

**Technical Document**

- [Application Note](#)
  - [HA0075E MCU Reset and Oscillator Circuits Application Note](#)

**Features**

- Operating voltage: 2.7V~3.6V
- System clock: 4MHz~8MHz
- Crystal and RC system oscillator
- 12 I/O pins
- I<sup>2</sup>C/SPI Bus Serial Interface, shared with PB
- 2K×15 OTP Program Memory
- Between 2M×8 bit and 128K×8 bit flash type data memory
- 80×8 Data Memory
- Two 8-bit programmable timer counter with 8-stage prescaler and one time base counter
- 12-bit high quality voltage type D/A output
- PWM circuit direct drive speaker
- Watchdog Timer function
- 4-level subroutine nesting
- 2.7V Low voltage detection, tolerance 5%
- Integrated LDO regulator in HT83F10P/20P/40P/60P/80P
- Power-down function and wake-up feature reduce power consumption
- Up to 1μs (0.5μs) instruction cycle with 4MHz (8MHz) system clock at V<sub>DD</sub>= 3.6V
- 63 powerful instructions
- One reset pin
- Flash Data Memory can be re-programmed up to 100,000 times
- Flash Data Memory data retention > 10 years
- 44-pin QFP package

**General Description**

The flash type voice series of MCUs have OTP type Program Memory and Flash type Voice Memory.

The devices are 8-bit high performance microcontrollers which include a voice synthesizer and tone generator. They are designed for applications which require multiple I/Os and sound effects, such as voice and melody. The devices can provide various sampling rates and beats, tone levels, tempos for speech synthesizer and melody generator.

They also include two integrated high quality, voltage type DAC outputs and voltage type PWM outputs.

The devices are excellent solutions for versatile voice and sound effect product applications with their efficient MCU instructions providing the user with programming capability for powerful custom applications. The system frequency can be up to 8MHz at an operating voltage of 2.7V and include a power-down function to reduce power consumption.

The MCU flash voice memory capacity ranges from 2M×8 bit to 128K×8 bit, into which the user can download their voice data repeatedly.

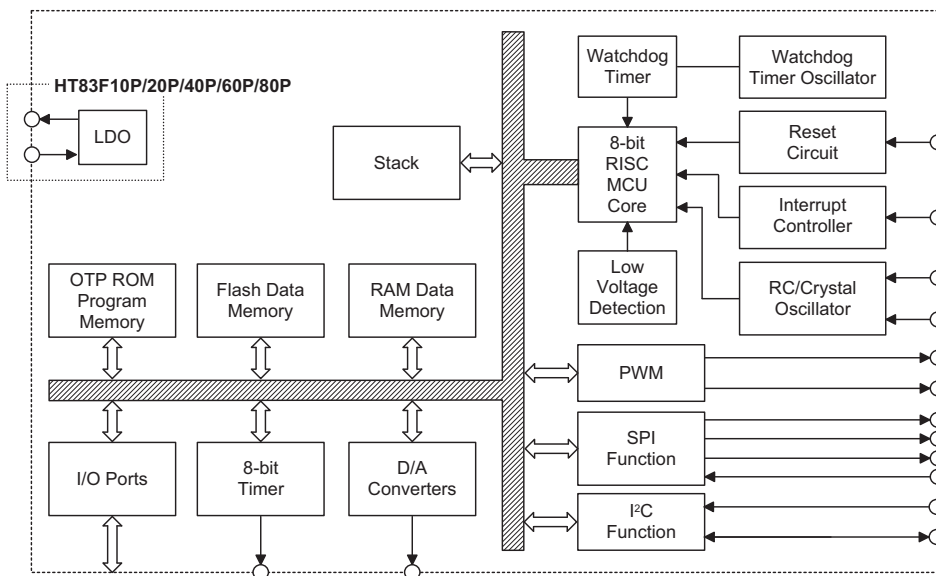
### Selection Table

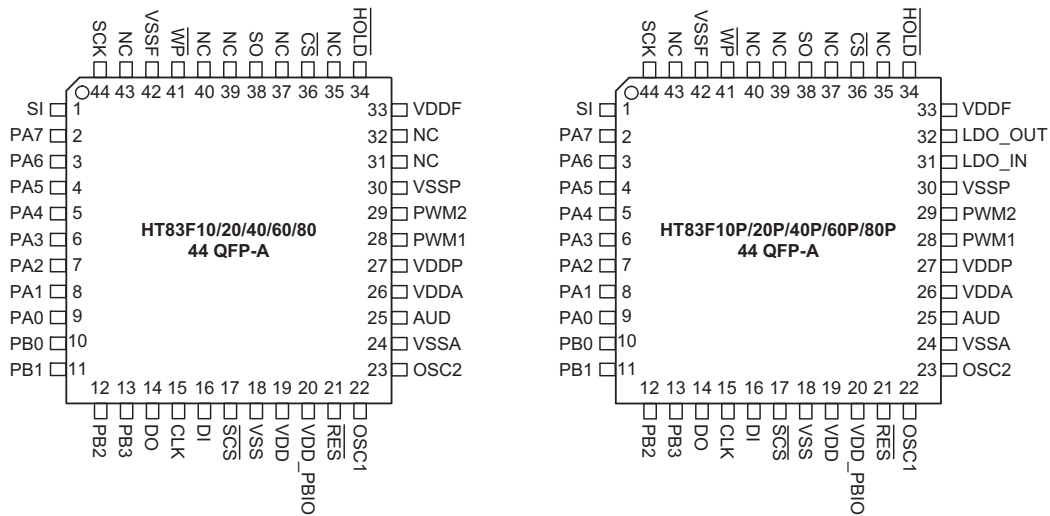
The devices include a comprehensive range of features, with most features common to all devices. The main features distinguishing them are Flash Voice Memory capacity. The functional differences between the devices are shown in the following table.

Part No.	VDD	VIN	OTP Program Memory	Data Memory	Flash Voice Memory	Voice Capacity	I/O	8-bit Timer	I <sup>2</sup> C/SPI	D/A	Package Types
HT83F10	2.7V~3.6V	—	2K×15	80×8	128K×8	32sec	12	2	√	12-bit, PWM	44QFP
HT83F10P	3.3V	3.6V~24V									
HT83F20	2.7V~3.6V	—	2K×15	80×8	256K×8	64sec	12	2	√	12-bit, PWM	44QFP
HT83F20P	3.3V	3.6V~24V									
HT83F40	2.7V~3.6V	—	2K×15	80×8	512K×8	128sec	12	2	√	12-bit, PWM	44QFP
HT83F40P	3.3V	3.6V~24V									
HT83F60	2.7V~3.6V	—	2K×15	80×8	1024K×8	256sec	12	2	√	12-bit, PWM	44QFP
HT83F60P	3.3V	3.6V~24V									
HT83F80	2.7V~3.6V	—	2K×15	80×8	2048K×8	512sec	12	2	√	12-bit, PWM	44QFP
HT83F80P	3.3V	3.6V~24V									

Note: For devices that exist in more than one package formats, the table reflects the situation for the larger package.  
Voice length is estimated by 32K-bit data rate, or 8K sampling rate and 4 bit ADPCM compress mode.

### Block Diagram



**Pin Assignment**

**Pin Description**

Pin Name	I/O	Options	Description
PA0~PA7	I/O	Wake-up, Pull-high or None	Bidirectional 8-bit I/O port. Each bit can be configured as a wake-up input by configuration option. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on this port have pull-high resistors.
PB0/SDO/SDA PB1/SCK/SCL PB2/SDI PB3/SCS	I/O	Pull-high or None	Bidirectional 4-bit I/O port. Each bit can be configured as a wake-up input by configuration option. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on this port have pull-high resistors. Pins PB0~PB3 are pin-shared with SPI flash control and I <sup>2</sup> C control pins SDO/SDA, SCK/SCL, SDI and SCS.
DO	O	—	Data output pin
CLK	O	—	Clock output pin.
DI	I	—	Data input pin.
SCS	O	—	Select signal.
AUD	O	CMOS	Audio output for driving external transistor or power amplifier.
PWM1, PWM2	O	—	PWM circuit direct speaker drive
RES	I	—	Schmitt trigger reset input. Active low
OSC1 OSC2	—	Crystal or RC	OSC1, OSC2 are connected to an external RC network or external crystal, determined by configuration option, for the internal system clock. If the RC system clock option is selected, pin OSC2 can be used to measure the system clock at 1/4 frequency.
VDD	—	—	Positive digital power supply
VSS	—	—	Negative digital power supply, ground
VSSP	—	—	PWM negative power supply, ground
VDDP	—	—	PWM positive power supply
VSSA	—	—	Negative DAC circuit power supply, ground
VDDA	—	—	Positive DAC circuit Power supply

Pin Name	I/O	Options	Description
VDD_PBIO	—	—	PB I/O external positive power supply (determine by option)
LDO_OUT	O	—	LDO output
LDO_IN	I	—	LDO input
$\overline{\text{CS}}$	I	—	Flash data memory chip select pin
SI	I	—	Flash data memory data input pin
SO	O	—	Flash data memory data output pin
SCK	I	—	Flash data memory clock input pin
$\overline{\text{HOLD}}$	I	—	Hold, pause the device without deselecting Flash data memory
$\overline{\text{WP}}$	I	—	Flash data memory write protect pin
VDDF	—	—	Positive Flash data memory Power supply
VSSF	—	—	Negative Flash data memory Power supply, ground

Note: Each pin on PA can be programmed through a configuration option to have a wake-up function.  
Individual pins can be selected to have pull-high resistors.

### Absolute Maximum Ratings

Supply Voltage.....	$V_{SS}+2.7V$ to $V_{SS}+3.6V$	Storage Temperature.....	$-50^{\circ}\text{C}$ to $+125^{\circ}\text{C}$
Input Voltage.....	$V_{SS}-0.3V$ to $V_{DD}+0.3V$	Operating Temperature.....	$-40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$
$I_{OL}$ Total .....	150mA	$I_{OH}$ Total .....	-100mA
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

 $T_a=25^{\circ}\text{C}$ 

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		$V_{DD}$	Conditions				
$V_{DD}$	Operating Voltage	—	$f_{SYS}=4\text{MHz}/8\text{MHz}$	2.7	—	3.6	V
$f_{SYS}$	System Frequency	3V	$R_{OSC}=275\text{k}\Omega$	—	4	—	MHz
			$R_{OSC}=144\text{k}\Omega$	—	8	—	MHz
$I_{DD}$	Operating Current	3V	No load, $f_{SYS}=4\text{MHz}$	—	—	3	mA
			No load, $f_{SYS}=8\text{MHz}$	—	—	5	mA
$I_{STB1}$	Standby Current (WDT Off)	3V	No load, system HALT	—	—	1	$\mu\text{A}$
$I_{STB2}$	Standby Current (WDT On)	3V	No load, system HALT	—	—	7	$\mu\text{A}$
$V_{IL1}$	Input Low Voltage for I/O Ports	3V	—	—	1	—	V
$V_{IH1}$	Input High Voltage for I/O Ports	3V	—	—	2	—	V
$V_{IL2}$	Input Low Voltage ( $\overline{\text{RES}}$ )	3V	—	—	1.4	—	V
$V_{IH2}$	Input High Voltage ( $\overline{\text{RES}}$ )	3V	—	—	2.1	—	V
$V_{LVD}$	Low Voltage Detection	—	LVD 2.7V	2.565	2.700	2.835	V
$V_{LDO}$	LDO Output Voltage	—	$V_{LDO\_IN}>3.6V$	3.2	3.3	3.4	V
$V_{LDO\_IN}$	LDO Input Voltage	—	—	3.6	—	24	V

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V <sub>DD</sub>	Conditions				
I <sub>LDO</sub>	LDO Output Current	—	V <sub>LDO_IN</sub> =5.5V	60	100	—	mA
I <sub>OL1</sub>	I/O Port Sink Current	3V	V <sub>OL</sub> =0.1V <sub>DD</sub>	7	—	—	mA
I <sub>OH1</sub>	I/O Port Source Current	3V	V <sub>OH</sub> =0.9V <sub>DD</sub>	-3.5	—	—	mA
I <sub>OL2</sub>	PWM1/PWM2 Sink Current	3V	V <sub>OL</sub> =0.1V <sub>DD</sub>	50	—	—	mA
I <sub>OH2</sub>	PWM1/PWM2 Sink Current	3V	V <sub>OH</sub> =0.9V <sub>DD</sub>	-14.5	—	—	mA
I <sub>AUD</sub>	AUD Source Current	3V	V <sub>OH</sub> =0.9V <sub>DD</sub>	—	-3	—	mA
R <sub>PH</sub>	Pull-high Resistance	3V	—	20	60	100	kΩ

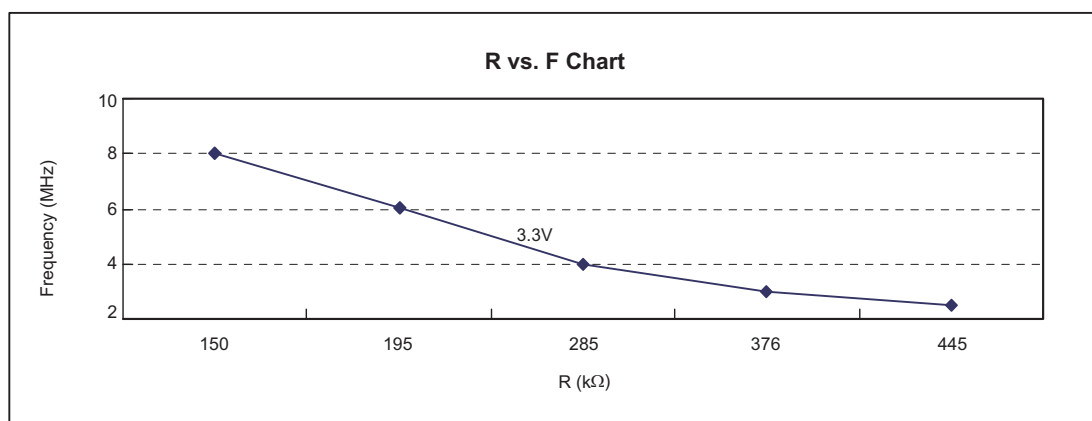
**A.C. Characteristics**

Ta=25°C

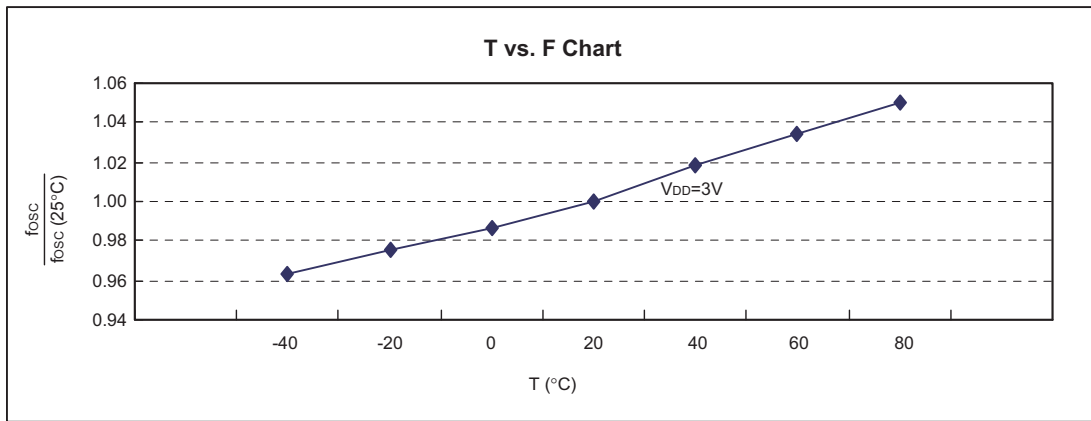
Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V <sub>DD</sub>	Conditions				
f <sub>SYS</sub>	System Clock (RC OSC, Crystal OSC)	—	2.7V~3.6V	4	—	8	MHz
f <sub>TIMER</sub>	Timer Inut Frequency	—	2.7V~3.6V	0	—	8	MHz
t <sub>WDTOSC</sub>	Watchdog Oscillator Period	3V	—	45	90	180	μs
t <sub>WDT1</sub>	Watchdog Time-out Period (WDT OSC)	3V	Without WDT prescaler	12	23	45	ms
t <sub>WDT2</sub>	Watchdog Time-out Period (System Clock)	—	Without WDT prescaler	—	1024	—	ms
t <sub>RES</sub>	External Reset Low Pulse Width	—	—	1	—	—	μs
t <sub>SST</sub>	System Start-up Timer Period	—	Wake-up from HALT	—	1024	—	*t <sub>SYS</sub>
t <sub>INT</sub>	Interrupt Pulse Width	—	—	1	—	—	μs

