HT86Axx/HT86ARxx
A/D Type Voice 8-Bit MCU

Technical Document

- Application Note
  - HA0075E MCU Reset and Oscillator Circuits Application Note

Features

- HT86Axx Operating voltage: 2.0V~5.5V
  HT86ARxx Operating voltage: 2.2V~5.5V
- System clock: 4MHz~8MHz
- Crystal and RC system oscillator
- 40 I/O pins
- 8K×16-bit Program Memory
- 384×8-bit Data Memory
- 768/1536K-bit voice ROM size
- 36/72 sec voice length
- External interrupt input
- 12-bit high quality voltage type D/A output
- 4 channels 12-bit resolution A/D Converter
- SPI series protocol interface
- Integrated 1W power amplifier to drive 8Ω Speaker
- Four 8-bit programmable Timer with overflow interrupt and 7-stage prescaler
- One optional 32768Hz crystal oscillator for RTC time base
- 8-bit counter with 3-bit prescaler
- Watchdog timer function
- 8-level subroutine nesting
- Low voltage reset and low voltage detect function
- Integrated voice ROM with various capacities
- Power-down function and wake-up feature reduce power consumption
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD}= 5V
- 63 powerful instructions
- 44-pin LQFP and 64-pin LQFP packages

General Description

The devices are Voice type series 8-bit high performance microcontrollers which include a voice synthesiser and tone generator. They are specifically designed for applications which require multiple I/Os and sound effects, such as voice and melodies. They can provide various sampling rates and beats, tone levels, tempos for speech synthesisers and melody generators. They also include an integrated high quality, voltage type DAC output, an integrated series protocol interface, multi-channel A/D Converter and integrated power amplifier for speaker driving. The external interrupt can be triggered with falling edges or both falling and rising edges.

The devices are fully supported by the Holtek range of fully functional development and programming tools, providing a means for fast and efficient product development cycles.

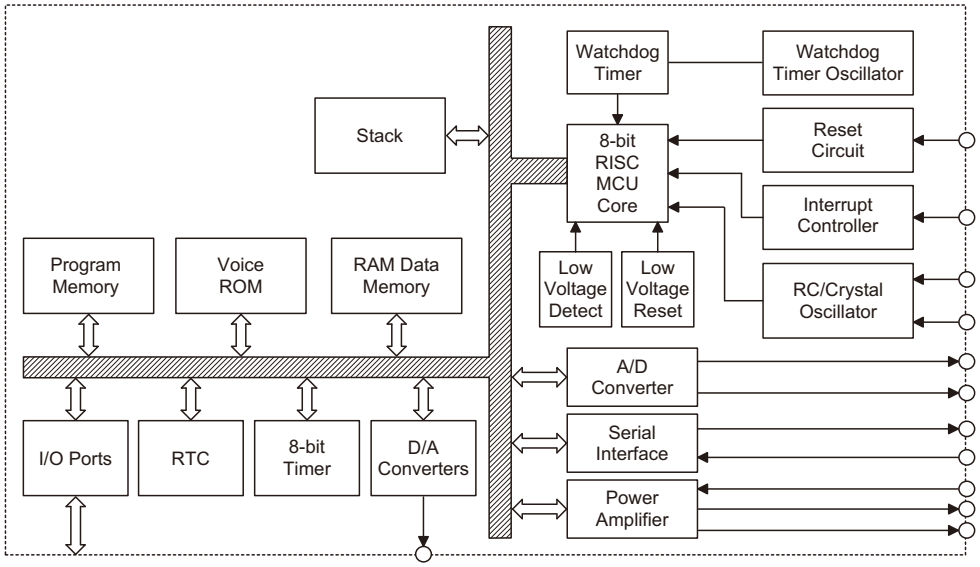
Selection Table

The devices include a comprehensive range of features, with most features common to all devices. The main features distinguishing them are Voice ROM capacity and power supply voltage. The functional differences between the devices are shown in the following table. Devices which include an "A" in their part number are Mask type while devices which contain an "R" in their part number are OTP type.

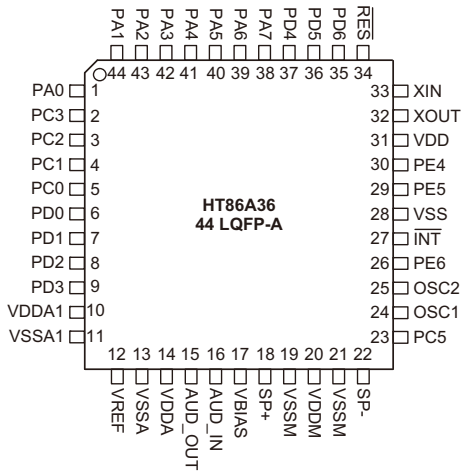
Part No.	VDD	Program Memory	Data Memory	Voice ROM	Voice Capacity	I/O	Timer	D/A	Stack	Package Types
HT86A36	2.0V~5.5V	8K×16	384×8	96×8	36sec	40	8-bit×4	12-bit×1	8	44/64LQFP
HT86A72				192×8	72sec					
HT86AR72	2.2V~5.5V									

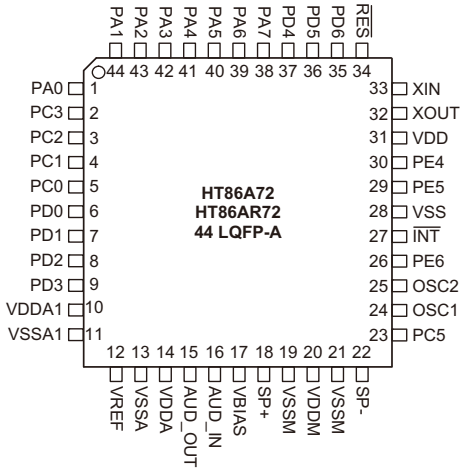
Note: Voice length is estimated by 21K-bit data rate

