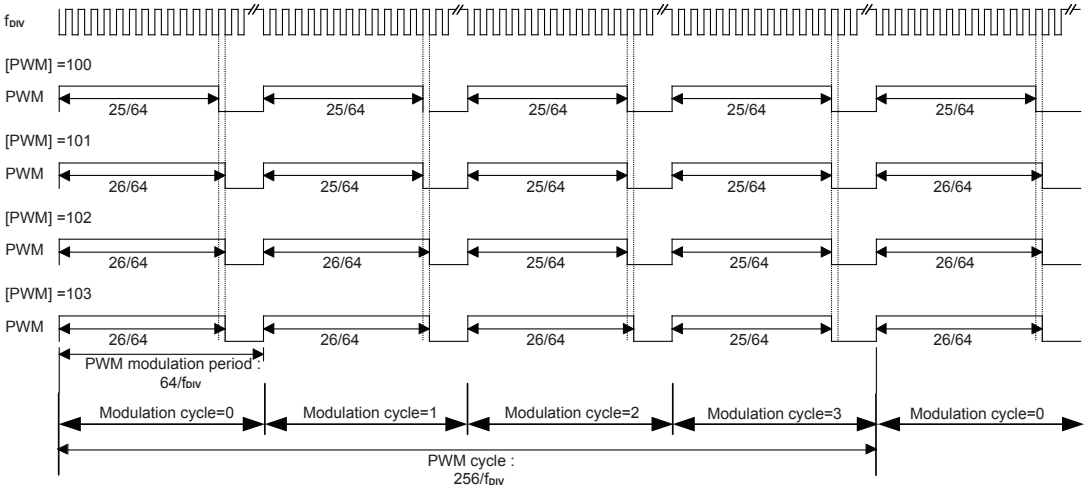
(6+2) Bits PWM Mode Modulation

A (6+2) bits mode PWM cycle is divided into four modulation cycles, which are named as Modulation cycle 0~Modulation cycle 3. Each modulation cycle has 64 PWM input clock period. In a (6+2) bits PWM mode, the contents of the PWM register is divided into two groups. Group 1 of the PWM register is denoted by DC which is the value of D7~D2 bits in the PWMnDATA register. The group 2 is denoted by AC which is the value of D1~D0 bits in the PWMnDATA register. In a (6+2) bits mode PWM cycle, the duty cycle of each modulation cycle is shown in the following table.

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

6+2 Mode Modulation Cycle Values

The following diagram illustrates the waveforms associated with the (6+2) bits mode of PWM operation. It is important to note how the single PWM cycle is subdivided into 4 individual modulation cycles, numbered from 0~3 and how the AC value is related to the PWM value. The waveforms of PWM outputs are as shown.



(6+2) Bits PWM Mode Modulation Waveform

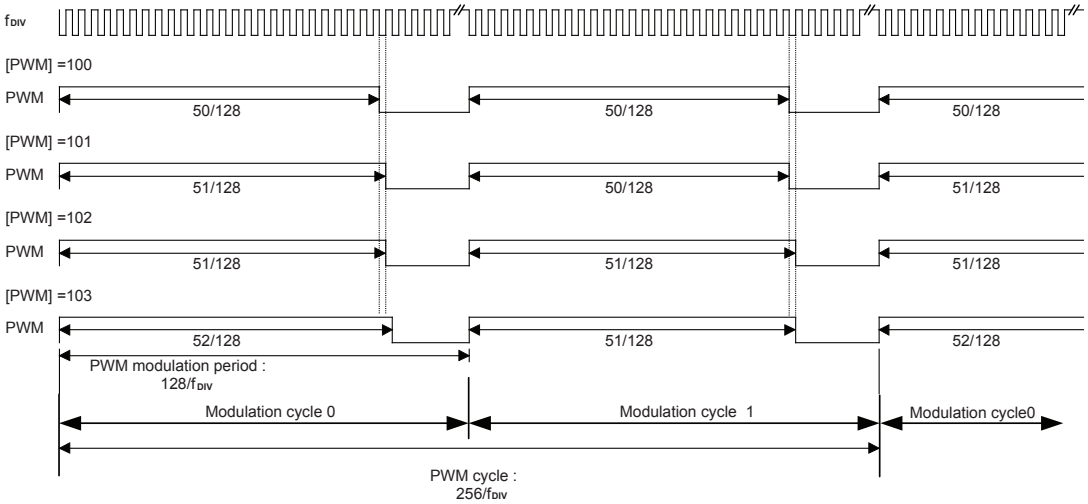
(7+1) Bits PWM Mode Modulation

A (7+1) bits mode PWM cycle is divided into two modulation cycles, which is named as Modulation cycle 0~Modulation cycle 1. Each modulation cycle has 128 PWM input clock period. In a (7+1) bits PWM mode, the contents of the PWM register is divided into two groups. Group 1 of the PWM register is denoted by DC which is the value of D7~D1 bits in the PWMnDATA register. The group 2 is denoted by AC which is the value of D0 bit in the PWMnDATA register. In a (7+1) bits mode PWM cycle, the duty cycle of each modulation cycle is shown in the following table.

