Note that the HT46F46E/HT46F48E devices and the HT46F49E 24/28SKDIP packages, although mentioned in this datasheet, have already been phased out and are presently no longer available.

Features

- Operating voltage:
  f_{SYS}=4MHz: 2.2V~5.5V
  f_{SYS}=8MHz: 3.3V~5.5V
  f_{SYS}=12MHz: 4.5V~5.5V
- 13 to 23 bidirectional I/O lines
- External interrupt input shared with an I/O line
- 8-bit programmable Timer/Event Counter with overflow interrupt and 7-stage prescaler
- On-chip crystal and RC oscillator
- Watchdog Timer function
- PFD for audio frequency generation
- Power down and wake-up functions to reduce power consumption
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD}=5V
- 4 or 6-level subroutine nesting
- 4 channels 8 or 9-bit resolution A/D converter
- 1 or 2 channel 8-bit PWM output shared with I/O lines
- Bit manipulation instruction
- Table read instructions
- 63 powerful instructions
- All instructions executed in one or two machine cycles
- Low voltage reset function
- Flash program memory can be re-programmed up to 10,000 times
- True EEPROM data memory can be re-programmed up to 100,000 times
- Flash program memory data retention > 10 years
- True EEPROM data memory data retention > 10 years
- ISP (In-System Programming) interface
- Range of packaging types

General Description

The Cost-Effective A/D Flash Type MCU with true EEPROM Devices are a series of 8-bit high performance RISC architecture microcontrollers, designed especially for applications that interface directly to analog signals, such as those from sensors. All devices include an integrated multi-channel Analog to Digital Converter in addition to one or two Pulse Width Modulation outputs. The usual Holtek MCU features such as power down and wake-up functions, oscillator options, programmable frequency divider, etc. combine to ensure user applications require a minimum of external components.

The benefits of integrated A/D and PWM functions, in addition to low power consumption, high performance, I/O flexibility and low-cost, provide these devices with the versatility to suit a wide range of application possibilities such as sensor signal processing, motor driving, industrial control, consumer products, subsystem controllers, etc. Many features are common to all devices, however, they differ in areas such as I/O pin count, Program Memory capacity, A/D resolution, stack capacity and package types.

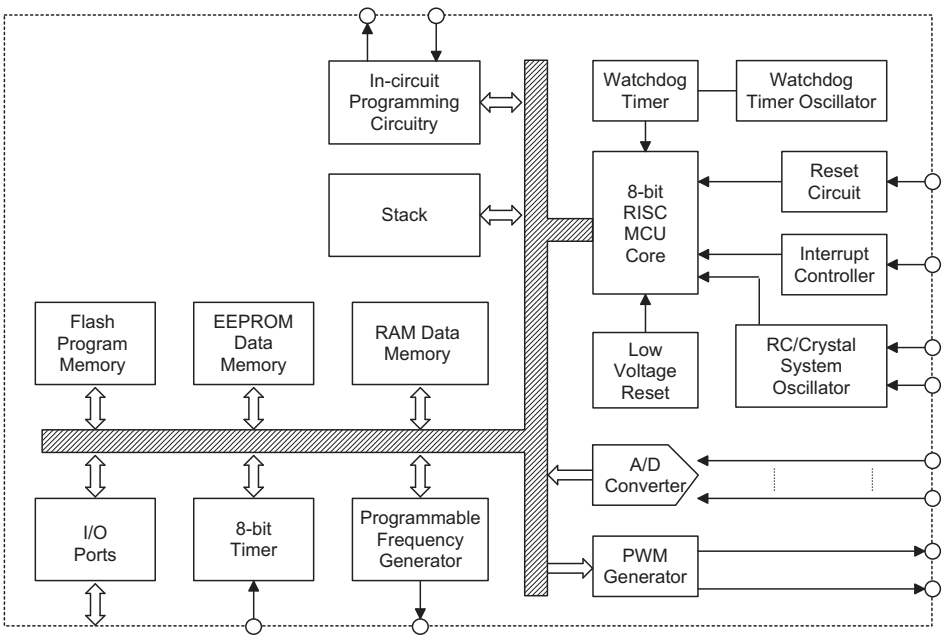
Selection Table

Most features are common to all devices, the main feature distinguishing them are Program Memory capacity, I/O count, A/D resolution, stack capacity and package types. The following table summarises the main features of each device.

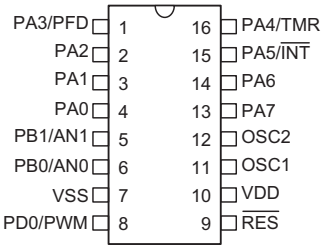
Part No.	VDD	Program Memory	Data Memory	Data EEPROM	I/O	Timer	Int.	A/D	PWM	Stack	Package Types
HT46F46E	2.2V~5.5V	1K×14	64×8	128×8	13	8-bit×1	3	8-bit×4	8-bit×1	4	16NSOP, 18SOP
HT46F47E	2.2V~5.5V	2K×14	64×8	128×8	13	8-bit×1	3	9-bit×4	8-bit×1	6	18SOP 20SSOP
HT46F48E	2.2V~5.5V	2K×14	88×8	128×8	19	8-bit×1	3	9-bit×4	8-bit×1	6	24SOP/SSOP
HT46F49E	2.2V~5.5V	4K×15	128×8	256×8	23	8-bit×1	3	9-bit×4	8-bit×2	6	24SOP/SSOP 28SOP

Note: For devices that exist in two package formats, the table reflects the situation for the larger package.

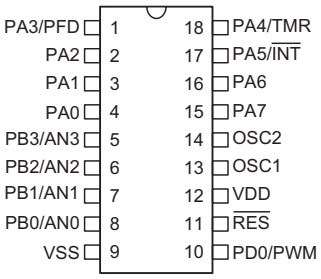
Block Diagram



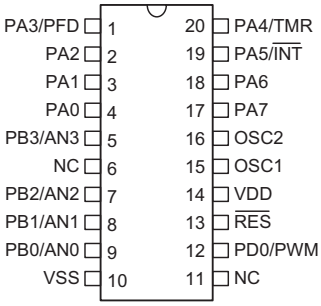
Pin Assignment



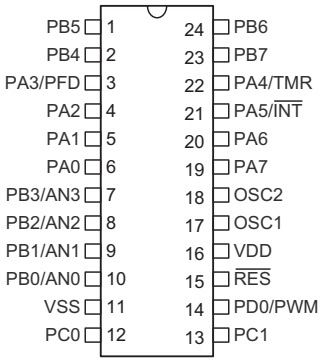
HT46F46E
16 NSOP-A



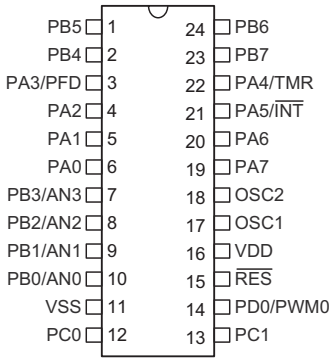
HT46F46E/HT46F47E
18 SOP-A



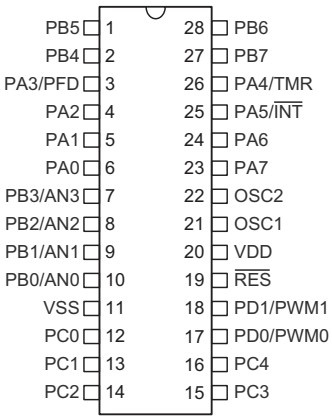
HT46F47E
20SSOP-A



HT46F48E
24 SOP-A/SSOP-A



HT46F49E
24 SOP-A/SSOP-A



HT46F49E
28 SOP-A

Pin Description

HT46F46E, HT46F47E

Pin Name	I/O	Configuration Option	Description
PA0~PA2 PA3/PFD PA4/TMR PA5/INT PA6~PA7	I/O	Pull-high Wake-up PA3 or PFD	Bidirectional 8-bit input/output port. Each individual pin on this port can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors. Pins PA3, PA4 and PA5 are pin-shared with PFD, TMR and INT, respectively.
PB0/AN0 PB1/AN1 PB2/AN2 PB3/AN3	I/O	Pull-high	Bidirectional 4-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors. PB is pin-shared with the A/D input pins. The A/D inputs are selected via software instructions. Once selected as an A/D input, the I/O function and pull-high resistor options are disabled automatically.
PD0/PWM	I/O	Pull-high PD0 or PWM	Bidirectional 1-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if this pin has a pull-high resistor. The PWM output is pin-shared with pin PD0 selected via a configuration option.
OSC1 OSC2	I O	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.
RES	I	—	Schmitt Trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground

Note: 1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
2. Individual pins can be selected to have a pull-high resistor.
3. Pins PB2/AN2~PB3/AN3 exist but are not bonded out on the 16-pin package.
4. Unbonded pins should be setup as outputs or as inputs with pull-high resistors to conserve power.

HT46F48E

Pin Name	I/O	Configuration Option	Description
PA0~PA2 PA3/PFD PA4/TMR PA5/INT PA6~PA7	I/O	Pull-high Wake-up PA3 or PFD	Bidirectional 8-bit input/output port. Each individual pin on this port can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors. Pins PA3, PA4 and PA5 are pin-shared with PFD, TMR and INT, respectively.
PB0/AN0 PB1/AN1 PB2/AN2 PB3/AN3 PB4~PB7	I/O	Pull-high	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors. PB is pin-shared with the A/D input pins. The A/D inputs are selected via software instructions. Once selected as an A/D input, the I/O function and pull-high resistor options are disabled automatically.
PC0~PC1	I/O	Pull-high	Bidirectional 2-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors.
PD0/PWM	I/O	Pull-high I/O or PWM	Bidirectional 1-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration option determines if this pin has a pull-high resistor. The PWM output is pin-shared with pin PD0 selected via a configuration option.
OSC1 OSC2	I O	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.

Pin Name	I/O	Configuration Option	Description
$\overline{RES}$	I	—	Schmitt Trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground

Note: 1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
2. Individual pins can be selected to have a pull-high resistor.

HT46F49E

Pin Name	I/O	Configuration Option	Description
PA0~PA2 PA3/PFD PA4/TMR PA5/$\overline{INT}$ PA6~PA7	I/O	Pull-high Wake-up PA3 or PFD	Bidirectional 8-bit input/output port. Each individual pin on this port can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors. Pins PA3, PA4 and PA5 are pin-shared with PFD, TMR and $\overline{INT}$, respectively.
PB0/AN0 PB1/AN1 PB2/AN2 PB3/AN3 PB4~PB7	I/O	Pull-high	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors. PB is pin-shared with the A/D input pins. The A/D inputs are selected via software instructions. Once selected as an A/D input, the I/O function and pull-high resistor options are disabled automatically.
PC0~PC4	I/O	Pull-high	Bidirectional 5-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors.
PD0/PWM0 PD1/PWM1	I/O	Pull-high I/O or PWM	Bidirectional 2-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration option determines if this pin has a pull-high resistor. The PWM output are pin-shared with pins PD0 and PD1 selected via a configuration option.
OSC1 OSC2	I O	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.
$\overline{RES}$	I	—	Schmitt Trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground

Note: 1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
2. Individual pins can be selected to have a pull-high resistor.
3. Pins PC2~PC4 and pin PD1/PWM1 exist but are not bonded out on the 24-pin package.
4. Unbonded pins should be setup as outputs or as inputs with pull-high resistors to conserve power.

Absolute Maximum Ratings

Supply Voltage ..... V_{SS}–0.3V to V_{SS}+6.0V
Input Voltage ..... V_{SS}–0.3V to V_{DD}+0.3V
I_{OL} Total 150mA
Total Power Dissipation 500mW
Storage Temperature ..... –60°C to 150°C
Operating Temperature ..... –40°C to 85°C
I_{OH} Total ..... –100mA

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

D.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	f_{SYS}=4MHz	2.2	—	5.5	V
			f_{SYS}=8MHz	3.3	—	5.5	V
			f_{SYS}=12MHz	4.5	—	5.5	V
I_{DD1}	Operating Current (Crystal OSC)	3V	No load, f_{SYS}=4MHz ADC disable	—	0.6	1.5	mA
		5V		—	2	4	mA
I_{DD2}	Operating Current (RC OSC)	3V	No load, f_{SYS}=4MHz ADC disable	—	0.8	1.5	mA
		5V		—	2.5	4	mA
I_{DD3}	Operating Current (Crystal OSC, RC OSC)	5V	No load, f_{SYS}=8MHz ADC disable	—	4	8	mA
I_{DD4}	Operating Current (Crystal OSC, RC OSC)	5V	No load, f_{SYS}=12MHz ADC disable	—	5	10	mA
I_{STB1}	Standby Current (WDT Enabled)	3V	No load, system HALT	—	—	5	μA
		5V		—	—	10	μA
I_{STB2}	Standby Current (WDT Disabled)	3V	No load, system HALT	—	—	1	μA
		5V		—	—	2	μA
V_{IL1}	Input Low Voltage for I/O Ports, TMR and $\overline{INT}$	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports, TMR and $\overline{INT}$	—	—	$0.7V_{DD}$	—	V_{DD}	V
V_{IL2}	Input Low Voltage ($\overline{RES}$)	—	—	0	—	$0.4V_{DD}$	V
V_{IH2}	Input High Voltage ($\overline{RES}$)	—	—	$0.9V_{DD}$	—	V_{DD}	V
V_{LVR}	Low Voltage Reset Voltage	—	LVR enable, 2.1V option	1.98	2.10	2.22	V
		—	LVR enable, 3.15V option	2.98	3.15	3.32	V
		—	LVR enable, 4.2V option	3.98	4.20	4.42	V
I_{OL}	I/O Port Sink Current	3V	V_{OL}=0.1V_{DD}	4	8	—	mA
		5V	V_{OL}=0.1V_{DD}	10	20	—	mA
I_{OH}	I/O Port Source Current	3V	V_{OH}=0.9V_{DD}	–2	–4	—	mA
		5V	V_{OH}=0.9V_{DD}	–5	–10	—	mA
R_{PH}	Pull-high Resistance	3V	—	20	60	100	kΩ
		5V	—	10	30	50	kΩ
V_{AD}	A/D Input Voltage	—	—	0	—	V_{DD}	V
E_{AD}	A/D Conversion Error	—	—	—	±0.5	±1	LSB
I_{ADC}	Additional Power Consumption if A/D Converter is Used	3V	—	—	0.5	1	mA
		5V		—	1.5	3	mA

A.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock	—	2.2V~5.5V	400	—	4000	kHz
			3.3V~5.5V	400	—	8000	kHz
			4.5V~5.5V	400	—	12000	kHz
f_{TIMER}	Timer I/P Frequency (TMR)	—	2.2V~5.5V	0	—	4000	kHz
			3.3V~5.5V	0	—	8000	kHz
			4.5V~5.5V	0	—	12000	kHz
t_{WDTOSC}	Watchdog Oscillator Period	3V	—	45	90	180	μs
		5V	—	32	65	130	μs
t_{RES}	External Reset Low Pulse Width	—	—	1	—	—	μs
t_{SST}	System Start-up Timer Period	—	Wake-up from HALT	—	1024	—	*t_{SYS}
t_{LVR}	Low Voltage Reset Time	—	—	0.25	1	2	ms
t_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
t_{AD1}	A/D Clock Period – HT46F46E	—	—	0.5	—	—	μs
t_{AD2}	A/D Clock Period – HT46F47E/HT46F48E/HT46F49E	—	—	1	—	—	μs
t_{ADC1}	A/D Conversion Time – HT46F46E	—	—	—	64	—	t_{AD1}
t_{ADC2}	A/D Conversion Time – HT46F47E/HT46F48E/HT46F49E	—	—	—	76	—	t_{AD2}
t_{ADCS1}	A/D Sampling Time – HT46F46E	—	—	—	32	—	t_{AD1}
t_{ADCS2}	A/D Sampling Time – HT46F47E/HT46F48E/HT46F49E	—	—	—	32	—	t_{AD2}

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

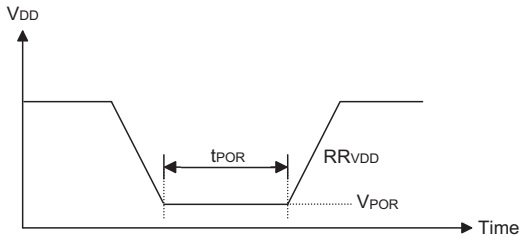
EEPROM A.C. Characteristics

Symbol	Parameter	V_{CC}=5V±10%		V_{CC}=2.2V±10%		Unit
		Min.	Max.	Min.	Max.	
f_{SK}	Clock Frequency	0	2	0	1	MHz
t_{SKH}	SK High Time	250	—	500	—	ns
t_{SKL}	SK Low Time	250	—	500	—	ns
t_{CSS}	CS Setup Time	50	—	100	—	ns
t_{CSH}	CS Hold Time	0	—	0	—	ns
t_{CDS}	CS Deselect Time	250	—	250	—	ns
t_{DIS}	DI Setup Time	100	—	200	—	ns
t_{DIH}	DI Hold Time	100	—	200	—	ns
t_{PD1}	DO Delay to "1"	—	250	—	500	ns
t_{PD0}	DO Delay to "0"	—	250	—	500	ns
t_{SV}	Status Valid Time	—	250	—	250	ns
t_{PR}	Write Cycle Time	—	5	—	5	ms

Power-on Reset Characteristics

Ta=25°C

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



System Architecture

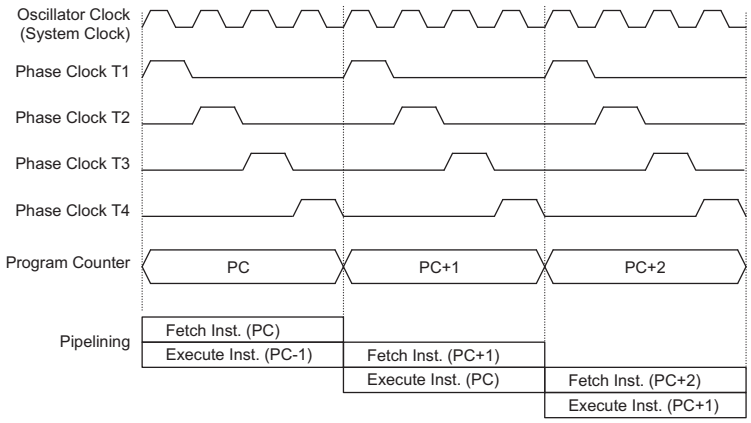
A key factor in the high-performance features of the Holtek range of Cost-Effective A/D Flash Type with EEPROM microcontrollers is attributed to the internal system architecture. The range of devices take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one cycle, with the exception of branch or call instructions. An 8-bit wide ALU is used in practically all operations of the instruction set. It carries out arithmetic operations, logic operations, rotation, increment, decrement, branch decisions, etc. The internal data path is simplified by moving data through the Accumulator and the ALU. Certain internal registers are implemented in the Data Memory and can be directly or indirectly addressed. The simple addressing methods of these registers along with additional architectural features ensure that a minimum of external components is required to provide a functional I/O and A/D control system with maximum reliability and flexibility. This makes these devices suitable for low-cost, high-volume production for controller applications requiring from 1K up to 4K words of Program Memory and 64 to 128 bytes of Data Memory storage.