 Note: \*t<sub>SYS</sub>=1/f<sub>SYS</sub>
**Characteristics Curves**

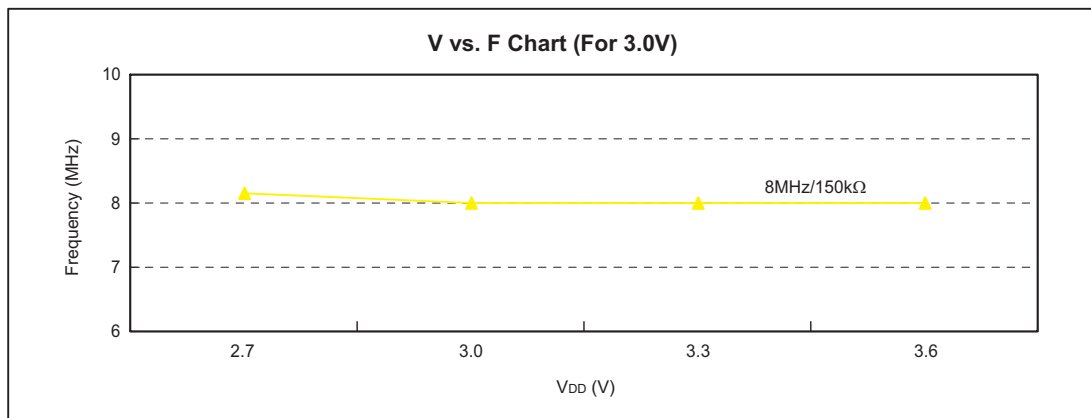
- R vs. F Chart Characteristics Curves



• T vs. F Chart Characteristics Curves



• V vs. F Chart Characteristics Curves – 3.0V



### System Architecture

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

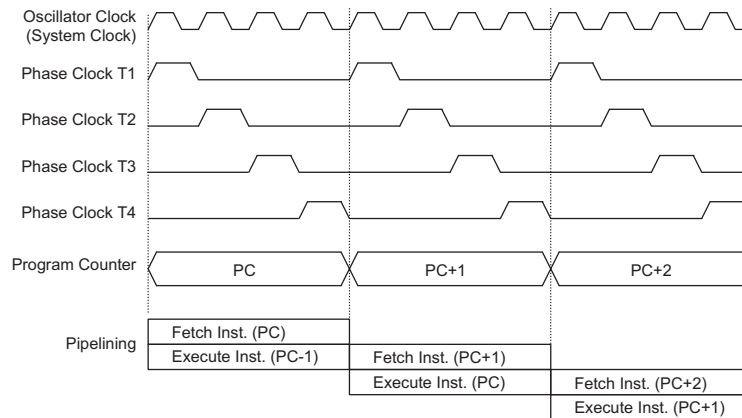
### Clocking and Pipelining

The main system clock, derived from either a Crystal/Resonator or RC oscillator is subdivided into four inter-

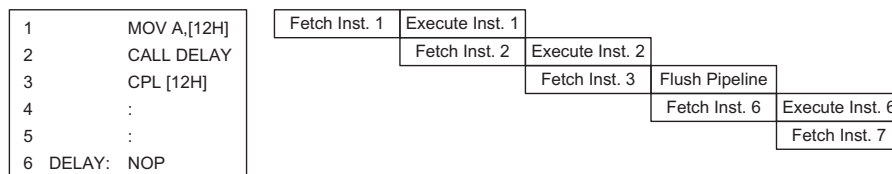
nally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. 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.

When the RC oscillator is used, OSC2 can be used as a T1 phase clock synchronizing pin. This T1 phase clock has a frequency of  $f_{SYS}/4$  with a 1:3 high/low duty cycle.

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



System Clocking and Pipelining



Instruction Fetching

### Program Counter

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

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

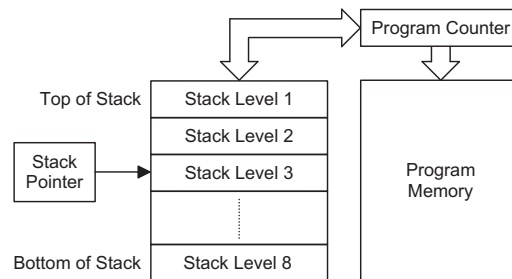
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writable register. By transferring data directly into this register, a short program jump can be executed directly, however, as only this low byte is available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

The lower byte of the Program Counter is fully accessible under program control. Manipulating the PCL might

cause program branching, so an extra cycle is needed to pre-fetch. Further information on the PCL register can be found in the Special Function Register section.

### Stack

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



If the stack is full and an enabled interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal will be inhibited. When the Stack Pointer is decremented, by RET or RETI, the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily.

Mode	Program Counter										
	*10	*9	*8	*7	*6	*5	*4	*3	*2	*1	*0
Initial Reset	0	0	0	0	0	0	0	0	0	0	0
Timer Base Overflow	0	0	0	0	0	0	0	0	1	0	0
Timer Counter 0 Overflow	0	0	0	0	0	0	0	1	0	0	0
Timer Counter 1 Overflow	0	0	0	0	0	0	0	1	1	0	0
SIM Interrupt	0	0	0	0	0	0	1	0	1	0	0
Skip	Program Counter + 2										
Loading PCL	*10	*9	*8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

### Program Counter

Note: \*10~\*0: Program counter bits  
#10~#0: Instruction code bits

S10~S0: Stack register bits  
@7~@0: PCL bits

However, when the stack is full, a CALL subroutine instruction can still be executed which will result in a stack overflow. Precautions should be taken to avoid such cases which might cause unpredictable program branching.

**Arithmetic and Logic Unit – ALU**

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

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

**Program Memory**

The Program Memory is the location where the user code or program is stored. By using the appropriate programming tools, this Program memory device offer users the flexibility to conveniently debug and develop their applications while also offering a means of field programming.

**Structure**

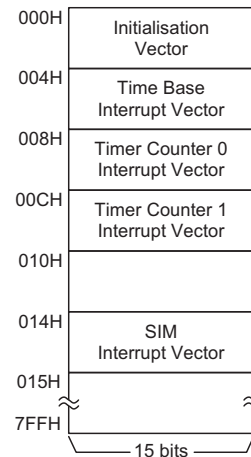
The program memory stores the program instructions that are to be executed. It also includes data, table and interrupt entries, addressed by the Program Counter along with the table pointer. The program memory size is 2K ×15 bits. Certain locations in the program memory are reserved for special usage.

**Special Vectors**

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

- Location 000H  
This vector is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.

- Location 004H  
This vector is used by the external interrupt. If the external interrupt pin on the device goes low, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full.
- Location 008H  
This vector is used by the 8-bit Timer 0. If a overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 00CH  
This vector is used by the 8-bit Timer1. If a overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 010H  
Reserved.
- Location 014H  
This vector is used by the SIM Bus interrupt service program. If the SIM Bus interrupt resulting from a slave address is matched or if 8 bits of data have been received or transmitted successfully from the I<sup>2</sup>C interface, or 8 bits of data have been received or transmitted successful from SPI interface, the program will jump to this location and begin execution if the interrupt is enabled and the stack is not full.



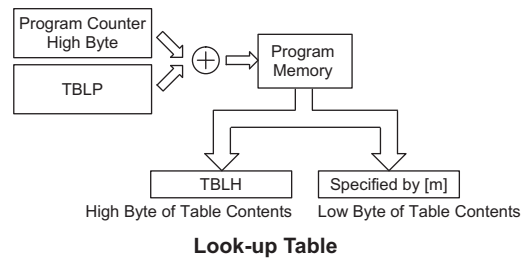
**Program Memory Structure**

**Look-up Table**

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, table pointers are used to setup the address of the data that is to be accessed from the Program Memory. However, as some devices possess only a low byte table pointer and other devices possess both a high and low byte pointer it should be noted that depending upon which device is used, accessing look-up table data is implemented in slightly different ways.

There are two Table Pointer Registers known as TBLP and TBHP in which the lower order and higher order address of the look-up data to be retrieved must be respectively first written. The additional TBHP register allows the complete address of the look-up table to be defined and consequently allow table data from any address and any page to be directly accessed. For this device, after setting up both the low and high byte table pointers, the table data can then be retrieved from any area of Program Memory using the "TABRDC [m]" instruction or from the last page of the Program Memory using the "TABRDL [m]" instruction. When either of these instructions are executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register. Any unused bits in this transferred higher order byte will be read as "0".

The following 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 devices. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "700H" which refers to the start address of the last page within the 2048x15-bit Program Memory of the microcontroller.

The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "706H" or 6 locations after the start of the last page.

Note that the value for the table pointer is referenced to the first address of the present page if the "TABRDC [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRDL [m]" instruction is executed.

```

tempreg1 db    ?    ; temporary register #1
tempreg2 db    ?    ; temporary register #2
:
:
mov  a,06h      ; initialise table pointer - note that this address is referenced
mov  tblp,a     ; to the last page or present page
:
:
tabrdl tempreg1 ; transfers value in table referenced by table pointer
                ; to tempreg1
                ; data at prog. memory address "706H" transferred to
                ; tempreg1 and TBLH

dec  tblp       ; reduce value of table pointer by one

tabrdl tempreg2 ; transfers value in table referenced by table pointer
                ; to tempreg2
                ; data at prog. memory address "705H" transferred to
                ; tempreg2 and TBLH
                ; in this example the data "1AH" is transferred to
                ; tempreg1 and data "0FH" to register tempreg2
                ; the value "00H" will be transferred to the high byte
                ; register TBLH
:
:
org  700h       ; sets initial address of HT83F10/20/40/60/80 last page

dc   00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
:
:

```

Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use 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.

Instruction	Table Location										
	*10	*9	*8	*7	*6	*5	*4	*3	*2	*1	*0
TABRDC [m]	P10	P9	P8	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL [m]	1	1	1	@7	@6	@5	@4	@3	@2	@1	@0

**Table Location**

Note: \*10~\*0: Current Program ROM table  
 @7~@0: Write @7~@0 to TBLP pointer register

P10~P8: Write P12~P8 to TBHP pointer register

**Data Memory**

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

**General Purpose Data Memory**

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

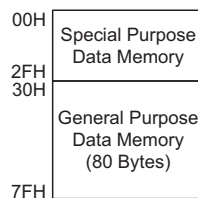
**Structure**

The Data Memory has a bank, known as Bank, which is implemented in 8-bit wide RAM. The RAM Data Memory is located in Bank 0 which is also subdivided into two sections, the Special Purpose Data Memory and the General Purpose Data Memory. The length of these sections is dictated by the type of microcontroller chosen.

**Special Purpose Data Memory**

This area of Data Memory, is located in Bank, where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers are both readable and writable 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".

The start address of the RAM Data Memory for all devices is the address "00H", and the last Data Memory address is "FFH". Registers which are common to all microcontrollers, such as ACC, PCL, etc., have the same Data Memory address.



**RAM Data Memory Structure**

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

## Special Function Registers

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

00H	IAR
01H	MP
02H	
03H	
04H	
05H	ACC
06H	PCL
07H	TBLP
08H	TBLH
09H	WDTs
0AH	STATUS
0BH	INTC
0CH	
0DH	TMR0
0EH	TMR0C
0FH	
10H	TMR1
11H	TMR1C
12H	PA
13H	PAC
14H	PB
15H	PBC
16H	
17H	
18H	
19H	
1AH	
1BH	
1CH	
1DH	
1EH	INTCH
1FH	
20H	SIMC0
21H	SIMC1
22H	SIMDR
23H	SIMAR/SIMC2
24H	DAL
25H	DAH
26H	PWMCR
27H	PWML
28H	PWMH
29H	VOL
2AH	
2BH	
2CH	
2DH	
2EH	
2FH	

■ : Unknown

### Special Purpose Data Memory Structure

#### Indirect Addressing Register – IAR

The Indirect Addressing Register, IAR, although having location in normal RAM register space, do not actually physically exist as normal registers.

The method of indirect addressing for RAM data manipulation uses the Indirect Addressing Register and Memory Pointer, in contrast to direct memory addressing, where the actual memory address is specified.

Actions on the IAR register will result in no actual read or write operation to these register but rather to the memory location specified by their corresponding Memory Pointer, MP. Acting as a pair, IAR and MP can together only access data. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Register indirectly will return a result of "00H" and writing to the registers indirectly will result in no operation.

#### Memory Pointer – MP

For all devices, Memory Pointer, known as MP is provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal register 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. MP, together with Indirect Addressing Register, IAR, are used to access data. Note that bit 7 of the Memory Pointers is not required to address the full memory space and will return a value of "1" if read.