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

(7+1) Bits Mode Modulation Cycle Values

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



(7+1) Bits PWM Mode Modulation Waveform

PWM Output Control

The PWM outputs are pin-shared with the HVOn pins. To operate as a PWM output and not as a high voltage output, the correct bits must be set in the HVOC1 register.

- **HVOC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	HVO1S1	HVO1S0	HVO0S1	HVO0S0	—	—	HVOD1	HVOD0
R/W	R/W	R/W	R/W	R/W	—	—	R/W	R/W
POR	0	0	0	0	—	—	0	0

Bit 7~6 **HVO1S1~HVO1S0**: High voltage driver output 1 selection
00: HVO1
01: HVO1
10: PWM0
11: PWM1

Bit 5~4 **HVO0S1~HVO0S0**: High voltage driver output 0 selection
00: HVO0
01: HVO0
10: PWM0
11: PWM1

Bit 3~2 Unimplemented, read as "0"

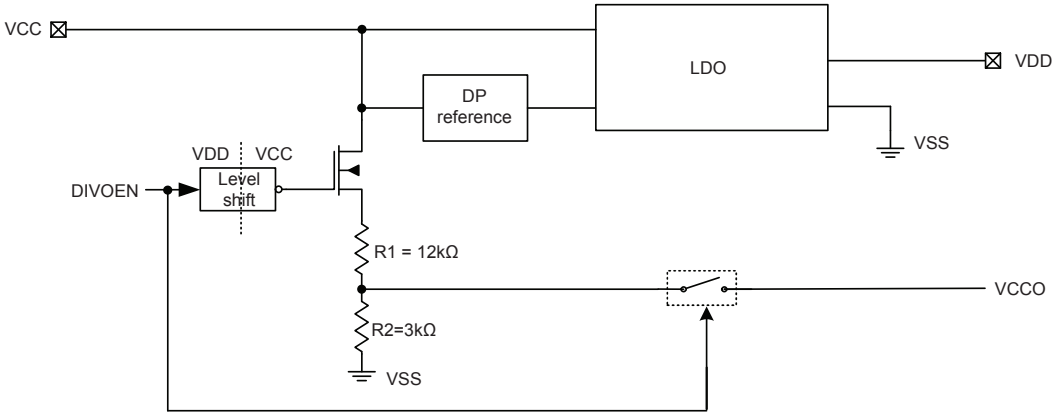
Bit 1 **HVOD1**: High voltage driver output 1 data
Described elsewhere

Bit 0 **HVOD0**: High voltage driver output 0 data
Described elsewhere

Low Dropout Regulator – LDO

The system is supplied by a higher system voltage approximately 6V to 12V on the input pin VCC.

An internal LDO reduces this higher voltage to a 5V level which is supplied on the output pin VDD. This lower voltage level is used by the internal logic circuits but as it can supply up to 40mA it can also be used by external circuitry.



Note: The Voltage Divider=R1:R2=12kΩ:3kΩ=4:1, V_{CCO}= R2/(R1+R2)×V_{CC} = 0.2×V_{CC}.

LDO Control Register

- **DIVOC Register**

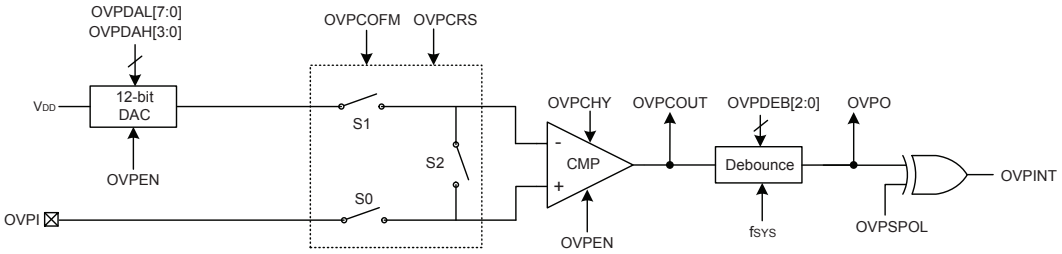
Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	—	DIVOEN
R/W	—	—	—	—	—	—	—	R/W
POR	—	—	—	—	—	—	—	0

Bit 7 ~1 Unimplemented, read as "0"

Bit 0 **DIVOEN**: V_{CC} divider output enable control
0: Disable
1: Enable
When the DIVOEN bit is cleared to 0, the VCCO is in a floating status.
When the DIVOEN bit is set high, the VCCO voltage is equal to V_{CC}/5.