Clocking and Pipelining

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

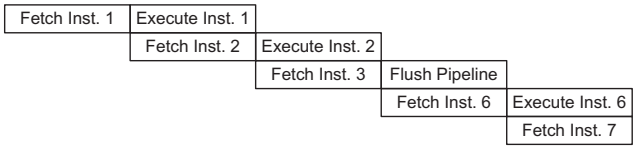
When the RC oscillator is used, OSC2 is freed for use as a T1 phase clock synchronizing pin. This T1 phase clock has a frequency of $f_{SYS}/4$ with a 1:3 high/low duty cycle.

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

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



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. For the Cost-Effective A/D Flash Type with EEPROM series of microcontrollers, 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.

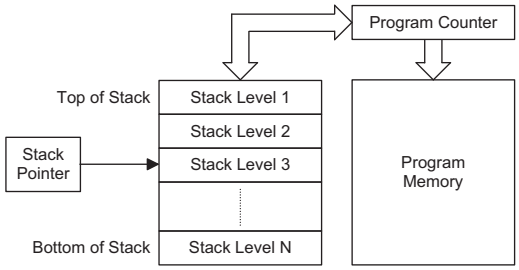
Mode	Program Counter Bits											
	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Initial Reset	0	0	0	0	0	0	0	0	0	0	0	0
External Interrupt	0	0	0	0	0	0	0	0	0	1	0	0
Timer/Event Counter Overflow	0	0	0	0	0	0	0	0	1	0	0	0
A/D Converter Interrupt	0	0	0	0	0	0	0	0	1	1	0	0
Skip	Program Counter + 2											
Loading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

Program Counter

Note: PC11~PC8: Current Program Counter bits
@7~@0: PCL bits
#11~#0: Instruction code address bits
S11~S0: Stack register bits
For the HT46F49E, the Program Counter is 12 bits wide, i.e. from b11~b0.
For the HT46F47E and HT46F48E, the Program Counter is 11 bits wide, i.e. From b10~b0, therefore the b11 column in the table is not applicable.
For the HT46F46E, the Program Counter is 10 bits wide, i.e. from b9~b0, therefore the b11 and b10 the columns in the table are not applicable.

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack can have either 4 or 6 levels depending upon which device is selected 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 writable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack. After a device reset, the Stack Pointer will point to the top of the stack.



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

Note: For the HT46F46E, 4 levels of stack are available and for the HT46F47E,HT46F48E and HT46F49E, 6 levels of stack are available.

Arithmetic and Logic Unit – ALU

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

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

Flash Program Memory

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

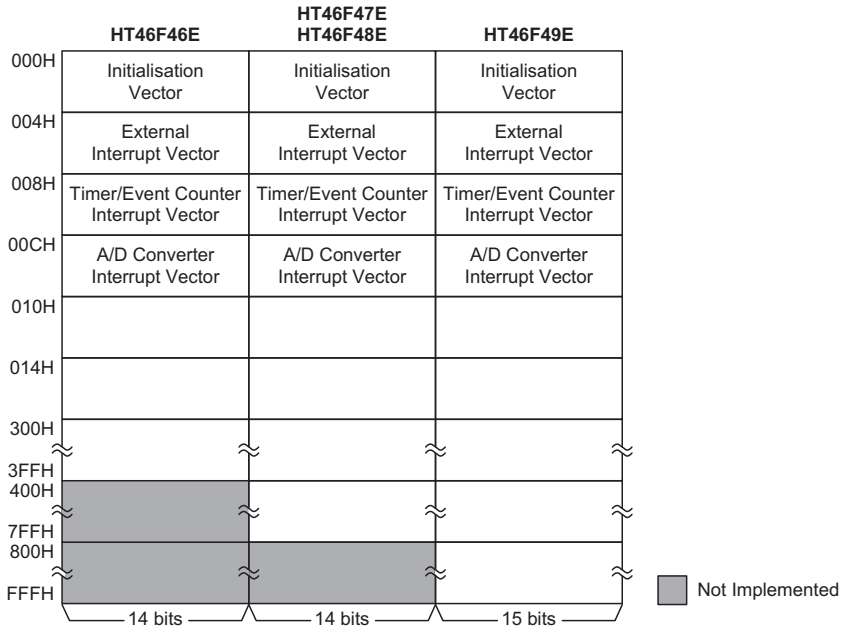
Structure

The Program Memory has a capacity of 1K by 14, 2K by 14 or 4K by 15 bits depending upon which device is selected. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by separate table pointer registers.

Special Vectors

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

- Location 000H
  This vector is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.
- Location 004H
  This vector is used by the 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 internal vector is used by the Timer/Event Counter. If a counter overflow occurs, the program will jump to this location and begin execution if the timer/event counter interrupt is enabled and the stack is not full.
- Location 00CH
  This internal vector is used by the A/D converter. When an A/D conversion cycle is complete, the program will jump to this location and begin execution if the A/D interrupt is enabled and the stack is not full.



Program Memory Structure

Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, the table pointer must first be setup by placing the lower order address of the look up data to be retrieved in the table pointer register, TBLP. This register defines the lower 8-bit address of the look-up table.

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

The following diagram illustrates the addressing/data flow of the look-up table:

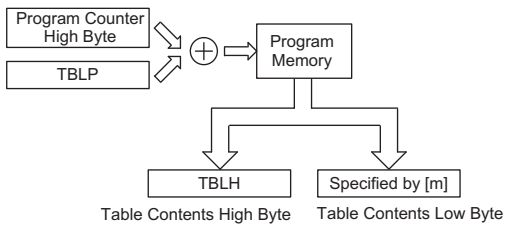


Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the HT46F47E microcontroller. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is ″700H″ which refers to the start address of the last page within the 2K Program Memory of the HT46F47E 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 last page (for HT46F47E)

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.

In Circuit Programming

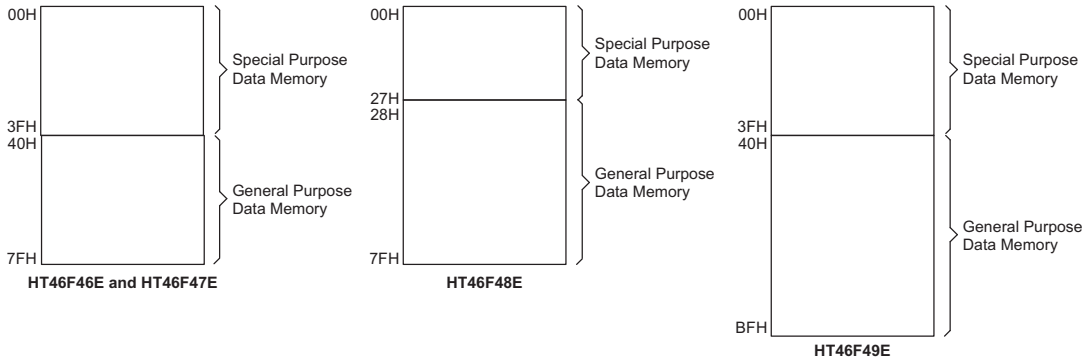
The provision of Flash Program Memory gives the user and designer the convenience of easy upgrades and modifications to their programs on the same device. As an additional convenience, Holtek has provided a means of programming the microcontroller in-circuit. This provides manufacturers with the possibility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

Pin Name	Function
PA0	Serial data input/output
PA4	Serial clock
$\overline{\text{RES}}$	Device reset
VDD	Power supply
VSS	Ground

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

Table Location

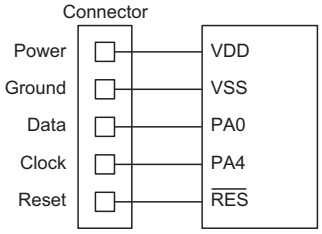
Note: PC11~PC8: Current Program Counter bits
@7~@0: Table Pointer TBLP bits
For the HT46F49E the Table address location is 12 bits, i.e. from b11~b0.
For the HT46F47E and HT46F48E, the Table address location is 11 bits, i.e. from b10~b0.
For the HT46F46E, the Table address location is 10 bits, i.e. from b9~b0.



Data Memory Structure

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

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



In-circuit Programming Interface

RAM Data Memory

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

Structure

The RAM Data Memory is subdivided into two banks, known as Bank 0 and Bank 1, all of which are implemented in 8-bit wide RAM. Most of 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 start address of the Data Memory for all devices is the address "00H". Registers which are common to all microcontrollers, such as ACC, PCL, etc., have the same Data Memory address.

Bank 1 of the RAM Data Memory contains only one special function register, known as the EECR register which is located at address "40H" for all devices. This register is used to access data from the EEPROM Data Memory.

Bank 1 RAM Data Memory Structure

General Purpose Data Memory

All microcontroller programs require an area of read/write memory where temporary data can be stored and retrieved for use later. It is this area of RAM memory that is known as General Purpose Data Memory. This area of Data Memory is fully accessible by the user 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.

Special Purpose Data Memory

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

Special Function Registers

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

Indirect Addressing Register – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 and IAR1 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointer, MP0 or MP1. Acting as a pair, IAR0 and MP0 can together only access data from Bank 0, while the IAR1 and MP1 register pair can access data from both Bank 0 and Bank 1. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointer – MP0, MP1

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

For devices with 64 or 88 bytes of RAM Data Memory, bit 7 of the Memory Pointer is not implemented. However, it must be noted that when the Memory Pointer for these devices is read, bit 7 will be read as high.

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

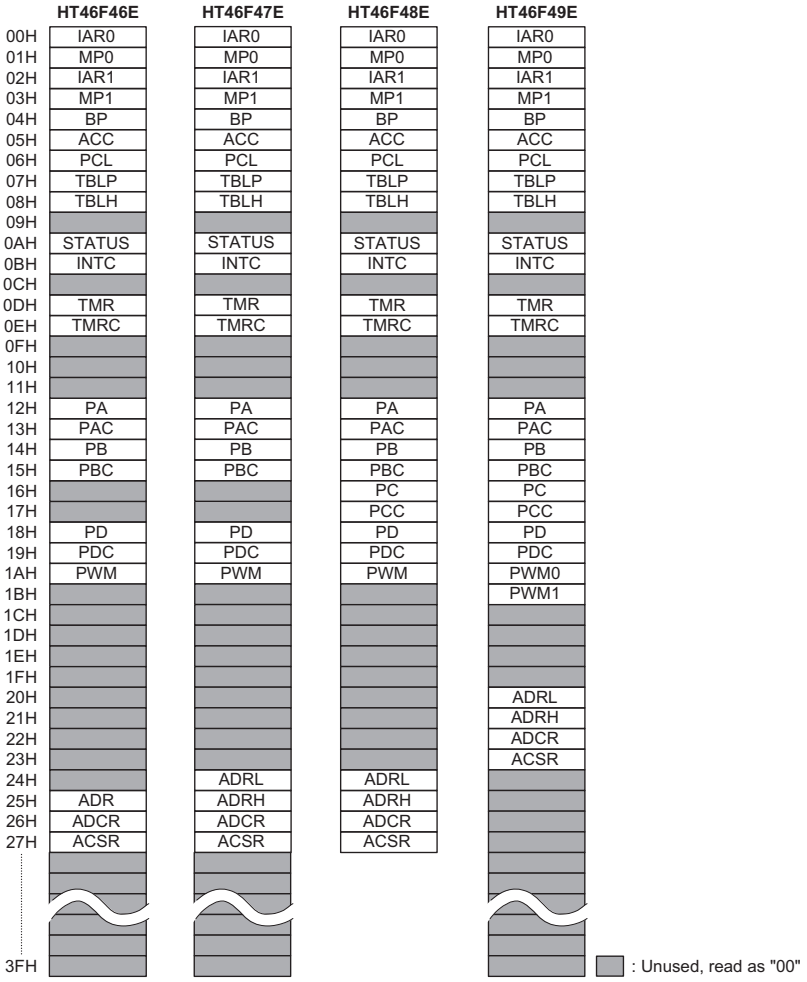
```
data .section  'data'
adres1    db ?
adres2    db ?
adres3    db ?
adres4    db ?
block     db ?
code .section at 0 'code'
org  00h

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

loop:
    clr  IAR0               ; clear the data at address defined by MP0
    inc  mp0                ; increment memory pointer
    sdz  block              ; check if last memory location has been cleared
    jmp  loop

continue:
```

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



Special Purpose Data Memory

Bank Pointer – BP

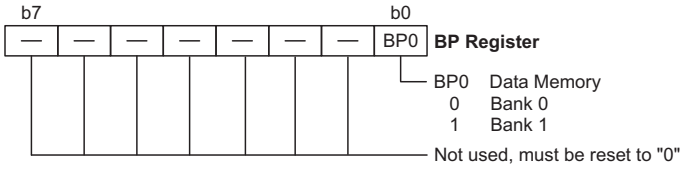
The RAM Data Memory is divided into two Banks, known as Bank 0 and Bank 1. With the exception of the EECR register, all of the Special Purpose Registers and General Purpose Registers are contained in Bank 0. Bank 1 contains only one register, which is the EEPROM Control Register, known as EECR. Selecting the required Data Memory area is achieved using the Bank Pointer. If data in Bank 0 is to be accessed, then the BP register must be loaded with the value ″00″, while if data in Bank 1 is to be accessed, then the BP register must be loaded with the value ″01″.

Using Memory Pointer MP0 and Indirect Addressing Register IAR0 will always access data from Bank 0, irrespective of the value of the Bank Pointer. The EECR register is located at memory location 40H in Bank 1 and can only be accessed indirectly using memory pointer MP1 and the indirect addressing register, IAR1, after the BP register has first been loaded with the value ″01″. Data can only be read from or written to the EEPROM via this register.

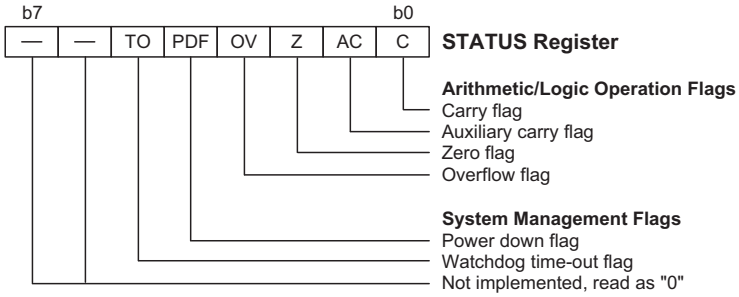
The Data Memory is initialised to Bank 0 after a reset, except for the WDT time-out reset in the Power Down Mode, in which case, the Data Memory bank remains unaffected. It should be noted that Special Function Data Memory is not affected by the bank selection, which means that the Special Function Registers can be accessed from within either Bank 0 or Bank 1. Directly addressing the Data Memory will always result in Bank 0 being accessed irrespective of the value of the Bank Pointer.

Accumulator – ACC

The Accumulator is central to the operation of any microcontroller and is closely related with operations carried out by the ALU. The Accumulator is the place where all intermediate results from the ALU are stored. Without the Accumulator it would be necessary to write the result of each calculation or logical operation such as addition, subtraction, shift, etc., to the Data Memory resulting in higher programming and timing overheads. Data transfer operations usually involve the temporary storage function of the Accumulator; for example, when



Bank Pointer



Status Register

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.

Status Register – STATUS

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

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

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

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

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

Interrupt Control Register – INTC

This 8-bit register, known as the INTC register, controls the operation of both external and internal timer interrupts. By setting various bits within this register using standard bit manipulation instructions, the enable/disable function of each interrupt 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.

Timer/Event Counter Registers – TMR, TMRC

All devices possess a single internal 8-bit count-up timer. An associated register known as TMR is the location where the timer's 8-bit value is located. This register can also be preloaded with fixed data to allow different time intervals to be setup. An associated control register, known as TMRC, contains the setup information for this timer, which determines in what mode the timer is to be used as well as containing the timer on/off control function.

Input/Output Ports and Control Registers

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

Pulse Width Modulator Registers – PWM, PWM0, PWM1

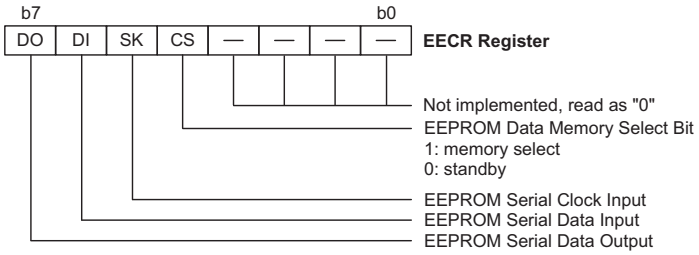
Each device in the Cost-Effective A/D Flash Type with EEPROM microcontroller range contains either one or two Pulse Width Modulators. Each one has its own related independent control register. For devices with a single PWM function this is register is known as PWM, while for devices with two PWM functions, their control register names are PWM0 and PWM1. The 8-bit contents of these registers, defines the duty cycle value for the modulation cycle of the corresponding Pulse Width Modulator.

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

Each device in the Cost-Effective A/D Flash Type with EEPROM microcontroller range contains a 4-channel 8-bit or 9-bit A/D converter. The correct operation of the A/D requires the use of one or two data registers, a control register and a clock source register. For the HT46F46E device, which has an 8-bit A/D converter, there is a single data register, known as ADR. For the other devices, which contain a 9-bit A/D converter, there are two data registers, a high byte data register known as ADRH, and a low byte data register known as ADRL. These are the register locations where the digital value is placed after the completion of an analog to digital conversion cycle. The channel selection and configuration of the A/D converter is setup via the control register ADCR while the A/D clock frequency is defined by the clock source register, ACSR.