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

```

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

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

loop:
    clr IAR                 ; clear the data at address defined by MP
    inc mp                  ; 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 RAM addresses.

### Accumulator – ACC

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

### Program Counter Low Register – PCL

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

### Look-up Table Registers – TBLP, TBLH

These two special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP is the table pointer and indicates the location where the table data is located. Its value must be setup before any table read commands are executed. Its 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.

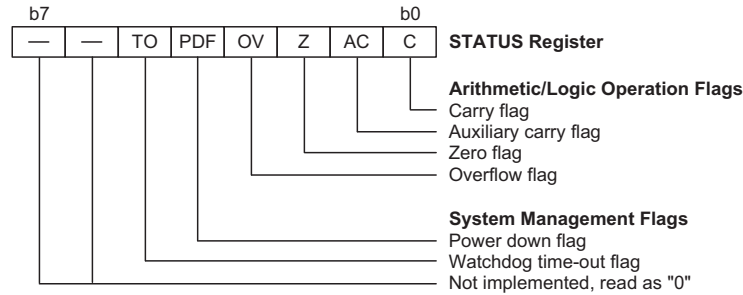
### Watchdog Timer Register – WDTS

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

### Status Register – STATUS

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

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most other registers. Any data written into the status register will not change the TO or PDF flag. In addition, 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



Status Register

time-out or by executing the "CLR WDT" or "HALT" instruction. The PDF flag is affected only by executing the "HALT" or "CLR WDT" instruction or during a system power-up.

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

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

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

**Interrupt Control Register – INTC, INTCH**

Two 8-bit register, known as the INTC and INTCH registers, controls the operation of both external and internal timer interrupts. By setting various bits within these registers using standard bit manipulation instructions, the enable/disable function of the external and timer interrupts can be independently controlled. A master interrupt bit within this register, the EMI bit, acts like a global enable/disable and is used to set all of the interrupt enable bits on or off. This bit is cleared when an interrupt

routine is entered to disable further interrupt and is set by executing the "RETI" instruction.

Note: In situations where other interrupts may require servicing within present interrupt service routines, the EMI bit can be manually set by the program after the present interrupt service routine has been entered.

**Timer Registers**

All devices contain two 8-bit Timers whose associated registers are known as TMR0 and TMR1 which is the location where the associated timer's 8-bit value is located. Their associated control registers, known as TMR0C and TMR1C, contain the setup information for these timers.

Note that all timer registers can be directly written to in order to preload their contents with fixed data to allow different time intervals to be setup.

**Input/Output Ports and Control Registers**

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

### Voice Control and Audio output Registers – DAL, DAH, VOL

The devices include a single 12-bit current type DAC function for driving an external 8Ω speaker through an external NPN transistor or Power Amplifier. The programmer must write the voice data to these DAL/DAH registers. The programmer can control the DAC volume with 7-levels via the VOL register.

### Pulse Width Modulator Registers – PWMC, PWML, PWMH

Each device contains a single 12-bit PWM function for driving an external 8Ω speaker. The programmer must write the voice data to PWML/PWMH register. The programmer can control the PWM volume with 8-levels via the VOL register.

### Serial Interface Module(SIM) Registers – SIMC0, SIMC1, SIMAR/SIMC2, SIMDR

Each SIM contains SPI and I<sup>2</sup>C function for communicating with other microcontroller or SPI Flash Memory. All devices contain an integrated I<sup>2</sup>C and SPI bus which interfaces to the external shared pins SDA, SCL and SCSB, SCK, SDI, SDO with PB on the microcontroller. The I<sup>2</sup>C correct setup and data transfer operation of this 2-line bidirectional bus utilizes 4 special function registers. The SIMAR register sets the slave address of the device while the SIMC0 is the control register that enables or disables the device as well as select whether it is in I<sup>2</sup>C or SPI mode. The SIMC1 register is the I<sup>2</sup>C status register while the SIMDR register is the input/output data register. The SPI correct setup and data transfer operation of this 3-line bidirectional bus utilizes 3 special function registers. The SIMC0 is the control register that enables or disables the device as well as select whether it is in I<sup>2</sup>C or SPI mode. The SIMC2 register is the SPI status register while the SIMDR register is the input/output data register.

### Flash Data Memory

The Data Memory is the location where the user Data is stored. For this device the Data Memory is a Flash type, which means it can be programmed and reprogrammed a large number of times, allowing the user the convenience of voice data modification using the same device. By using the appropriate programming tools, these devices offer users the flexibility to conveniently change and develop their applications while also offering a means of field programming.

#### Flash Data Memory Structure

The internal Flash Data Memory has a capacity of between 2M×8 bit and 128K×8 bit. Unlike the Program

Memory and RAM Data Memory, the Flash Data Memory is not directly mapped and is therefore not directly accessible in the same way as the other types of memory.

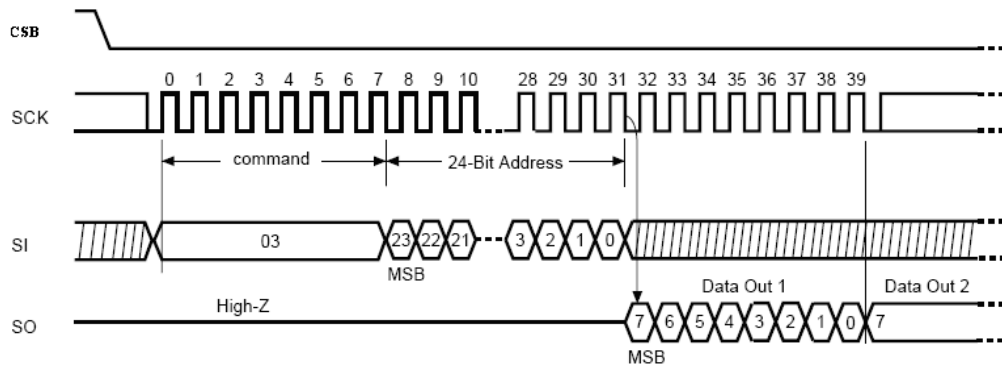
#### Accessing the Flash Data Memory

The Flash Data Memory is accessed using a set of Macros in the library. These instructions control all functions of the Flash such as read, write, erase, enable etc. The internal Flash structure is similar to that of a standard SPI Flash Memory, for which 4 pins are used for transfer of instruction, address and data information. These are the Chip Select pin,  $\overline{CS}$ , Serial Clock pin, SCK, Data In pin, SI and the Data Out pin, SO. All actions related to the Flash Memory must be conducted through each of these four Flash Memory download pins. By manipulating these four pin in the device, in accordance with the accompanying timing diagrams, the microcontroller can communicate with the Flash Memory and carry out the required read and write instructions.

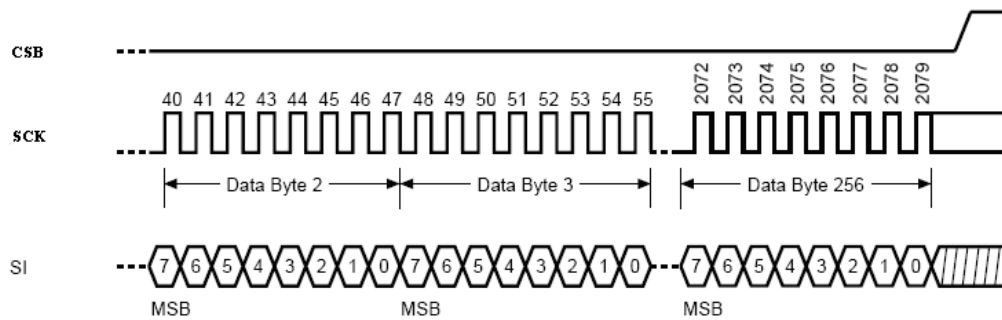
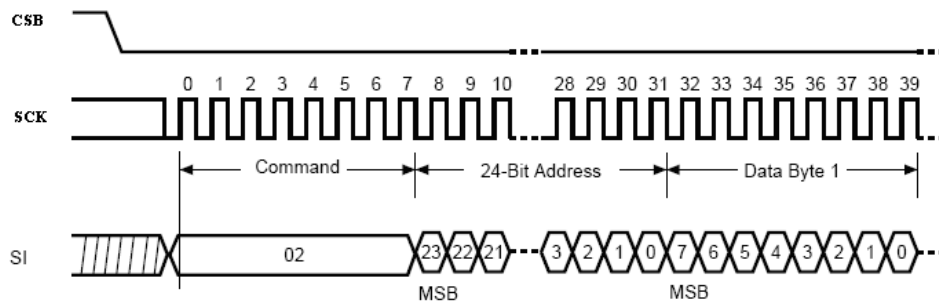
When reading data from the Flash Memory,  $\overline{CS}$  should be set to "0" to start the data transmission. The data will clocked out on the rising edge of SCK and appear on SO. The SO pin will normally be in a high-impedance condition unless a READ statement is being executed. When writing to the Flash Memory the data must be presented first on SI and then clocked in on the rising edge of SCK. After all the instruction, address and data information has been transmitted,  $\overline{CS}$  should be set to "1" to terminate the data transmission. Note that after power on the Flash Memory must be initialised as described.

#### READ

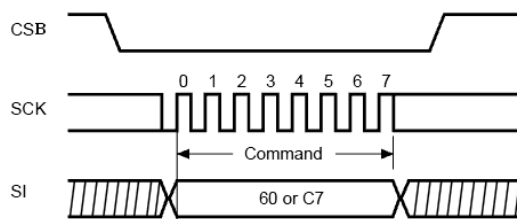
The "READ" instruction is used to read out one or more bytes of data from the Flash Data Memory. To instigate a "READ" instruction, the  $\overline{CS}$  bit should be set low, followed by a command instruction and then the instruction code "03", all transmitted via the SI bit. The address information should then follow with the MSB being transmitted first. After the last address bit, A0, has been transmitted, the data can be clocked out, bit D7 first, on the rising edge of the SCK clock signal and can be read via the SO bit. The data information will first precede the reading of the first data bit, D7. After the full byte has been read out, the internal address will be automatically incremented allowing the next consecutive data byte to be read out without entering further address data. As long as the  $\overline{CS}$  bit remains low, data bit D7 of the next address will automatically follow data bit D0 of the previous address being inserted between them. The address will keep incrementing in this way until CS returns to a high value. SO will normally be in a high impedance condition until the "READ" instruction is executed.



**Read Data Byte Timing**



**Page Program Timing**



**Erase All Timing**

**WRITE**

The "WRITE" instruction is used to write a page byte of data into the Flash Data Memory. To instigate a WRITE instruction, the  $\overline{CS}$  bit should be set low, then the instruction code "02", all transmitted via the SI bit. For this device, The address information should then follow with the MSB bit being transmitted first. After the last address bit, A0, has been transmitted, the data can be immediately transmitted MSB first. After all the WRITE instruction code, address and data have been transmitted, the data will be written into the Flash Data Memory when the  $\overline{CS}$  bit is set to high. The Flash Data Memory does this by executing an internal write-cycle, which will first erase and then write the previously transmitted data byte into the Flash Data Memory. This process takes place internally using the Flash Data Memory's own internal clock and does not require any action from the SCK clock. No further instructions can be accepted by the Flash Data Memory until this internal write-cycle has finished.

**ERAL**

The "ERAL" instruction is used to erase the whole contents of the Flash Data Memory. After it has been executed all the data in the Flash Data Memory will be set to "1". To instigate this instruction, the CSB bit should be set low. The instruction code "60" or "C7". Following on from this, a "60" or "C7" should then be transmitted. After the "ERAL" instruction code has been transmitted, the Flash Data Memory data will be erased when the CS bit is set to high.

The Flash Data Memory does this by executing an internal write-cycle. This process takes place internally using the Flash Data Memory's own internal clock and does not require any action from the SCK clock. No further instructions can be accepted by the Flash Data Memory until this internal write-cycle has finished. To determine when the write

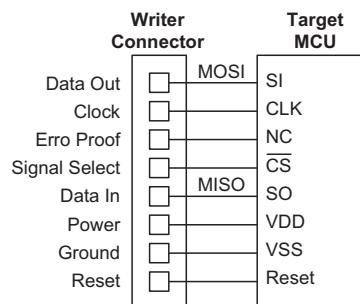
Instruction	Function	Instruction Code	Address	Data
READ	Read Out Data	03	A23~A0	D7~D0
WRITE	Write Data Page Byte	02	A23~A0	D7~D0
ERAL	Erase All	60 or C7	—	—

**Instruction Set Summary**
**In Circuit Programming**

The provision of Flash type Data Memory gives the user and designer the convenience of easy upgrades and modifications to their Data on the same device. As an additional convenience, Holtek has provided a means of programming the microcontroller in-circuit. 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 data releases without removal and re-insertion of the device.

Pin Name	Function
SI	Serial data input
SO	Serial data output
SCK	Serial clock
$\overline{CS}$	Signal Select
VDD	Power supply
VSS	Ground

The Data Memory can be programmed serially in-circuit using a 8-wire interface. Data is downloaded and uploaded serially on two SI/SO pins with an additional line for the clock. Two additional lines are required for the power supply and one line for the select signal. The technical details regarding the in-circuit programming of the devices are beyond the scope of this document and will be supplied in supplementary literature.


**In-circuit Programming Interface**

## Input/Output Ports

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

Depending upon which device or package is chosen, the microcontroller range provides from 12 bidirectional input/output lines labeled with port names PA, PB, etc. These I/O ports are mapped to the 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.

### Pull-high Resistors

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

### Port A Wake-up

Each device has a HALT instruction enabling the microcontroller to enter a Power Down Mode and preserve 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. After a "HALT" instruction forces the microcontroller into entering a HALT condition, the processor will remain idle or in a low-power state until the logic condition of the selected wake-up pin on Port A changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Note that each pin on Port A can be selected individually to have this wake-up feature.

### I/O Port Control Registers

Each I/O port has its own control register PAC and PBC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each 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.

### Pin-shared Functions

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

- Serial Interface Module

The device pins, PB0~PB3, are pin-shared with pins SDA, SCL,  $\overline{SCS}$ , SCK, SDI, SDO. The choice of which function is used is selected using the SIMC0 register.

- I/O Pin Structures

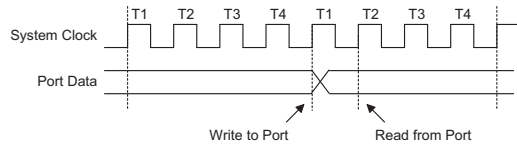
The following diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins.

Note also that the specified pins refer to the largest device package, therefore not all pins specified will exist on all devices.

**Programming Considerations**

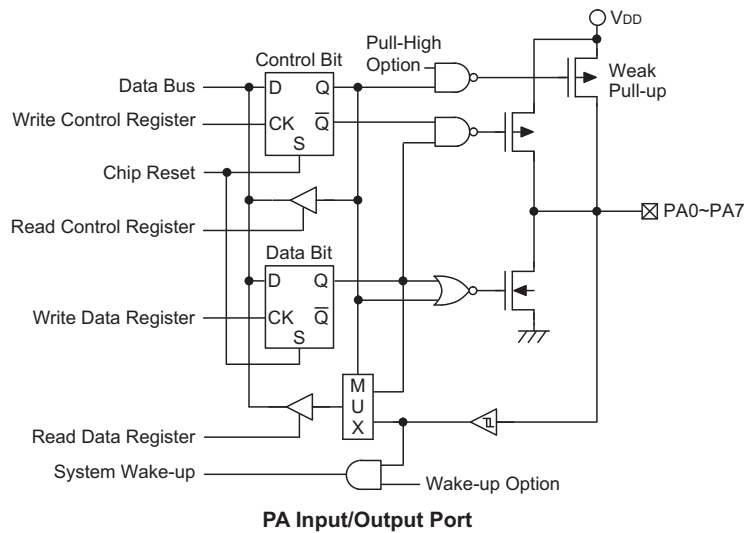
Within the user program, one of the first things to consider is port initialization. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high options have been selected. If the port control registers, PAC, PBC etc., are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated port data registers, PA, PB, etc., are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct values into the appropriate port control register or by programming individual bits in the port control register using the "SET [m].i" and "CLR [m].i" instructions. Note that when using these bit control instructions, a read-modify-write

operation takes place. The microcontroller must first read in the data on the entire port, modify it to the required new bit values and then rewrite this data back to the output ports.

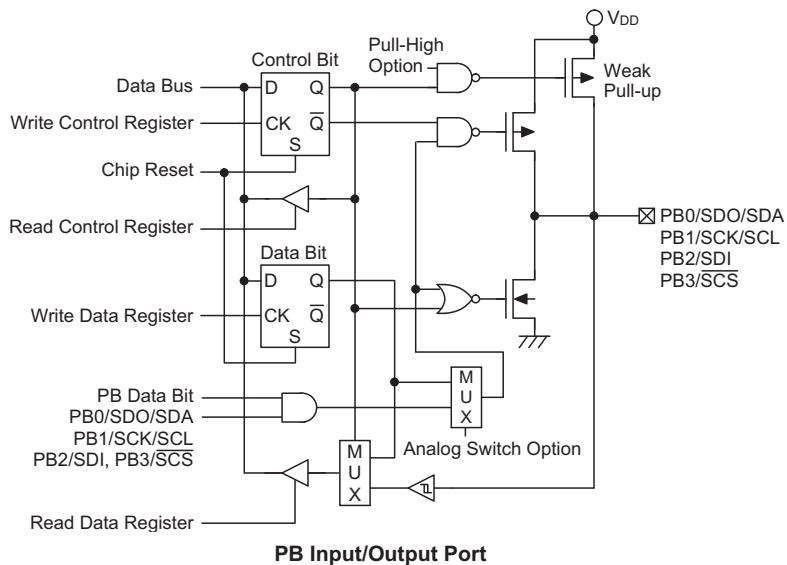


**Read/Write Timing**

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



**PA Input/Output Port**



**PB Input/Output Port**

## Timers

The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. These devices contain two count up timers of 8-bit capacity. The provision of an internal prescaler to the clock circuitry of the timer gives added range to the timer.

There are two types of register related to each Timer. 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. All devices can have the timer clock configured to come from the internal clock source.

### Configuring the Timer Input Clock Source

The clock source for the 8-bit timers is the system clock divided by four. The 8-bit timer clock source is also first divided by a, the division ratio of which is conditioned by the three lower bits of the associated timer control register.

### Timer Registers – TMR0, TMR1

The timer registers are special function registers located in the special purpose Data Memory and is the place where the actual timer value is stored. All devices contain two 8-bit timers, whose registers are known as TMR0 and TMR1. The value in the timer registers 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 at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting.

Note that to achieve a maximum full range count of FFH for the 8-bit timer, the preload registers must first be cleared to all zeros. It should be noted that after power-on, the preload registers will be in an unknown condition. Note that if the Timer Counters are in an OFF condition and data is written to their preload registers, 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. Note also that when the timer registers

are read, the timer clock will be blocked to avoid errors, however, as this may result in certain timing errors, programmers must take this into account.

### Timer Control Registers – TMR0C, TMR1C

Each timer has its respective timer control register, known as TMR0C and TMR1C. It is the timer control register together with their corresponding timer registers that control the full operation of the timers. Before the timers can be used, it is essential that the appropriate timer control register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialization. Bits 7 and 6 of the Timer Control Register, must be set to the required logic levels. Bit 6 of the registers must always be written with a "1", and bit 7 must always be written with a "0". The timer-on bit, which is bit 4 of the Timer Control Register and known as T0ON/ T1ON, depending upon which timer is used, provides the basic on/off control of the respective timer. setting the bit high allows the timer to run, clearing the bit stops the timer. For the 8-bit timers, which have prescalers, bits 0~2 of the Timer Control Register determines the division ratio of the input clock prescaler.

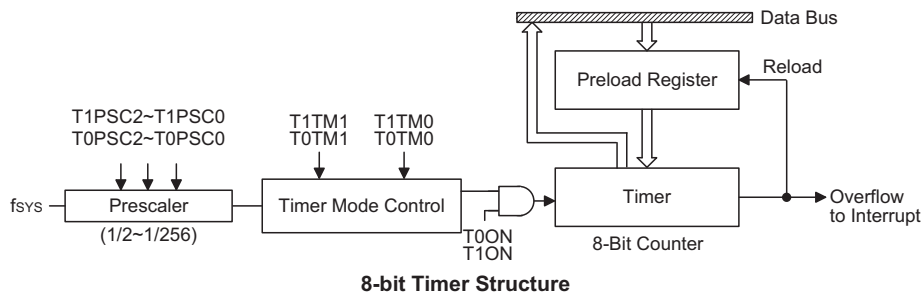
### Configuring the Timer

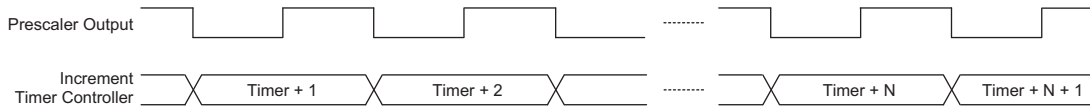
The Timer is used to measure fixed time intervals, providing an internal interrupt signal each time the Timer overflows. To do this the Operating Mode Select bit pair in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits

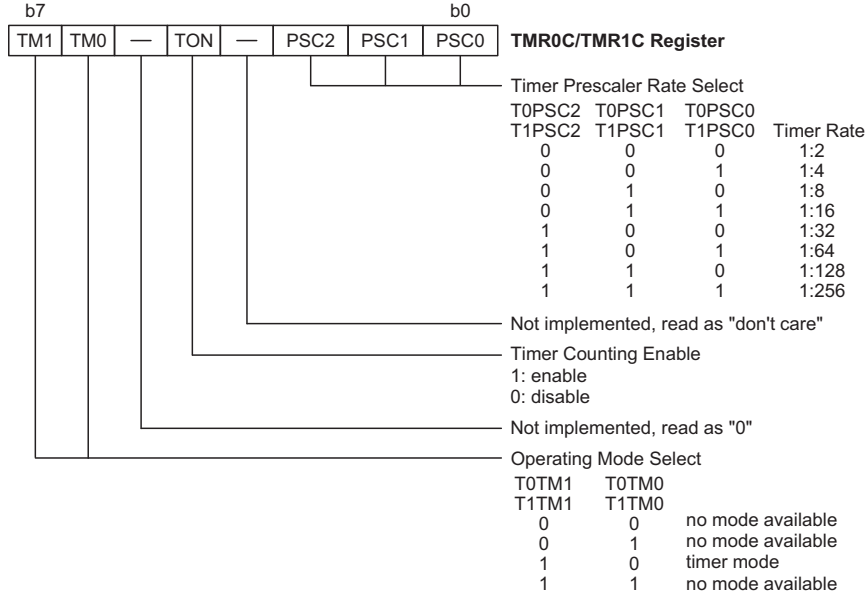
Bit7	Bit6
1	0

The internal clock,  $f_{SYS}$ , is used as the Timer clock. However, this clock source is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits, which are bits 0~2 in the Timer Control Register. After the other bits in the Timer Control Register have been setup, the enable bit, which is bit 4 of the Timer Control Register, can be set high to enable the Timer to run. Each time an internal clock cycle occurs, the Timer increments by one. When it is full and overflows, an interrupt signal is generated and the Timer will





Timer Mode Timing Diagram



Timer Control Register

reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer Interrupt Enable bit in the Interrupt Control Register, INTC, is reset to zero.

**Prescaler**

All of the 8-bit timers possess a prescaler. Bits 0~2 of their associated timer control register, define the pre-scaling stages of the internal clock source of the Timer. The Timer overflow signal can be used to generate signals for the Timer interrupt.

**Programming Considerations**

The internal system clock is used as the timer clock source and is therefore synchronized with the overall operation of the microcontroller. In this mode, when the appropriate timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector.

When the Timer is read, the clock is blocked to avoid errors, however as this may result in a counting error, this should be taken into account by the programmer. Care

must be taken to ensure that the timers are properly 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.

**Timer Program Example**

The following example program section is based on the HT83F60 device, which contain two 8-bit timers. Programming the timer for other devices is conducted in a very similar way. The program shows how the timer registers are setup along with how the interrupts are enabled and managed. Points to note in the example are how, for the 8-bit timer. Note how the timer is turned on by setting bit 4 of the respective timer control register. The timer can be turned off in a similar way by clearing the same bit. This example program sets the timer to be in the timer mode which uses the internal fsys as their clock source, and produce a timer 0 interrupt per 1ms.

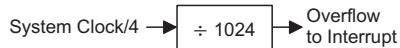
```
#include HT83F60.inc
    jmp begin
    :
org 04h          ; time base vector
    reti
org 08h          ; timer 0 interrupt vector
    jmp tmr0int  ; jump here when timer 0 overflows every 1ms
org 0Ch
    reti
org 10h
    reti
org 14h
    reti
    :
    ; internal timer 0 interrupt routine
Tmr0int:
    :
    ; timer 0 main program placed here
    :
    reti
    :
begin:
    ; setup timer 0 registers
    mov a,06h    ; setup timer 0 low byte
    mov tmr0,a  ; flow byte must be setup before high byte
    mov a,094   ; setup timer 0 control register
    mov tmr0c,a ; setup timer mode and clock source is fsys/32 prescaler
    ; setup interrupt register
    mov a,05h   ; enable global interrupt
    mov intc,a  ; enable timer 0 interrupt
```

## Time Base

The Time Base function will generate a regular interrupt signal synchronised to the system clock which can be used by the application as a time base signal.

### Time Base Operation

The Time Base operation is a very simple function for the generation of a regular time signal. This is implemented by generating a regular interrupt signal whose enable/disabled and request flags are in the INTC register. The clock source for the time base is the internal



$f_{\text{SYS}}/4$  clock source, which is then divided internally by a value of 1024. It is this divided signal that generates the internal interrupt. The Time Base Interrupt is enabled by the ETBI bit in the INTC register and interrupt request flag is the TBF flag in the same register. A time base of 1ms will therefore be generated from a system clock of 4MHz and a time base of 0.5ms will be generated from a system clock source of 8MHz.

### Time base Example

The following example program section is based on the HT83F60 device. The program shows how the Time Base registers are setup along with how the interrupts are enabled and managed. The points to note in the example are how the Time Base is turned on by setting bit 4 of the INTC register. The Time Base can be turned off in a similar way by clearing the same bit. This example program sets the Time Base which uses the internal system clock as their clock source, and produces a time base interrupt every 0.5ms from a system source clock of 8MHz.