Over Voltage Protection – OVP

The device includes an over voltage protection function, also known as OVP, which provides a protection mechanism for applications. To prevent the operating voltage from exceeding a specific level, the voltage on the OVPI pin is compared with a reference voltage generated by a 12-bit DAC. When an over voltage event occurs, an OVP interrupt will be generated if the corresponding interrupt control is enabled.



Over Voltage Protection Circuit

The on/off control for the switches S0, S1 and S2 is summarised in the following table.

OVPCOFM	OVPCRS	S0	S1	S2
0	x	ON	ON	OFF
1	0	OFF	ON	ON
1	1	ON	OFF	ON

"x": Don't care

Over Voltage Protection Operation

The source voltage is supplied on the OVPI pin and then connected to one input of the comparator. A DAC is used to generate a reference voltage. The comparator compares the reference voltage with the input voltage to produce the OVPO signal.

Over Voltage Protection Control Registers

Overall operation of the over voltage protection is controlled using several registers. One register is used to provide the reference voltages for the over voltage protection circuit. The remaining two registers are control registers which are used to control the OVP function, DAC reference voltage selection, comparator de-bounce time, comparator hysteresis function together with the comparator input offset calibration.

Register Name	Bit							
	7	6	5	4	3	2	1	0
OVPC0	OVPCOUT	OVPSPOL	OVPEN	OVPCHY	—	OVPDEB2	OVPDEB1	OVPDEB0
OVPC1	OVPO	OVPCOFM	OVPCRS	OVPCOF4	OVPCOF3	OVPCOF2	OVPCOF1	OVPCOF0
OVPDAL	D7	D6	D5	D4	D3	D2	D1	D0
OVPDAH	—	—	—	—	D3	D2	D1	D0

OVP Control Registers List

- **OVPC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	OVPCOUT	OVPSPOL	OVPEN	OVPCHY	—	OVPDEB2	OVPDEB1	OVPDEB0
R/W	R	R/W	R/W	R/W	—	R/W	R/W	R/W
POR	0	0	0	0	—	0	0	0

Bit 7 **OVPCOUT**: OVP comparator output
0: Positive input voltage < negative input voltage
1: Positive input voltage > negative input voltage

Bit 6 **OVPSPOL**: OVPO polarity control
0: Non-inverted
1: Inverted

Bit 5 **OVPEN**: OVP function enable control
0: Disable
1: Enable
If the OVPEN bit is cleared to 0, the over voltage protection function is disabled and no power will be consumed. This results in the comparator and D/A converter of OVP both being switched off.

Bit 4 **OVPCHY**: OVP comparator hysteresis function enable control
0: Disable
1: Enable

Bit 3 Unimplemented, read as "0"

Bit 2~0 **OVPDEB2~OVPDEB0**: OVP comparator debounce time control
000: No debounce
001 : (1~2) × $1/f_{SYS}$
010: (3~4) × $1/f_{SYS}$
011: (7~8) × $1/f_{SYS}$
100: (15~16) × $1/f_{SYS}$
101: (31~32) × $1/f_{SYS}$
110: (63~64) × $1/f_{SYS}$
111: (127~128) × $1/f_{SYS}$

- **OVPC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	OVPO	OVPCOFM	OVPCRS	OVPCOF4	OVPCOF3	OVPCOF2	OVPCOF1	OVPCOF0
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	1	0	0	0	0

Bit 7 **OVPO**: OVP comparator debounce output
The OVPO is the debounce version of OVPCOUT.

Bit 6 **OVPCOFM**: OVP comparator normal operation or input offset voltage calibration mode selection
0: Normal operation
1: Input offset voltage calibration mode

Bit 5 **OVPCRS**: OVP comparator input offset voltage calibration reference selection
0: Input reference voltage comes from negative input
1: Input reference voltage comes from positive input

Bit 4~0 **OVPCOF4~OVPCOF0**: OVP comparator input offset voltage calibration control

- **OVPDAL 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**: OVP DAC output voltage control

Note: The data in the OVPDAL register is only written into shadow buffer, and the data will be copied into the OVPDAL register until writing data into the OVPDAH register.

- **OVPDAH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	D3	D2	D1	D0
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as "0"

Bit 3~0 **D3~D0**: OVP DAC output voltage control

DAC $V_{OUT} = (V_{DD}/4096) \times \{OVPDAH[3:0],OVPDAL[7:0]\}$

Input Offset Calibration

The OVPCOFM bit in the OVPC1 register is used to select the OVP comparator operating mode, normal operation or input offset calibration mode. If set the bit high, the comparator will enter the offset voltage calibration mode. It is need to note that before offset calibration, the hysteresis voltage should be zero by clearing the OVPCHY bit to 0. Because the OVPI pin is pin-shared with I/O or other pin functions, it should be configured as comparator input first. For comparator input offset calibration, the procedures are summarised in the following steps.