EEPROM Control Register – EECR

One special features of this device is that it contains an area of internal EEPROM Data Memory. EEPROM, which stands for Electrically Erasable Programmable



EEPROM Control Register

Read Only Memory, is by its nature a non-volatile form of memory, with data retention even when its power supply is removed. By incorporating this kind of data memory in the device a whole new host of application possibilities are made available to the designer. The availability of EEPROM storage allows information such as product identification numbers, calibration values, specific user data, system setup data or other product information to be stored directly within the product microcontroller.

EEPROM Data Memory

Dependent upon which device is chosen, the EEPROM Data Memory capacity is either 128×8 bits or 256×8 bits. Unlike the Program Memory and RAM Data Memory, the EEPROM Data Memory is not directly mapped and is therefore not directly accessible in same way as the other types of memory. Instead it has to be accessed indirectly through the EEPROM Control Register.

Device	EEPROM Memory Capacity
Except HT46F49E	128×8
HT46F49E	256×8

EEPROM Data Memory Capacity

Accessing the EEPROM Data Memory

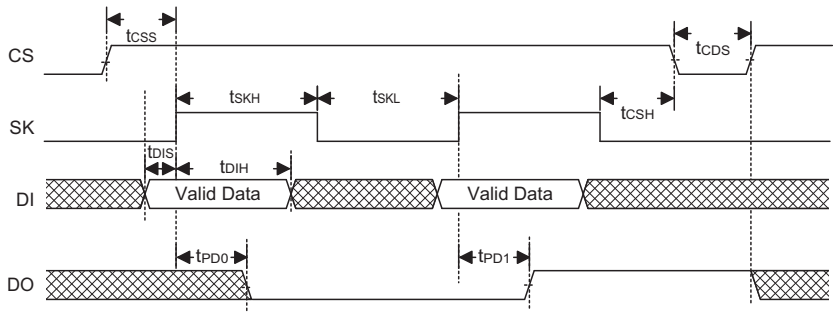
The EEPROM Data Memory is accessed using a set of seven instructions. These instructions control all functions of the EEPROM such as read, write, erase, enable etc. The internal EEPROM structure is similar to that of a standard 3-wire EEPROM, for which four pins are used for transfer of instruction, address and data information. These are the Chip Select pin, CS, Serial Clock pin, SK, Data In pin, DI and the Data Out pin, DO. All actions related to the EEPROM must be conducted through the EECR register which is located in Bank 1 of the RAM Data Memory, in which each of these four EEPROM pins is represented by a bit in the EECR register. By manipulating these four bits in the EECR register, in accordance with the accompanying timing diagrams, the microcontroller can communicate with the EEPROM and carry out the required functions, such as reading and writing data.

Bit No.	Label	EEPROM Function
0~3	—	Not implemented bit, read as ″0″
4	CS	EEPROM Data Memory select
5	SK	Serial Clock: Used to clock data into and out of the EEPROM
6	DI	Data Input: Instructions, address and data information are written to the EEPROM on this pin
7	DO	Data Output: Data from the EEPROM is readout with this bit. Will be in a high-impedance condition if no data is being read.

EECR Register EEPROM Control Bit Functions

When reading data from the EEPROM, the data will clocked out on the rising edge of SK and appear on DO. The DO pin will normally be in a high-impedance condition unless a READ statement is being executed. When writing to the EEPROM the data must be presented first on DI and then clocked in on the rising edge of SK. After all the instruction, address and data information has been transmitted, CS should be cleared to ″0″ to terminate the instruction transmission. Note that after power on the EEPROM must be initialised as described.

As indirect addressing is the only way to access the EECR register, all read and write operations to this register must take place using the Indirect Addressing Register, IAR1, and the Memory Pointer, MP1. Because the EECR control register is located in Bank 1 of the RAM Data Memory at location 40H, the MP1 Memory Pointer must first be set to the value 40H and the Bank Pointer set to ″1″.



Clocking Data In and Out of the EEPROM

EEPROM Data Memory Instruction Set

Control over the internal EEPROM, to execute functions such as read, write, disable, enable etc, is implemented through instructions of which there are a total of seven. The related instruction is transmitted to the EEPROM via the DI bit, after CS has first been set to "1" to enable the EEPROM and a start bit "1" has been transmitted.

For the READ, WRITE and ERASE instructions, each of the three instructions has its own two bit related instruction code. The address should then be transmitted, which in the case of devices with a 128×8 capacity EEPROM is 7-bits. For devices with a 256×8 capacity EEPROM, a 9-bit address is transmitted, however the first bit is a dummy bit and can have any value. The address is transmitted in MSB first format.

For the other four instructions, "EWEN", "EWDS", "ERAL" and "WRAL", after the start bit has been transmitted a "00" instruction code should then follow. The address information should then follow which is 7-bits long for devices with a 128×8 capacity EEPROM, and 9-bits long for devices with a 256×8 capacity EEPROM. The first two bits of this address is instruction dependant as shown in the table while the remaining bits have don't care values and can be either high or low.

After any write or erase instruction is issued, the internal write function of the EEPROM will be used to write the data into the device. As this internal write operation uses the EEPROM's own internal clock, no further instructions will be accepted by the EEPROM until the internal write function has ended. After power on and before any instruction is issued the EEPROM must be properly initialised to ensure proper operation.

Instruction	Function	Start Bit	Instruction Code	Address	Data
READ	Read Out Data Byte(s)	1	10	A6~A0	D7~D0
ERASE	Erase Single Data Byte	1	11	A6~A0	—
WRITE	Write Single Data Byte	1	01	A6~A0	D7~D0
EWEN	Erase/Write Enable	1	00	11 XXXXX	—
EWDS	Erase/Write Disable	1	00	00 XXXXX	—
ERAL	Erase All	1	00	10 XXXXX	—
WRAL	Write All	1	00	01 XXXXX	D7~D0

EEPROM Instruction Set Summary – Except HT46F49E

Instruction	Function	Start Bit	Instruction Code	Address	Data
READ	Read Out Data Byte(s)	1	10	X, A7~A0	D7~D0
ERASE	Erase Single Data Byte	1	11	X, A7~A0	—
WRITE	Write Single Data Byte	1	01	X, A7~A0	D7~D0
EWEN	Erase/Write Enable	1	00	11 XXXXXXX	—
EWDS	Erase/Write Disable	1	00	00 XXXXXXX	—
ERAL	Erase All	1	00	10 XXXXXXX	—
WRAL	Write All	1	00	01 XXXXXXX	D7~D0

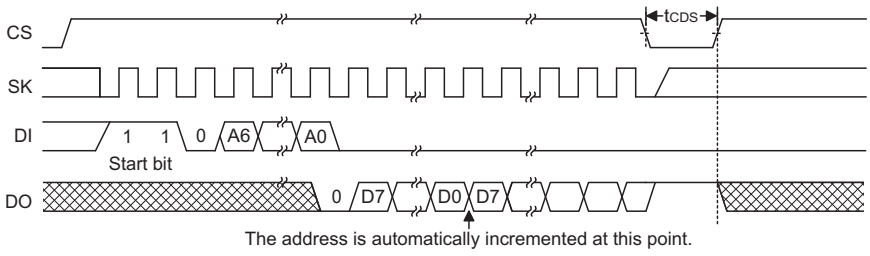
EEPROM Instruction Set Summary – HT46F49E

READ

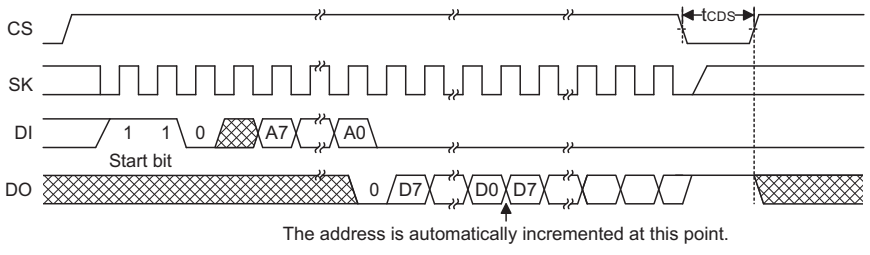
The ″READ″ instruction is used to read out one or more bytes of data from the EEPROM Data Memory. To instigate a ″READ″ instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″10″, all transmitted via the DI bit. The address information should then follow with the MSB being transmitted first. For the HT46F49E device, a dummy bit must be inserted between the last bit of the instruction code and the MSB of the address. After the last address bit, A0, has been transmitted, the data can be clocked out, bit D7 first, on the rising edge of the SK clock signal and can be read via the DO bit. However, a dummy ″0″ bit 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 CS bit remains high, data bit D7 of the next address will automatically follow data bit D0 of the previous address with no dummy ″0″ being inserted between them. The address will keep incrementing in this way until CS returns to a low value. DO will normally be in a high impedance condition until the ″READ″ instruction is executed. Note that as the ″READ″ instruction is not affected by the condition of the ″EWEN″ or ″EWDS″ instruction, the READ command is always valid and independent of these two instructions.

WRITE

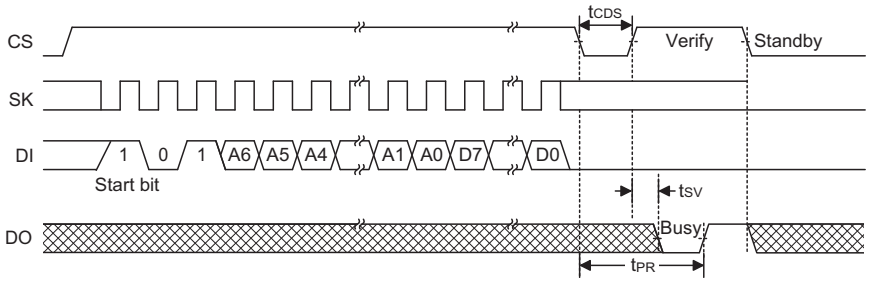
The ″WRITE″ instruction is used to write a single byte of data into the EEPROM. To instigate a WRITE instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″01″, all transmitted via the DI bit. The address information should then follow with the MSB bit being transmitted first. After the last address bit, A0, has been transmitted, the data can be immediately transmitted MSB first. For the HT46F49E device, a dummy bit must be inserted between the last bit of the instruction code and the MSB of the address. After all the WRITE instruction code, address and data have been transmitted, the data will be written into the EEPROM when the CS bit is cleared to zero. The EEPROM does this by executing an internal write-cycle, which will first erase and then write the previously transmitted data byte into the EEPROM. This process takes place internally using the EEPROM′s own internal clock and does not require any action from the SK clock. No further instructions can be accepted by the EEPROM until this internal write-cycle has finished. To determine when the write cycle has ended, CS should be again brought high and the DO bit polled. If DO is low this indicates that the internal write-cycle is still in progress, however, the DO bit will change to a high value when the internal write-cycle has ended. Before a ″WRITE″ instruction is transmitted an ″EWEN″ instruction must have been transmitted at some point earlier to ensure that the erase/write function of the EEPROM is enabled.



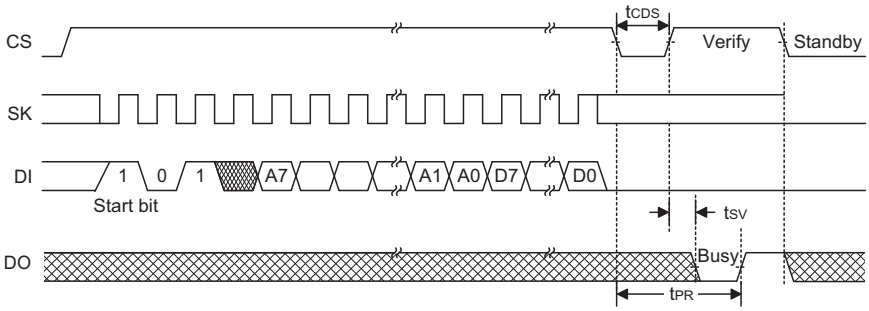
READ Timing – Except HT46F49E



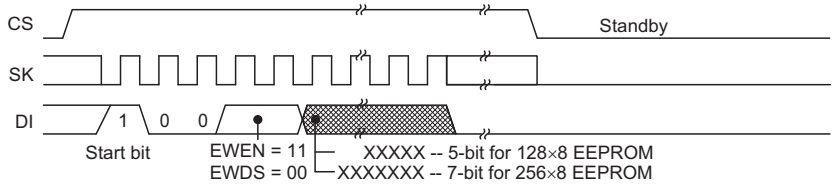
READ Timing – HT46F49E



WRITE Timing – Except HT46F49E



WRITE Timing – HT46F49E



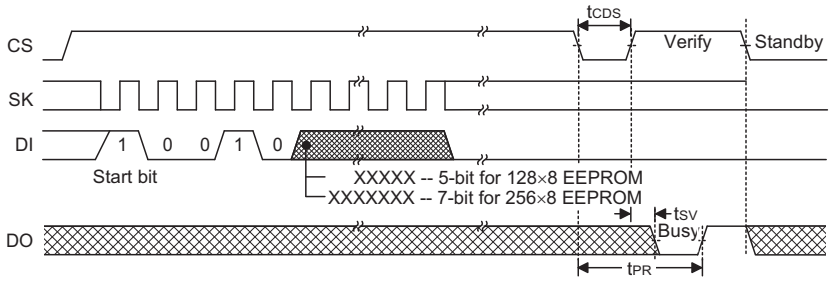
EWEN/EWDS Timing

EWEN/EWDS

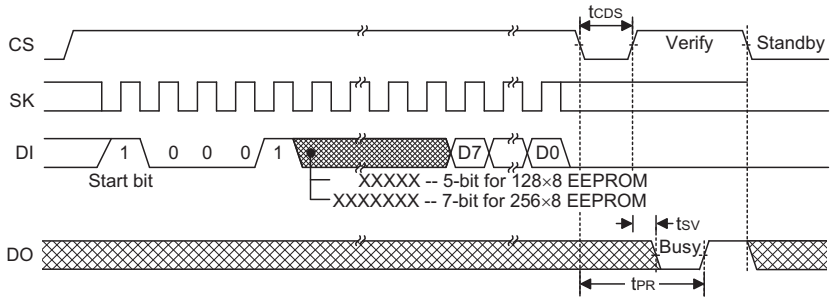
The ″EWEN″ instruction is the Erase/Write Enable instruction and the ″EWDS″ instruction is the Erase/Write Disable instruction. To instigate an ″EWEN″ or ″EWDS″ instruction, the CS bit should first be set high, followed by a high start bit and then the instruction code ″00″. For the ″EWEN″ instruction, a ″11″ should then be transmitted and for the ″EWDS″ instruction a ″00″ should be transmitted. Following on from this, and depending on whether the internal EEPROM has a 128×8 or 256×8 capacity, either 5-bits or 7-bits respectively, of ″don′t-care″ data should then be transmitted to complete the instruction. If the device is already in the Erase Write Disable mode then no write or erase operations can be executed thus protecting the internal EEPROM data. Before any write or erase instruction is executed an ″EWEN″ instruction must be issued. After the ″EWEN″ instruction is executed, the device will remain in the Erase Write Enable mode until a subsequent ″EWDS″ instruction is issued or until the device is powered down.

ERAL

The ″ERAL″ instruction is used to erase the whole contents of the EEPROM memory. After it has been executed all the data in the EEPROM will be set to ″1″. To instigate this instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″00″. Following on from this, a ″10″ should then be transmitted, and depending on whether the internal EEPROM has a 128×8 or 256×8 capacity, this should be followed by either 5-bits or 7-bits respectively, of ″don′t-care″ data to complete the instruction. After the ″ERAL″ instruction code has been transmitted, the EEPROM data will be erased when the CS bit is cleared to zero. The EEPROM does this by executing an internal write-cycle. This process takes place internally using the EEPROM′s own internal clock and does not require any action from the SK clock. No further instructions can be accepted by the EEPROM until this internal write-cycle has finished. To determine when the write cycle has ended, CS should be again brought high and the DO bit polled. If D0 is low this indicates that the internal write-cycle is still in progress, however the D0 bit will change to a high value when the internal write-cycle has



ERAL Timing



WRAL Timing

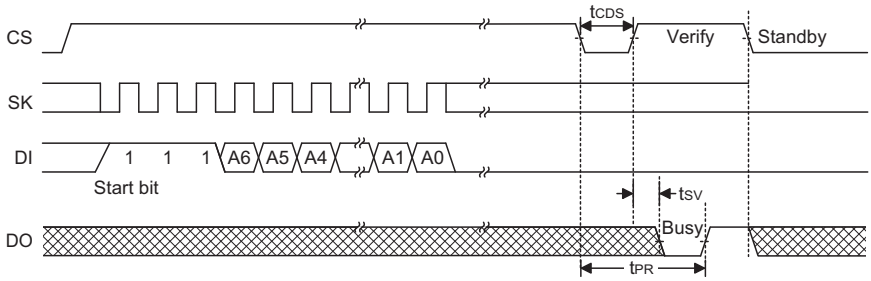
ended. Before an ″ERAL″ instruction is transmitted an ″EWEN″ instruction must have been transmitted at some point earlier to ensure that the erase/write function of the EEPROM is enabled.