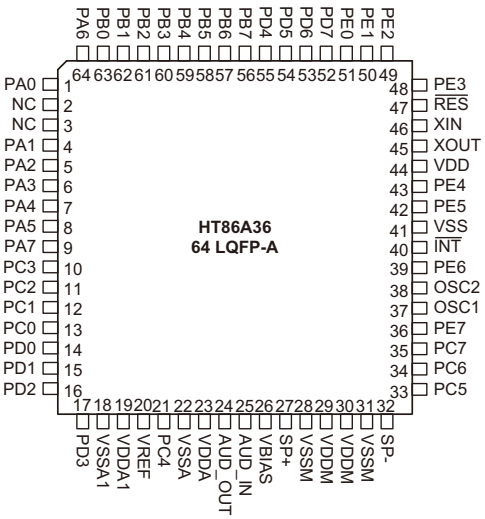
Block Diagram

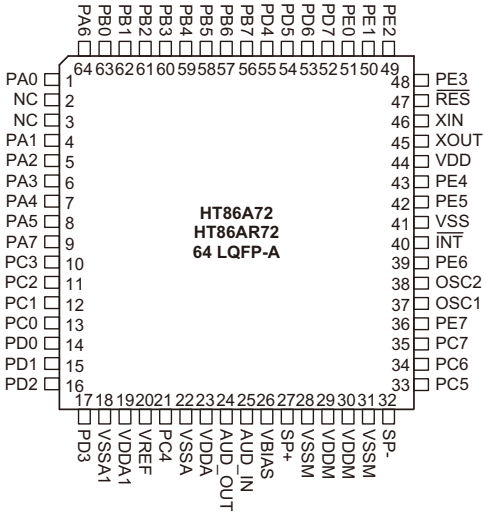


Pin Assignment



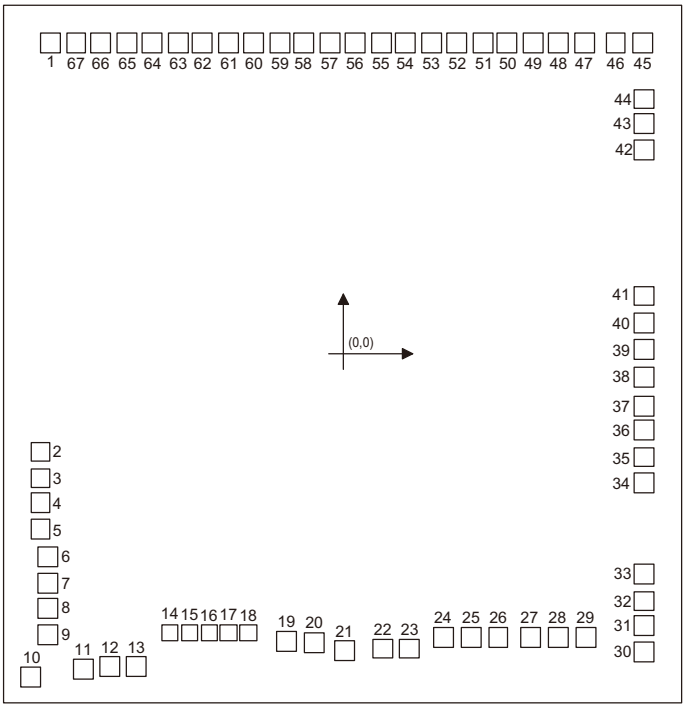






Pad Assignment

HT86A36



Chip Size: 2655 × 2725 (μm)^2

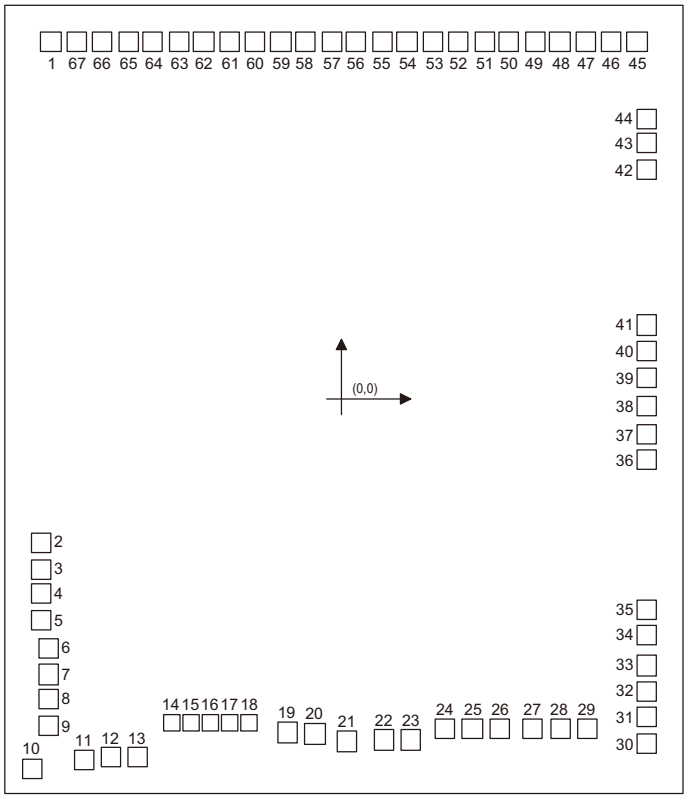
* The IC substrate should be connected to VSS in the PCB layout artwork.

Pad Coordinates

Unit: μm

Pad No.	Pad Name	X	Y	Pad No.	Pad Name	X	Y
1	PA0	–1141.487	1211.389	35	OSC2	1175.250	–400.731
2	PC3	–1177.862	–380.061	36	PE6	1175.250	–296.331
3	PC2	–1177.862	–483.061	37	$\overline{\text{INT}}$	1175.250	–201.331
4	PC1	–1177.862	–578.061	38	VSS	1174.567	–90.426
5	PC0	–1177.862	–681.061	39	PE5	1174.567	19.807
6	PD0	–1150.059	–789.182	40	PE4	1174.567	122.807
7	PD1	–1150.059	–892.182	41	VDD	1174.567	226.177
8	PD2	–1150.059	–987.182	42	XOUT	1174.450	794.097
9	PD3	–1150.059	–1090.182	43	XIN	1174.450	897.097
10	VSSA1	–1215.319	–1257.457	44	$\overline{\text{RES}}$	1174.450	992.097
11	VDDA1	–1008.914	–1225.000	45	PE3	1166.013	1211.389
12	VREF	–906.508	–1213.900	46	PE2	1063.013	1211.389
13	PC4	–803.508	–1213.900	47	PE1	941.513	1211.389
14	NC	–672.868	–1082.529	48	PE0	838.513	1211.389
15	NC	–596.868	–1082.529	49	PD7	743.513	1211.389
16	NC	–520.868	–1082.529	50	PD6	640.513	1211.389
17	NC	–444.868	–1082.529	51	PD5	545.513	1211.389
18	NC	–368.868	–1082.529	52	PD4	442.513	1211.389
19	VSSA	–220.317	–1116.909	53	PB7	347.513	1211.389
20	VDDA	–113.217	–1121.909	54	PB6	244.513	1211.389
21	AUD_OUT	8.383	–1151.309	55	PB5	149.513	1211.389
22	AUD_IN	156.311	–1146.077	56	PB4	46.513	1211.389
23	VBIAS	289.311	–1146.077	57	PB3	–48.487	1211.389
24	SP+	391.136	–1101.247	58	PB2	–151.487	1211.389
25	VSSM	498.085	–1101.247	59	PB1	–246.487	1211.389
26	VDDM	605.036	–1101.247	60	PB0	–349.487	1211.389
27	VDDM	733.086	–1101.247	61	PA7	–444.487	1211.389
28	VSSM	840.037	–1101.247	62	PA6	–547.487	1211.389
29	SP-	946.986	–1101.247	63	PA5	–642.487	1211.389
30	PC5	1174.567	–1158.350	64	PA4	–745.487	1211.389
31	PC6	1174.567	–1055.350	65	PA3	–840.487	1211.389
32	PC7	1174.567	–960.350	66	PA2	–943.487	1211.389
33	PE7	1174.567	–857.350	67	PA1	–1038.487	1211.389
34	OSC1	1175.250	–499.731				

HT86A72



Chip Size: 2651 × 3078 (μm)^2

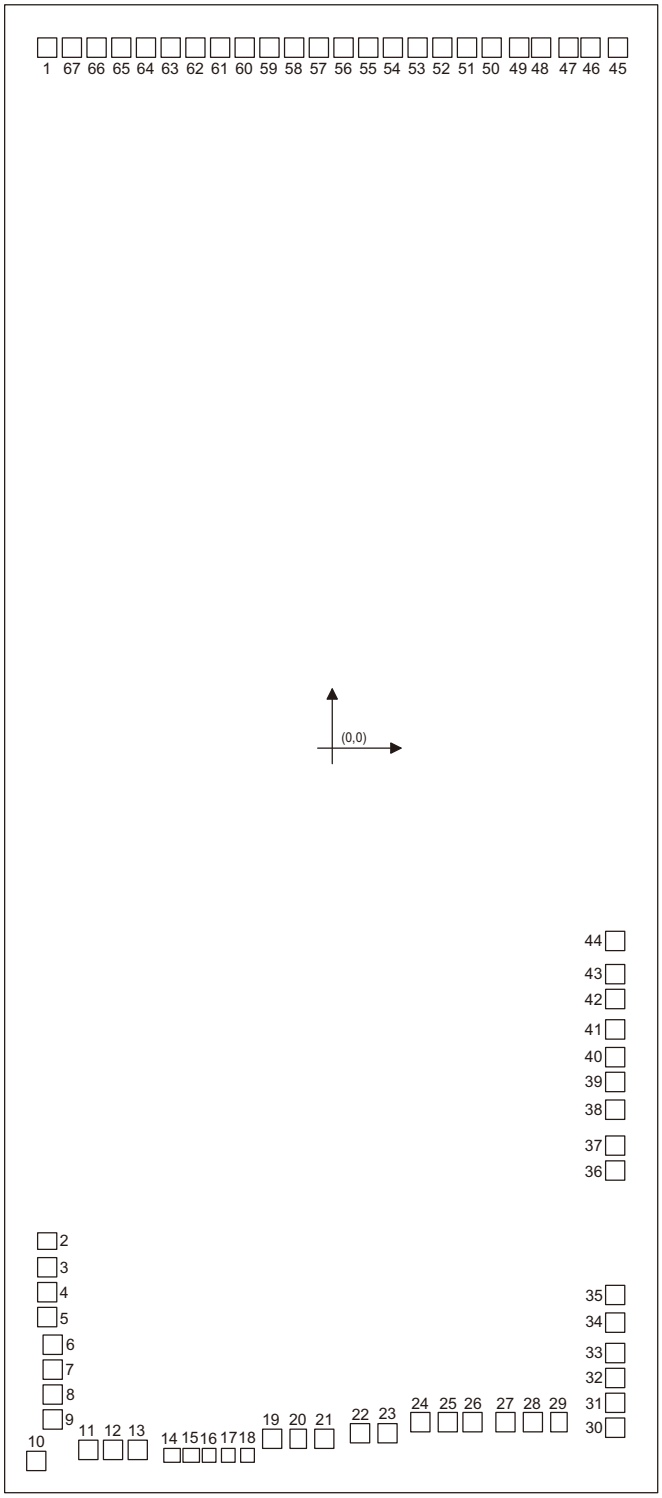
* The IC substrate should be connected to VSS in the PCB layout artwork.