Step1: Set OVPCOFM=1, OVPCRS=1, the OVP is now in the comparator calibration mode, S0 and S2 on. To make sure V_{OS} as minimise as possible after calibration, the input reference voltage in calibration mode should be the same as input DC operating voltage in normal mode operation.

Step2: Set OVPCOF[4:0] =00000 then read the OVPO bit status.

Step3: Let OVPCOF[4:0]=OVPCOF[4:0]+1 then read the OVPO bit status, if OVPO is changed, record the OVPCOF[4:0] data as V_{OS1}.

Step4: Set OVPCOF[4:0] =11111 then read the OVPO bit status.

Step5: Let OVPCOF[4:0]=OVPCOF[4:0]-1 then read the OVPO bit status, if OVPO data is changed, record the OVPCOF[4:0] data as V_{OS2}.

Step6: Restore $V_{OS} = (V_{OS1} + V_{OS2})/2$ to the OVPCOF[4:0] bits. The calibration is finished.

If $(V_{OS1} + V_{OS2})/2$ is not integral, discard the decimal. Residue $V_{OS} = V_{OUT} - V_{IN}$

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a touch action or Timer/Counter overflow 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 an external interrupt and several internal interrupts functions. The external interrupts are generated by the action of the external INT pin, while the internal interrupts are generated by various internal functions such as the Touch Keys, Timer/Event Counter, Time Base, and Over Voltage Protection.

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 series of registers, located in the Special Purpose Data Memory, as shown in the accompanying table. The number of registers falls into two categories. The first is the INTC0~INTC1 registers which setup the primary interrupts, the second is the INTEG register which setup the external interrupt trigger edge type.

Each 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
Global	EMI	—
INT Pin	INTE	INTF
Touch Key	TKME	TKMF
Timer/Event Counter	TE	TF
Over Voltage Protection	OVPE	OVPF
Time Base	TBE	TBF

Interrupt Register Bit Naming Conventions

Register Name	Bit							
	7	6	5	4	3	2	1	0
INTEG	—	—	—	—	—	—	INTS1	INTS0
INTC0	—	TF	TKMF	INTF	TE	TKME	INTE	EMI
INTC1	—	—	TBF	OVPF	—	—	TBE	OVPE

Interrupt Registers List

- **INTEG Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	INTS1	INTS0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **INTS1~INTS0**: Interrupt edge control for INT pin
00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

- **INTC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	TF	TKMF	INTF	TE	TKME	INTE	EMI
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as "0"

Bit 6 **TF**: Timer/Event Counter interrupt request flag
0: No request
1: Interrupt request

Bit 5 **TKMF**: Touch key module interrupt request flag
0: No request
1: Interrupt request

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

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

Bit 2 **TKME**: Touch key module interrupt control
0: Disable
1: Enable

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

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

- **INTC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	TBF	OVPF	—	—	TBE	OVPE
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **TBF**: Time Base interrupt request flag
0: No request
1: Interrupt request

Bit 4 **OVPF**: OVP interrupt request flag
0: No request
1: Interrupt request

Bit 3~2 Unimplemented, read as "0"

Bit 1 **TBE**: Time Base interrupt control
0: Disable
1: Enable

Bit 0 **OVPE**: OVP interrupt control
0: Disable
1: Enable

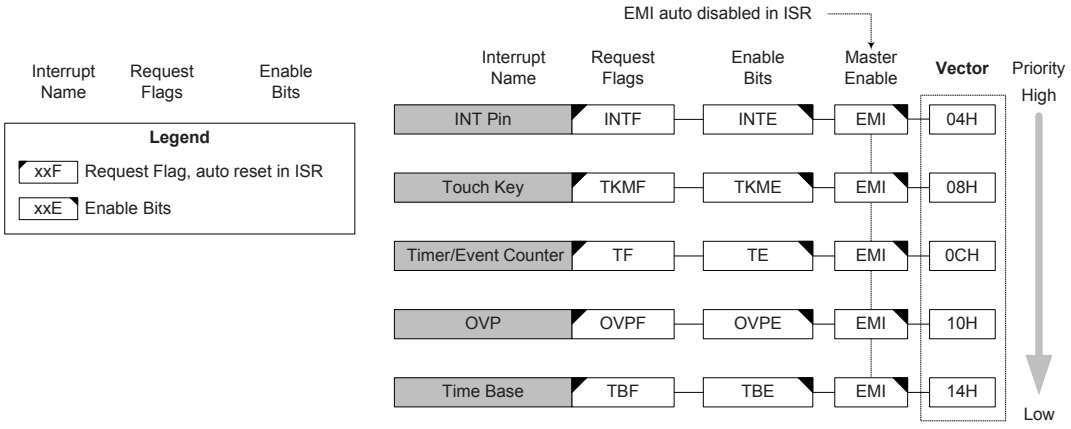
Interrupt Operation

When the conditions for an interrupt event occur, such as a Timer/Event Counter or touch key time slot counter overflows 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.