WRAL

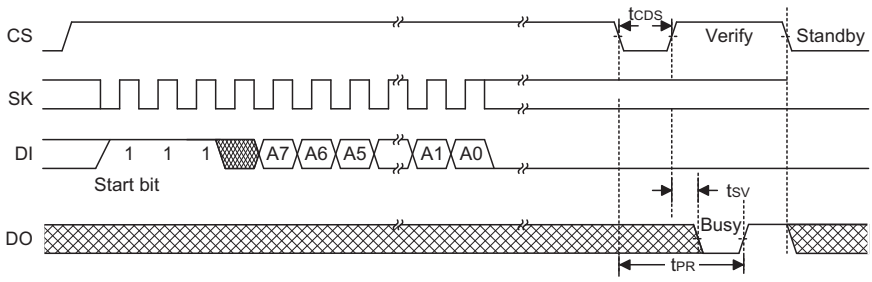
The WRAL instruction is used to write the same data into the entire EEPROM. To instigate this instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″00″. Following on from this, a ″01″ should then be transmitted, and depending on whether the internal EEPROM has a 128×8 or 256×8 capacity, this should be followed by either 5-bits or 7-bits respectively, of ″don′t-care″ data. The data information should then follow with the MSB bit being transmitted first. After the instruction code and data have been transmitted, the data will be written into the EEPROM when the CS bit is cleared to zero. The EEPROM does this by executing an internal write-cycle. This process takes place internally using the EEPROM′s own internal clock and does not require any action from the SK clock. No further instructions can be accepted by the EEPROM until this internal write-cycle has finished. To determine when the write cycle has ended, CS should be again brought high and the DO bit polled. If D0 is low this indicates that the internal write-cycle is still in progress, however the D0 bit will change to a high value when the internal write-cycle has ended. Before a ″WRAL″ instruction is transmitted an ″EWEN″ instruction must have been transmitted at some point earlier to ensure that the erase/write function of the EEPROM is enabled. The WRAL instruction will automatically erase any previously written data making it unnecessary to first issue an erase instruction.

ERASE

The ″ERASE″ instruction is used to erase data at a specified addresses. The data at the address specified will be set to ″1″. To instigate an ″ERASE″ instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″11″, all transmitted via the DI bit. The address information should then follow with the MSB bit being transmitted first. For the HT46F49E device, a dummy bit must be inserted between the last bit of the instruction code and the MSB of the address. After all the ″ERASE″ instruction code and address have been transmitted, the data at the specified address will be erased when the CS bit is cleared to zero. The EEPROM does this by executing an internal write cycle which will set all data at the specified address to ″1″. This process takes place internally using the EEPROM′s own internal clock and does not require any action from the SK clock. No further instructions can be accepted by the EEPROM until the write cycle has finished. To determine when the write cycle has ended, the CS should be again brought high and the DO bit polled. If the DO bit is low this indicates that the write-cycle is still in progress, however, the DO bit will change to a high value when the write-cycle has ended. Before an ″ERASE″ instruction is transmitted, an ″EWEN″ instruction must have been transmitted at some point earlier to ensure that the erase/write function of the EEPROM is enabled.



ERASE Timing – Except HT46F49E

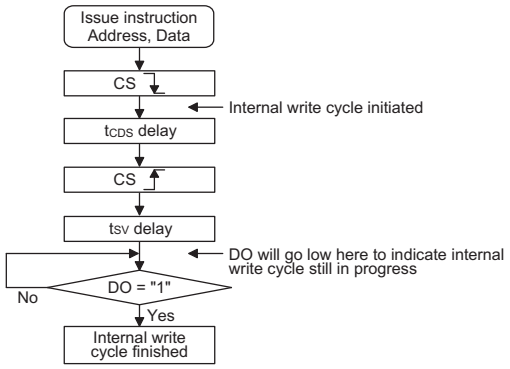


ERASE Timing – HT46F49E

Internal Write Cycle

The write or erase instructions, "WRITE", "ERASE", "ERAL" or "WRAL" will all use the EEPROM's internal write cycle function. As this function is completely internally timed, the SK clock is not required. As the MCU has no control over the timing of this write cycle, it must still have some way of knowing when the internal write cycle has completed. This is because, when the internal write cycle is executing, the EEPROM will not accept any further instructions from the MCU. The MCU must therefore wait until the write cycle has finished before sending any further instructions.

One way for the MCU to know when the write cycle has terminated is to poll the DO bit after the CS bit has issued a low pulse. The low going edge of this CS bit pulse will initiate the internal write cycle, when the bit is returned high the DO bit will go low to indicated that the write cycle is in progress. When the DO bit returns high this indicates that the internal write cycle has ended and that the EEPROM is ready to receive further instructions.



Internal Write Cycle Busy Polling

Initialising the EEPROM

After the MCU is powered on and if the EEPROM is to be used, it must be initialised in a specific way before any user instructions are transmitted. This is achieved by first transmitting an EWEN instruction, then by issuing a WRITE instruction to write random data to any single address in the EEPROM. The initialisation procedure can then be terminated by issuing an EWDS instruction, however at this point, if actual user data is to be imminently written to the EEPROM, this last step is optional.

The following program example shows how to initialise the EEPROM after power-on:

```
mov      A,01h
mov      BP,A          ; set to bank 1
mov      A,40h
mov      MP1,A         ; set MP1 to EECR address
Call     EWEN          ; subroutine to run EWEN instructions
mov      A, 7Fh
mov      EEADDR, A
mov      A, 55h
mov      EEDATA, A
call     WRITE         ; subroutine to run WRITE instruction
                       ; write 55h data to address 7Fh
call     EWDS          ; optional subroutine to run EWDS instruction
```

EEPROM Program Examples

The following short programs gives examples of how to send instructions, read and write to the EEPROM. These programs can form a basis of understanding as to how the internal EEPROM memory is to be used to store and retrieve data. The programs are for use with the HT46F49E device, which has the same capacity internal EEPROM memory of 256×8 bits. For the other devices, which have a smaller 128×8 bit EEPROM memory capacity, the dummy bit which is inserted between the instruction code transmission and the address MSB, is not transmitted.

Example 1 – Definitions and Sending Instructions to the EEPROM

```
_CS EQU  IAR1.4               ; EEPROM lines setup to have a corresponding
_SK EQU  IAR1.5               ; Bit in the Indirect Addressing Register IAR1
_DI EQU  IAR1.6               ; EEPROM can only be indirectly addressed using MP1
_DO EQU  IAR1.7
_EECR    EQU   40H            ; Setup address of the EEPROM control register
C_Addr_Length  EQU 8          ; Address length – 8-bits for this device
C_Data_Length  EQU 8          ; Data length – always 8-bits
;
DATA   .SECTION at 70h 'DATA'
EE_command DB  ?              ; Stores the read or write instruction information
ADDR       DB  ?              ; Store write data or read data address
WR_Data    DB  ?              ; Store read or write data
COUNT      DB  ?              ; Temporary counter
;
WriteCommand:                 ; Write instruction code subroutine
    MOV  A,3                  ; Read, write and erase instructions are 3 bits long
    MOV  COUNT,A
WriteCommand_0:
    CLR  _DI                  ; Prepare the transmitted bit
    SZ   EE_command.7         ; Check value of highest instruction code bit
    SET  _DI
    SET  _SK
    CLR  _SK
    CLR  C
    RLC  EE_command           ; Get next bit of instruction code
    SDZ  COUNT                ; Check if last bit has been transmitted
    JMP  WriteCommand_0
    CLR  _DI
    RET
```

Example 2 – Transmitting an Address to the EEPROM

```
WriteAddr:                      ; Write address subroutine
    MOV  A,C_Addr_Length        ; Setup address length – 8 bits for HT46F49E device
    MOV  COUNT,A
    SET  _SK                    ; Dummy bit transmission for HT46F49E only
    CLR  _SK                    ; Not required for other devices
WriteAddr_0:
    CLR  _DI
    SZ   ADDR.7                 ; Check value of address MSB
    SET  _DI
    CLR  C
    RLC  ADDR                   ; Get next address bit
    SET  _SK
    CLR  _SK
    SDZ  COUNT                  ; Check if address LSB has been written
    JMP  WriteAddr_0
    CLR  _DI
    RET
```

Example 3 – Writing Data to the EEPROM

```
WriteData:
    MOV  A,C_Data_Length        ; Setup data length
    MOV  COUNT,A
WriteData_0:
    CLR  _DI
    SZ   WR_Data.7              ; Check value of data MSB
    SET  _DI
    CLR  C
    RLC  WR_Data                ; Get next address bit
    SET  _SK
    CLR  _SK
    SDZ  COUNT                  ; Check if data LSB has been written
    JMP  WriteData_0
    CLR  _CS                    ; CS low edge initiates internal write cycle
    SET  _CS                    ; CS high edge allows DO to be used to indicate
                                ; end of write cycle
    SNZ  _DO                    ; Poll for DO high to indicate end of write cycle
    JMP  $-1
    RET
```

Example 4 – Reading Data from the EEPROM

```
ReadData:
    MOV  A,C_Data_Length        ; Setup data length
    MOV  COUNT,A
    CLR  WR_Data
ReadData_0:
    CLR  C
    RLC  WR_Data
    SET  _SK
    SZ   _DO                    ; check value of data MSB
    SET  WR_Data.0
    CLR  _SK
    SDZ  COUNT                  ; check if LSB has been received
    JMP  ReadData_0
    MOV  A,WR_Data
    RET
```

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 13 to 23 bidirectional input/output lines labeled with port names PA, PB, PC and PD. These I/O ports are mapped to the RAM Data Memory with specific addresses as shown in the Special Purpose Data Memory table. All of these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction "MOV A,[m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Pull-high Resistors

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

Port A Wake-up

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, PBC, PCC and PDC, 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.

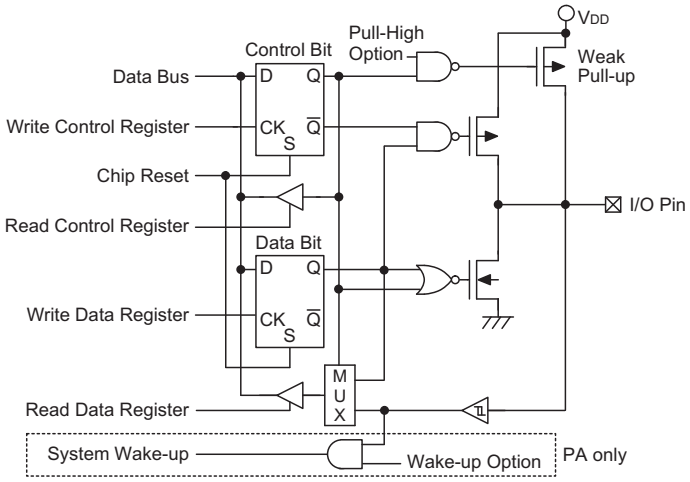
Pin-shared Functions

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.

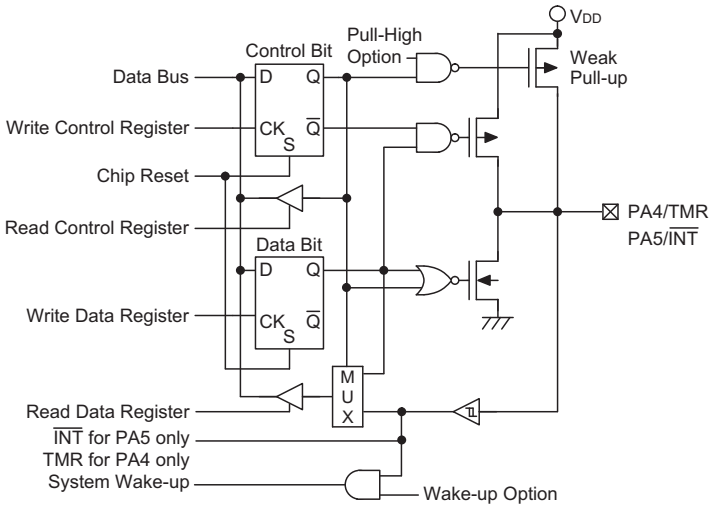
- External Interrupt Input
  The external interrupt pin $\overline{INT}$ is pin-shared with the I/O pin PA5. For applications not requiring an external interrupt input, the pin-shared external interrupt pin can be used as a normal I/O pin, however to do this, the external interrupt enable bits in the INTC register must be disabled.
- External Timer Clock Input
  The external timer pin TMR is pin-shared with the I/O pin PA4. To configure it to operate as a timer input, the corresponding control bits in the timer control register must be correctly set. For applications that do not require an external timer input, the pin can be used as a normal I/O pin. Note that if used as a normal I/O pin the timer mode control bits in the timer control register must select the timer mode, which has an internal clock source, to prevent the input pin from interfering with the timer operation.
- PFD Output
  Each device contains a PFD function whose single output is pin-shared with PA3. The output function of this pin is chosen via a configuration option and remains fixed after the device is programmed. Note that the corresponding bit of the port control register, PAC.3, must setup the pin as an output to enable the PFD output. If the PAC port control register has setup the pin as an input, then the pin will function as a normal logic input with the usual pull-high option, even if the PFD configuration option has been selected.

- PWM Outputs
  All devices contain one or two PWM outputs pin shared with pins PD0 and PD1. The PWM output functions are chosen via configuration options and remain fixed after the device is programmed. Note that the corresponding bit or bits of the port control register, PDC, must setup the pin as an output to enable the PWM output. If the PDC port control register has setup the pin as an input, then the pin will function as a normal logic input with the usual pull-high option, even if the PWM configuration option has been selected.

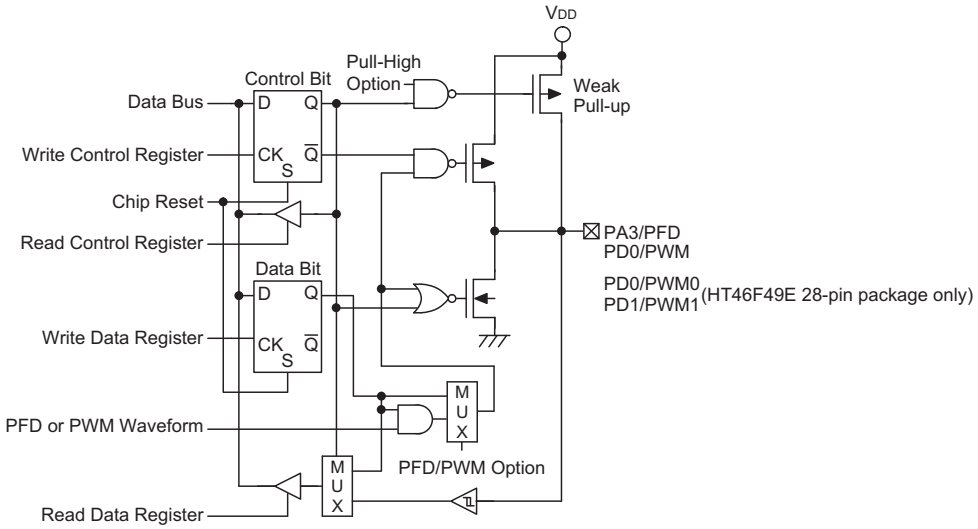
- A/D Inputs
  Each device has four A/D converter inputs. All of these analog inputs are pin-shared with I/O pins on Port B. If these pins are to be used as A/D inputs and not as normal I/O pins then the corresponding bits in the A/D Converter Control Register, ADCR, must be properly set. There are no configuration options associated with the A/D function. If used as I/O pins, then full pull-high resistor configuration options remain, however if used as A/D inputs then any pull-high resistor options associated with these pins will be automatically disconnected.



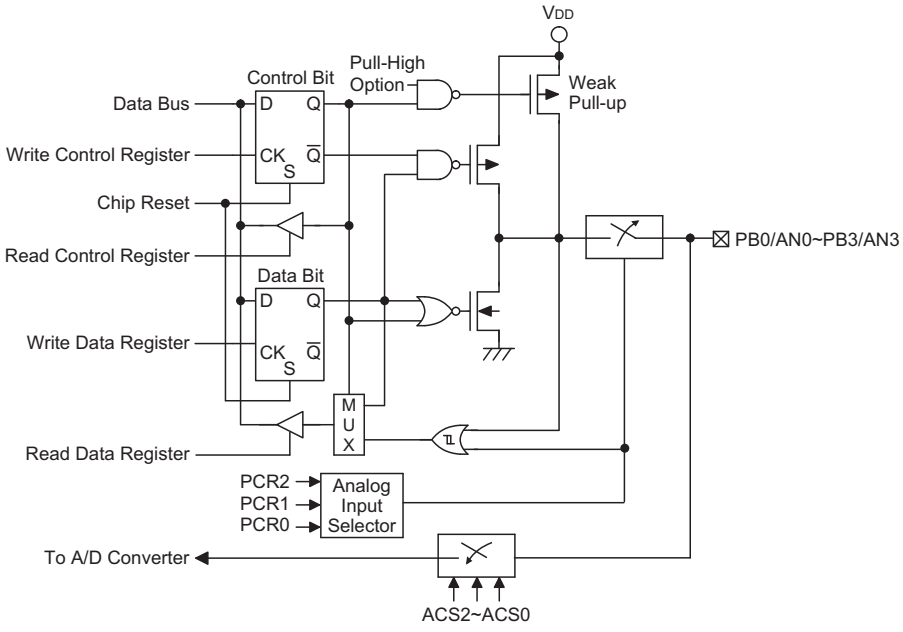
Non-pin-shared Function Input/Output Ports



PA4/PA5 Input/Output Ports



PA3/PFD and PD/PWM Input/Output Ports



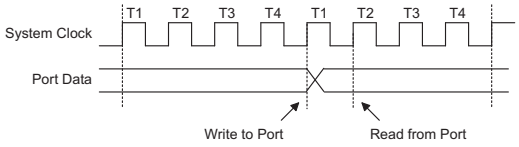
PB Input/Output Ports

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.

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, PCC and PDC, 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, PC and PD, 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.

Note that some devices have different package types which may result in some I/O pins not being bonded out. If these pins are setup as inputs they may oscillate and increase power consumption, especially notable if the device is in the Power Down Mode. It is therefore recommended that any unbonded pins should be setup as outputs, or if setup as inputs, then they should be connected to pull-high resistors.

Timer/Event Counters

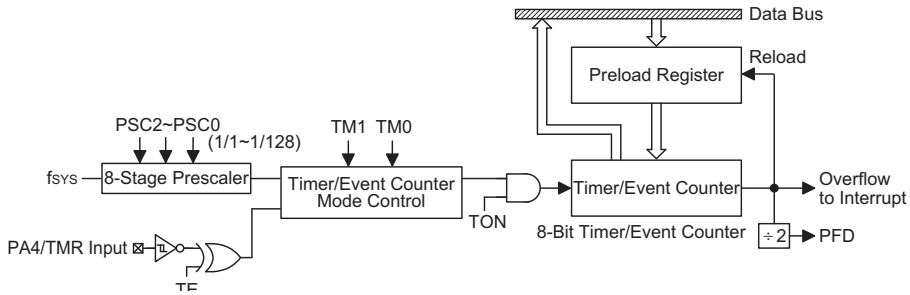
The provision of timers form an important part of any microcontroller giving the designer a means of carrying out time related functions. Each device contains an internal 8-bit count-up timer. With three operating modes, the timers can be configured to operate as a general timer, external event counter or as a pulse width measurement device. The provision of an internal 8-stage prescaler to the timer clock circuitry gives added range to the timer.