Pad Coordinates

Unit: μm

Pad No.	Pad Name	X	Y	Pad No.	Pad Name	X	Y
1	PA0	−1139.487	1389.643	35	OSC2	1177.250	−817.231
2	PC3	−1175.862	−556.476	36	PE6	1177.250	−232.827
3	PC2	−1175.862	−659.476	37	$\overline{\text{INT}}$	1177.250	−137.827
4	PC1	−1175.862	−754.476	38	VSS	1176.567	−27.172
5	PC0	−1175.862	−857.476	39	PE5	1176.567	82.881
6	PD0	−1148.059	−965.628	40	PE4	1176.567	185.881
7	PD1	−1148.059	1068.682	41	VDD	1176.567	289.431
8	PD2	−1148.059	−1163.682	42	XOUT	1176.450	892.351
9	PD3	−1148.059	−1266.682	43	XIN	1176.450	995.351
10	VSSA1	−1213.319	−1433.997	44	$\overline{\text{RES}}$	1176.450	1090.351
11	VDDA1	−1009.034	−1401.500	45	PE3	1141.513	1389.643
12	VREF	−906.613	−1390.400	46	PE2	1038.513	1389.643
13	PC4	−803.613	−1390.400	47	PE1	943.513	1389.643
14	NC	−671.507	−1258.234	48	PE0	840.513	1389.643
15	NC	−595.507	−1258.234	49	PD7	745.513	1389.643
16	NC	−519.507	−1258.234	50	PD6	642.513	1389.643
17	NC	−443.507	−1258.234	51	PD5	547.513	1389.643
18	NC	−367.507	−1258.234	52	PD4	444.513	1389.643
19	VSSA	−218.317	−1293.409	53	PB7	349.513	1389.643
20	VDDA	−111.217	−1298.409	54	PB6	246.513	1389.643
21	AUD_OUT	10.383	−1327.809	55	PB5	151.513	1389.643
22	AUD_IN	158.311	−1322.577	56	PB4	48.513	1389.643
23	VBIAS	261.311	−1322.577	57	PB3	−46.487	1389.643
24	SP+	393.136	−1277.747	58	PB2	−149.487	1389.643
25	VSSM	500.085	−1277.747	59	PB1	−244.487	1389.643
26	VDDM	607.036	−1277.747	60	PB0	−347.487	1389.643
27	VDDM	735.086	−1277.747	61	PA7	−442.487	1389.643
28	VSSM	842.037	−1277.747	62	PA6	−545.487	1389.643
29	SP-	948.986	−1277.747	63	PA5	−640.487	1389.643
30	PC5	1176.567	−1334.850	64	PA4	−743.487	1389.643
31	PC6	1176.567	−1231.850	65	PA3	−838.487	1389.643
32	PC7	1176.567	−1136.850	66	PA2	−941.487	1389.643
33	PE7	1176.567	−1033.850	67	PA1	−1036.487	1389.643
34	OSC1	1177.250	−916.231				

HT86AR72



Chip Size: 2580 × 5870 (μm)^2

* The IC substrate should be connected to VSS in the PCB layout artwork.

Pad Coordinates

Unit: μm

Pad No.	Pad Name	X	Y	Pad No.	Pad Name	X	Y
1	PA0	−1138.054	2785.633	35	OSC2	1138.850	−2171.224
2	PC3	−1138.629	−1952.576	36	PE6	1138.668	−1669.854
3	PC2	−1138.629	−2055.576	37	$\overline{\text{INT}}$	1138.668	−1574.854
4	PC1	−1138.629	−2150.576	38	VSS	1138.985	−1432.154
5	PC0	−1138.629	−2253.576	39	PE5	1138.985	−1321.921
6	PD0	−1110.826	−2361.132	40	PE4	1138.985	−1218.921
7	PD1	−1110.826	−2464.132	41	VDD	1138.900	−1110.526
8	PD2	−1110.826	−2559.132	42	XOUT	1138.908	−994.167
9	PD3	−1110.826	−2662.132	43	XIN	1138.908	−891.167
10	VSSA1	−1176.086	−2827.757	44	$\overline{\text{RES}}$	1138.906	−761.007
11	VDDA1	−968.465	−2785.850	45	PE3	1142.946	2785.633
12	VREF	−873.465	−2785.850	46	PE2	1039.946	2785.633
13	PC4	−770.465	−2785.850	47	PE1	944.946	2785.633
14	NC	−638.243	−2804.019	48	PE0	841.946	2785.633
15	NC	−562.243	−2804.019	49	PD7	746.946	2785.633
16	NC	−486.243	−2804.019	50	PD6	643.946	2785.633
17	NC	−410.243	−2804.019	51	PD5	548.946	2785.633
18	NC	−334.243	−2804.019	52	PD4	445.946	2785.633
19	VSSA	−232.480	−2738.114	53	PB7	350.946	2785.633
20	VDDA	−131.115	−2739.664	54	PB6	247.946	2785.633
21	AUD_OUT	−28.525	−2739.664	55	PB5	152.946	2785.633
22	AUD_IN	120.644	−2718.027	56	PB4	49.946	2785.633
23	VBIAS	223.644	−2718.027	57	PB3	−45.054	2785.633
24	SP+	355.469	−2673.197	58	PB2	−148.054	2785.633
25	VSSM	462.418	−2673.197	59	PB1	−243.054	2785.633
26	VDDM	569.369	−2673.197	60	PB0	−346.054	2785.633
27	VDDM	697.419	−2673.197	61	PA7	−441.054	2785.633
28	VSSM	804.370	−2673.197	62	PA6	−544.054	2785.633
29	SP-	911.319	−2673.197	63	PA5	−639.054	2785.633
30	PC5	1138.900	−2696.357	64	PA4	−742.054	2785.633
31	PC6	1138.900	−2593.357	65	PA3	−837.054	2785.633
32	PC7	1138.900	−2498.357	66	PA2	−940.054	2785.633
33	PE7	1138.900	−2395.357	67	PA1	−1035.054	2785.633
34	OSC1	1138.900	−2274.826				

Pin Description

Pin Name	I/O	Options	Description
PA0~PA7	I/O	Wake-up, Pull-high	Bidirectional 8-bit I/O port. Each 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 a pull-high resistor.
PB0~PB7	I/O	Pull-high	Bidirectional 8-bit I/O port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on the port have a pull-high resistor.
PC0/AD0 PC1/AD1 PC2/AD2 PC3/AD3 PC4~PC7	I/O	Pull-high	Bidirectional 8-bit I/O port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on the port have a pull-high resistor. Pins PC0~PC3 are pin-shared with A/D converter input pins AD0~AD3.
PD0/SCS PD1/SCK PD2/SDI PD3/SDO PD4~PD7	I/O	Pull-high	Bidirectional 8-bit I/O port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pin on the port have a pull-high resistor. Pins PD0~PD3 are pin-shared with SPI interface pins SCS, SCK, SDI, SDO.
PE0~PE7	I/O	Pull-high	Bidirectional 8-bit I/O port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. Configuration options determine which pins on the port have a pull-high resistor.
AUD_OUT	O	—	Audio output for driving an external transistor or for driving HT82V739
AUD_IN	I	—	Power amplifier input pin
SP-, SP+		—	Power amplifier output pins
VBIAS	O	—	Voltage bias pin
$\overline{RES}$	I	—	Schmitt trigger reset input. Active low.
$\overline{INT}$	I	Falling Edge Trigger or Falling/Rising Edge Trigger	External interrupt Schmitt trigger input without pull-high resistor. A configuration option determines if the interrupt active edge is a falling edge only or both a falling and rising edge.
OSC1 OSC2	—	Crystal or RC	OSC1, OSC2 are connected to an external RC network or external crystal, determined by configuration option, for the internal system clock. If the RC system clock option is selected, pin OSC2 can be used to measure the system clock at 1/4 frequency.
XIN XOUT	—	Crystal	Connected to an external 32kHz crystal
VREF	I	—	A/D circuit reference voltage
VDD	—	—	Positive digital power supply
VSS	—	—	Negative digital power supply, ground.
VDDA	—	—	Positive DAC circuit power supply
VSSA	—	—	Negative DAC circuit power supply, ground
VDDA1	—	—	Positive A/D circuit power supply
VSSA1	—	—	Negative A/D circuit power supply, ground
VDDM	—	—	Positive Power Amp. power supply
VSSM	—	—	Negative Power Amp. 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 pull-high resistors.