```

#include HT83F60.inc
    jmp begin
    :
org 04h                ; time base vector
    jmp time_base_int  ; jump here when time base overflows per 0.5ms
org 08h
    reti
org 0Ch
    reti
org 10h
    reti
org 14h
    reti
    :
                                ; time base interrupt routine
time_base_int:
    :
                                ; time base main program placed here
    :
    reti
    :
begin:
                                ; setup interrupt register
    mov a,03h                  ; enable global and time base interrupt
    mov intc,a                 ; enable time base
  
```



	Master/Slave (SIMEN=0)	Master (SIMEN=1)		Slave (SIMEN=1)		
		CSEN=1	CSEN=0	CSEN=0	SCS line=0 (CSEN=1)	SCS line=1 (CSEN=1)
SCS	Z	L	Z	Z	I, Z	I, Z
SDO	Z	O	O	O	O	Z
SDI	Z	I, Z	I, Z	I, Z	I, Z	Z
SCK	Z	L(CPOL=1) H(CPOL=0)	L(CPOL=1) H(CPOL=0)	I, Z	I, Z	Z

"Z" floating, "H" output high, "L" output low, "I" Input, "O" output level, "I,Z" input floating (no pull-high)

#### SPI Interface Pin Status

- SPI Registers

The SIMDR register is used to store the data being transmitted and received. There are two control registers associated with the SPI interface, SIMC0 and SIMC2 and one data register known as SIMDR. The SIMC1 register is not used by the SPI function. Register SIMC0 is used to control the enable/disable function, the power down control and to set the data transmission clock frequency. Register SIMC2 is used for other control functions such as LSB/MSB selection, write collision flag etc.

The following gives further explanation of each bit:

- ♦ SIMEN

The SIMEN bit is the overall on/off control for the SPI interface. When the SIMEN bit is cleared to zero to disable the SPI interface, the SDI, SDO, SCK and SCS lines will be in a floating condition and the SPI operating current will be reduced to <0.1µA at 5V. When the bit is high the SPI interface is enabled. Note that when the SIMEN bit changes from low to high the contents of the SPI control registers will be in an unknown condition and should therefore be initialised by the application program.

- ♦ SIM0~SIM2

These three bits control the Master/Slave selection and also setup the SPI interface clock speed when in the Master Mode. The SPI clock is a function of the system clock whether it be RC type or Crystal type. If the Slave Mode is selected then the clock will be supplied by the external Master device. The following gives further explanation of each bit:

- ♦ TRF

The TRF bit is the Transmit/Receive Complete flag and is cleared by the application program and can be used to generate an interrupt. When the bit is high the data has been transmitted or received. If the bit is low the data is being transmitted or has not yet been received.

- ♦ WCOL

The WCOL bit is used to detect if a data collision has occurred. If this bit is high it means that data has been attempted to be written to the SIMDR register during a data transfer operation. This writing operation will be ignored if data is being transferred. The bit can be cleared by the application program. Note that using the CSEN bit can be disabled or enabled via configuration option.

- ♦ CSEN

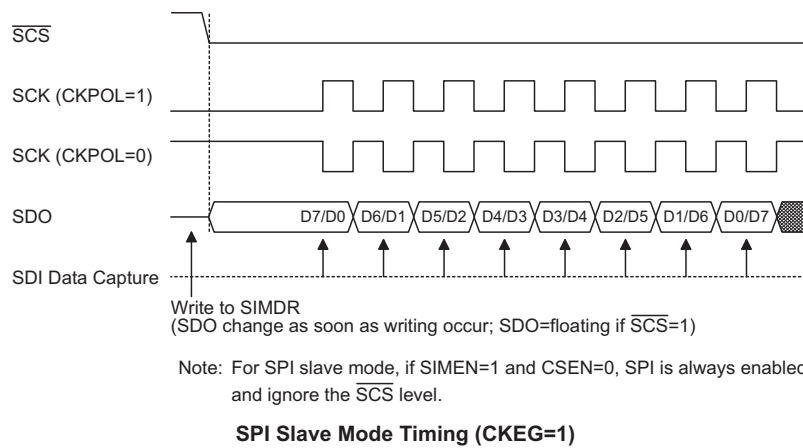
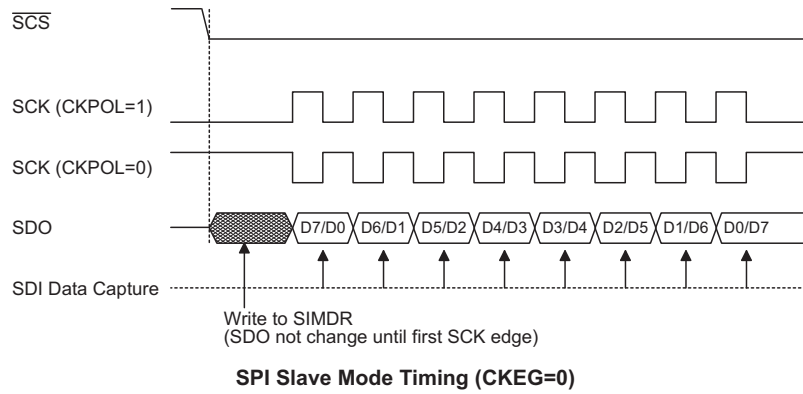
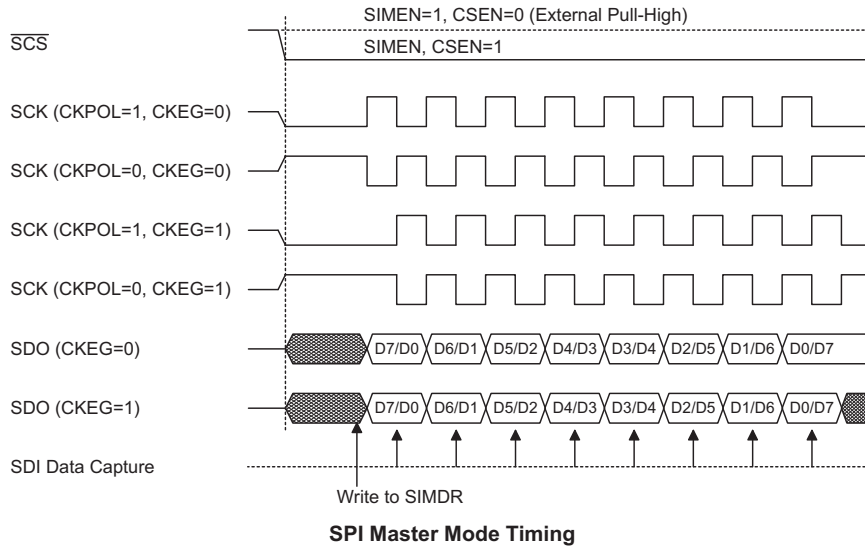
The CSEN bit is used as an on/off control for the SCS pin. If this bit is low then the SCS pin will be disabled and placed into a floating condition. If the bit is high the SCS pin will be enabled and used as a select pin.

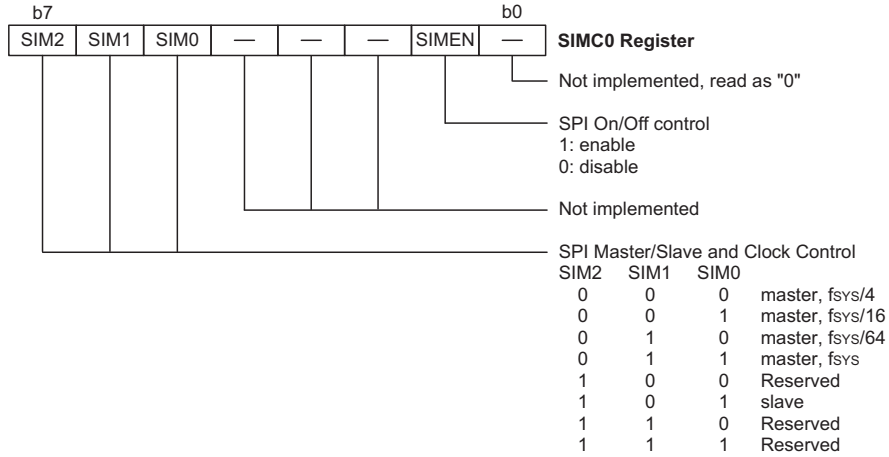
- ♦ MLS

The MLS is used to select how the data is transferred, either MSB or LSB first. Setting the bit high will select MSB first and low for LSB first. Note that the SIMC2 register is the same as the SIMAR register used by the I<sup>2</sup>C interface.

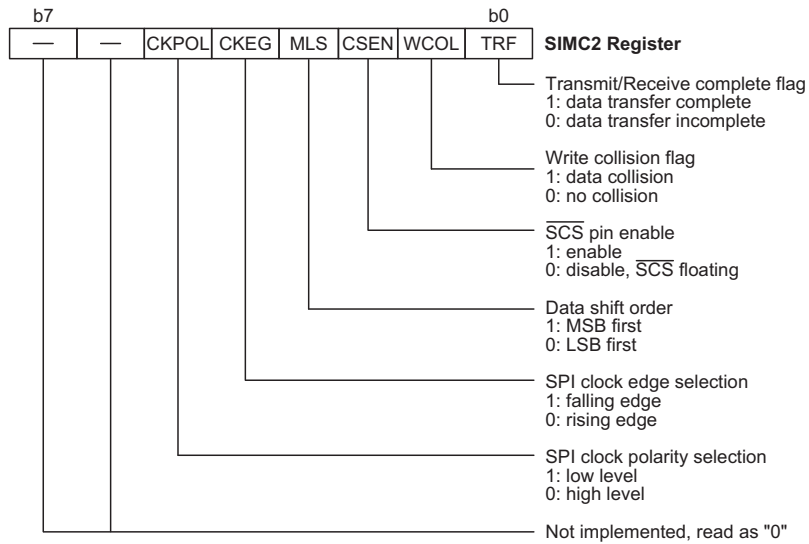
- SPI Communication

After the SPI interface is enabled by setting the SIMEN bit high, then in the Master Mode, when data is written to the SIMDR register, transmission/reception will begin simultaneously. When the data transfer is complete, the TRF flag will be set automatically. In the Slave Mode, when the clock signal from the master has been received, any data in the SIMDR register will be transmitted and any data on the SDI pin will be shifted into the SIMDR register. The master should output an SCS signal before a clock signal is provided and slave data transfers should be enabled/disabled before/after an SCS signal is received.

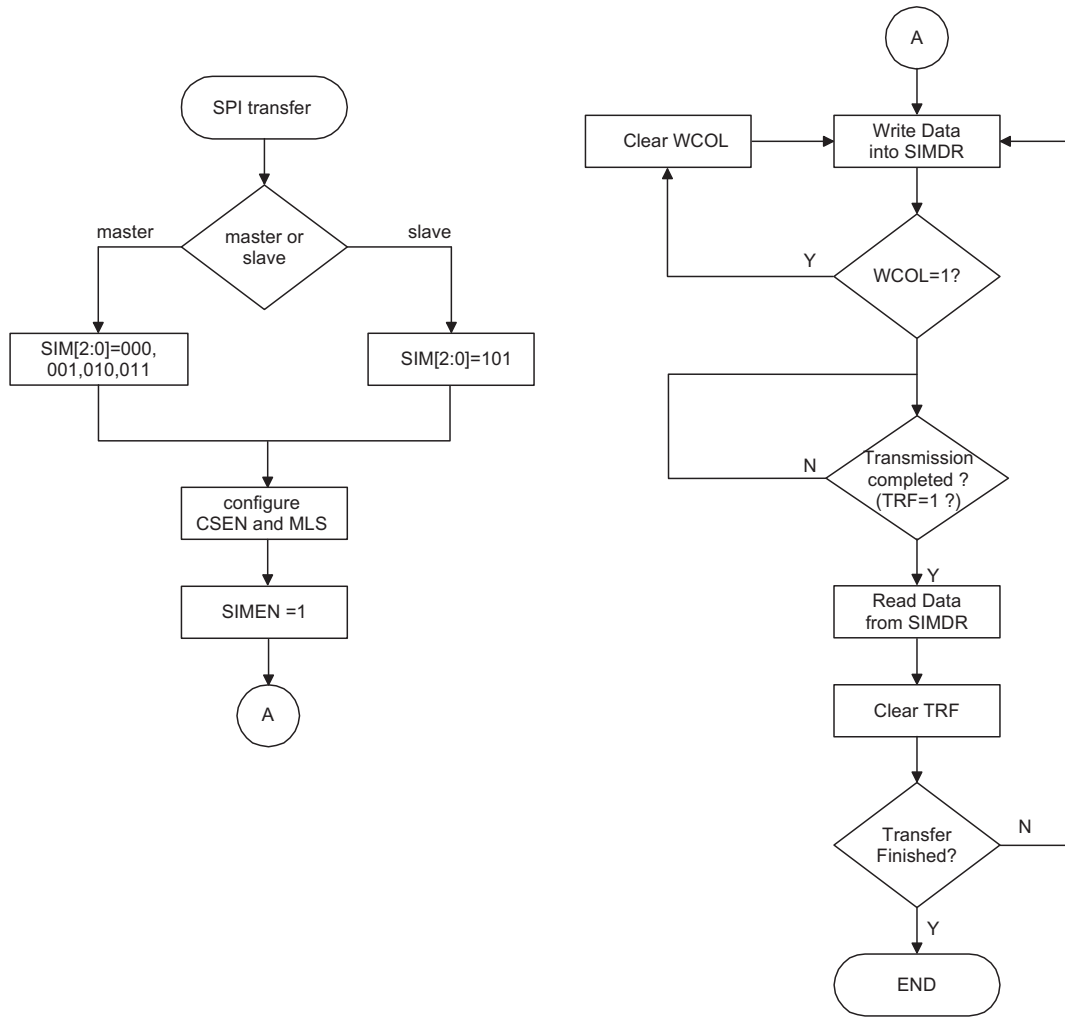




**SPI Control Register – SIMC0**



**SPI Control Register – SIMC2**



SPI Transfer Control Flowchart

**I<sup>2</sup>C Interface**

The I<sup>2</sup>C bus is a bidirectional 2-line communication interface originally developed by Philips. The possibility of transmitting and receiving data on only 2 lines offers many new application possibilities for microcontroller based applications.

• I<sup>2</sup>C Interface Operation

As the I<sup>2</sup>C interface pins are pin-shared with segment pins and with the SPI function pins, the I<sup>2</sup>C interface must first be enabled by selecting the correct configuration option.

There are two lines associated with the I<sup>2</sup>C bus, the first is known as SDA and is the Serial Data line, the second is known as SCL line and is the Serial Clock line. As many devices may be connected together on the same bus, their outputs are both open drain types. For this reason it is necessary that external pull-high resistors are connected to these outputs. Note that no chip select line exists, as each device on the I<sup>2</sup>C bus is identified by a unique address which will be transmitted and received on the I<sup>2</sup>C bus.

When two devices communicate with each other on the bidirectional I<sup>2</sup>C bus, one is known as the master device and one as the slave device. Both master and slave can transmit and receive data, however, it is the master device that has overall control of the bus. For this device, which only operates in slave mode, there are two methods of transferring data on the I<sup>2</sup>C bus, the slave transmit mode and the slave receive mode.

• I<sup>2</sup>C Registers

There are three control registers associated with the I<sup>2</sup>C bus, SIMC0, SIMC1 and SIMAR and one data register, SIMDR.

The SIMDR register is used to store the data being transmitted and received on the I<sup>2</sup>C bus. Before the microcontroller writes data to the I<sup>2</sup>C bus, the actual data to be transmitted must be placed in the SIMDR register. After the data is received from the I<sup>2</sup>C bus, the microcontroller can read it from the SIMDR register. Any transmission of data to the I<sup>2</sup>C bus or reception of data from the I<sup>2</sup>C bus must be made via the SIMDR register.

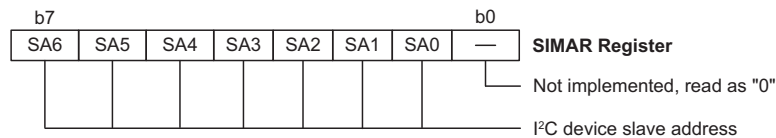
The SIMAR register is the location where the slave address of the microcontroller is stored. Bits 1~7 of the SIMAR register define the microcontroller slave address. Bit 0 is not defined. When a master device, which is connected to the I<sup>2</sup>C bus, sends out an address, which matches the slave address in the SIMAR register, the microcontroller slave device will be selected.

Note that the SIMAR register is the same register as SIMC2 which is used by the SPI interface.

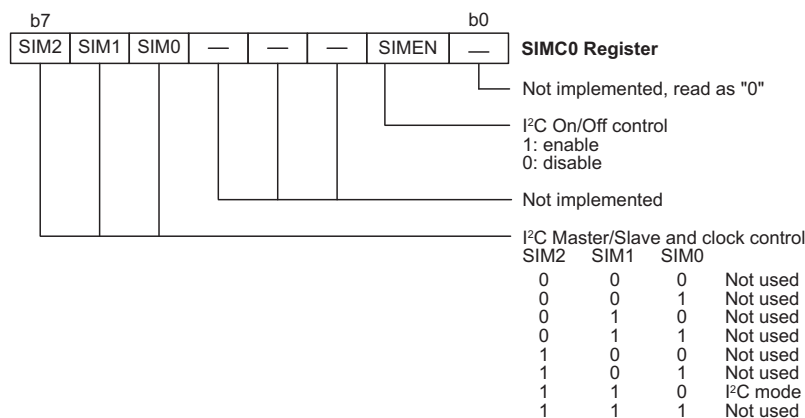
The SIMC0 register is used for the I<sup>2</sup>C overall on/off control.

• I<sup>2</sup>C Configuration Option

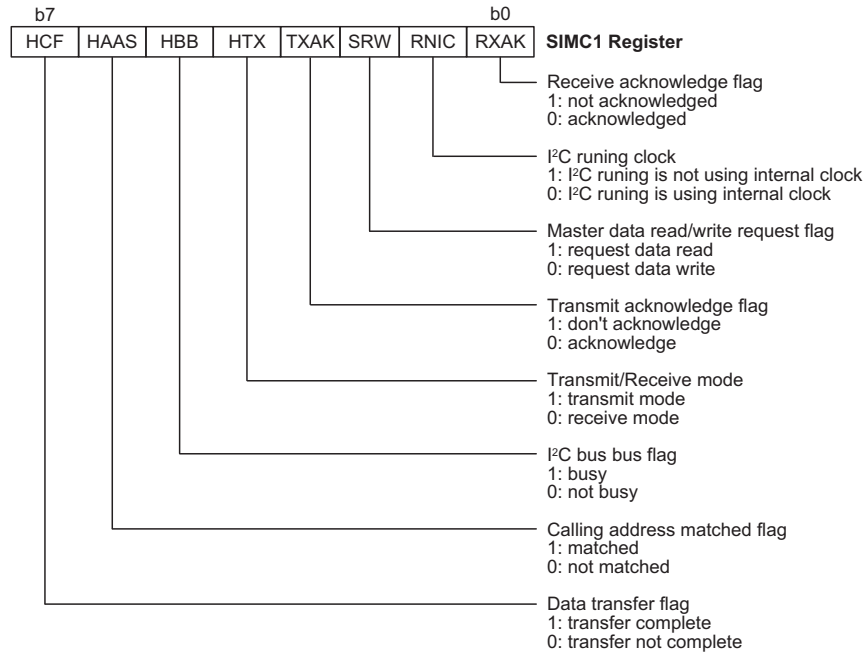
There are several configuration options associated with the I<sup>2</sup>C interface. One of these is to enable the RNIC bit function which selects the RNIC bit in SIMC1 register. Another configuration option determines the debounce time of the I<sup>2</sup>C interface. This add a debounce delay time to the external clock to reduce the possibility of glitches on the clock line causing erroneous operation. The debounce time if selected can be chosen to be either 1 or 2 system clocks.



**Slave Address Register – SIMAR**



**I<sup>2</sup>C Control Register – SIMC0**

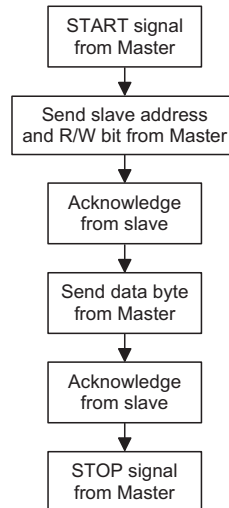


I<sup>2</sup>C Control Register – SIMC1

The following gives further explanation of each bit:

- ♦ SIMEN  
The SIMEN bit determines if the I<sup>2</sup>C bus is enabled or disabled. If data is to be transferred or received on the I<sup>2</sup>C bus then this bit must be set high.
  
- The following gives further explanation of each bit:
- ♦ HCF  
The HCF flag is the data transfer flag. This flag will be zero when data is being transferred. Upon completion of an 8-bit data transfer the flag will go high and an interrupt will be generated.
- ♦ HASS  
The HASS flag is the address match flag. This flag is used to determine if the slave device address is the same as the master transmit address. If the addresses match then this bit will be high, if there is no match then the flag will be low.
- ♦ HBB  
The HBB flag is the I<sup>2</sup>C busy flag. This flag will be high when the I<sup>2</sup>C bus is busy which will occur when a START signal is detected. The flag will be reset to zero when the bus is free which will occur when a STOP signal is detected.
- ♦ HTX  
The HTX flag is the transmit/receive mode bit. This flag should be set high to set the transmit mode and low for the receive mode.
- ♦ TXAK  
The TXAK flag is the transmit acknowledge flag. After the receipt of 8-bits of data, this bit will be transmitted to the bus on the 9th clock. To continue receiving more data, this bit has to be reset to zero before further data is received.

- ♦ SRW  
The SRW bit is the Slave Read/Write bit. This bit determines whether the master device wishes to transmit or receive data from the I<sup>2</sup>C bus. When the transmitted address and slave address match, that is when the HAAS bit is set high, the device will check the SRW bit to determine whether it should be in transmit mode or receive mode. If the SRW bit is high, the master is requesting to read data from the bus, so the device should be in transmit mode. When the SRW bit is zero, the master will write data to the bus, therefore the device should be in receive mode to read this data.