There are two registers related to the Timer/Event Counter, TMR and TMRC. The TMR register is the register that contains the actual timing value. Writing to TMR places an initial starting value in the Timer/Event Counter preload register while reading TMR retrieves the contents of the Timer/Event Counter. The TMRC register is a Timer/Event Counter control register, which defines the timer options, and determines how the timer is to be used. The timer clock source can be configured to come from the internal system clock source or from an external clock on shared pin PA4/TMR.

Configuring the Timer/Event Counter Input Clock Source

The internal timer's clock source can originate from either the system clock or from an external clock source. The system clock input timer source is used when the timer is in the timer mode or in the pulse width measurement mode. The internal timer clock also passes through a prescaler, the value of which is conditioned by the bits PSC0, PSC1 and PSC2.

An external clock source is used when the timer is in the event counting mode, the clock source being provided on pin-shared pin PA4/TMR. Depending upon the condition of the TE bit, each high to low, or low to high transition on the PA4/TMR pin will increment the counter by one.



8-bit Timer/Event Counter Structure

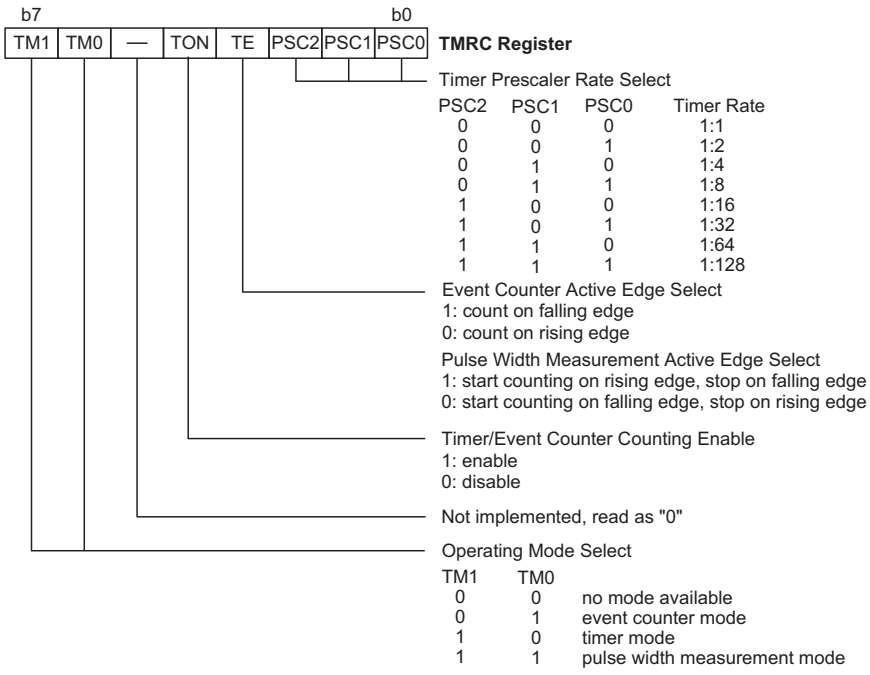
Timer Register – TMR

The TMR register is an 8-bit special function register location within the special purpose Data Memory where the actual timer value is stored. The value in the timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the PA4/TMR pin. The timer will count from the initial value loaded by the preload register to the full count value of FFH at which point the timer overflows and an internal interrupt signal generated. The timer value will then be reset with the initial preload register value and continue counting. For a maximum full range count of 00H to FFH the preload register must first be cleared to 00H. It should be noted that after power-on the preload register will be in an unknown condition. Note that if the Timer/Event Counter is not running and data is written to its preload register, this data will be immediately written into the actual counter. However, if the counter is enabled and counting, any new data written into the preload register during this period will remain in the preload register and will only be written into the actual counter the next time an overflow occurs.

Timer Control Register – TMRC

The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of the Timer Control Register TMRC. Together with the TMR register, these two registers control the full operation of the Timer/Event Counters. Before the timer can be used, it is essential that the TMRC register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialisation.

To choose which of the three modes the timer is to operate in, the timer mode, the event counting mode or the pulse width measurement mode, bits TM0 and TM1 must be set to the required logic levels. The timer-on bit TON or bit 4 of the TMRC register provides the basic on/off control of the timer, setting the bit high allows the counter to run, clearing the bit stops the counter. Bits 0~2 of the TMRC register determine the division ratio of the input clock prescaler. The prescaler bit settings have no effect if an external clock source is used. If the timer is in the event count or pulse width measurement mode the active transition edge level type is selected by the logic level of the TE or bit 3 of the TMRC register.



Timer/Event Counter Control Register

Configuring the Timer Mode

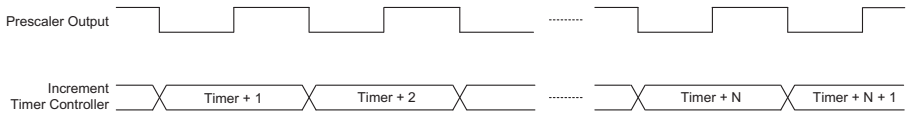
In this mode, the timer can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the counter overflows. To operate in this mode, bits TM1 and TM0 of the TMRC register must be set to 1 and 0 respectively. In this mode, the internal clock is used as the timer clock. The input clock frequency to the timer is f_{SYS} divided by the value programmed into the timer prescaler, the value of which is determined by bits PSC0~PSC2 of the TMRC register. The timer-on bit, TON must be set high to enable the timer to run. Each time an internal clock high to low transition occurs, the timer increments by one. When the timer is full and overflows, the timer will be reset to the value already loaded into the preload register and continue counting. If the timer interrupt is enabled, an interrupt signal will also be generated. The timer interrupt can be disabled by ensuring that the ETI bit in the INTC register is cleared to zero. It should be noted that a timer overflow is one of the wake-up sources.

Configuring the Event Counter Mode

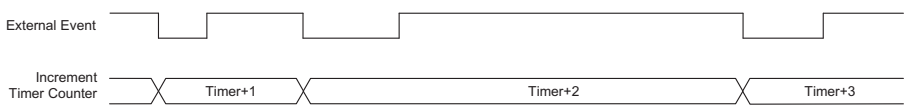
In this mode, a number of externally changing logic events, occurring on external pin PA4/TMR, can be recorded by the internal timer. For the timer to operate in the event counting mode, bits TM1 and TM0 of the TMRC register must be set to 0 and 1 respectively. The timer-on bit, TON must be set high to enable the timer to count. With TE low, the counter will increment each time the PA4/TMR pin receives a low to high transition. If the TE bit is high, the counter will increment each time TMR receives a high to low transition. As in the case of the other two modes, when the counter is full and overflows, the timer will be reset to the value already loaded into the preload register and continue counting. If the timer interrupt is enabled, an interrupt signal will also be generated. The timer interrupt can be disabled by ensuring that the ETI bit in the INTC register is cleared to zero. To ensure that the external pin PA4/TMR is configured to operate as an event counter input pin, two things have to happen. The first is to ensure that the TM0 and TM1 bits place the timer/event counter in the event counting mode, the second is to ensure that the port control register configures the pin as an input. It should be noted that a timer overflow is one of the wake-up sources. Also in the Event Counting mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input pin, even if the microcontroller is in the Power Down Mode. As a result when the timer overflows it will generate a wake-up and if the interrupts are enabled also generate a timer interrupt signal.

Configuring the Pulse Width Measurement Mode

In this mode, the width of external pulses applied to the pin-shared external pin PA4/TMR can be measured. In the Pulse Width Measurement Mode, the timer clock source is supplied by the internal clock. For the timer to operate in this mode, bits TM0 and TM1 must both be set high. If the TE bit is low, once a high to low transition has been received on the PA4/TMR pin, the timer will start counting until the PA4/TMR pin returns to its original high level. At this point the TON bit will be automatically reset to zero and the timer will stop counting. If the TE bit is high, the timer will begin counting once a low to high transition has been received on the PA4/TMR pin and stop counting when the PA4/TMR pin returns to its original low level. As before, the TON bit will be automatically reset to zero and the timer will stop counting. It is important to note that in the Pulse Width Measurement Mode, the TON bit is automatically reset to zero when the external control signal on the external timer pin returns to its original level, whereas in the other two modes the TON bit can only be reset to zero under program control. The residual value in the timer, which can now be read by the program, therefore represents the length of the pulse received on pin PA4/TMR. As the TON bit has now been reset any further transitions on the PA4/TMR pin will be ignored. Not until the TON bit is again set high by the program can the timer begin further pulse width measurements. In this way single shot pulse measurements can be easily made. It should be noted that in this mode the counter is controlled by logical transitions on the PA4/TMR pin and not by the logic level.



Timer Mode Timing Chart



Event Counter Mode Timing Chart

As in the case of the other two modes, when the counter is full and overflows, the timer will be reset to the value already loaded into the preload register. If the timer interrupt is enabled, an interrupt signal will also be generated. To ensure that the external pin PA4/TMR is configured to operate as a pulse width measuring input pin, two things have to happen. The first is to ensure that the TM0 and TM1 bits place the timer/event counter in the pulse width measuring mode, the second is to ensure that the port control register configures the pin as an input. It should be noted that a timer overflow is one of the wake-up sources.

Programmable Frequency Divider – PFD

The PFD output is pin-shared with the I/O pin PA3. The PFD function is selected via configuration option, however, if not selected, the pin can operate as a normal I/O pin. The timer overflow signal is the clock source for the PFD circuit. The output frequency is controlled by loading the required values into the timer prescaler registers to give the required division ratio. The counter, driven by the system clock which is divided by the prescaler value, will begin to count-up from this preload register value until full, at which point an overflow signal is generated, causing the PFD output to change state. The counter will then be automatically reloaded with the preload register value and continue counting-up.

For the PFD output to function, it is essential that the corresponding bit of the Port A control register PAC bit 3 is setup as an output. If setup as an input the PFD output will not function, however, the pin can still be used as a normal input pin. The PFD output will only be activated if bit PA3 is set to ″1″. This output data bit is used as the on/off control bit for the PFD output. Note that the PFD output will be low if the PA3 output data bit is cleared to ″0″.

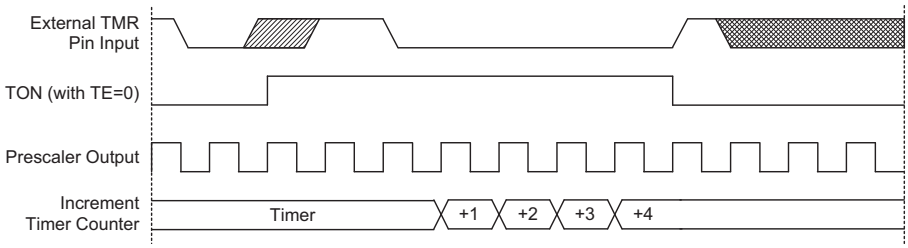
Using this method of frequency generation, and if a crystal oscillator is used for the system clock, very precise values of frequency can be generated.

Prescaler

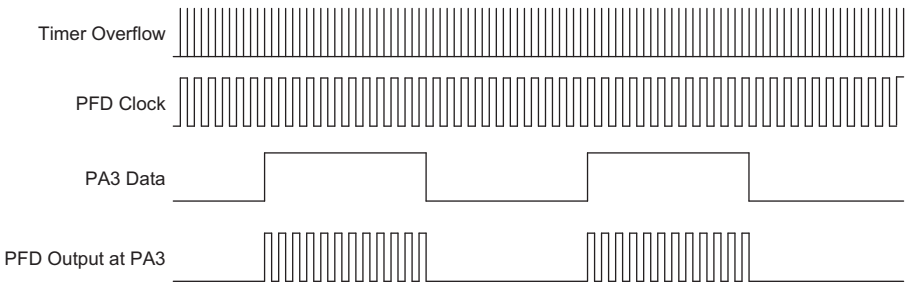
Bits PSC0~PSC2 of the TMRC register can be used to define the pre-scaling stages of the internal clock sources of the Timer/Event Counter. The Timer/Event Counter overflow signal can be used to generate signals for the PFD and Timer Interrupt.

I/O Interfacing

The Timer/Event Counter, when configured to run in the event counter or pulse width measurement mode, require the use of the external PA4/TMR pin for correct operation. As this pin is a shared pin it must be configured correctly to ensure it is setup for use as a Timer/Event Counter input and not as a normal I/O pin. This is implemented by ensuring that the mode select bits in the Timer/Event Counter control register, select either the event counter or pulse width measurement mode. Additionally the Port Control Register PAC bit 4 must be set high to ensure that the pin is setup as an input. Any pull-high resistor configuration option on this pin will remain valid even if the pin is used as a Timer/Event Counter input.



Pulse Width Measure Mode Timing Chart



PFD Output Control

Programming Considerations

When configured to run in the timer mode, 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. For the pulse width measurement mode, the internal system clock is also used as the timer clock source but the timer will only run when the correct logic condition appears on the external timer input pin. As this is an external event and not synchronized with the internal timer clock, the microcontroller will only see this external event when the next timer clock pulse arrives. As a result, there may be small differences in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to be in the event counting mode, which again is an external event and not synchronized with the internal system or timer clock.

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

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

Timer Program Example

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

```
org 04h             ; external interrupt vector
reti
org 08h             ; Timer/Event Counter  interrupt vector
jmp tmrint          ; jump here when Timer overflows
:
org 20h             ; main program
;internal Timer/Event Counter interrupt routine
tmrint:
:
; Timer/Event Counter main program placed here
:
reti
:
:

begin:
;setup Timer registers
mov a,09bh          ; setup Timer preload value
mov tmr,a;
mov a,081h          ; setup Timer control register
mov tmrc,a          ; timer mode and prescaler set to /2
; setup interrupt register
mov a,005h          ; enable master interrupt and timer interrupt
mov intc,a
set tmrc.4          ; start Timer - note mode bits must be previously setup
```

Pulse Width Modulator

Each microcontroller in the Cost-Effective A/D Flash Type with EEPROM MCU series contains either one or two Pulse Width Modulation (PWM) outputs. Useful for such applications such as motor speed control, the PWM function provides outputs with a fixed frequency but with a duty cycle that can be varied by setting particular values into the corresponding PWM register.

For devices with one PWM output, a single register, located in the Data Memory is assigned to the Pulse Width Modulator and is known as the PWM register. For devices with two PWM outputs, two registers are provided and are known as PWM0 and PWM1. It is here that the 8-bit value, which represents the overall duty cycle of one modulation cycle of the output waveform, should be placed. To increase the PWM modulation frequency, each modulation cycle is subdivided into two or four individual modulation subsections, known as the 7+1 mode or 6+2 mode respectively. Each device can choose which mode to use by selecting the appropriate configuration option. When a mode configuration option is chosen, it applies to all PWM outputs on that device. Note that when using the PWM, it is only necessary to write the required value into the appropriate PWM register and select the required mode configuration option, the subdivision of the waveform into its sub-modulation cycles is done automatically within the microcontroller hardware.

For all devices, the PWM clock source is the system clock f_{SYS}.

Device	Channels	PWM Mode	Output Pins	Register Name
HT46F49E	2	6+2 or 7+1	PD0/ PD1	PWM0/ PWM1
Other Devices	1	6+2 or 7+1	PD0	PWM

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

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

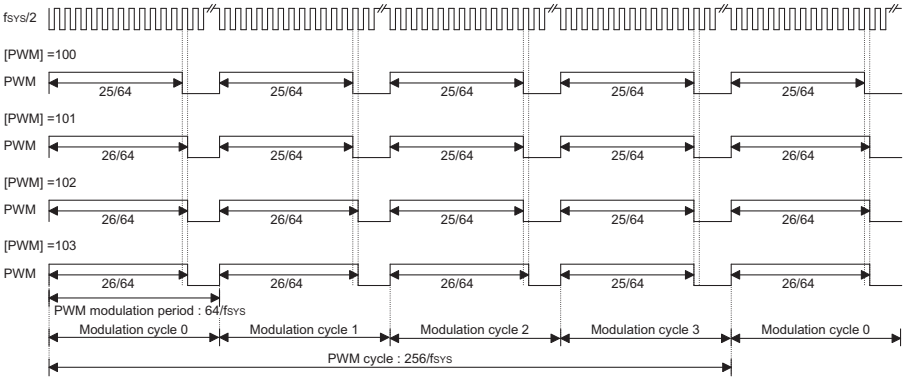
6+2 PWM Mode

Each full PWM cycle, as it is controlled by an 8-bit PWM, PWM0 or PWM1 register, has 256 clock periods. However, in the 6+2 PWM mode, each PWM cycle is subdivided into four individual sub-cycles known as modulation cycle 0 ~ modulation cycle 3, denoted as i in the table. Each one of these four sub-cycles contains 64 clock cycles. In this mode, a modulation frequency increase of four is achieved. The 8-bit PWM, PWM0 or PWM1 register value, which represents the overall duty cycle of the PWM waveform, is divided into two groups. The first group which consists of bit2~bit7 is denoted here as the DC value. The second group which consists of bit0~bit1 is known as the AC value. In the 6+2 PWM mode, the duty cycle value of each of the four modulation sub-cycles is shown in the following table.