Absolute Maximum Ratings

Supply Voltage ..... V_{SS}+2.0V to V_{SS}+5.5V
Input Voltage ..... V_{SS}–0.3V to V_{DD}+0.3V
I_{OL} Total 150mA
Total Power Dissipation 500mW

Storage Temperature ..... –50°C to 125°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}	HT86Axx Operating Voltage	—	f_{SYS}=4MHz/8MHz	2.0	—	5.5	V
V_{DD}	HT86ARxx Operating Voltage	—	f_{SYS}=4MHz/8MHz	2.2	—	5.5	V
I_{DD1}	Operating Current	3V	No load, f_{SYS}=4MHz, RTC enable, DAC disable	—	—	1.5	mA
		5V		—	—	5.0	mA
		3V	No load, f_{SYS}=8MHz, RTC enable, DAC disable	—	—	3.0	mA
		5V		—	—	7.0	mA
I_{DD2}	Operating Current	3V	No load, f_{SYS}=4MHz, RTC enable, DAC enable	—	—	10	mA
		5V		—	—	18	mA
		3V	No load, f_{SYS}=8MHz, RTC enable, DAC enable	—	—	20	mA
		5V		—	—	36	mA
I_{DD3}	Operating Current	3V	No load, f_{SYS}=4MHz, RTC disable, DAC disable	—	—	1.5	mA
		5V		—	—	5.0	mA
		3V	No load, f_{SYS}=8MHz, RTC disable, DAC disable	—	—	3.0	mA
		5V		—	—	7.0	mA
I_{DD4}	Operating Current	3V	No load, f_{SYS}=4MHz, RTC disable, DAC enable	—	—	10	mA
		5V		—	—	18	mA
		3V	No load, f_{SYS}=8MHz, RTC disable, DAC enable	—	—	20	mA
		5V		—	—	36	mA
I_{STB1}	Standby Current	3V	No load, system HALT, WDT enable, RTC enable	—	—	5	μA
		5V		—	—	10	μA
I_{STB2}	Standby Current	3V	No load, system HALT, WDT disable, RTC enable	—	—	2	μA
		5V		—	—	4	μA
I_{STB3}	Standby Current	3V	No load, system HALT, WDT enable, RTC disable	—	—	4	μA
		5V		—	—	8	μA
I_{STB4}	Standby Current	3V	No load, system HALT, WDT disable, RTC disable	—	—	1.0	μA
		5V		—	—	2.0	μA
V_{IL1}	Input Low Voltage for I/O Ports	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports	—	—	$0.7V_{DD}$	—	V_{DD}	V

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
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	—	LVR 2.1V option	2.0	2.1	2.2	V
V_{LVD1}	Low Voltage Detect	—	LVD 2.2V	1.9	—	2.5	V
V_{LVD2}	Low Voltage Detect	—	LVD 2.3V	2.0	—	2.6	V
V_{LVD3}	Low Voltage Detect	—	LVD 2.4V	2.1	—	2.7	V
V_{LVD4}	Low Voltage Detect	—	LVD 2.5V	2.2	—	2.8	V
I_{OL1}	I/O Port Sink Current (For PA0~PA7, PB0~PB7, PC0~PC7, PD4~PD7, PE0~PE7)	3V	$V_{OL}=0.1V_{DD}$	4	—	—	mA
		5V		10	—	—	mA
I_{OH1}	I/O Port Source Current (For PA0~PA7, PB0~PB7, PC0~PC7, PD4~PD7, PE0~PE7)	3V	$V_{OH}=0.9V_{DD}$	–2	—	—	mA
		5V		–5	—	—	mA
I_{OL2}	I/O Port Sink Current (For PD0~PD3)	3V	$V_{OL}=0.1V_{DD}$	8	—	—	mA
		5V		20	—	—	mA
I_{OH2}	I/O Port Source Current (For PD0~PD3)	3V	$V_{OH}=0.9V_{DD}$	–2	—	—	mA
		5V		–5	—	—	mA
I_{AUD}	AUD Source Current	3V	$V_{OH}=0.9V_{DD}$	–1.5	—	—	mA
		5V		–3	—	—	mA
R_{PH}	Pull-high Resistance	3V	—	20	60	100	kΩ
		5V		10	30	50	kΩ
V_{AD}	A/D Input/Output	—	—	0	—	V_{REF}	V
V_{REF}	ADC Input Reference Voltage Range	3V	$AV_{DD}=3V$	1.3	—	AV_{DD}	V
		5V	$AV_{DD}=5V$	1.5	—	AV_{DD}	V
DNL	ADC Differential Non-Linear	—	$AV_{DD}=5V$, $V_{REF}=AV_{DD}$, $t_{AD}=1\mu s$	—	—	±2.0	LSB
INL	ADC Integral Non-Linear	—	$AV_{DD}=5V$, $V_{REF}=AV_{DD}$, $t_{AD}=1\mu s$	—	±2.5	±4.0	LSB
RESOLU	Resolution	—	—	—	—	12	bits
I_{ADC}	Additional Power Consumption if A/D Converter is Used	3V	No load, $t_{AD}=1\mu s$	—	0.5	1.0	mA
		5V		—	1.5	3.0	mA

A.C. Characteristics

Ta=25°C

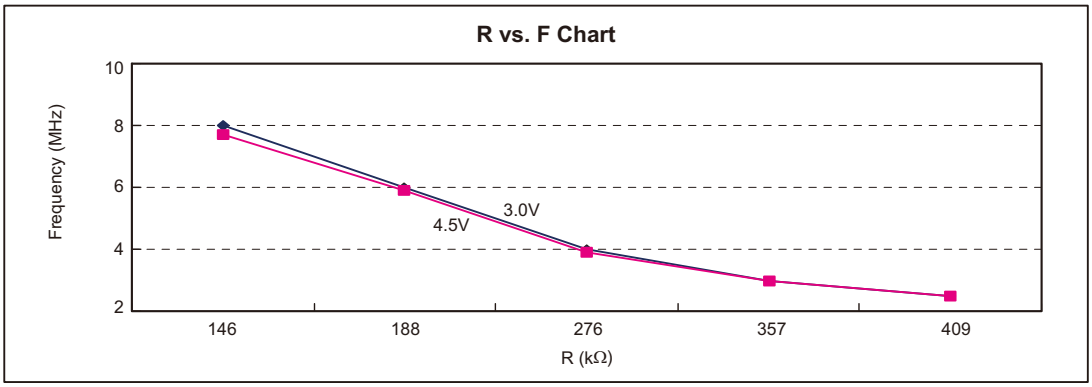
Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock (RC OSC, Crystal OSC)	—	HT86Axx:2.0V~5.5V HT86ARxx:2.2V~5.5V	4	—	8	MHz
t_{WDTOSC}	Watchdog Oscillator Period	3V	—	45	90	180	μs
		5V	—	32	65	130	μs
t_{RES}	External Reset Low Pulse Width	—	—	1	—	—	μs
t_{SST}	System Start-up Timer Period	—	Power-up or Wake-up from HALT	—	1024	—	*t_{SYS}
t_{LVR}	Low Voltage Reset Time	—	—	2	—	—	ms
t_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
t_{AD}	A/D Clock Period	—	—	1.0	—	—	μs
t_{ADC}	A/D Conversion Time	—	—	—	80	—	t_{AD}
t_{ADC}	A/D Sampling Time	—	—	—	32	—	t_{AD}
t_{MAT}	Circumscribe Memory Access Time	—	HT86Axx:2.0V~5.5V HT86ARxx:2.2V~5.5V	—	—	400	ns