- ♦ RNIC  
The RNIC bit is used as I<sup>2</sup>C running clock from Internal or external clock. If this bit is low then I<sup>2</sup>C running using internal clock and it will not wake-up when I<sup>2</sup>C interrupts in the Power Down Mode. If the bit is high I<sup>2</sup>C running using external clock and it will wake-up when I<sup>2</sup>C interrupts in the Power Down Mode.
- ♦ RXAK  
The RXAK flag is the receive acknowledge flag. When the RXAK bit has been reset to zero it means that a correct acknowledge signal has been received at the 9th clock, after 8 bits of data have been transmitted. When in the transmit mode, the transmitter checks the RXAK bit to determine if the receiver wishes to receive the next byte. The transmitter will therefore continue sending out data until the RXAK bit is set to "1". When this occurs, the transmitter will release the SDA line to allow the master to send a STOP signal to release the bus.



**I<sup>2</sup>C Bus Communication**

Communication on the I<sup>2</sup>C bus requires four separate steps, a START signal, a slave device address transmission, a data transmission and finally a STOP signal. When a START signal is placed on the I<sup>2</sup>C bus, all devices on the bus will receive this signal and be notified of the imminent arrival of data on the bus. The first seven bits of the data will be the slave address with the first bit being the MSB. If the address of the microcontroller matches that of the transmitted address, the HAAS bit in the SIMC1 register will be set and an I<sup>2</sup>C interrupt will be generated. After entering the interrupt service routine, the microcontroller slave device must first check the condition of the HAAS bit to determine whether the interrupt source originates from an address match or from the completion of an 8-bit data transfer. During a data transfer, note that after the 7-bit slave address has been transmitted, the following bit, which is the 8th bit, is the read/write bit whose value will be placed in the SRW bit. This bit will be checked by the microcontroller to determine whether to go into transmit or receive mode. Before any transfer of data to or from the I<sup>2</sup>C bus, the microcontroller must initialise the bus, the following are steps to achieve this:

**Step 1**

Write the slave address of the microcontroller to the I<sup>2</sup>C bus address register SIMAR.

**Step 2**

Set the SIMEN bit in the SIMC0 register to "1" to enable the I<sup>2</sup>C bus.

**Step 3**

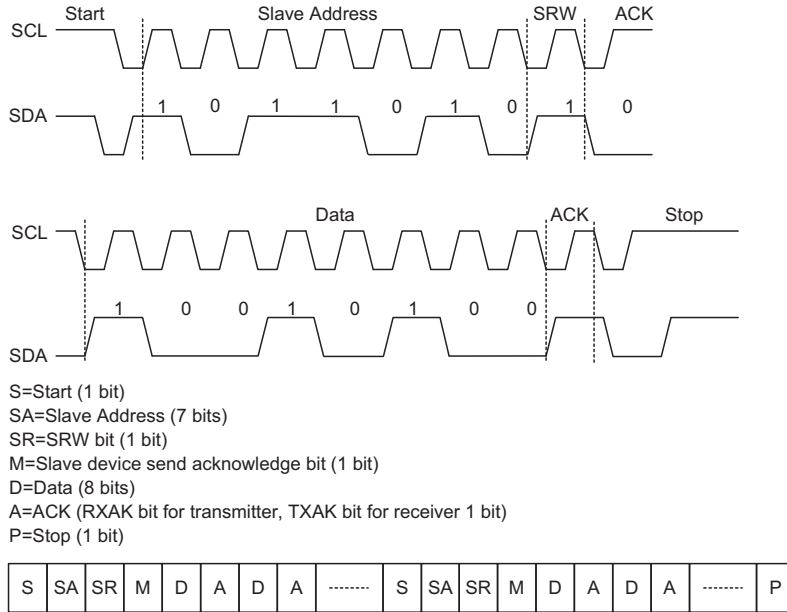
Set the EHI bit of the interrupt control register to enable the I<sup>2</sup>C bus interrupt.

- **Start Signal**

The START signal can only be generated by the master device connected to the I<sup>2</sup>C bus and not by the microcontroller, which is only a slave device. This START signal will be detected by all devices connected to the I<sup>2</sup>C bus. When detected, this indicates that the I<sup>2</sup>C bus is busy and therefore the HBB bit will be set. A START condition occurs when a high to low transition on the SDA line takes place when the SCL line remains high.

- **Slave Address**

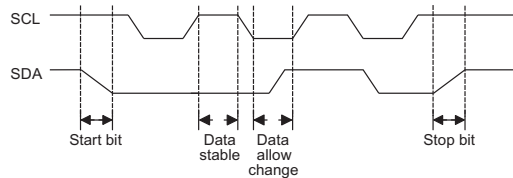
The transmission of a START signal by the master will be detected by all devices on the I<sup>2</sup>C bus. To determine which slave device the master wishes to communicate with, the address of the slave device will be sent out immediately following the START signal. All slave devices, after receiving this 7-bit address data, will compare it with their own 7-bit slave address. If the address sent out by the master matches the internal address of the microcontroller slave device, then an internal I<sup>2</sup>C bus interrupt signal will be generated. The next bit following the address, which is the 8th bit, defines the read/write status and will be saved to the SRW bit of the SIMC1 register. The device will then transmit an acknowledge bit, which is a low level, as the 9th bit. The microcontroller slave device will also set the status flag HAAS when the addresses match. As an I<sup>2</sup>C bus interrupt can come from two sources, when the program enters the interrupt subroutine, the HAAS bit should be examined to see whether the interrupt source has come from a matching slave address or from the completion of a data byte transfer. When a slave address is matched, the device must be placed in either the transmit mode and then write data to the SIMDR register, or in the receive mode where it must implement a dummy read from the SIMDR register to release the SCL line.



I<sup>2</sup>C Communication Timing Diagram

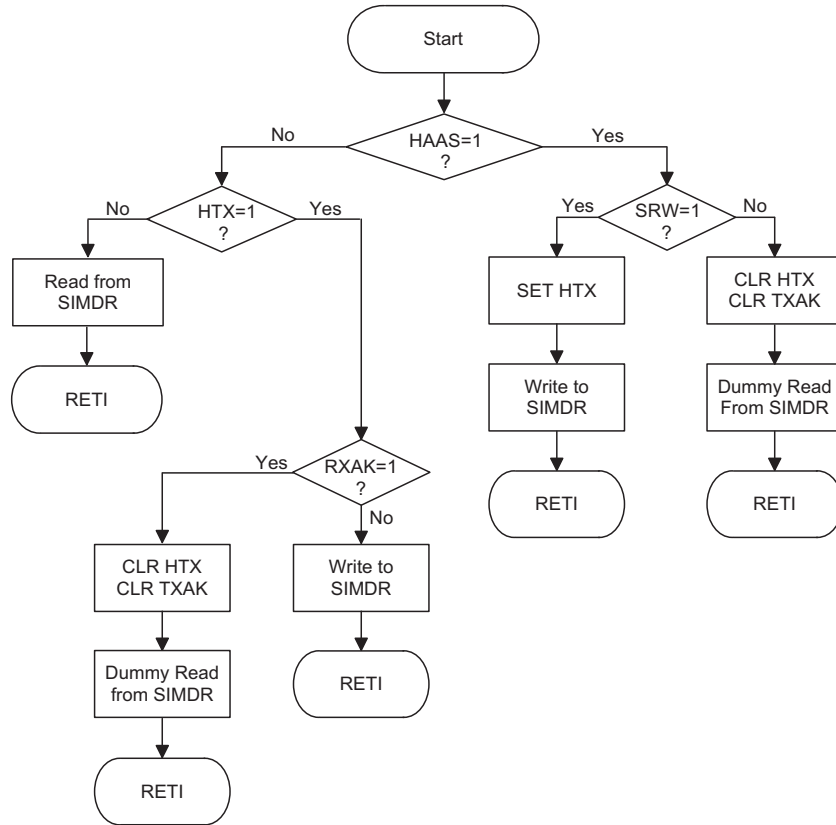
- SRW Bit**  
 The SRW bit in the SIMC1 register defines whether the microcontroller slave device wishes to read data from the I<sup>2</sup>C bus or write data to the I<sup>2</sup>C bus. The microcontroller should examine this bit to determine if it is to be a transmitter or a receiver. If the SRW bit is set to "1" then this indicates that the master wishes to read data from the I<sup>2</sup>C bus, therefore the microcontroller slave device must be setup to send data to the I<sup>2</sup>C bus as a transmitter. If the SRW bit is "0" then this indicates that the master wishes to send data to the I<sup>2</sup>C bus, therefore the microcontroller slave device must be setup to read data from the I<sup>2</sup>C bus as a receiver.
- Acknowledge Bit**  
 After the master has transmitted a calling address, any slave device on the I<sup>2</sup>C bus, whose own internal address matches the calling address, must generate an acknowledge signal. This acknowledge signal will inform the master that a slave device has accepted its calling address. If no acknowledge signal is received by the master then a STOP signal must be transmitted by the master to end the communication. When the HAAS bit is high, the addresses have matched and the microcontroller slave device must check the SRW bit to determine if it is to be a transmitter or a receiver. If the SRW bit is high, the microcontroller slave device should be setup to be a transmitter so the HTX bit in the SIMC1 register should be set to "1" if the SRW bit is low then the microcontroller slave device should be setup as a receiver and the HTX bit in the SIMC1 register should be set to "0".
- Data Byte**  
 The transmitted data is 8-bits wide and is transmitted after the slave device has acknowledged receipt of its slave address. The order of serial bit transmission is

the MSB first and the LSB last. After receipt of 8-bits of data, the receiver must transmit an acknowledge signal, level "0", before it can receive the next data byte. If the transmitter does not receive an acknowledge bit signal from the receiver, then it will release the SDA line and the master will send out a STOP signal to release control of the I<sup>2</sup>C bus. The corresponding data will be stored in the SIMDR register. If setup as a transmitter, the microcontroller slave device must first write the data to be transmitted into the SIMDR register. If setup as a receiver, the microcontroller slave device must read the transmitted data from the SIMDR register.

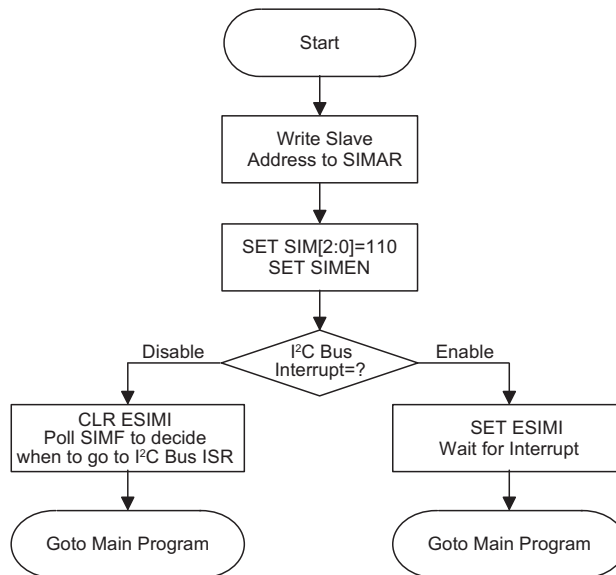


Data Timing Diagram

- Receive Acknowledge Bit**  
 When the receiver wishes to continue to receive the next data byte, it must generate an acknowledge bit, known as TXAK, on the 9th clock. The microcontroller slave device, which is setup as a transmitter will check the RXAK bit in the SIMC1 register to determine if it is to send another data byte, if not then it will release the SDA line and await the receipt of a STOP signal from the master.



I<sup>2</sup>C Bus ISR Flow Chart



I<sup>2</sup>C Bus Initialisation Flow Chart

## Interrupts

Interrupts are an important part of any microcontroller system. When an internal function such as a Time Base or Timer 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. Each device contains a Time Base interrupt and two internal timer interrupt functions. The Time Base interrupt is controlled by bit 1 of INTC register, while the internal interrupt is controlled by the Timer Counter overflow.

### Interrupt Register

Overall interrupt control, which means interrupt enabling and flag setting, is controlled using two registers, known as INTC and INTCH, which are located in the Data Memory. By controlling the appropriate enable bits in these registers each individual interrupt can be enabled or disabled. Also when an interrupt occurs, the corresponding request flag will be set by the microcontroller. The global enable flag if cleared to zero will disable all interrupts.

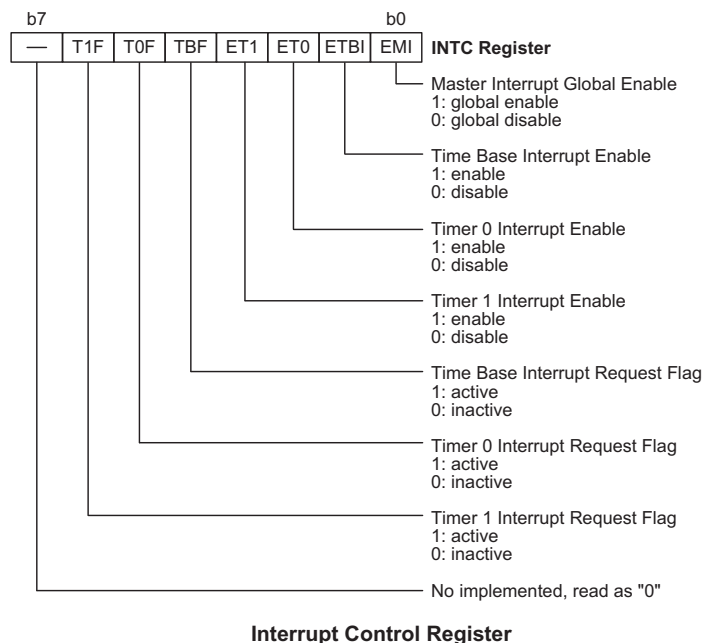
### Interrupt Operation

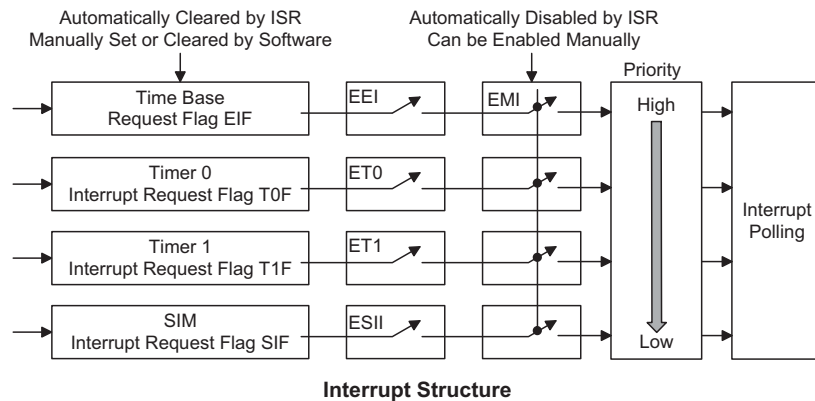
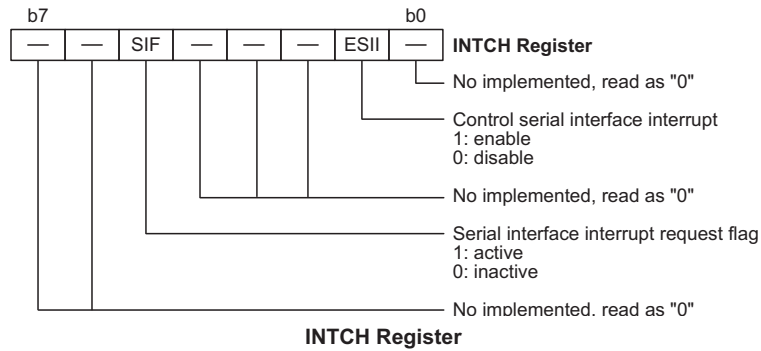
A timer or Time Base overflow or by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding inter-

rupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a JMP statement which will take program execution to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI statement, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

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

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





**Interrupt Priority**

Interrupts, occurring in the interval between the rising edges of two consecutive T2 pulses, will be serviced on the latter of the two T2 pulses, if the corresponding interrupts are enabled. In case of simultaneous requests, the accompanying table shows the priority that is applied.

Interrupt Source	Interrupt Vector	HT83FXX Priority
Time Base Interrupt	04H	1
Timer 0 Overflow	08H	2
Timer 1 Overflow	0CH	3
SIM Interrupt	14H	4

Suitable masking of the individual interrupts using the INTC and INTCH registers can prevent simultaneous occurrences.