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



Interrupt Structure

External Interrupt

The external interrupts are controlled by signal transitions on the INT pin. An external interrupt request will take place when the external interrupt request flag, INTF, are set, which will occur when a transition, whose type is chosen by the edge select bits, appears on the external interrupt pin. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and respective external interrupt enable bit, INTE, must first be set. Additionally the correct interrupt edge type must be selected using the INTEG register to enable the external interrupt function and to choose the trigger edge type. As the external interrupt pin is pin-shared with I/O pin, it can only be configured as external interrupt pin if their external interrupt enable bit in the corresponding interrupt register has been set. The pin must also be setup as an input by setting the corresponding bit in the port control register. When the interrupt is enabled, the stack is not full and the correct transition type appears on the external interrupt pin, a subroutine call to the external interrupt vector, will take place. When the interrupt is serviced, the external interrupt request flag, INTF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor selection on the external interrupt pin will remain valid even if the pin is used as an external interrupt input.

The INTEG register is used to select the type of active edge that will trigger the external interrupt. A choice of either rising or falling or both edge types can be chosen to trigger an external interrupt. Note that the INTEG register can also be used to disable the external interrupt function.

Touch Key Module Interrupt

A Touch Key Module Interrupt request will take place when the Touch Key Module Interrupt request flag, TKMF, is set, which occurs when the touch key time slot counter overflows. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and the Touch Key Module Interrupt enable bit, TKME, must first be set. When the interrupt is enabled, the stack is not full and the touch key time slot counter overflows, a subroutine call to the respective Interrupt vector, will take place. When the interrupt is serviced, the Touch Key Module Interrupt flag, TKMF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

Timer/Event Counter Interrupt

An actual Timer/Event Counter interrupt request will take place when the Timer/Event Counter interrupt request flag, TF, 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, TE, must first be set. When the interrupt is enabled, the stack is not full and a Timer/Event Counter overflows, a subroutine call to the respective Interrupt vector, will take place. When the interrupt is serviced, the Timer/Event Counter Interrupt flag, TF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

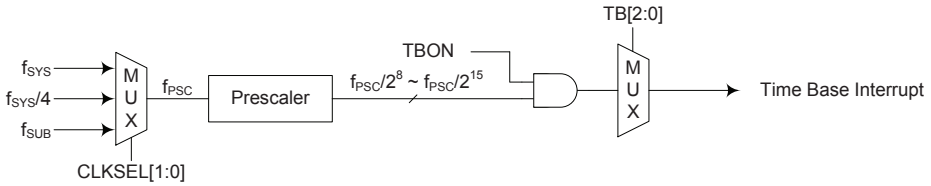
Over Voltage Protection Interrupt

The OVP Interrupt is controlled by detecting the input voltage. An OVP Interrupt request will take place when the OVP Interrupt request flag, OVPF, is set, which occurs when a large voltage is detected. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and OVP Interrupt enable bit, OVPE, must first be set. When the interrupt is enabled, the stack is not full and a large voltage is detected, a subroutine call to the OVP Interrupt vector, will take place. When the interrupt is serviced, the OVP Interrupt flag, OVPF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

Time Base Interrupt

The function of the Time Base Interrupts is to provide regular time signal in the form of an internal interrupt. They are controlled by the overflow signals from their respective timer functions. When these happens their respective interrupt request flag, TBF will be set. To allow the program to branch to their respective interrupt vector addresses, the global interrupt enable bit, EMI and Time Base enable bit, TBE, must first be set. When the interrupt is enabled, the stack is not full and the Time Base overflows, a subroutine call to their respective vector locations will take place. When the interrupt is serviced, the respective interrupt request flag, TBF, will be automatically reset and the EMI bit will be cleared to disable other interrupts.

The purpose of the Time Base Interrupt is to provide an interrupt signal at fixed time periods. Its clock source, f_{PSC}, originates from the internal clock source f_{SYS}, $f_{SYS}/4$ or f_{SUB} and then passes through a divider, the division ratio of which is selected by programming the appropriate bits in the TBC register to obtain longer interrupt periods whose value ranges. The clock source which in turn controls the Time Base interrupt period is selected using the CLKSEL1~CLKSEL0 bits in the PSCR register.