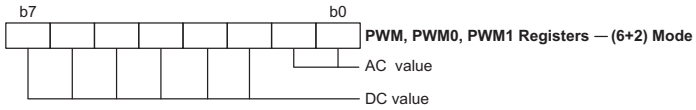
Parameter	AC (0~3)	DC (Duty Cycle)
Modulation cycle i (i=0~3)	i<AC	$\frac{DC+1}{64}$
	i≥AC	$\frac{DC}{64}$

6+2 Mode Modulation Cycle Values

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



6+2 PWM Mode



6+2 Mode Pulse Width Modulation Register

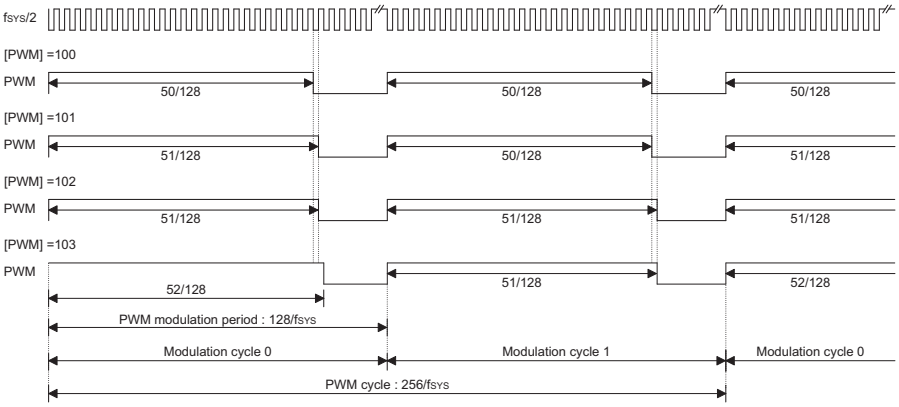
7+1 PWM Mode

Each full PWM cycle, as it is controlled by an 8-bit PWM, PWM0 or PWM1 register, has 256 clock periods. However, in the 7+1 PWM mode, each PWM cycle is subdivided into two individual sub-cycles known as modulation cycle 0 ~ modulation cycle 1, denoted as "i" in the table. Each one of these two sub-cycles contains 128 clock cycles. In this mode, a modulation frequency increase of two is achieved. The 8-bit PWM, PWM0 or PWM1 register value, which represents the overall duty cycle of the PWM waveform, is divided into two groups. The first group which consists of bit1~bit7 is denoted here as the DC value. The second group which consists of bit0 is known as the AC value. In the 7+1 PWM mode, the duty cycle value of each of the two modulation sub-cycles is shown in the following table.

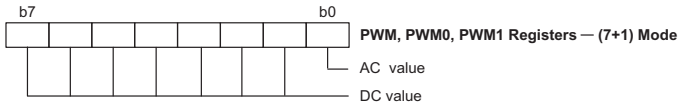
Parameter	AC (0~1)	DC (Duty Cycle)
Modulation cycle i (i=0~1)	i<AC	$\frac{DC+1}{128}$
	i≥AC	$\frac{DC}{128}$

7+1 Mode Modulation Cycle Values

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



7+1 PWM Mode



7+1 Mode Pulse Width Modulation Register

PWM Output Control

On all devices, the PWM outputs are pin-shared with pins PD0 or PD1. To operate as PWM outputs and not as I/O pins, the correct PWM configuration options must be selected. A ″0″ must also be written to the corresponding bits in the I/O port control register PDC to ensure that the required PWM output pin is setup as an output. After these two initial steps have been carried out, and of course after the required PWM value has been written into the PWM register, writing a ″1″ to the corresponding bit in the PD output data register will enable the PWM data to appear on the pin. Writing a ″0″ to the corresponding bit in the PD output data register will disable the PWM output function and force the output low. In this way, the Port D data output register can be used as an on/off control for the PWM function. Note that if the configuration options have selected the PWM function, but a ″1″ has been written to its corresponding bit in the PDC control register to configure the pin as an input, then the pin can still function as a normal input line, with pull-high resistor options.

PWM Programming Example

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

```
clr PDC.0       ; set pin PD0 as output
clr PDC.1       ; set pin PD1 as output

set pd.0        ; PD.0=1; enable pin "PD0/PWM0" to be the PWM channel 0
mov a,64h       ; PWM0=100D=64H
mov pwm0,a

set pd.1        ; PD.1=1; enable pin "PD1/PWM1" to be the PWM channel 1
mov a,65h       ; PWM1=101D=65H
mov pwm1,a


clr pd.0        ; disable PWM0 output – PD.0 will remain low
clr pd.1        ; disable PWM1 output – PD.1 will remain low
```

Analog to Digital Converter

The need to interface to real world analog signals is a common requirement for many electronic systems. However, to properly process these signals by a microcontroller, they must first be converted into digital signals by A/D converters. By integrating the A/D conversion electronic circuitry into the microcontroller, the need for external components is reduced significantly with the corresponding follow-on benefits of lower costs and reduced component space requirements.

A/D Overview

Each of the devices contains a 4-channel analog to digital converter which can directly interface to external analog signals, such as that from sensors or other control signals and convert these signals directly into either an 8-bit or 9-bit digital value.

Device	Input Channels	Conversion Bits	Input Pins
HT46F46E	4	8	PB0~PB3
HT46F47E	4	9	PB0~PB3
HT46F48E	4	9	PB0~PB3
HT46F49E	4	9	PB0~PB3

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

A/D Converter Data Registers – ADR, ADRL, ADRH

For the HT46F46E device, which has an 8-bit A/D converter, a single register, known as ADR, is used to store the 8-bit analog to digital conversion value. For the remaining devices, which have a 9-bit A/D converter, two registers are required, a high byte register, known as ADRH, and a low byte register, known as ADRL. After the conversion process takes place, these registers can be directly read by the microcontroller to obtain the digitised conversion value. For devices which use two A/D Converter Data Registers, note that only the high byte register ADRH utilises its full 8-bit contents. The low byte register utilises only 1 bit of its 8-bit contents as it contains only the lowest bit of the 9-bit converted value.

In the following tables, D0~D8 are the A/D conversion data result bits.

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADR	D7	D6	D5	D4	D3	D2	D1	D0

A/D Data Register – HT46F46E

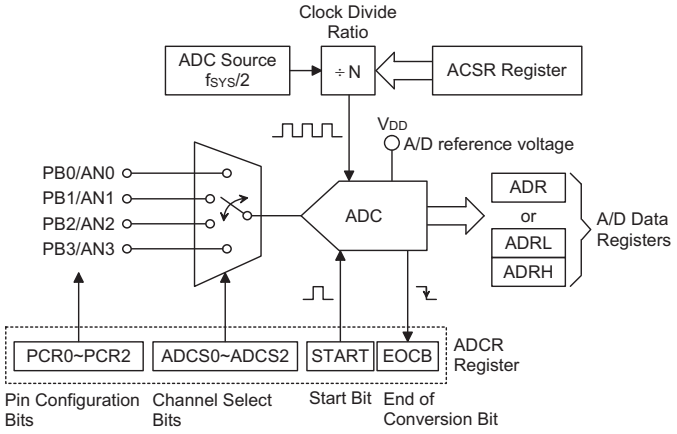
Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADRL	D0	—	—	—	—	—	—	—
ADRH	D8	D7	D6	D5	D4	D3	D2	D1

A/D Data Register – Other Devices

A/D Converter Control Register – ADCR

To control the function and operation of the A/D converter, a control register known as ADCR is provided. This 8-bit register defines functions such as the selection of which analog channel is connected to the internal A/D converter, which pins are used as analog inputs and which are used as normal I/Os as well as controlling the start function and monitoring the A/D converter end of conversion status.

One section of this register contains the bits ACS2~ACS0 which define the channel number. As each of the devices contains only one actual analog to digital converter circuit, each of the individual 4 analog inputs must be routed to the converter. It is the function of the ACS2~ACS0 bits in the ADCR register to determine which analog channel is actually connected to the internal A/D converter. Note that the ACS2 bit must always be assigned a zero value.



A/D Converter Structure

The ADCR control register also contains the PCR2~PCR0 bits which determine which pins on Port B are used as analog inputs for the A/D converter and which pins are to be used as normal I/O pins. If the 3-bit address on PCR2~PCR0 has a value of "100" or higher, then all four pins, namely AN0, AN1, AN2 and AN3 will all be set as analog inputs. Note that if the PCR2~PCR0 bits are all set to zero, then all the Port B pins will be setup as normal I/Os and the internal A/D converter circuitry will be powered off to reduce the power consumption.

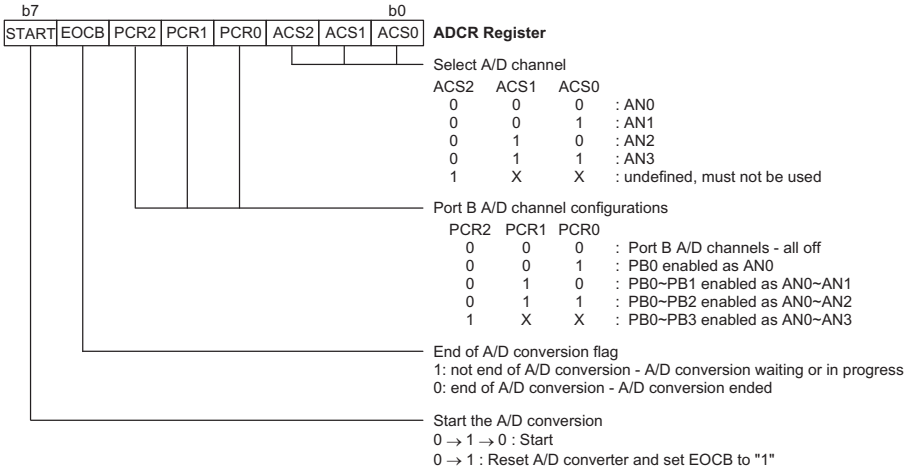
The START bit in the ADCR register is used to start and reset the A/D converter. When the microcontroller sets this bit from low to high and then low again, an analog to digital conversion cycle will be initiated. When the START bit is brought from low to high but not low again, the EOCB bit in the ADCR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

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

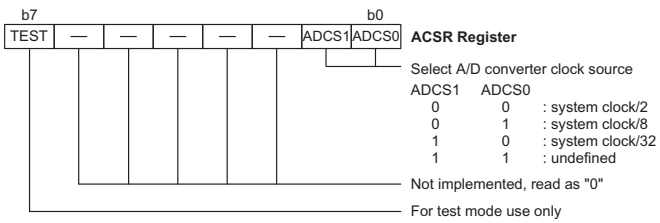
A/D Converter Clock Source Register – ACSR

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

Although the A/D clock source is determined by the system clock f_{SYS}, and by bits ADCS1 and ADCS0, there are some limitations on the maximum A/D clock source speed that can be selected. As the minimum value of permissible A/D clock period, t_{AD}, is 0.5μs for the HT46F46E, device, and 1μs for the other devices, care must be taken for system clock speeds in excess of 2MHz. With the exception of the HT46F46E device, for system clock speeds in excess of 2MHz, the ADCS1 and ADCS0 bits should not be set to "00". For the HT46F46E device, for system clock speeds in excess of 4MHz, the ADCS1 and ADCS0 bits should not be set to "00". Doing so will give A/D clock periods that are less than the minimum A/D clock period which may result in inaccurate A/D conversion values. Refer to the following table for examples, where values marked with an asterisk * show where, depending upon the device, special care must be taken, as the values may be less than the specified minimum A/D Clock Period.



A/D Converter Control Register



A/D Converter Clock Source Register

f_{SYS}	A/D Clock Period (t_{AD})			
	ADCS1, ADCS0=00 ($f_{SYS}/2$)	ADCS1, ADCS0=01 ($f_{SYS}/8$)	ADCS1, ADCS0=10 ($f_{SYS}/32$)	ADCS1, ADCS0=11
1MHz	2μs	8μs	32μs	Undefined
2MHz	1μs	4μs	16μs	Undefined
4MHz	500ns*	2μs	8μs	Undefined
8MHz	250ns*	1μs	4μs	Undefined
12MHz	166.67ns*	0.67μs	2.67μs	Undefined

A/D Clock Period Examples

A/D Input Pins

All of the A/D analog input pins are pin-shared with the I/O pins on Port B. Bits PCR2~PCR0 in the ADCR register, not configuration options, determine whether the input pins are setup as normal Port B input/output pins or whether they are setup as analog inputs. In this way, pins can be changed under program control to change their function from normal I/O operation to analog inputs and vice versa. Pull-high resistors, which are setup through configuration options, apply to the input pins only when they are used as normal I/O pins, if setup as A/D inputs the pull-high resistors will be automatically disconnected. Note that it is not necessary to first setup the A/D pin as an input in the PBC port control register to enable the A/D input, when the PCR2~PCR0 bits enable an A/D input, the status of the port control register will be overridden. The VDD power supply pin is used as the A/D converter reference voltage, and as such analog inputs must not be allowed to exceed this value. Appropriate measures should also be taken to ensure that the VDD pin remains as stable and noise free as possible.

Initialising the A/D Converter

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

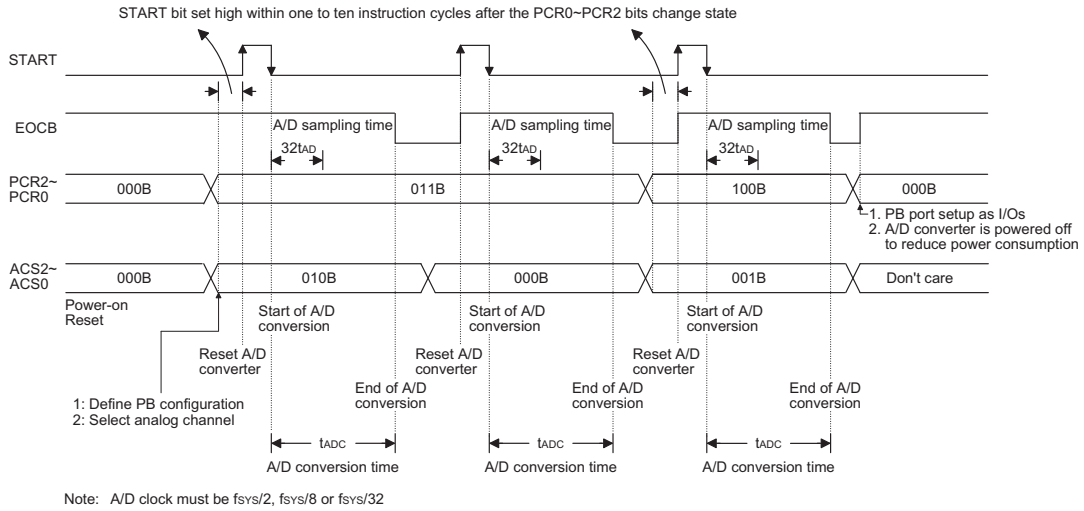
Summary of A/D Conversion Steps

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

- Step 1
  Select the required A/D conversion clock by correctly programming bits ADCS1 and ADCS0 in the ACSR register.
- Step 2
  Select which channel is to be connected to the internal A/D converter by correctly programming the ACS2~ACS0 bits which are also contained in the ADCR register.
- Step 3
  Select which pins on Port B are to be used as A/D inputs and configure them as A/D input pins by correctly programming the PCR2~PCR0 bits in the ADCR register. Note that this step can be combined with Step 2 into a single ADCR register programming operation.
- Step 4
  If the interrupts are to be used, the interrupt control registers must be correctly configured to ensure the A/D converter interrupt function is active. The master interrupt control bit, EMI, in the INTC interrupt control register must be set to ″1″ and the A/D converter interrupt bit, EADI, in the INTC register must also be set to ″1″.
- Step 5
  The analog to digital conversion process can now be initialised by setting the START bit in the ADCR register from ″0″ to ″1″ and then to ″0″ again. Note that this bit should have been originally set to ″0″.
- Step 6
  To check when the analog to digital conversion process is complete, the EOCB bit in the ADCR register can be polled. The conversion process is complete when this bit goes low. When this occurs the A/D data registers ADRL and ADRH can be read to obtain the conversion value. As an alternative method if the interrupts are enabled and the stack is not full, the program can wait for an A/D interrupt to occur.

Note: When checking for the end of the conversion process, if the method of polling the EOCB bit in the ADCR register is used, the interrupt enable step above can be omitted.

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



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

A/D Conversion Timing

The setting up and operation of the A/D converter function is fully under the control of the application program as there are no configuration options associated with the A/D converter. After an A/D conversion process has been initiated by the application program, the microcontroller internal hardware will begin to carry out the conversion, during which time the program can continue with other functions. The time taken for the A/D conversion is dependent upon the device chosen and is a function of the A/D clock period t_{AD} as shown in the table.

Device	A/D Conversion Time
HT46F46E	$64t_{AD}$
Other Devices	$76t_{AD}$

A/D Conversion Time

Programming Considerations

When programming, special attention must be given to the A/D channel selection bits in the ADCR register. If these bits are all cleared to zero no external pins will be selected for use as A/D input pins allowing the pins to be used as normal I/O pins. When this happens the power supplied to the internal A/D circuitry will be reduced resulting in a reduction of supply current. This ability to reduce power by turning off the internal A/D function by clearing the A/D channel selection bits may be an important consideration in battery powered applications.