**Time Base Interrupt**

Each device contains a Time Base whose corresponding interrupt enable bits are known as ETBI and is located in the INTC register. For a Time Base generated interrupt to occur, the corresponding Time Base interrupt enable bit must be first set. Time Base also has a corresponding Time Base interrupt request flag, which is known as TBF, also located in the INTC register. When the master interrupt and corresponding timer interrupt enable bits are enabled, the stack is not full, and when the corresponding timer overflows a subroutine call to the corresponding Time Base interrupt vector will occur. The corresponding Program Memory vector locations for the Time Base is 04H. After entering the interrupt execution routine, the corresponding interrupt request flag, TBF will be reset and the EMI bit will be cleared to disable other interrupts.

### Timer Interrupt

For a timer generated interrupt to occur, the corresponding timer interrupt enable bit must be first set. Each device contains two 8-bit timers whose corresponding interrupt enable bits are known as ET0 and ET1 and are located in the INTC register. Each timer also has a corresponding timer interrupt request flag, which are known as T0F and T1F, also located in the INTC register. When the master interrupt and corresponding timer interrupt enable bits are enabled, the stack is not full, and when the corresponding timer overflows a subroutine call to the corresponding timer interrupt vector will occur. The corresponding Program Memory vector locations for Timer 0 and Timer 1 are 08H and 0CH. After entering the interrupt execution routine, the corresponding interrupt request flags, T0F or T1F will be reset and the EMI bit will be cleared to disable other interrupts.

### Serial Interface Module - SIM - Interrupt

SIM Interrupts include both the SPI and I<sup>2</sup>C Interrupts. The SIM Mode is determined by the SIM2, SIM1 and SIM0 bits in the SIMC0 register.

For a SPI Interrupt to occur, the global interrupt enable bit, EMI, and the corresponding SIM interrupt enable bit, ESII, must be first set. The SIMEN bit in the SIMC0 register must also be set. An actual SPI Interrupt will take place when the flag, SIF, is set, a situation that will occur when 8-bits of data are transferred or received from either of the SPI interfaces. When the interrupt is enabled, the stack is not full and an SIM interrupt occurs, a subroutine call to the SIM interrupt vector at location 14H, will take place. When the interrupt is serviced, the SPI interrupt request flag, SIF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

For an I<sup>2</sup>C interrupt to occur, the corresponding interrupt enable bit ESII must be first set. An actual I<sup>2</sup>C interrupt will be initialized when the SIM interrupt request flag, SIF, is set, a situation that will occur when a matching I<sup>2</sup>C slave address is received or from the completion of an I<sup>2</sup>C data byte transfer. When the interrupt is enabled, the stack is not full and a SIM interrupt occurs, a subroutine call to the SIM interrupt vector at location 14H, will take place. When an I<sup>2</sup>C interrupt occurs, the interrupt request flag SIF will be reset and the EMI bit will be cleared to disable other interrupts.

### Programming Considerations

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

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

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

### Reset and Initialisation

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

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

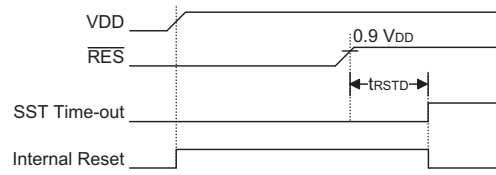
### Reset Functions

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

- Power-on Reset

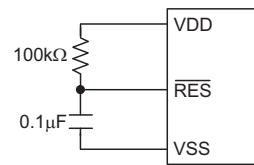
The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions.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

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



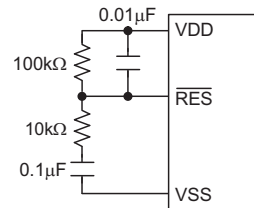
Power-On Reset Timing Chart

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



Basic Reset Circuit

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

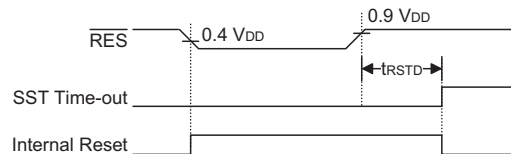


Enhanced Reset Circuit

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

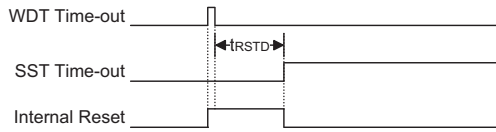
- $\overline{\text{RES}}$  Pin Reset

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



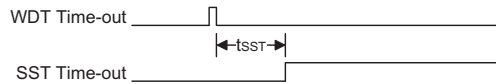
RES Reset Timing Chart

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



**WDT Time-out Reset during Normal Operation  
Timing Chart**

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



**WDT Time-out Reset during Power Down  
Timing Chart**

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

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

#### Reset Initial Conditions

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

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

Note: "u" stands for unchanged

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

Register	Reset (Power-on)	WDT Time-out (Normal Operation)	RES Reset (Normal Operation)	RES Reset (HALT)	WDT Time-out from HALT
MP	-xxx xxxx	-uuu uuuu	-uuu uuuu	-uuu uuuu	-uuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
WDS	0000 0111	0000 0111	0000 0111	0000 0111	uuuu uuuu
STATUS	--00 xxxx	--1u uuuu	--uu uuuu	--01 uuuu	--11 uuuu
INTC	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TMR0C	0100 0000	0100 0000	0100 0000	0100 0000	uuuu uuuu
TMR1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TMR1C	0100 0000	0100 0000	0100 0000	0100 0000	uuuu uuuu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	---- 1111	---- 1111	---- 1111	---- 1111	---- uuuu
PBC	---- 1111	---- 1111	---- 1111	---- 1111	---- uuuu
INTCH	--00 --00	--00 --00	--00 --00	--00 --00	--uu --uu
SIMC0	111x xx0-	111x xx0-	111x xx0-	111x xx0-	uuux xxu-
SIMC1	100x x0x1	100x x0x1	100x x0x1	100x x0x1	uuux xuxu
SIMDR	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx
SIMAR/ SIMC2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
DAL	xxxx ----	uuuu ----	uuuu ----	uuuu ----	uuuu ----
DAH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PWMCR	0--- 0000	0--- 0000	0--- 0000	0--- 0000	u--- uuuu
PWML	xxxx ----	uuuu ----	uuuu ----	uuuu ----	uuuu ----
PWMH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
VOL	xxxx -xxx	uuuu -uuu	uuuu -uuu	uuuu -uuu	uuuu -uuu

Note: "u" stands for unchanged  
"x" stands for unknown  
"--" stands for undefined

## Oscillator

Various oscillator options offer the user a wide range of functions according to their various application requirements. Two types of system clocks can be selected while various clock source options for the Watchdog Timer are provided for maximum flexibility. All oscillator options are selected through the configuration options.

The two methods of generating the system clock are:

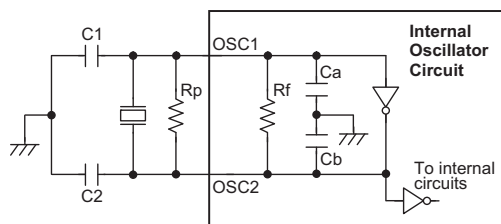
- External crystal/resonator oscillator
- External RC oscillator

One of these two methods must be selected using the configuration options.

More information regarding the oscillator is located in Application Note HA0075E on the Holtek website.

### External Crystal/Resonator Oscillator

The simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation, and will normally not require external capacitors. However, for some crystals and most resonator types, to ensure oscillation and accurate frequency generation, it may be necessary to add two small value external capacitors, C1 and C2. The exact values of C1 and C2 should be selected in consultation



Note: 1. Rp is normally not required.  
2. Although not shown OSC1/OSC2 pins have a parasitic capacitance of around 7pF.

### Crystal/Resonator Oscillator

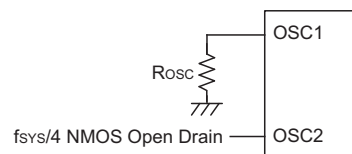
with the crystal or resonator manufacturer's specification. The external parallel feedback resistor, Rp, is normally not required but in some cases may be needed to assist with oscillation start up.

Internal Ca, Cb, Rf Typical Values @ 5V, 25°C		
Ca	Cb	Rf
11~13pF	13~15pF	800kΩ

Oscillator Internal Component Values

### External RC Oscillator

Using the external system RC oscillator requires that a resistor, with a value between 150kΩ and 300kΩ, is connected between OSC1 and VSS. The generated system clock divided by 4 will be provided on OSC2 as an output which can be used for external synchronization purposes. Note that as the OSC2 output is an NMOS open-drain type, a pull high resistor should be connected if it to be used to monitor the internal frequency. Although this is a cost effective oscillator configuration, the oscillation frequency can vary with VDD, temperature and process variations and is therefore not suitable for applications where timing is critical or where accurate oscillator frequencies are required. For the value of the external resistor R<sub>OSC</sub> refer to the Holtek website for typical RC Oscillator vs. Temperature and VDD characteristics graphics. Note that it is the only microcontroller internal circuitry together with the external resistor, that determine the frequency of the oscillator. The external capacitor shown on the diagram does not influence the frequency of oscillation.



External RC Oscillator

### Watchdog Timer Oscillator

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

## Power Down Mode and Wake-up

### Power Down Mode

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

### Entering the Power Down Mode

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

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

### Standby Current Considerations

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

### Wake-up

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

- An external reset
- An external falling edge on Port A
- A system interrupt
- A WDT overflow

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

Each pin on Port A can be setup via an individual configuration option 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 to "1" before entering the Power Down Mode, the wake-up function of the related interrupt will be disabled.

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