Time Base Interrupt

- **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 selection
00: f_{SYS}
01: $f_{SYS}/4$
1x: f_{SUB}

- **TBC Register**

Bit	7	6	5	4	3	2	1	0
Name	TBON	—	—	—	—	TB2	TB1	TB0
R/W	R/W	—	—	—	—	R/W	R/W	R/W
POR	0	—	—	—	—	0	0	0

Bit 7 **TBON**: Time Base Control
0: Disable
1: Enable

Bit 6~3 Unimplemented, read as "0"

Bit 2~0 **TB2~TB0**: Select Time Base Time-out Period
000: $2^8/f_{PSC}$
001: $2^9/f_{PSC}$
010: $2^{10}/f_{PSC}$
011: $2^{11}/f_{PSC}$
100: $2^{12}/f_{PSC}$
101: $2^{13}/f_{PSC}$
110: $2^{14}/f_{PSC}$
111: $2^{15}/f_{PSC}$

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 transition on the external interrupt pin 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 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 – Dimming and Toning Touch Desk Lights

Introduction

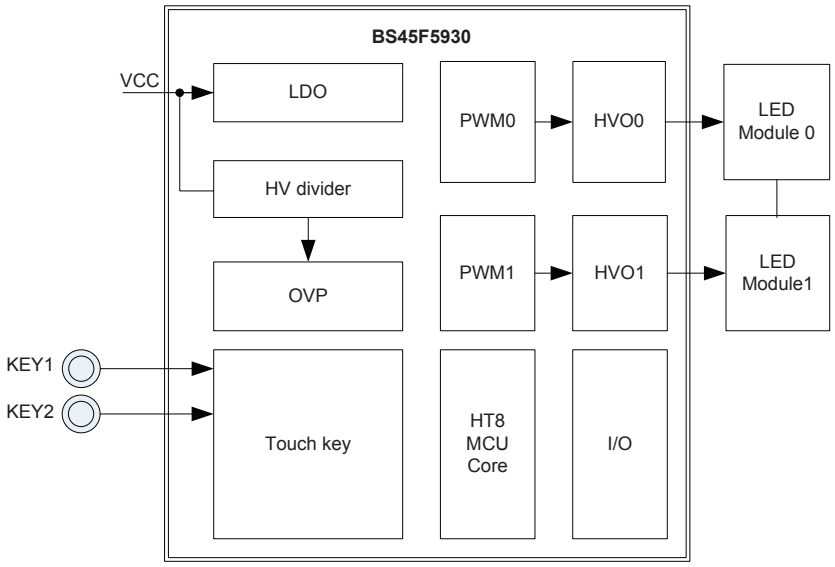
The dimming and toning touch desk lights have mainly three functions such as touch detecting, PWM regulation and high voltage driving. If the device with the input voltage of 12V has detected that the key is pressed, the light brightness will be changed. When the two LED modules use yellow light and white light respectively, different light colors will be generated through different yellow and white light ratios. The functional principle will be described as follows.

Functional Description

The system power supply is 12V. Each LED module has three serial LEDs and a current limiting resistor, which can limit the maximum current flowing through LEDs to prevent the LEDs from burning because of too large current. The device also provides over voltage protection function, which can detect the VCC voltage with High Voltage divider. If the VCC voltage is too high, the HVO output will be off to avoid burning LEDs because of too large current. In this application, the LED module 0 use white LED, while the LED module1 use warm white LED, users can adjust these two modules PWM duty to get different color temperature by mixing different brightness ratios.

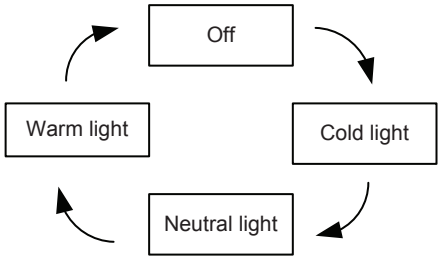
There are two touch keys in the light control section, and the KEY1 is used to adjust color temperature, while the KEY2 is used to adjust brightness.

Hardware Block Diagram



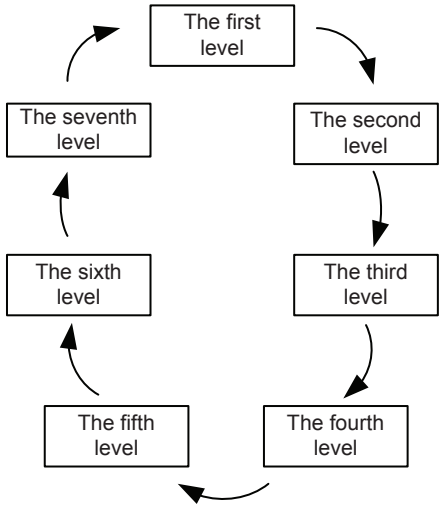
Color Temperature Adjustment Key (KEY1)

The KEY1 function setting is circular. After power on reset, the KEY1 is preseted to turn off. The light will be switched to cold light through touching the KEY1, then the light will be cut to neutral light through touching the KEY1 again and so on. This process will be shown in the following figure.