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

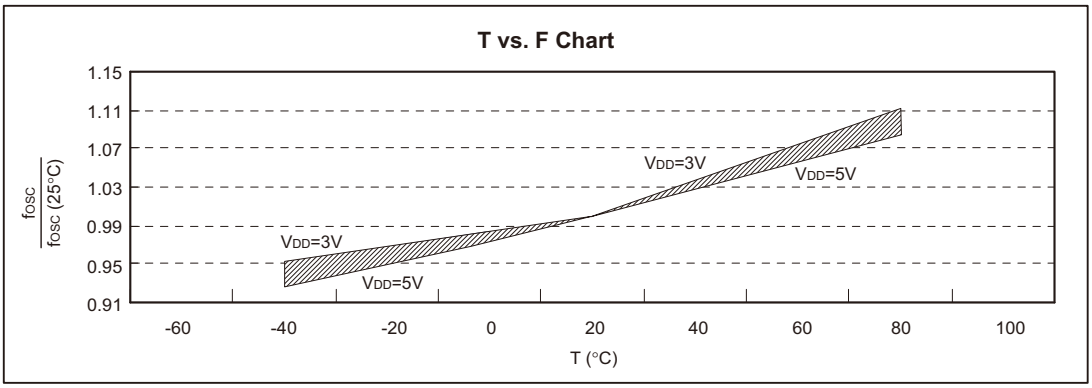
Characteristics Curves

HT86Axx

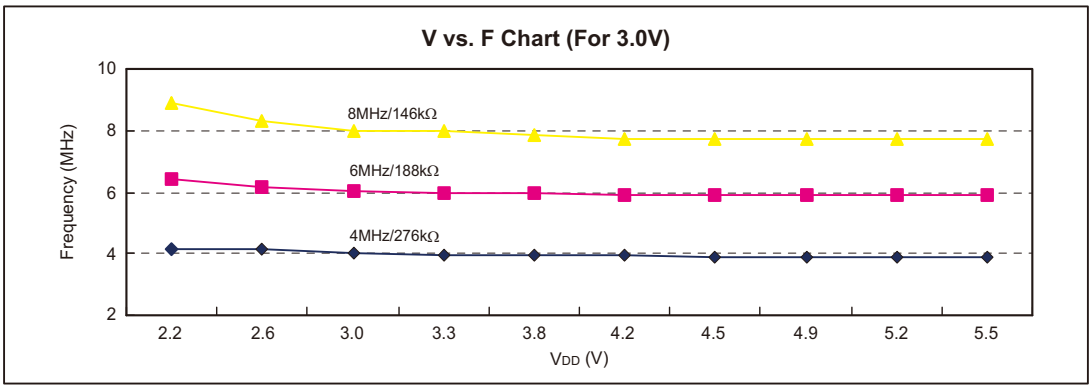
- R vs. F Chart Characteristics Curves



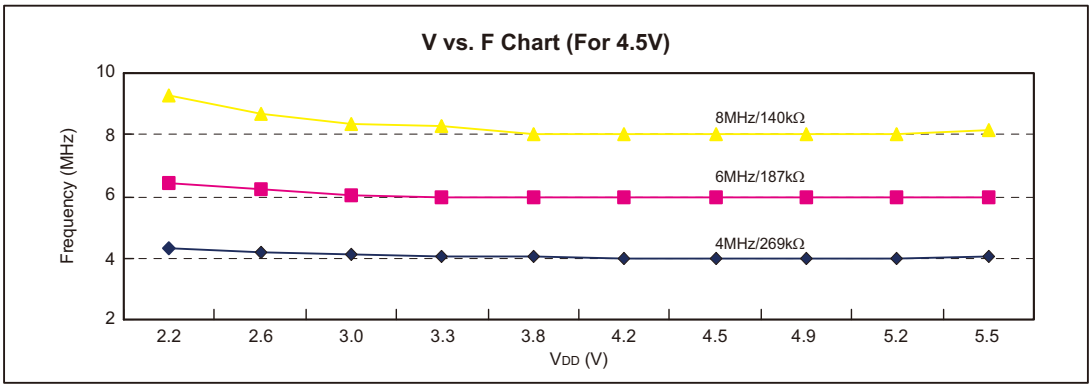
- T vs. F Chart Characteristics Curves



• V vs. F Chart Characteristics Curves – 3.0V

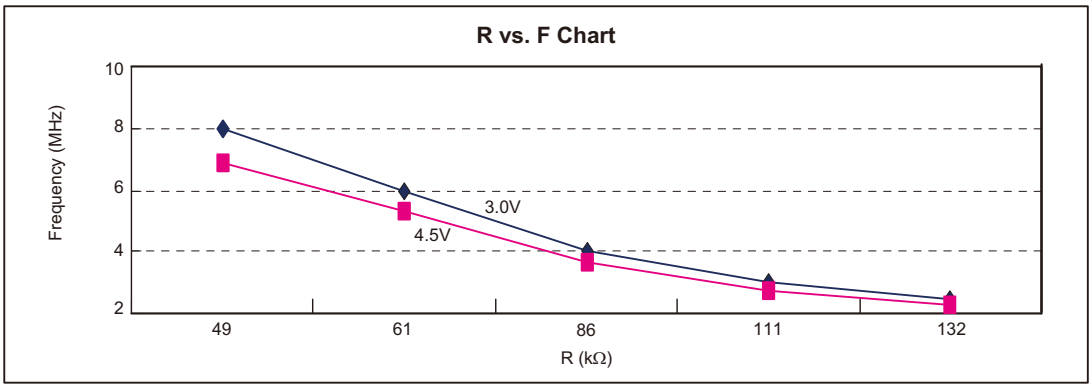


• V vs. F Chart Characteristics Curves – 4.5V

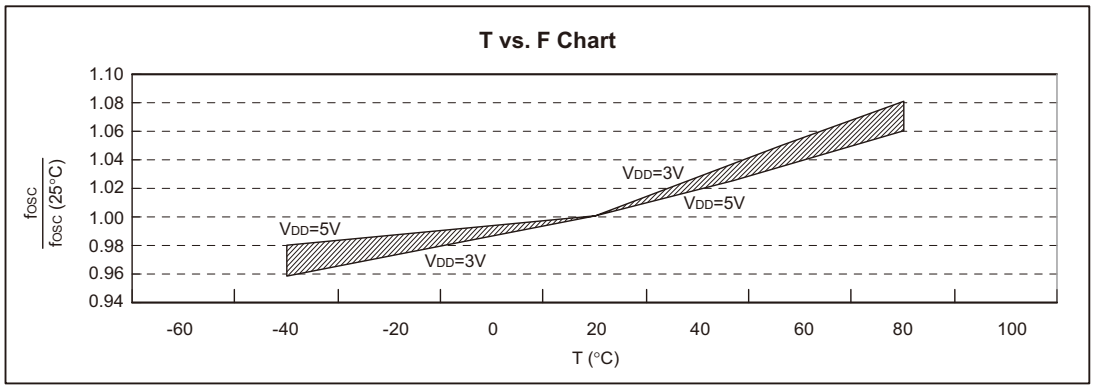


HT86ARxx

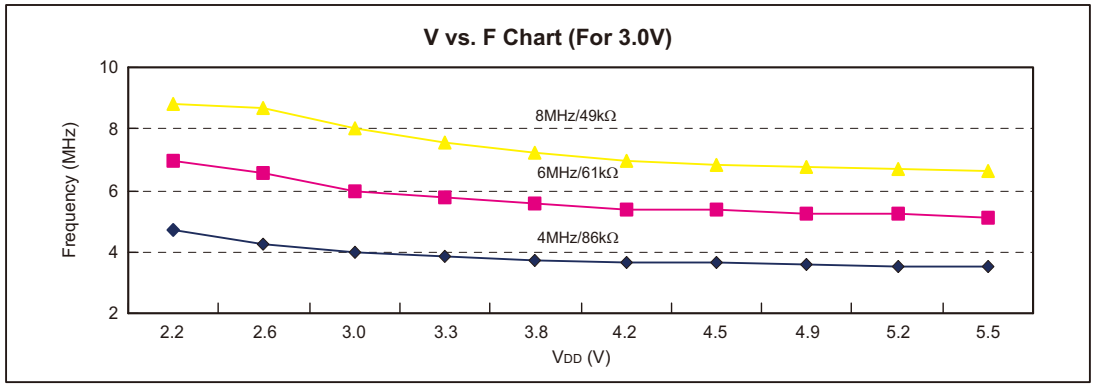
• R vs. F Chart Characteristics Curves



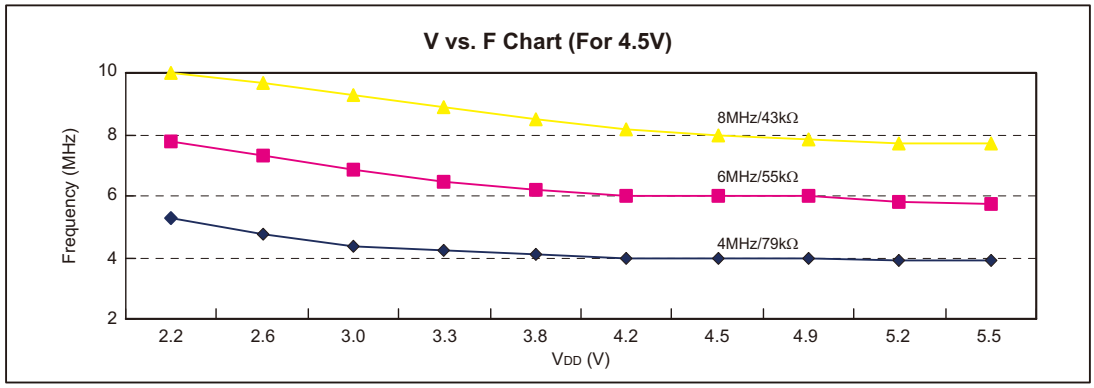
- T vs. F Chart Characteristics Curves



- V vs. F Chart Characteristics Curves – 3.0V



- V vs. F Chart Characteristics Curves – 4.5V



System Architecture

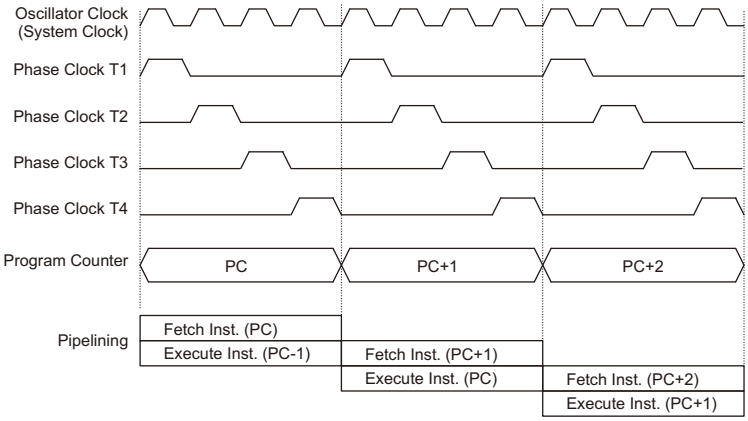
A key factor in the high-performance features of the Holtek range of Voice microcontrollers is attributed to the internal system architecture. The range of devices take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance.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