## Low Drop Output – LDO

All device include a fully integrated LDO regulator which can be used to provide a fixed voltage for user applications. The integrated LDO is a simple three terminal device with an external input pin, LDO\_IN, external output pin, LDO\_OUT, and a ground pin connected to the device VSS pin. Implemented in CMOS technology, it can deliver a 100mA output current and allow an input voltage as high as 24V. It will supply a fixed output voltage level of 3.3V. Using CMOS technology ensures that the regulator has a low dropout voltage and a low quiescent current.

## Low Voltage Detector – LVD

The Low Voltage Detector internal function provides a means for the user to monitor when the power supply voltage falls below a certain fixed level as specified in the DC characteristics.

### Operation

The Low Voltage Detector must first be enabled using a configuration option.

The LVD control bit is bit 2 of the PWMCR register and is known as LVDF. Under normal operation, and when the power supply voltage is above the specified VLVD value in the DC characteristic section, the LVDF bit will remain at a zero value. If the power supply voltage should fall below this VLVD value then the LVDF bit will change to a high value indicating a low voltage condition. Note that the LVDF bit is a read-only bit. By polling the LVDF bit in the PWMCR register, the application program can therefore determine the presence of a low voltage condition.

## Watchdog Timer

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

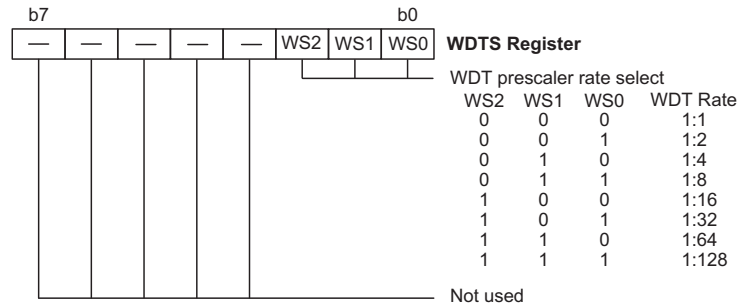
The internal WDT oscillator has an approximate period of 65 $\mu$ s at a supply voltage of 5V. If selected, it is first divided by 256 via an 8-stage counter to give a nominal

period of 17ms. Note that this period can vary with VDD, temperature and process variations. For longer WDT time-out periods the WDT prescaler can be utilized. By writing the required value to bits 0, 1 and 2 of the WDTS register, known as WS0, WS1 and WS2, longer time-out periods can be achieved. With WS0, WS1 and WS2 all equal to 1, the division ratio is 1:128 which gives a maximum time-out period of about 2.1s.

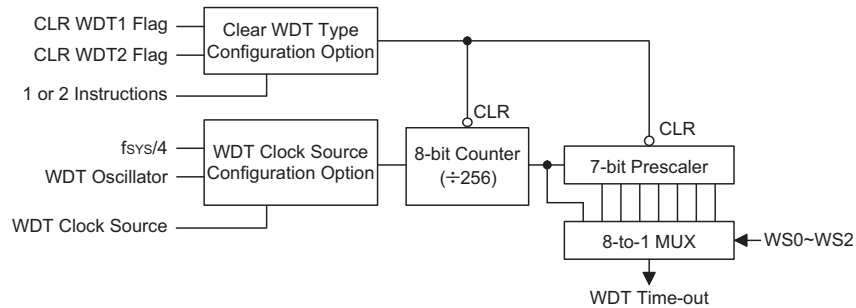
A configuration option can select the instruction clock, which is the system clock divided by 4, as the WDT clock source instead of the internal WDT oscillator. If the instruction clock is used as the clock source, it must be noted that when the system enters the Power Down Mode, as the system clock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

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

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



**Watchdog Timer Register**



**Watchdog Timer**

**Voice Output**

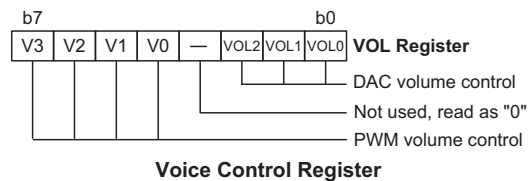
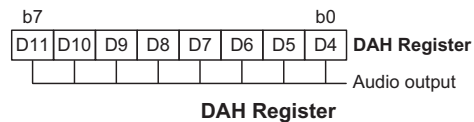
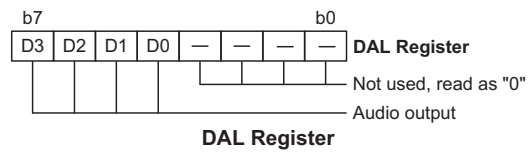
The device contains an internal 12-bit DAC function which can be used for audio signal generation.

**Voice Control**

Two internal registers DAL and DAH contain the 12-bit digital value for conversion by the internal DAC. There is also a DAC enable/disable control bit in the PWMCR control register for overall on/off control of the DAC circuit. If the DAC circuit is not enabled, the DAH/DAL value outputs will be invalid. Writing a "1" to the DAC bit in bit1 of PWMCR will enable the enable DAC circuit, while writing a "0" to the DAC bit will disable the DAC circuit.

**Audio Output and Volume Control – DAL, DAH, VOL**

The audio output is 12-bits wide whose highest 8-bits are written into the DAH register and whose lowest four bits are written into the highest four bits of the DAL register. Bits 0~3 of the DAL register are always read as zero. There are 8 levels of volume which are setup using the VOL register. Only the lowest 3-bits of this register are used for volume control.



**Voice Control Register**

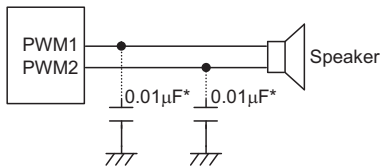
### Pulse Width Modulation Output

All devices include a single 12-bit PWM function which can directly drive external audio components such as speakers.

#### Pulse Width Modulator Operation

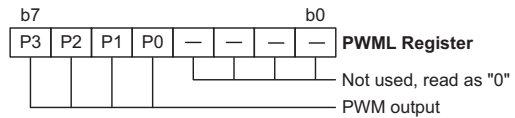
The PWM output is provided on two complimentary outputs on the PWM1 and PWM2 pins, providing a differential output pair and thus capable of higher drive power. These two pins can directly drive a piezo buzzer or an 8 ohm speaker without using external components. The PWM outputs can also be used single ended, where the signal is provided on the PWM1 output, and again can also be used by itself alone to drive a piezo buzzer or an 8 ohm speaker without external components. This single end output drive type is chosen using the Single\_PWM bit in the PWMCR register.

If the MSB\_SIGN bit is low, then the signal that is provided on PWM1 and PWM2 will obtain a GND level voltage after setting the PWMCC bit high. If the MSB\_SIGN bit is high, then the signal that is provided on PWM2 and PWM1 will have a GND level voltage when the PWMCC bit is set high.

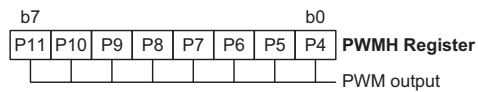


Note: "\*" For reducing the digital noise that PWM may cause, can consider increment capacitors.

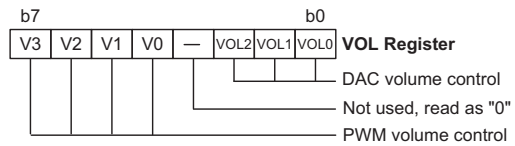
The two PWM outputs will initially be at low levels, and if the PWM function is stopped will also return to a low level. If the PWMCC bit changes from low to high then the PWM function will start running and latch new data. If the data is not updated then the old value will remain. If the PWMCC bit changes from high to low, at the end of the duty cycle, the PWM output will stop.



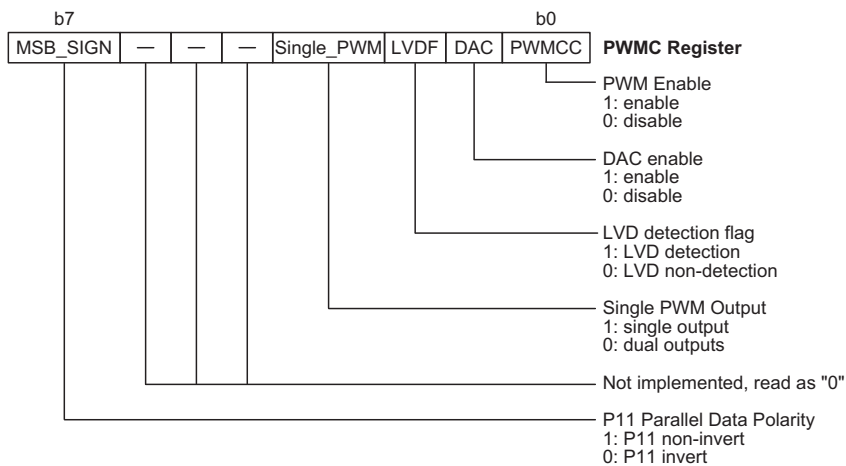
Pulse Width Modulator Data Low Register



Pulse Width Modulator Data High Register



Voice Control Register



Pulse Width Modulator Control Register

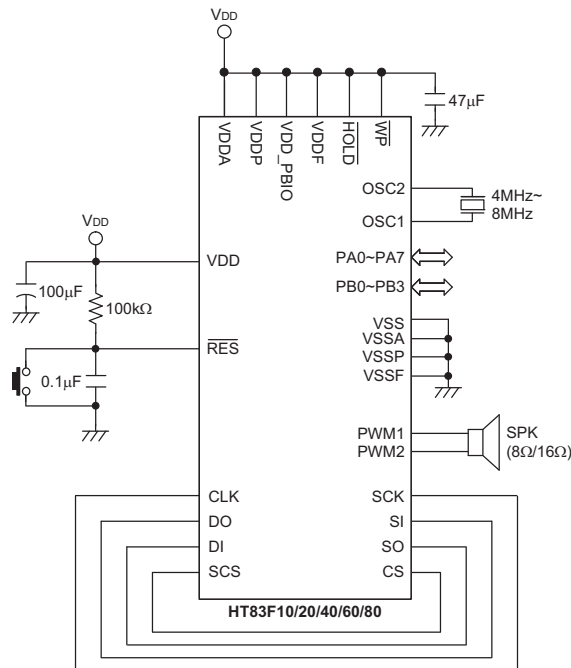
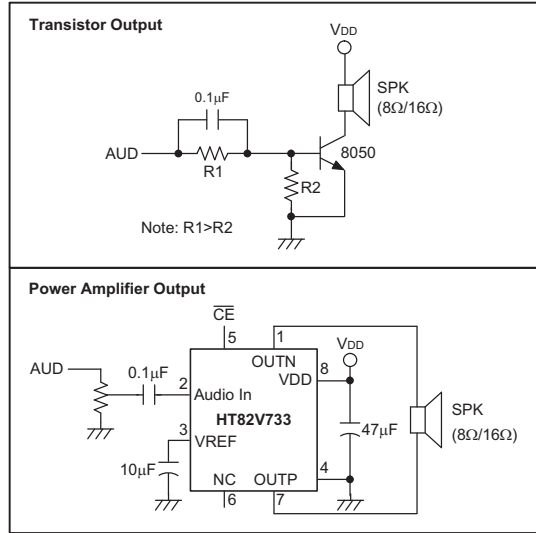
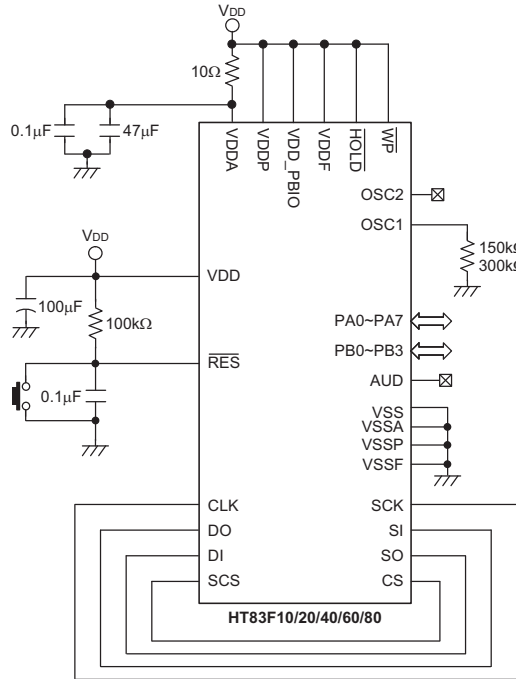
## Configuration Options

Configuration options refer to certain options within the MCU that are programmed into the device during the programming process. During the development process, these options are selected using the HT-IDE software development tools. As these options are programmed into the device using the hardware programming tools, once they are selected they cannot be changed later by the application software.

No.	Options
<b>I/O Options</b>	
1	PA0~PA7: wake-up enable or disable
2	PA0~PA7: pull-high enable or disable
3	PB0~PB3: pull-high enable or disable
<b>Oscillator Options</b>	
4	OSC type selection: RC or crystal
<b>Watchdog Options</b>	
5	WDT: enable or disable
6	WDT clock source: WDROSC or T1
<b>PB I/O Port Output Voltage Options</b>	
7	VDD_PBIO/VDD type selection: VDD_PBIO or VDD for Port B, SPI, I <sup>2</sup> C I/O per bit
<b>LVD Options</b>	
8	LVD function: enable or disable
<b>SIM Options</b>	
9	SIM Function: enable or disable
10	SPI S/W CSEN: enable or disable
11	SPI S/W WCOL: enable or disable
<b>I<sup>2</sup>C Options</b>	
12	I <sup>2</sup> C RNIC: enable or disable
13	I <sup>2</sup> C debounce time: 0/1/2 system clocks

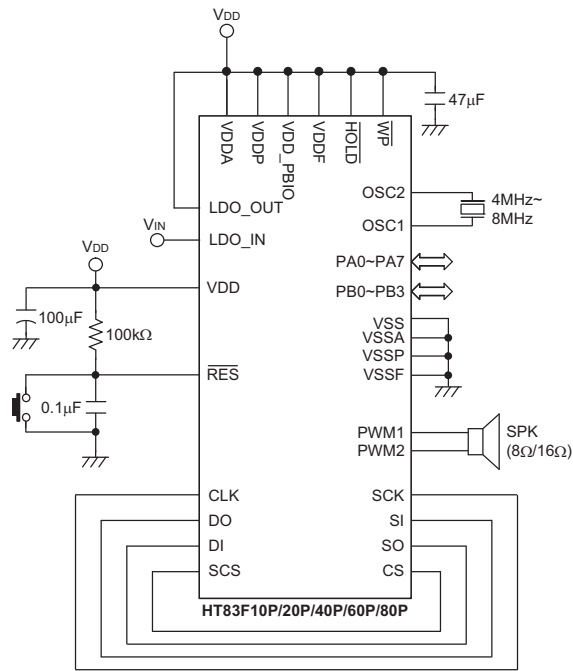
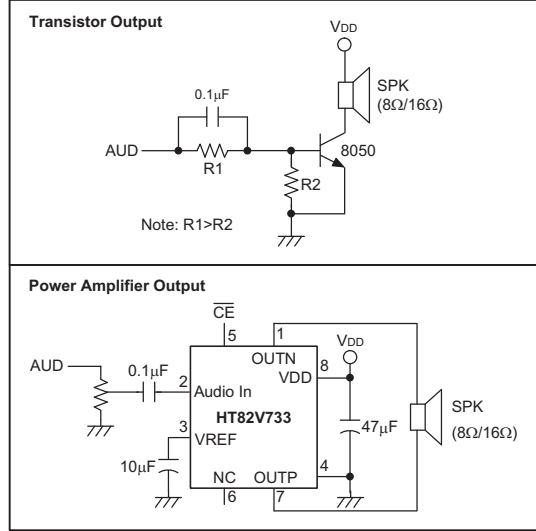
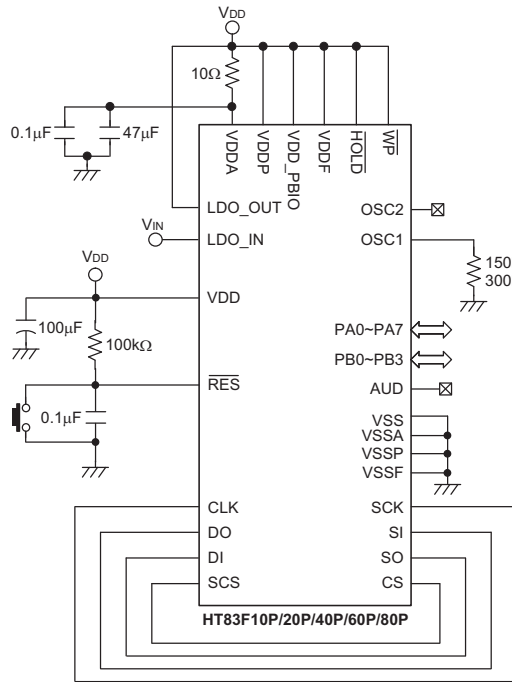
Application Circuits

V<sub>DD</sub>=2.7V~3.6V



Note: The PWM application refer to the description of Pulse Width Modulation Output.

$V_{IN}=3.6V\sim 24V$



Note: The PWM application refer to the description of Pulse Width Modulation Output.

## Instruction Set

### Introduction

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

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

### Instruction Timing

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

### Moving and Transferring Data

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

### Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and

subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

### Logical and Rotate Operations

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

### Branches and Control Transfer

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

### Bit Operations

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

### Table Read Operations

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

### Other Operations

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

### Instruction Set Summary

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

Table conventions:

x: Bits immediate data

m: Data Memory address

A: Accumulator

i: 0-7 number of bits

addr: Program memory address

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

Mnemonic	Description	Cycles	Flag Affected
<b>Rotate</b>			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	↑ <sup>Note</sup>	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	↑ <sup>Note</sup>	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	↑ <sup>Note</sup>	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	↑ <sup>Note</sup>	C
<b>Data Move</b>			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	↑ <sup>Note</sup>	None
MOV A,x	Move immediate data to ACC	1	None
<b>Bit Operation</b>			
CLR [m].i	Clear bit of Data Memory	↑ <sup>Note</sup>	None
SET [m].i	Set bit of Data Memory	↑ <sup>Note</sup>	None
<b>Branch</b>			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	↑ <sup>Note</sup>	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	↑ <sup>note</sup>	None
SZ [m].i	Skip if bit i of Data Memory is zero	↑ <sup>Note</sup>	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	↑ <sup>Note</sup>	None
SIZ [m]	Skip if increment Data Memory is zero	↑ <sup>Note</sup>	None
SDZ [m]	Skip if decrement Data Memory is zero	↑ <sup>Note</sup>	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	↑ <sup>Note</sup>	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	↑ <sup>Note</sup>	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
<b>Table Read</b>			
TABRDC [m]	Read table (current page) to TBLH and Data Memory	2 <sup>Note</sup>	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2 <sup>Note</sup>	None
<b>Miscellaneous</b>			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	↑ <sup>Note</sup>	None
SET [m]	Set Data Memory	↑ <sup>Note</sup>	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	↑ <sup>Note</sup>	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**

<b>ADC A,[m]</b>	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	$ACC \leftarrow ACC + [m] + C$
Affected flag(s)	OV, Z, AC, C