Dimming Key (KEY2)

The KEY2 function setting is circular. After power on reset, the KEY2 is preseted to the first level. The light will be switched to the second level through touching the KEY2, then the light will be switched to the third level through touching the KEY2 again and so on. There are seven brightness level options in the application, which is shown in the following figure.

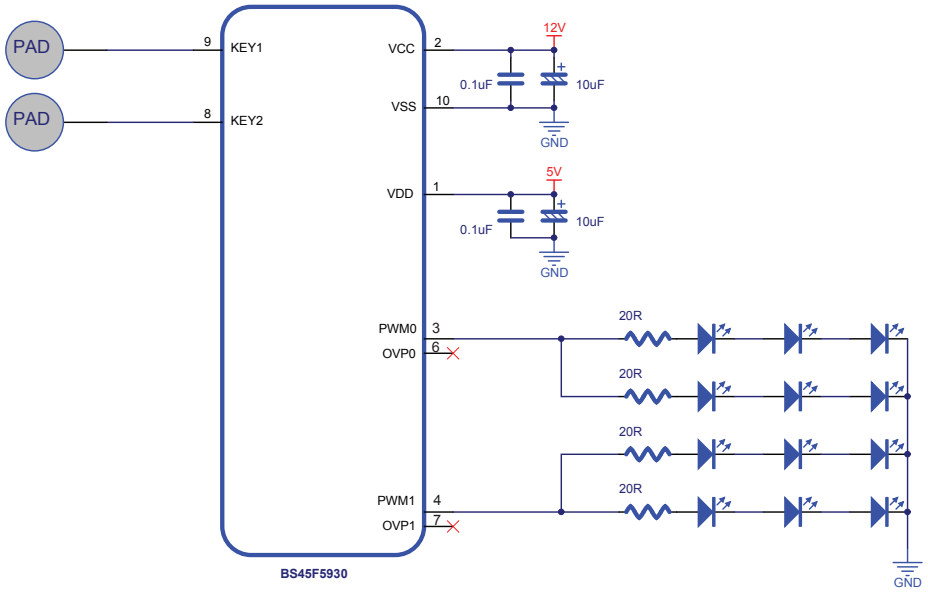


Color Temperature Definition

The color temperature definition is shown in the following table. The PWM duty cycle is marked by number, which can change brightness without affecting color temperature in the same color temperature mode. When the brightness is enhanced, the duty cycle in two LED modules will be not increased by an equal ratio. The values in the following table are for reference only and the different LED beads will have different setting.

PWM Duty	Warm Light		Neutral Light		Cold Light	
	Warm White	White	Warm White	White	Warm White	White
The first level	9%	2%	5%	5%	2%	9%
The second level	23%	3%	12%	12%	3%	23%
The third level	36%	4%	20%	20%	4%	36%
The fourth level	50%	6%	28%	28%	6%	50%
The fifth level	63%	7%	35%	35%	7%	63%
The sixth level	77%	9%	43%	43%	9%	77%
The seventh level	90%	10%	50%	50%	10%	90%

Hardware Circuit Diagram



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 three kinds of MOV instructions, data can be transferred from registers to the Accumulator and vice-versa as well as being able to move specific immediate data directly into the Accumulator. One of the most important data transfer applications is to receive data from the input ports and transfer data to the output ports.

Arithmetic Operations

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

Logical and Rotate 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 set as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

Other Operations

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

Instruction Set Summary

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

Table Conventions

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

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

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

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

Instruction Definition

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

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

Operation ACC ← ACC + [m] + C

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

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

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

Operation [m] ← ACC + [m] + C

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

ADD A,[m] Add Data Memory to ACC

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

Operation ACC ← ACC + [m]

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

ADD A,x Add immediate data to ACC

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

Operation ACC ← ACC + x

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

ADDM A,[m] Add ACC to Data Memory

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

Operation [m] ← ACC + [m]

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

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

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

Operation ACC ← ACC "AND" [m]

Affected flag(s) Z

AND A,x Logical AND immediate data to ACC

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

Operation ACC ← ACC "AND" x

Affected flag(s) Z

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

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

Operation [m] ← ACC "AND" [m]

Affected flag(s) Z

CALL addr Subroutine call

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

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

Affected flag(s): None

CLR [m] Clear Data Memory

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

Operation: [m] ← 00H

Affected flag(s): None

CLR [m].i Clear bit of Data Memory

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

Operation: [m].i ← 0

Affected flag(s): None

CLR WDT Clear Watchdog Timer

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

Operation: WDT cleared
TO ← 0
PDF ← 0

Affected flag(s): TO, PDF

CLR WDT1 Pre-clear Watchdog Timer

Description: The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect.