Clocking and Pipelining

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

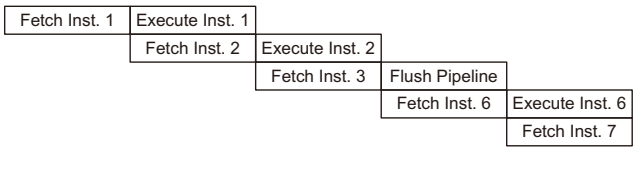
When the RC oscillator is used, OSC2 is freed for use as a T1 phase clock synchronizing pin. This T1 phase clock has a frequency of $f_{SYS}/4$ with a 1:3 high/low duty cycle.

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. Note that the Program Counter width varies with the Program Memory capacity depending upon which device is selected. However, it must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by user.

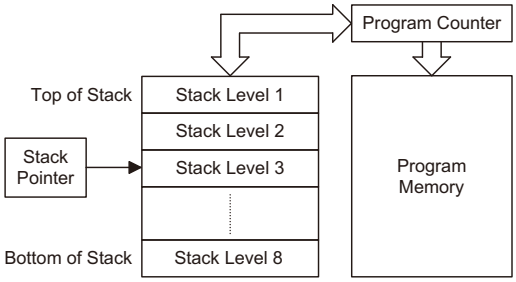
When executing instructions requiring jumps to non-consecutive addresses such as a jump instruction, a subroutine call, interrupt or reset, etc., the microcontroller manages program control by loading the required address into the Program Counter. For conditional skip instructions, once the condition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

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

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

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has 8 levels and is neither part of the data nor part of the program space, and is neither readable nor writable. The activated level is indexed by the Stack Pointer, SP, and is neither readable nor 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.



Mode	Program Counter												
	*12	*11	*10	*9	*8	*7	*6	*5	*4	*3	*2	*1	*0
Initial Reset	0	0	0	0	0	0	0	0	0	0	0	0	0
External Interrupt	0	0	0	0	0	0	0	0	0	0	1	0	0
Timer 0 Overflow	0	0	0	0	0	0	0	0	0	1	0	0	0
Timer 1 Overflow	0	0	0	0	0	0	0	0	0	1	1	0	0
Timer 2 Overflow	0	0	0	0	0	0	0	0	1	0	0	0	0
Timer 3 Overflow	0	0	0	0	0	0	0	0	1	0	1	0	0
A/D Converter Interrupt	0	0	0	0	0	0	0	0	1	1	0	0	0
Skip	Program Counter + 2												
Loading PCL	*12	*11	*10	*9	*8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#12	#11	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Program Counter

Note: *12~*0: Program counter bits
#12~#0: Instruction code bits
S12~S0: Stack register bits
@7~@0: PCL bits

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

Arithmetic and Logic Unit – ALU

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

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

Program Memory

The Program Memory is the location where the user code or program is stored.

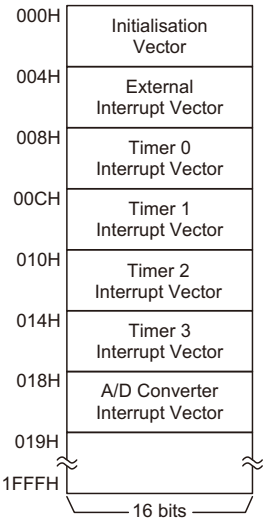
Structure

The program memory stores the program instructions that are to be executed. It also includes data, table and interrupt entries, addressed by the Program Counter along with the table pointer. The program memory size is 8192×16 bits. Certain locations in the program memory are reserved for special usage.

Special Vectors

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

- Location 000H
  This vector is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.
- Location 004H
  This vector is used by the external interrupt. If the external interrupt pin on the device goes low, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full.
- Location 008H
  This internal vector is used by the 8-bit Timer 0. If a overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 00CH
  This internal vector is used by the 8-bit Timer1. If a overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 010H
  This internal vector is used by the 8-bit Timer2. If a overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 014H
  This internal vector is used by the 8-bit Timer3. If a overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 018H
  This internal vector is used by the A/D Converter. If an A/D converter conversion completes, the program will jump to this location and begin execution if the A/D converter interrupt is enabled and the stack is not full.



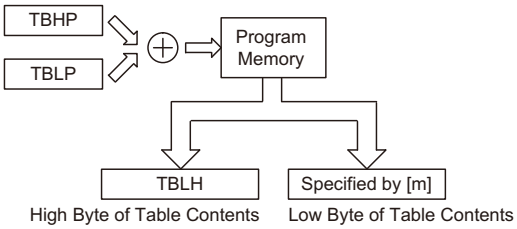
Program Memory Structure

Look-up Table

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

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

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



Look-up Table

Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the devices. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "1F00H" which refers to the start address of the last page within the Program Memory of the microcontroller. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "1F06H" 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 "1F06H" 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 "1F05H" 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         1F00h           ; sets initial address of HT86A72 last page

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

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

Instruction	Table Location												
	*12	*11	*10	*9	*8	*7	*6	*5	*4	*3	*2	*1	*0
TABRDC [m]	P12	P11	P10	P9	P8	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL [m]	1	1	1	1	1	@7	@6	@5	@4	@3	@2	@1	@0

Table Location

Note: *12~*0: Current Program ROM table
@7~@0: Write @7~@0 to TBLP pointer register
P12~P8: Write P12~P8 to TBHP pointer register

Data Memory

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

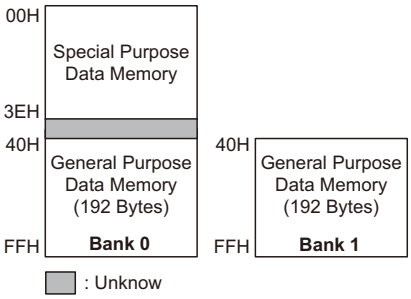
Structure

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

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

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.



RAM Data Memory Structure – Bank 0 and Bank 1

Special Purpose Data Memory

This area of Data Memory, is located in Bank 0, where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers are both readable and writeable but some are protected and are readable only, the details of which are located under the relevant Special Function Register section. Note that for locations that are unused, any read instruction to these addresses will return the value ″00H″. Although the Special Purpose Data Memory registers are located in Bank 0, they will still be accessible even if the Bank Pointer has selected Bank 1.

00H	IAR0
01H	MP0
02H	IAR1
03H	MP1
04H	BP
05H	ACC
06H	PCL
07H	TBLP
08H	TBLH
09H	WDTS
0AH	STATUS
0BH	INTC
0CH	
0DH	TMR0
0EH	TMR0C
0FH	
10H	TMR1
11H	TMR1C
12H	PA
13H	PAC
14H	PB
15H	PBC
16H	PC
17H	PCC
18H	LATCH0H
19H	LATCH0M
1AH	LATCH0L
1BH	LATCH1H
1CH	LATCH1M
1DH	LATCH1L
1EH	INTCH
1FH	TBHP
20H	
21H	TMR2
22H	TMR2C
23H	
24H	TMR3
25H	TMR3C
26H	VOICEC
27H	DAL
28H	DAH
29H	VOL
2AH	LATCHD
2BH	
2CH	
2DH	
2EH	PD
2FH	PDC
30H	ADRL
31H	ADRH
32H	ADCR
33H	ACSR
34H	LVDC
35H	
36H	
37H	
38H	
39H	
3AH	PE
3BH	PEC
3CH	SBCR
3DH	SBDR

: Unknow

Special Purpose Data Memory Structure

Special Function Registers

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

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.

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.