<b>ADCM A,[m]</b>	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	$[m] \leftarrow ACC + [m] + C$
Affected flag(s)	OV, Z, AC, C
<b>ADD A,[m]</b>	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	$ACC \leftarrow ACC + [m]$
Affected flag(s)	OV, Z, AC, C
<b>ADD A,x</b>	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	$ACC \leftarrow ACC + x$
Affected flag(s)	OV, Z, AC, C
<b>ADDM A,[m]</b>	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	$[m] \leftarrow ACC + [m]$
Affected flag(s)	OV, Z, AC, C
<b>AND A,[m]</b>	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	$ACC \leftarrow ACC \text{ "AND" } [m]$
Affected flag(s)	Z
<b>AND A,x</b>	Logical AND immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	$ACC \leftarrow ACC \text{ "AND" } x$
Affected flag(s)	Z
<b>ANDM A,[m]</b>	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	$[m] \leftarrow ACC \text{ "AND" } [m]$
Affected flag(s)	Z

<b>CALL addr</b>	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 $\leftarrow$ Program Counter + 1 Program Counter $\leftarrow$ addr
Affected flag(s)	None
<b>CLR [m]</b>	Clear Data Memory
Description	Each bit of the specified Data Memory is cleared to 0.
Operation	[m] $\leftarrow$ 00H
Affected flag(s)	None
<b>CLR [m].i</b>	Clear bit of Data Memory
Description	Bit i of the specified Data Memory is cleared to 0.
Operation	[m].i $\leftarrow$ 0
Affected flag(s)	None
<b>CLR WDT</b>	Clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared.
Operation	WDT cleared TO $\leftarrow$ 0 PDF $\leftarrow$ 0
Affected flag(s)	TO, PDF
<b>CLR WDT1</b>	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. Repeatedly executing this instruction without alternately executing CLR WDT2 will have no effect.
Operation	WDT cleared TO $\leftarrow$ 0 PDF $\leftarrow$ 0
Affected flag(s)	TO, PDF
<b>CLR WDT2</b>	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. Repeatedly executing this instruction without alternately executing CLR WDT1 will have no effect.
Operation	WDT cleared TO $\leftarrow$ 0 PDF $\leftarrow$ 0
Affected flag(s)	TO, PDF

<b>CPL [m]</b>	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
<b>CPLA [m]</b>	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
<b>DAA [m]</b>	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	$[m] \leftarrow ACC + 00H$ or $[m] \leftarrow ACC + 06H$ or $[m] \leftarrow ACC + 60H$ or $[m] \leftarrow ACC + 66H$
Affected flag(s)	C
<b>DEC [m]</b>	Decrement Data Memory
Description	Data in the specified Data Memory is decremented by 1.
Operation	$[m] \leftarrow [m] - 1$
Affected flag(s)	Z
<b>DECA [m]</b>	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	$ACC \leftarrow [m] - 1$
Affected flag(s)	Z
<b>HALT</b>	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	$TO \leftarrow 0$ $PDF \leftarrow 1$
Affected flag(s)	TO, PDF

<b>INC [m]</b>	Increment Data Memory
Description	Data in the specified Data Memory is incremented by 1.
Operation	$[m] \leftarrow [m] + 1$
Affected flag(s)	Z
<b>INCA [m]</b>	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	$ACC \leftarrow [m] + 1$
Affected flag(s)	Z
<b>JMP addr</b>	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	$Program\ Counter \leftarrow addr$
Affected flag(s)	None
<b>MOV A,[m]</b>	Move Data Memory to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator.
Operation	$ACC \leftarrow [m]$
Affected flag(s)	None
<b>MOV A,x</b>	Move immediate data to ACC
Description	The immediate data specified is loaded into the Accumulator.
Operation	$ACC \leftarrow x$
Affected flag(s)	None
<b>MOV [m],A</b>	Move ACC to Data Memory
Description	The contents of the Accumulator are copied to the specified Data Memory.
Operation	$[m] \leftarrow ACC$
Affected flag(s)	None
<b>NOP</b>	No operation
Description	No operation is performed. Execution continues with the next instruction.
Operation	No operation
Affected flag(s)	None
<b>OR A,[m]</b>	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	$ACC \leftarrow ACC \text{ "OR" } [m]$
Affected flag(s)	Z

<b>OR A,x</b>	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
<b>ORM A,[m]</b>	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
<b>RET</b>	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
<b>RET A,x</b>	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
<b>RETI</b>	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
<b>RL [m]</b>	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
<b>RLA [m]</b>	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

<b>RLC [m]</b>	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	$[m].(i+1) \leftarrow [m].i; (i = 0\sim6)$ $[m].0 \leftarrow C$ $C \leftarrow [m].7$
Affected flag(s)	C
<b>RLCA [m]</b>	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	$ACC.(i+1) \leftarrow [m].i; (i = 0\sim6)$ $ACC.0 \leftarrow C$ $C \leftarrow [m].7$
Affected flag(s)	C
<b>RR [m]</b>	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	$[m].i \leftarrow [m].(i+1); (i = 0\sim6)$ $[m].7 \leftarrow [m].0$
Affected flag(s)	None
<b>RRA [m]</b>	Rotate Data Memory right with result in ACC
Description	Data in the specified Data Memory and the carry flag are rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation	$ACC.i \leftarrow [m].(i+1); (i = 0\sim6)$ $ACC.7 \leftarrow [m].0$
Affected flag(s)	None
<b>RRC [m]</b>	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	$[m].i \leftarrow [m].(i+1); (i = 0\sim6)$ $[m].7 \leftarrow C$ $C \leftarrow [m].0$
Affected flag(s)	C
<b>RRCA [m]</b>	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	$ACC.i \leftarrow [m].(i+1); (i = 0\sim6)$ $ACC.7 \leftarrow C$ $C \leftarrow [m].0$
Affected flag(s)	C

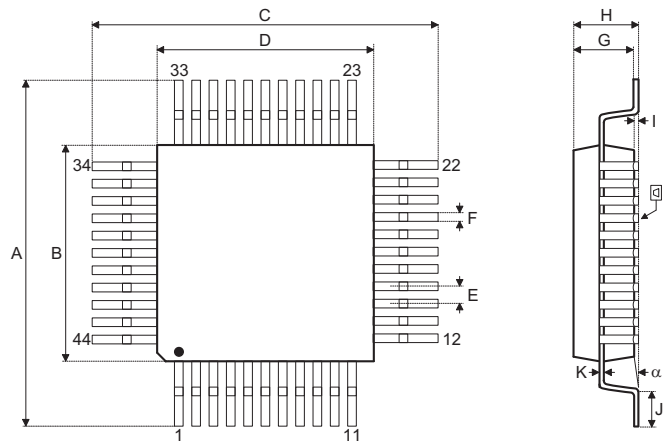
<b>SBC A,[m]</b>	Subtract Data Memory from ACC with Carry
Description	The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$ACC \leftarrow ACC - [m] - \bar{C}$
Affected flag(s)	OV, Z, AC, C
<b>SBCM A,[m]</b>	Subtract Data Memory from ACC with Carry and result in Data Memory
Description	The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$[m] \leftarrow ACC - [m] - \bar{C}$
Affected flag(s)	OV, Z, AC, C
<b>SDZ [m]</b>	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	$[m] \leftarrow [m] - 1$ Skip if $[m] = 0$
Affected flag(s)	None
<b>SDZA [m]</b>	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	$ACC \leftarrow [m] - 1$ Skip if $ACC = 0$
Affected flag(s)	None
<b>SET [m]</b>	Set Data Memory
Description	Each bit of the specified Data Memory is set to 1.
Operation	$[m] \leftarrow FFH$
Affected flag(s)	None
<b>SET [m].i</b>	Set bit of Data Memory
Description	Bit i of the specified Data Memory is set to 1.
Operation	$[m].i \leftarrow 1$
Affected flag(s)	None

<b>SIZ [m]</b>	Skip if increment Data Memory is 0
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	$[m] \leftarrow [m] + 1$ Skip if $[m] = 0$
Affected flag(s)	None
<b>SIZA [m]</b>	Skip if increment Data Memory is zero with result in ACC
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	$ACC \leftarrow [m] + 1$ Skip if $ACC = 0$
Affected flag(s)	None
<b>SNZ [m].i</b>	Skip if bit i of Data Memory is not 0
Description	If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.
Operation	Skip if $[m].i \neq 0$
Affected flag(s)	None
<b>SUB A,[m]</b>	Subtract Data Memory from ACC
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$ACC \leftarrow ACC - [m]$
Affected flag(s)	OV, Z, AC, C
<b>SUBM A,[m]</b>	Subtract Data Memory from ACC with result in Data Memory
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$[m] \leftarrow ACC - [m]$
Affected flag(s)	OV, Z, AC, C
<b>SUB A,x</b>	Subtract immediate data from ACC
Description	The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	$ACC \leftarrow ACC - x$
Affected flag(s)	OV, Z, AC, C

<b>SWAP [m]</b>	Swap nibbles of Data Memory
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged.
Operation	$[m].3 \sim [m].0 \leftrightarrow [m].7 \sim [m].4$
Affected flag(s)	None
<b>SWAPA [m]</b>	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	$ACC.3 \sim ACC.0 \leftarrow [m].7 \sim [m].4$ $ACC.7 \sim ACC.4 \leftarrow [m].3 \sim [m].0$
Affected flag(s)	None
<b>SZ [m]</b>	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
<b>SZA [m]</b>	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	$ACC \leftarrow [m]$ Skip if $[m] = 0$
Affected flag(s)	None
<b>SZ [m].i</b>	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
<b>TABRDC [m]</b>	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	$[m] \leftarrow$ program code (low byte) $TBLH \leftarrow$ program code (high byte)
Affected flag(s)	None
<b>TABRDL [m]</b>	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	$[m] \leftarrow$ program code (low byte) $TBLH \leftarrow$ program code (high byte)
Affected flag(s)	None

---

<b>XOR A,[m]</b>	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
<b>XORM A,[m]</b>	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
<b>XOR A,x</b>	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

**44-pin QFP (10mm×10mm) Outline Dimensions**


Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.512	—	0.528
B	0.390	—	0.398
C	0.512	—	0.528
D	0.390	—	0.398
E	—	0.031	—
F	—	0.012	—
G	0.075	—	0.087
H	—	—	0.106
I	0.010	—	0.020
J	0.029	—	0.037
K	0.004	—	0.008
L	—	0.004	—
$\alpha$	0°	—	7°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	13.00	—	13.40
B	9.90	—	10.10
C	13.00	—	13.40
D	9.90	—	10.10
E	—	0.80	—
F	—	0.30	—
G	1.90	—	2.20
H	—	—	2.70
I	0.25	—	0.50
J	0.73	—	0.93
K	0.10	—	0.20
L	—	0.10	—
$\alpha$	0°	—	7°

**Holtek Semiconductor Inc. (Headquarters)**

No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan  
Tel: 886-3-563-1999  
Fax: 886-3-563-1189  
<http://www.holtek.com.tw>

**Holtek Semiconductor Inc. (Taipei Sales Office)**

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

**Holtek Semiconductor Inc. (Shenzhen Sales Office)**

5F, Unit A, Productivity Building, No.5 Gaoxin M 2nd Road, Nanshan District, Shenzhen, China 518057  
Tel: 86-755-8616-9908, 86-755-8616-9308  
Fax: 86-755-8616-9722

**Holtek Semiconductor (USA), Inc. (North America Sales Office)**

46729 Fremont Blvd., Fremont, CA 94538  
Tel: 1-510-252-9880  
Fax: 1-510-252-9885  
<http://www.holtek.com>

Copyright © 2010 by HOLTEK SEMICONDUCTOR INC.

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