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

A/D Programming Example

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

Example: using an EOCB polling method to detect the end of conversion for the HT46F46E

```
    clr  EADI                 ; disable ADC interrupt
    mov  a,00000001B
    mov  ACSR,a               ; setup the ACSR register to select fSYS/8 as
                              ; the A/D clock
    mov  a,00100000B          ; setup ADCR register to configure Port PB0~PB3
                              ; as A/D inputs
    mov  ADCR,a               ; and select AN0 to be connected to the A/D
                              ; converter
         :
         :                    ; As the Port B channel bits have changed the
                              ; following START
                              ; signal (0-1-0) must be issued within 10
                              ; instruction cycles
         :
Start_conversion:
    clr  START
    set  START                ; reset A/D
    clr  START                ; start A/D
Polling_EOC:
    sz   EOCB                 ; poll the ADCR register EOCB bit to detect end
                              ; of A/D conversion
    jmp  polling_EOC          ; continue polling
    mov  a,ADR                ; read conversion result value from the ADR
                              ; register
    mov  adr_buffer,a         ; save result to user defined memory
         :
         :
    jmp  start_conversion     ; start next A/D conversion
```

Example: using an interrupt method to detect the end of conversion for the HT46F46E

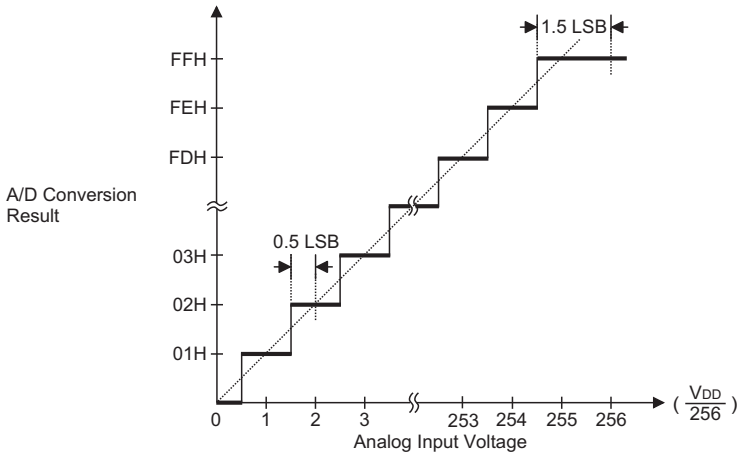
```
    clr  EADI                  ; disable ADC interrupt
    mov  a,00000001B
    mov  ACSR,a                ; setup the ACSR register to select fSYS/8 as
                               ; the A/D clock

    mov  a,00100000B           ; setup ADCR register to configure Port PB0~PB3
                               ; as A/D inputs
    mov  ADCR,a                ; and select AN0 to be connected to the A/D
                               ; converter
         :
                               ; As the Port B channel bits have changed the
                               ; following START
                               ; signal (0-1-0) must be issued within 10
                               ; instruction cycles
         :
Start_conversion:
    clr  START
    set  START                 ; reset A/D
    clr  START                 ; start A/D
    clr  ADF                   ; clear ADC interrupt request flag
    set  EADI                  ; enable ADC interrupt
    set  EMI                   ; enable global interrupt
         :
         :
         :
; ADC interrupt service routine
ADC_ISR:
    mov  acc_stack,a           ; save ACC to user defined memory
    mov  a,STATUS
    mov  status_stack,a        ; save STATUS to user defined memory
         :
         :
    mov  a,ADR                 ; read conversion result value from the ADR
                               ; register
    mov  adr_buffer,a          ; save result to user defined register
         :
         :
EXIT_INT_ISR:
    mov  a,status_stack
    mov  STATUS,a              ; restore STATUS from user defined memory
    mov  a,acc_stack           ; restore ACC from user defined memory
    reti
```

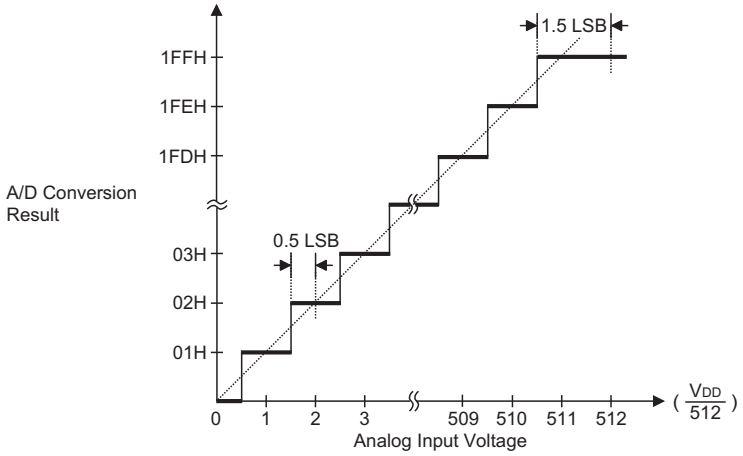
A/D Transfer Function

As the HT46F46E device contain an 8-bit A/D converter, their full-scale converted digitized value is equal to 0FFH. Since the full-scale analog input value is equal to the voltage, this gives a single bit analog input value of $V_{DD}/256$. For the other devices which each contain a 9-bit A/D converter, their full-scale converted digitised value is equal to 1FFH giving a single bit analog input value of $V_{DD}/512$. The following graphs show the ideal transfer function between the analog input value and the digitised output value for the A/D converters.

Note that to reduce the quantisation error, a 0.5 LSB offset is added to the A/D Converter input. Except for the digitised zero value, the subsequent digitised values will change at a point 0.5 LSB below where they would change without the offset, and the last full scale digitised value will change at a point 1.5 LSB below the V_{DD} level.



Ideal A/D Transfer Function – HT46F46E



Ideal A/D Transfer Function – Other Devices

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer/Event Counter or an A/D converter 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 in this series contains a single external interrupt and two internal interrupts functions. The external interrupt is controlled by the action of the external $\overline{INT}$ pin, while the internal interrupts are controlled by the Timer/Event Counter overflow and the A/D converter interrupt.

Interrupt Register

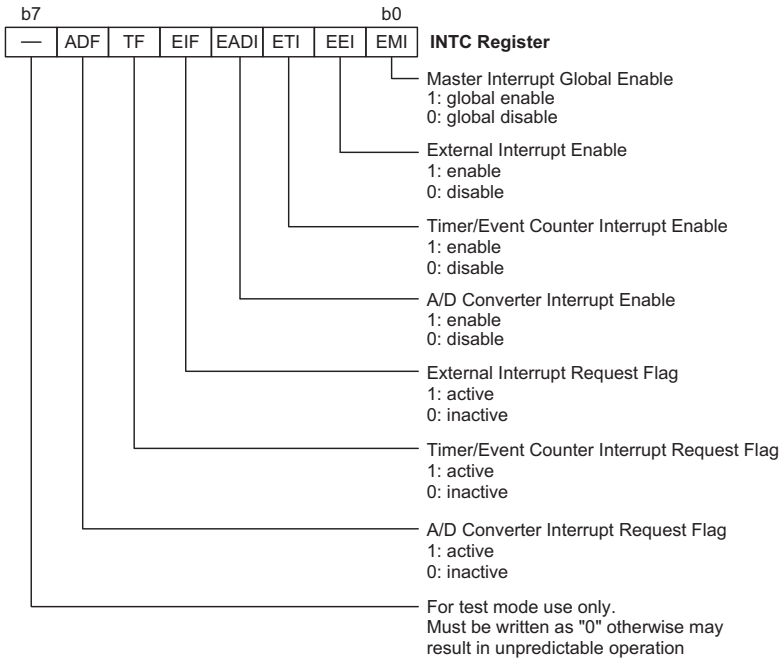
Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by a single INTC register, which is located in Data Memory. By controlling the appropriate enable bits in this register each individual interrupt can be enabled or disabled. Also when an interrupt occurs,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

Interrupt Operation

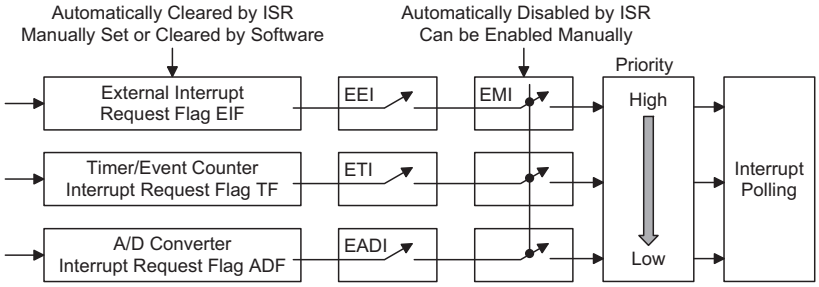
A Timer/Event Counter overflow, an end of A/D conversion or the external interrupt line being pulled low will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a JMP statement which will jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI statement, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

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

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



Interrupt Control Register



Interrupt Structure

Interrupt Priority

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

Interrupt Source	All Devices Priority
External Interrupt	1
Timer/Event Counter Overflow	2
A/D Converter Interrupt	3

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

External Interrupt

For an external interrupt to occur, the global interrupt enable bit, EMI, and external interrupt enable bit, EEI, must first be set. Additionally the correct interrupt configuration options must be selected to enable the external interrupt function and to choose the trigger edge type. An actual external interrupt will take place when the external interrupt request flag, EIF, is set, a situation that will occur when a transition, whose type is chosen by configuration option, appears on the $\overline{INT}$ line. The external interrupt pin is pin-shared with the I/O pin PA5 and can only be configured as an external interrupt pin if the corresponding external interrupt enable bit in the INTC register has been set. The pin must also be setup as an input by setting the corresponding PAC.5 bit in the port control register. When the interrupt is enabled, the stack is not full and the correct transition type appears on the external interrupt pin, a subroutine call to the external interrupt vector at location 04H, will take place. When the interrupt is serviced, the external interrupt request flag, EIF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor configuration options on this pin will remain valid even if the pin is used as an external interrupt input.

Timer/Event Counter Interrupt

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

A/D Interrupt

For an A/D interrupt to occur, the global interrupt enable bit, EMI, and the corresponding interrupt enable bit, EADI, must be first set. An actual A/D interrupt will take place when the A/D converter request flag, ADF, is set, a situation that will occur when an A/D conversion process has completed. When the interrupt is enabled, the stack is not full and an A/D conversion process finishes execution, a subroutine call to the A/D interrupt vector at location 0CH, will take place. When the interrupt is serviced, the A/D interrupt request flag, ADF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Programming Considerations

By disabling the interrupt enable bits, a requested interrupt can be prevented from being serviced, however, once an interrupt request flag is set, it will remain in this condition in the INTC 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.

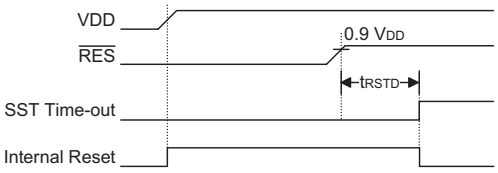
Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{\text{RES}}$ reset is implemented in situations where the power supply voltage falls below a certain threshold.

Reset Functions

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

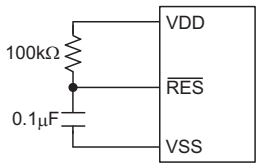
- Power-on Reset
  The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to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

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



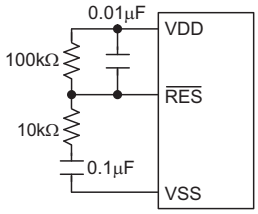
Power-On Reset Timing Chart

For most applications a resistor connected between VDD and the $\overline{\text{RES}}$ 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 $\overline{\text{RES}}$ pin should be kept as short as possible to minimise any stray noise interference.



Basic Reset Circuit

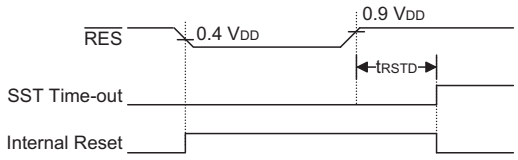
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Enhanced Reset Circuit

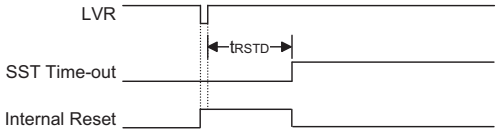
More information regarding external reset circuits is located in Application Note HA0075E on the Holtek website.

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



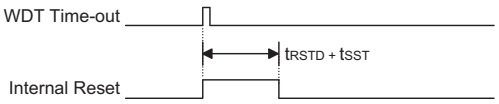
$\overline{RES}$ Reset Timing Chart

- Low Voltage Reset – LVR
  The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device, which is selected via a configuration option. If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing the battery, the LVR will automatically reset the device internally. The LVR includes the following specifications: For a valid LVR signal, a low voltage, i.e., a voltage in the range between 0.9V~V_{LVR} must exist for greater than the value t_{LVR} specified in the A.C. characteristics. If the low voltage state does not exceed 1ms, the LVR will ignore it and will not perform a reset function.



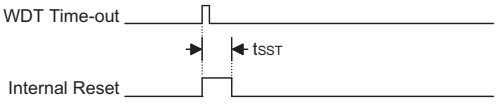
Low Voltage Reset Timing Chart

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



WDT Time-out Reset during Normal Operation Timing Chart

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



WDT Time-out Reset during Power Down Timing Chart

Reset Initial Conditions

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

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

Note: "u" stands for unchanged

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

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

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

HT46F46E

Register	Reset (Power-on)	$\overline{\text{RES}}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
MP0	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu
MP1	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu
BP	xxxx xxx0	xxxx xxx0	xxxx xxx0	xxxx xxxu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	--xx xxxx	--uu uuuu	--uu uuuu	--uu uuuu
STATUS	--00 xxxx	--uu uuuu	--1u uuuu	--11 uuuu
INTC	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMRC	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	---- 1111	---- 1111	---- 1111	---- uuuu
PBC	---- 1111	---- 1111	---- 1111	---- uuuu
PD	---- ---1	---- ---1	---- ---1	---- ---u
PDC	---- ---1	---- ---1	---- ---1	---- ---u
PWM	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADR	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR	0100 0000	0100 0000	0100 0000	uuuu uuuu
ACSR	1--- --00	1--- --00	1--- --00	u--- --uu
EECR	1000 ----	1000 ----	1000 ----	uuuu ----

"u" stands for unchanged
"x" stands for unknown
"-" stands for unimplemented

HT46F47E

Register	Reset (Power-on)	$\overline{\text{RES}}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
MP0	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu
MP1	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu
BP	xxxx xxx0	xxxx xxx0	xxxx xxx0	xxxx xxxu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	--xx xxxx	--uu uuuu	--uu uuuu	--uu uuuu
STATUS	--00 xxxx	--uu uuuu	--1u uuuu	--11 uuuu
INTC	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMRC	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	---- 1111	---- 1111	---- 1111	---- uuuu
PBC	---- 1111	---- 1111	---- 1111	---- uuuu
PD	---- ---1	---- ---1	---- ---1	---- ---u
PDC	---- ---1	---- ---1	---- ---1	---- ---u
PWM	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADRL	x--- ----	x--- ----	x--- ----	u--- ----
ADRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR	0100 0000	0100 0000	0100 0000	uuuu uuuu
ACSR	1--- --00	1--- --00	1--- --00	u--- --uu
EECR	1000 ----	1000 ----	1000 ----	uuuu ----

"u" stands for unchanged
"x" stands for unknown
"–" stands for unimplemented

HT46F48E

Register	Reset (Power-on)	$\overline{\text{RES}}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
MP0	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu
MP1	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu
BP	xxxx xxx0	xxxx xxx0	xxxx xxx0	xxxx xxxu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	--xx xxxx	--uu uuuu	--uu uuuu	--uu uuuu
STATUS	--00 xxxx	--uu uuuu	--1u uuuu	--11 uuuu
INTC	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMRC	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	---- --11	---- --11	---- --11	---- --uu
PCC	---- --11	---- --11	---- --11	---- --uu
PD	---- ---1	---- ---1	---- ---1	---- ---u
PDC	---- ---1	---- ---1	---- ---1	---- ---u
PWM	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADRL	x--- ----	x--- ----	x--- ----	u--- ----
ADRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR	0100 0000	0100 0000	0100 0000	uuuu uuuu
ACSR	1--- --00	1--- --00	1--- --00	u--- --uu
EECR	1000 ----	1000 ----	1000 ----	uuuu ----

"u" stands for unchanged
"x" stands for unknown
"-" stands for unimplemented

HT46F49E

Register	Reset (Power-on)	$\overline{\text{RES}}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
MP0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
BP	xxxx xxx0	xxxx xxx0	xxxx xxx0	xxxx xxxu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	-xxx xxxx	-uuu uuuu	-uuu uuuu	-uuu uuuu
STATUS	--00 xxxx	--uu uuuu	--1u uuuu	--11 uuuu
INTC	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMRC	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	---1 1111	---1 1111	---1 1111	---u uuuu
PCC	---1 1111	---1 1111	---1 1111	---u uuuu
PD	---- --11	---- --11	---- --11	---- --uu
PDC	---- --11	---- --11	---- --11	---- --uu
PWM0	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
PWM1	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADRL	x--- ----	x--- ----	x--- ----	u--- ----
ADRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR	0100 0000	0100 0000	0100 0000	uuuu uuuu
ACSR	1--- --00	1--- --00	1--- --00	u--- --uu
EECR	1000 ----	1000 ----	1000 ----	uuuu ----

"u" stands for unchanged
"x" stands for unknown
"-" stands for unimplemented

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:

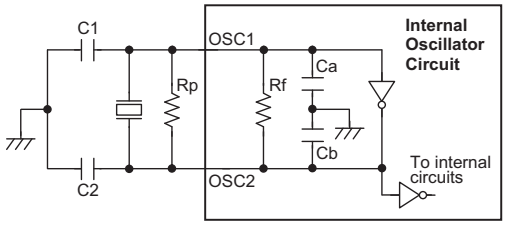
- External crystal/resonator oscillator
- External RC oscillator

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	470kΩ

Oscillator Internal Component Values

Crystal Oscillator C1 and C2 Values			
Crystal Frequency	C1	C2	CL
8MHz	TBD	TBD	TBD
4MHz	TBD	TBD	TBD
1MHz	TBD	TBD	TBD
Note: 1. C1 and C2 values are for guidance only. 2. CL is the crystal manufacturer specified load capacitor value.			

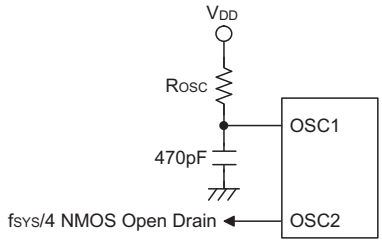
Crystal Recommended Capacitor Values

Resonator C1 and C2 Values		
Resonator Frequency	C1	C2
3.58MHz	TBD	TBD
1MHz	TBD	TBD
455kHz	TBD	TBD
Note: C1 and C2 values are for guidance only.		

Resonator Recommended Capacitor Values

External RC Oscillator

Using the external system RC oscillator requires that a resistor, with a value between 15kΩ and 750KΩ, is connected between OSC1 and VDD, and a capacitor is connected to ground. 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 ROSC 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.



RC Oscillator

Watchdog Timer Oscillator

The WDT oscillator is a fully integrated free running RC oscillator with a typical period of 65μs at 5V, requiring no external components. It is selected via configuration option. If selected, when the device enters the Power Down Mode, the system clock will stop running, however the WDT oscillator will continue to run and keep the watchdog function active. However, as the WDT will consume a certain amount of power when in the Power Down Mode, for low power applications, it may be desirable to disable the WDT oscillator by configuration option.

Power Down Mode and Wake-up

Power Down Mode

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

Entering the Power Down Mode

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

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

Standby Current Considerations

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

Wake-up

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

- An external reset
- An external falling 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 subroutine execution will be delayed by an additional one or more cycles. If the wake-up results in the execution of the next instruction following the ″HALT″ instruction, this will be executed immediately after the 1024 system clock period delay has ended.