Bank Pointer – BP

The RAM Data Memory is divided into two Banks, known as Bank 0 and Bank 1. With the exception of the BP register, all of the Special Purpose Registers and General Purpose Registers are contained in Bank 0. 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.

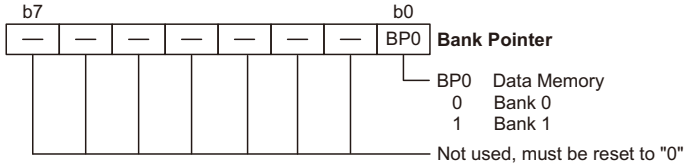
The Data Memory is initialised to Bank 0 after a reset, except for the WDT time-out reset in the Power Down Mode, in which case, the Data Memory bank remains unaffected. It should be noted that Special Function Data Memory is not affected by the bank selection, which means that the Special Function Registers 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 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.



Bank Pointer – BP

Look-up Table Registers – TBLP, TBHP, TBLH

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

Watchdog Timer Register – WDTS

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

Status Register – STATUS

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

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

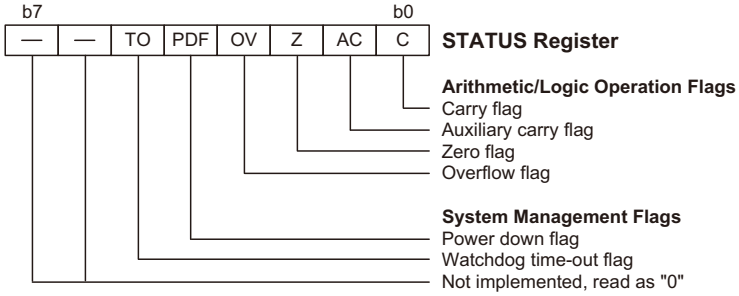
The Z, OV, AC and C flags generally reflect the status of the latest operations.

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

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

Interrupt Control Register – INTC, INTCH

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



Status Register

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

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

Timer Registers

All devices contain four 8-bit Timers whose associated registers are known as TMR0, TMR1, TMR2 and TMR3, which are the locations where the associated timer's 8-bit value is located. Their associated control registers, known as TMR0C, TMR1C, TMR2C and TMR3C, contain the setup information for these timers. Note that all timer registers can be directly written to in order to preload their contents with fixed data to allow different time intervals to be setup.

Input/Output Ports and Control Registers

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

Voice ROM Data Address Latch Counter Registers

These are the LATCH0H/LATCH0M/LATCH0L, LATCH1H/LATCH1M/LATCH1L and the Voice ROM data registers. The voice ROM data address latch counter provides the handshaking between the microcontroller and the voice ROM, where the voice codes are stored. Eight bits of voice ROM data will be addressed by using the 21-bit address latch counter, which is composed of LATCH0H/LATCH0M/LATCH0L or LATCH1H/LATCH1M/LATCH1L. After the 8-bit voice ROM data is addressed, several instruction cycles of at least 4μs at least, will be required to latch the voice ROM data, after which the microcontroller can read the voice data from LATCHD.

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

The device includes a single 12-bit current type DAC function for driving an external 8Ω speaker through an external NPN transistor. The programmer must write the voice data to the DAL/DAH registers.

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

The device contains a 4-channel 12-bit A/D converter. The correct operation of the A/D requires the use of two data registers, a control register and a clock source register. It contain a 12-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.

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 40 bidirectional input/output lines labeled with port names PA, PB, PC, PD, PE, etc. 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. Note that if the pull-high option is

selected, then all I/O pins on that port will be connected to pull-high resistors, individual pins can be selected for pull-high resistor options.

Port A Wake-up

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

I/O Port Control Registers

Each I/O port has its own control register PAC, PBC, PCC, PDC, PEC, etc., 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.

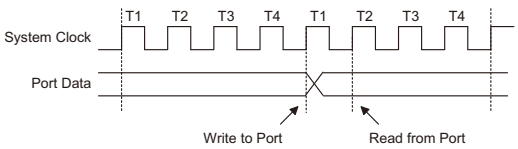
Pin-shared Functions

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.

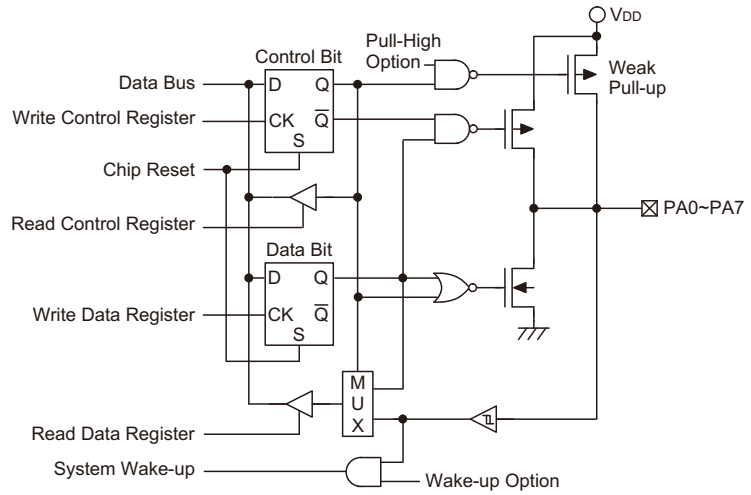
- A/D Inputs
  The device have 4 A/D converter channel inputs. All of these analog inputs are pin-shared with PC0 to PC3. If these pins are to be used as A/D inputs and not as normal I/O pins then the corresponding bits in the A/D Converter Control Register that must be properly set. There are no configuration options associated with the A/D function. If used as I/O pins, then full pull-high resistor selections remain, however if used as A/D inputs then any pull-high resistor selections associated with these pins will be automatically disconnected.
- Serial Interface
  The Serial Interface are pin-shared with PD0 to PD3. If there are to be used as Serial Interface inputs and not as normal I/O pins then the corresponding bits in the Serial Interface Control Register that must be properly set. There are a configuration option associated with the Serial Interface function. If used as I/O pins, then full pull-high resistor selections remain, however if used as the Serial Interface inputs then any pull-high resistor selections associated with these pins will be automatically disconnected.
- I/O Pin Structures
  The following diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins. Note also that the specified pins refer to the largest device package, therefore not all pins specified will exist on all devices.

Programming Considerations

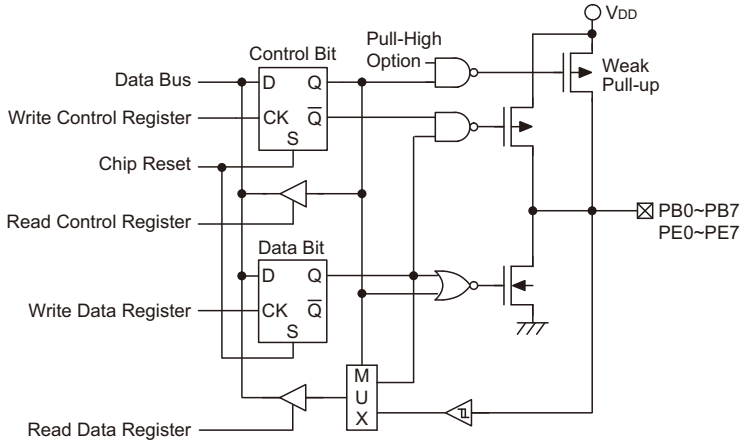
Within the user program, one of the first things to consider is port initialization. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high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