Operation: WDT cleared
TO ← 0
PDF ← 0

Affected flag(s): TO, PDF

CLR WDT2 Pre-clear Watchdog Timer

Description: The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect.

Operation: WDT cleared
TO ← 0
PDF ← 0

Affected flag(s): TO, PDF

CPL [m] Complement Data Memory

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

Operation: $[m] \leftarrow \overline{[m]}$

Affected flag(s): Z

CPLA [m] Complement Data Memory with result in ACC

Description Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

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

Affected flag(s) Z

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

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

Operation [m] ← 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

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

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

Operation [m] ← ACC − [m] − C

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

SDZ [m] Skip if decrement Data Memory is 0

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

Operation [m] ← [m] − 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] − 1
Skip if ACC=0

Affected flag(s) None

SET [m] Set Data Memory

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

Operation [m] ← FFH

Affected flag(s) None

SET [m].i Set bit of Data Memory

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

Operation [m].i ← 1

Affected flag(s) None

SIZ [m] Skip if increment Data Memory is 0

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

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

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

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

Operation Skip if [m].i ≠ 0

Affected flag(s) None

SUB A,[m] Subtract Data Memory from ACC

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

Operation ACC ← ACC − [m]

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

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

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

Operation [m] ← ACC − [m]

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

SUB A,x Subtract immediate data from ACC

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

Operation ACC ← ACC − x

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

SWAP [m] Swap nibbles of Data Memory

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

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

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

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

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

Affected flag(s) None

SZ [m] Skip if Data Memory is 0

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

Operation Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

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

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

Operation Skip if [m].i=0

Affected flag(s) None

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

Description The low byte of the program code (specific page) addressed by the table pointer pair (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

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

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s) Z

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

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

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s) Z

XOR A,x Logical XOR immediate data to ACC

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

Operation ACC ← ACC ″XOR″ x

Affected flag(s) Z

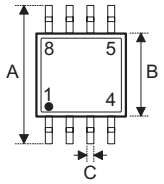
Package Information

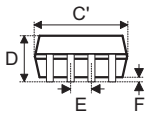
Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the Package/Carton Information.

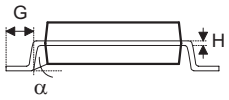
Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

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

8-pin SOP (150mil) Outline Dimensions



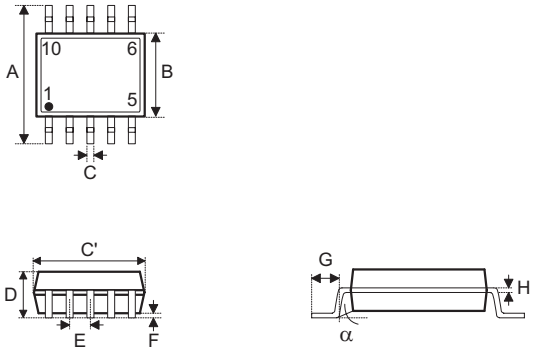




Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.236 BSC	—
B	—	0.154 BSC	—
C	0.012	—	0.020
C’	—	0.193 BSC	—
D	—	—	0.069
E	—	0.050 BSC	—
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6 BSC	—
B	—	3.9 BSC	—
C	0.31	—	0.51
C’	—	4.9 BSC	—
D	—	—	1.75
E	—	1.27 BSC	—
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°

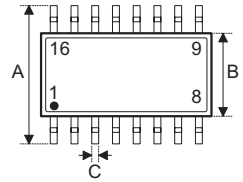
10-pin SOP (150mil) Outline Dimensions

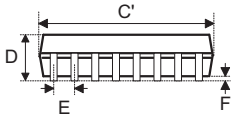


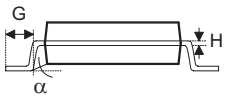
Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.236 BSC	—
B	—	0.154 BSC	—
C	0.012	—	0.018
C’	—	0.193 BSC	—
D	—	—	0.069
E	—	0.039 BSC	—
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6.00 BSC	—
B	—	3.90 BSC	—
C	0.30	—	0.45
C’	—	4.90 BSC	—
D	—	—	1.75
E	—	1.00 BSC	—
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°

16-pin NSOP (150mil) Outline Dimensions







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

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6 BSC	—
B	—	3.9 BSC	—
C	0.31	—	0.51
C’	—	9.9 BSC	—
D	—	—	1.75
E	—	1.27 BSC	—
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°