Watchdog Timer

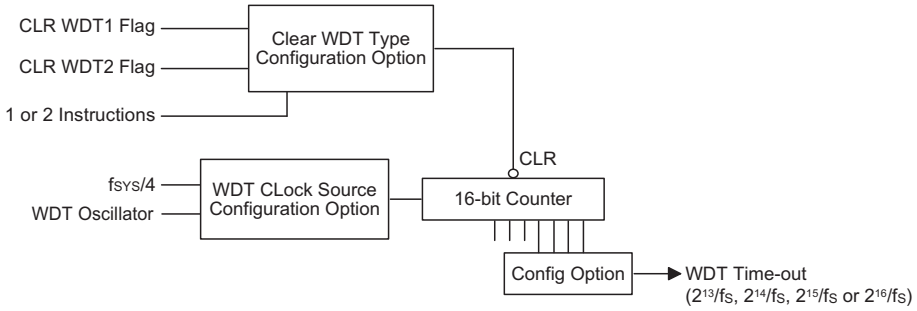
The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by one of two sources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

In the Cost-Effective A/D Flash Type with EEPROM series of microcontrollers, all Watchdog Timer options, such as enable/disable, WDT clock source and clear instruction type all selected through configuration options. There are no internal registers associated with the WDT in the Cost-Effective A/D Flash Type MCU series. One of the WDT clock sources is an internal oscillator which has an approximate period of 65μs at a supply voltage of 5V. However, it should be noted that this specified internal clock period can vary with VDD, temperature and process variations. The other WDT clock source option is the $f_{SYS}/4$ clock. Whether the WDT clock source is its own internal WDT oscillator, or from $f_{SYS}/4$, it is divided by 2^{13}~2^{16} (by options to get the WDT time-out period). The max time out period is around 4.3s when the 2^{16} is selected. This time-out period may vary with temperature, VDD and process variations. As the clear instruction only resets the last stage of the divider chain, for this reason the actual division ratio and corresponding Watchdog Timer time-out can vary by a factor of two. The exact division ratio depends upon the residual value in the Watchdog Timer counter before the clear instruction is executed. It is important to realise that as there are no independent internal registers or configuration options associated with the length of the Watchdog Timer time-out, it is completely dependent upon the frequency of $f_{SYS}/4$ or the internal WDT oscillator.

If the $f_{SYS}/4$ clock is used as the WDT clock source, it should be noted that when the system enters the Power Down Mode, then the instruction clock is stopped and the WDT will lose its protecting purposes. For systems that operate in noisy environments, using the internal WDT oscillator is strongly recommended.

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, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the WDT. The first is an external hardware reset, which means a low level on the $\overline{RES}$ 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.



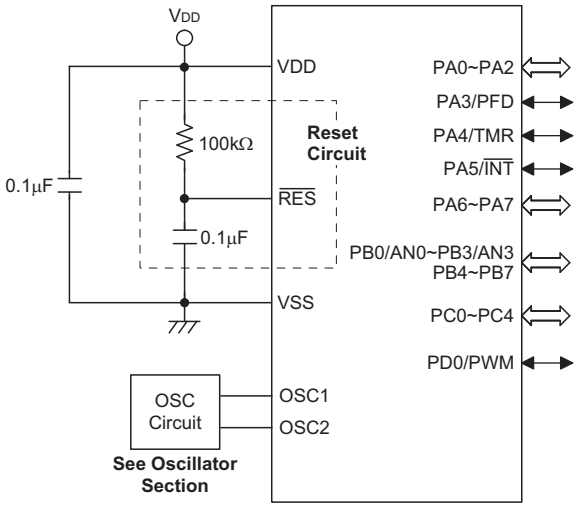
Watchdog Timer

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 as the application software has no control over the configuration options. All options must be defined for proper system function, the details of which are shown in the table.

No.	Options
1	Watchdog Timer clock source: WDT oscillator or $f_{SYS}/4$
2	Watchdog Timer function: enable or disable
3	WDT time-out period: $2^{13}/f_S$, $2^{14}/f_S$, $2^{15}/f_S$, or $2^{16}/f_S$
4	CLRWDT instructions: 1 or 2 instructions
5	System oscillator: Crystal or RC
6	PA, PB and PD: pull-high enable or disable PC: pull-high enable or disable - HT46F48E and HT46F49E only
7	PWM: enable or disable - Except HT46F49E PWM0, PWM1: enable or disable - HT46F49E only PWM mode: 6+2 or 7+1 mode selection
8	PA0~PA7: wake-up enable or disable - bit option
9	PFD: normal I/O or PFD output
10	LVR function: enable or disable Low voltage reset voltage: 2.1V, 3.15V or 4.2V
11	External interrupt $\overline{INT}$ trigger edge: disable, rising, falling, or double (rising or falling)

Application Circuits



Instruction Set

Introduction

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

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

Instruction Timing

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

Moving and Transferring Data

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

Arithmetic Operations

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

Logical and Rotate Operations

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

Branches and Control Transfer

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

Bit Operations

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

Table Read Operations

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

Other Operations

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

Instruction Set Summary

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

Table conventions:

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

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

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

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

Instruction Definition

ADC A,[m]	Add Data Memory to ACC with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C
ADCM A,[m]	Add ACC to Data Memory with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C
ADD A,[m]	Add Data Memory to ACC
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m]
Affected flag(s)	OV, Z, AC, C
ADD A,x	Add immediate data to ACC
Description	The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + x
Affected flag(s)	OV, Z, AC, C
ADDM A,[m]	Add ACC to Data Memory
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m]
Affected flag(s)	OV, Z, AC, C
AND A,[m]	Logical AND Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ [m]
Affected flag(s)	Z
AND A,x	Logical AND immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ x
Affected flag(s)	Z
ANDM A,[m]	Logical AND ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″AND″ [m]
Affected flag(s)	Z

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

CPL [m] Complement Data Memory

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

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

Affected flag(s) Z

CPLA [m] Complement Data Memory with result in ACC

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

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

Affected flag(s) Z

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

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

Operation $[m] \leftarrow ACC + 00H$ or
$[m] \leftarrow ACC + 06H$ or
$[m] \leftarrow ACC + 60H$ or
$[m] \leftarrow ACC + 66H$

Affected flag(s) C

DEC [m] Decrement Data Memory

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

Operation $[m] \leftarrow [m] - 1$

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

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

Operation $ACC \leftarrow [m] - 1$

Affected flag(s) Z

HALT Enter power down mode

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

Operation $TO \leftarrow 0$
$PDF \leftarrow 1$

Affected flag(s) TO, PDF

INC [m]	Increment Data Memory
Description	Data in the specified Data Memory is incremented by 1.
Operation	[m] ← [m] + 1
Affected flag(s)	Z
INCA [m]	Increment Data Memory with result in ACC
Description	Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC ← [m] + 1
Affected flag(s)	Z
JMP addr	Jump unconditionally
Description	The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction.
Operation	Program Counter ← addr
Affected flag(s)	None
MOV A,[m]	Move Data Memory to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator.
Operation	ACC ← [m]
Affected flag(s)	None
MOV A,x	Move immediate data to ACC
Description	The immediate data specified is loaded into the Accumulator.
Operation	ACC ← x
Affected flag(s)	None
MOV [m],A	Move ACC to Data Memory
Description	The contents of the Accumulator are copied to the specified Data Memory.
Operation	[m] ← ACC
Affected flag(s)	None
NOP	No operation
Description	No operation is performed. Execution continues with the next instruction.
Operation	No operation
Affected flag(s)	None
OR A,[m]	Logical OR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ [m]
Affected flag(s)	Z

OR A,x	Logical OR immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ x
Affected flag(s)	Z
ORM A,[m]	Logical OR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″OR″ [m]
Affected flag(s)	Z
RET	Return from subroutine
Description	The Program Counter is restored from the stack. Program execution continues at the restored address.
Operation	Program Counter ← Stack
Affected flag(s)	None
RET A,x	Return from subroutine and load immediate data to ACC
Description	The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address.
Operation	Program Counter ← Stack ACC ← x
Affected flag(s)	None
RETI	Return from interrupt
Description	The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. If an interrupt was pending when the RETI instruction is executed, the pending Interrupt routine will be processed before returning to the main program.
Operation	Program Counter ← Stack EMI ← 1
Affected flag(s)	None
RL [m]	Rotate Data Memory left
Description	The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0.
Operation	[m].(i+1) ← [m].i; (i = 0~6) [m].0 ← [m].7
Affected flag(s)	None
RLA [m]	Rotate Data Memory left with result in ACC
Description	The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation	ACC.(i+1) ← [m].i; (i = 0~6) ACC.0 ← [m].7
Affected flag(s)	None

RLC [m] Rotate Data Memory left through Carry

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

RR [m] Rotate Data Memory right

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

RRC [m] Rotate Data Memory right through Carry

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

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

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

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

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

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

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

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

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

SDZ [m] Skip if decrement Data Memory is 0

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

Operation $[m] \leftarrow [m] - 1$
Skip if $[m] = 0$

Affected flag(s) None

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

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

Operation $ACC \leftarrow [m] - 1$
Skip if $ACC = 0$

Affected flag(s) None

SET [m] Set Data Memory

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

Operation $[m] \leftarrow FFH$

Affected flag(s) None

SET [m].i Set bit of Data Memory

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

Operation $[m].i \leftarrow 1$

Affected flag(s) None

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

SWAP [m]	Swap nibbles of Data Memory
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged.
Operation	[m].3~[m].0 ↔ [m].7 ~ [m].4
Affected flag(s)	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC.3 ~ ACC.0 ← [m].7 ~ [m].4 ACC.7 ~ ACC.4 ← [m].3 ~ [m].0
Affected flag(s)	None
SZ [m]	Skip if Data Memory is 0
Description	If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	Skip if [m] = 0
Affected flag(s)	None
SZA [m]	Skip if Data Memory is 0 with data movement to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	ACC ← [m] Skip if [m] = 0
Affected flag(s)	None
SZ [m].i	Skip if bit i of Data Memory is 0
Description	If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.
Operation	Skip if [m].i = 0
Affected flag(s)	None
TABRDC [m]	Read table (current page) to TBLH and Data Memory
Description	The low byte of the program code (current page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory
Description	The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

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

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

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

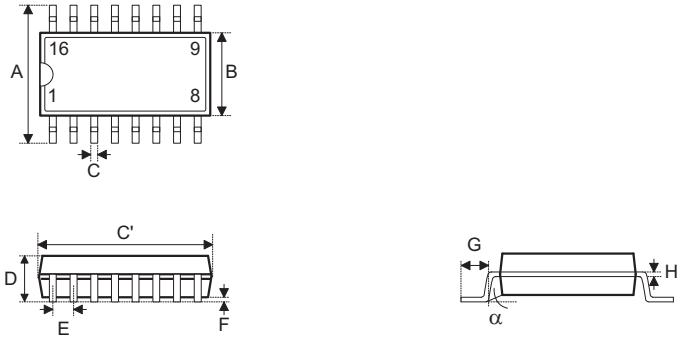
Package Information

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

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

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

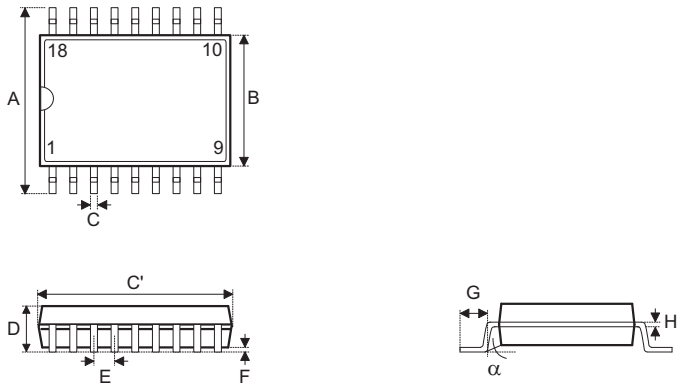
16-pin NSOP (150mil) Outline Dimensions



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

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A		6.00 BSC	
B		3.90 BSC	
C	0.31	—	0.51
C′		9.90 BSC	
D	—	—	1.75
E		1.27 BSC	
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°

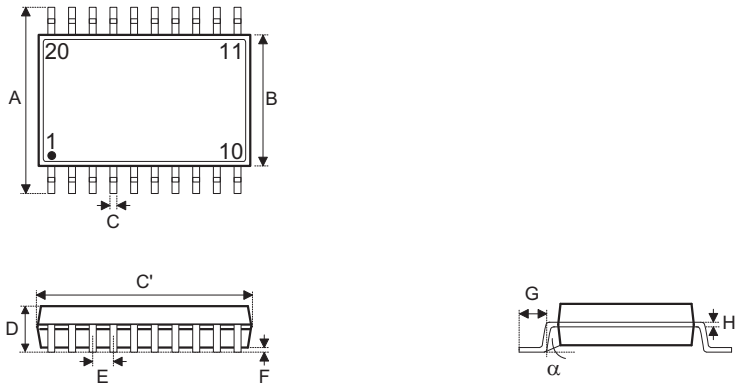
18-pin SOP (300mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A		0.406 BSC	
B		0.295 BSC	
C	0.012	—	0.020
C′		0.455 BSC	
D	—	—	0.104
E		0.050 BSC	
F	0.004	—	0.012
G	0.016	—	0.050
H	0.008	—	0.013
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A		10.30 BSC	
B		7.50 BSC	
C	0.31	—	0.51
C′		11.55 BSC	
D	—	—	2.65
E		1.27 BSC	
F	0.10	—	0.30
G	0.40	—	1.27
H	0.20	—	0.33
α	0°	—	8°

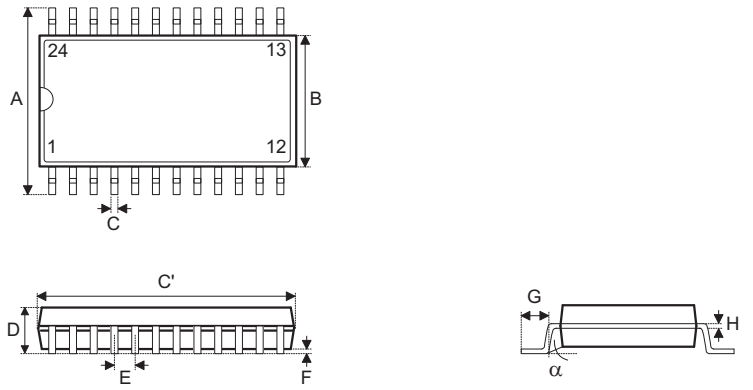
20-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A		0.236 BSC	
B		0.154 BSC	
C	0.008	—	0.012
C′		0.341 BSC	
D	—	—	0.069
E		0.025 BSC	
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A		6.00 BSC	
B		3.90 BSC	
C	0.20	—	0.30
C′		8.66 BSC	
D	—	—	1.75
E		0.635 BSC	
F	0.10	—	0.25
G	0.41	—	1.27
H	0.10	—	0.25
α	0°	—	8°

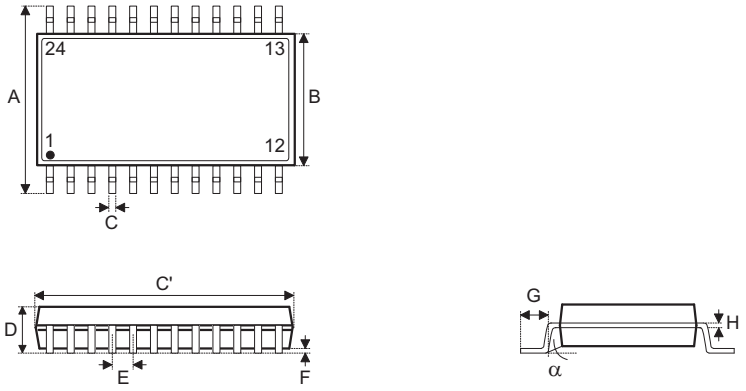
24-pin SOP (300mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A		0.406 BSC	
B		0.295 BSC	
C	0.012	—	0.020
C′		0.606 BSC	
D	—	—	0.104
E		0.050 BSC	
F	0.004	—	0.012
G	0.016	—	0.050
H	0.008	—	0.013
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A		10.30 BSC	
B		7.50 BSC	
C	0.31	—	0.51
C′		15.40 BSC	
D	—	—	2.65
E		1.27 BSC	
F	0.10	—	0.30
G	0.40	—	1.27
H	0.20	—	0.33
α	0°	—	8°

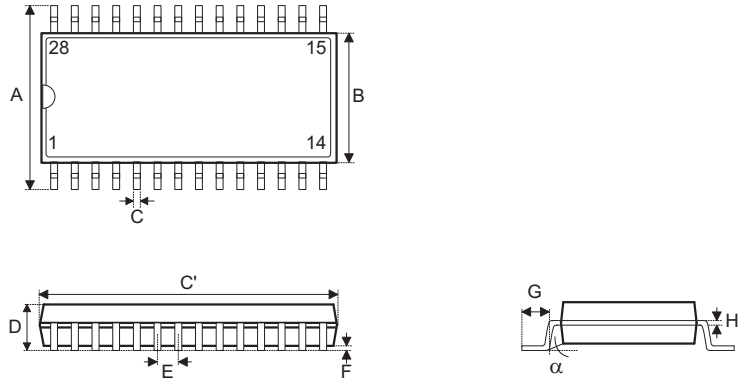
24-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A		0.236 BSC	
B		0.154 BSC	
C	0.008	—	0.012
C′		0.341 BSC	
D	—	—	0.069
E		0.025 BSC	
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A		6.00 BSC	
B		3.90 BSC	
C	0.20	—	0.30
C′		8.66 BSC	
D	—	—	1.75
E		0.635 BSC	
F	0.10	—	0.25
G	0.41	—	1.27
H	0.10	—	0.25
α	0°	—	8°

28-pin SOP (300mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A		0.406 BSC	
B		0.295 BSC	
C	0.012	—	0.020
C′		0.705 BSC	
D	—	—	0.104
E		0.050 BSC	
F	0.004	—	0.012
G	0.016	—	0.050
H	0.008	—	0.013
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A		10.30 BSC	
B		7.50 BSC	
C	0.31	—	0.51
C′		17.90 BSC	
D	—	—	2.65
E		1.27 BSC	
F	0.10	—	0.30
G	0.40	—	1.27
H	0.20	—	0.33
α	0°	—	8°