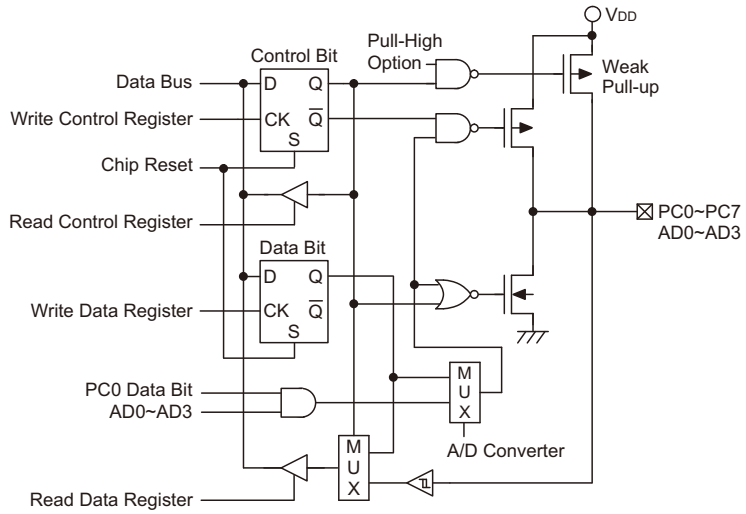
Read/Write Timing



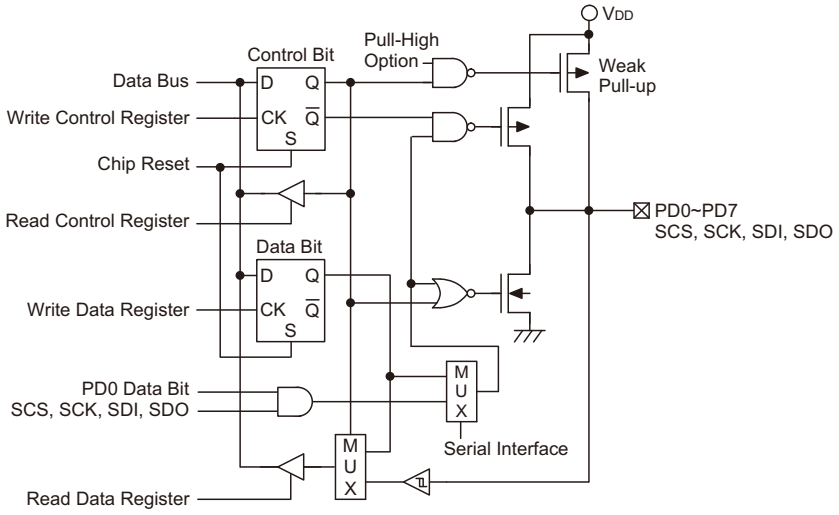
PA Input/Output Port



PB/PE Input/Output Ports



PC Input/Output Port



PD Input/Output Port

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.

Timers

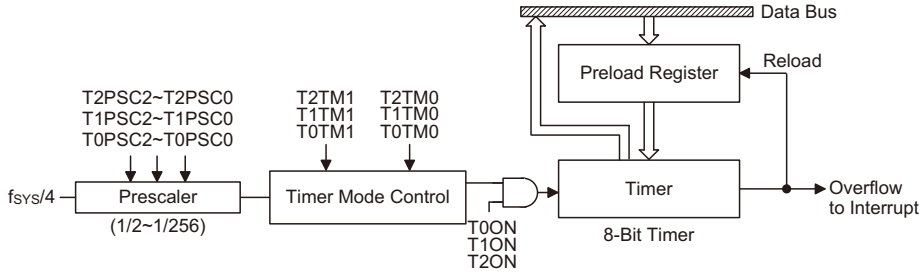
The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. The devices in the Voice Type MCU series contain four count up timers of 8-bit capacity. Each timer has only one operating mode, which is to act as a general timer. The provision of an internal prescaler to the clock circuitry of the timers gives added range to the timer.

Configuring the Timer Input Clock Source

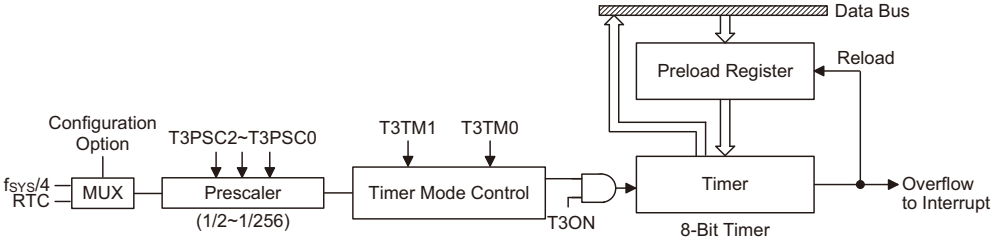
The clock source of Timer0/Timer1/Tmier2 is the system clock divided by four and the clock source of Timer3 is the system clock divided by four or the RTC clock which is derived from a external 32kHz crystal. A configuration option determines which clock is selected. If the RTC clock is selected then note that it will continue to run when the device is powered down using the HALT instruction. The 8-bit timer clock source is also first divided by a prescaler, the division ratio of which is conditioned by the three lower bits of the associated timer control register.

Timer Registers – TMR0, TMR1, TMR2, TMR3

The timer registers are special function registers located in the special purpose Data Memory and is the place where the actual timer value is stored. The value in the timer registers increases by one each time an internal clock pulse is received. The timer will count from the initial value loaded by the preload register to the full count of FFH at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting.



8-bit Timer Structure – TMR0, TMR1, TMR2



8-bit Timer Structure – TMR3

Note that to achieve a maximum full range count of FFH for the 8-bit timer, the preload registers must first be cleared to all zeros. It should be noted that after power-on, the preload registers will be in an unknown condition. Note that if the Timer Counters are in an OFF condition and data is written to their preload registers, this data will be immediately written into the actual counter. However, if the counter is enabled and counting, any new data written into the preload data register during this period will remain in the preload register and will only be written into the actual counter the next time an overflow occurs. Note also that when the timer registers are read, the timer clock will be blocked to avoid errors, however, as this may result in certain timing errors, programmers must take this into account.

Timer Control Registers – TMR0C, TMR1C, TMR2C, TMR3C

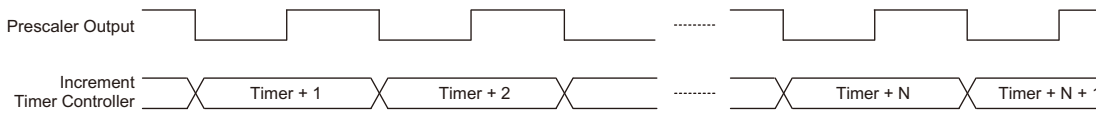
The timers are setup using their respective control register. These registers are known as TMR0C, TMR1C, TMR2C and TMR3C. It is the timer control register together with its corresponding timer registers that control the full operation of the Timer. Before the timers can be used, it is essential that the appropriate timer control register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialization.

Bits 7 and 6 of the Timer Control Register, which are known as the bit pair TM1/TM0, must be set to 10 respectively to ensure correct Timer operation. The timer-on bit, which is bit 4 of the Timer Control Register and known as TON, provides the basic on/off control of the respective timer. setting the bit high allows the timer to run, clearing the bit stops the timer. Bits 0~2 of the Timer Control Register determine the division ratio of the input clock prescaler.

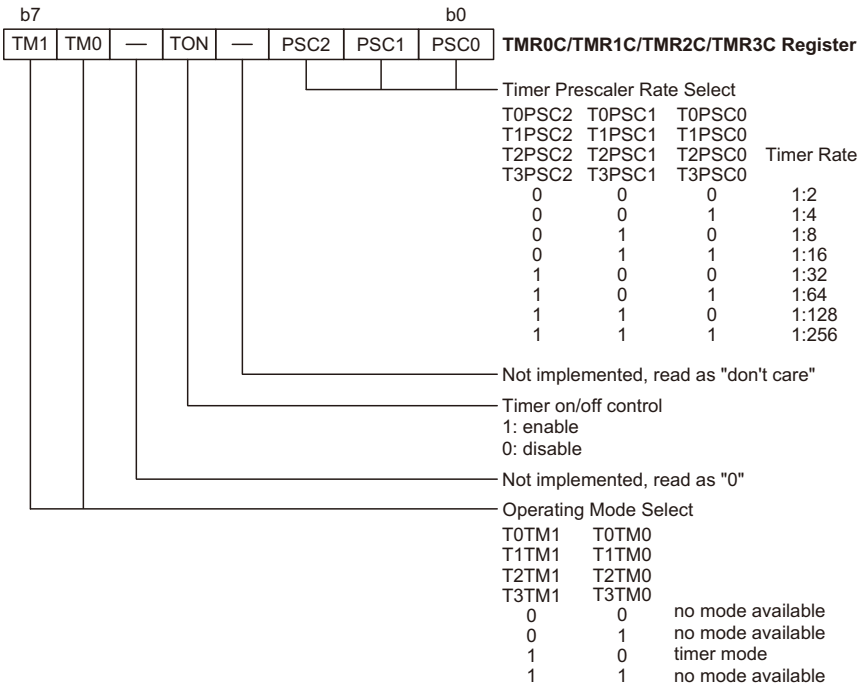
Configuring the Timer

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

The Timer clock source is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits, which are bits 0~2 in the Timer Control Register. After the other bits in the Timer Control Register have been setup, the enable bit, which is bit 4 of the Timer Control Register, can be set high to enable the Timer to run. Each time an internal clock cycle occurs, the Timer increments by one. When it is full and overflows, an interrupt signal is generated and the Timer will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer Interrupt Enable bit in the Interrupt Control Register, INTC, is reset to zero.



Timer Mode Timing Diagram



Timer Control Register – All Devices

Prescaler

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

Programming Considerations

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

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

Timer Program Example

The following example program section is based on the devices, which contain a single internal 8-bit timer. Programming the timer for other devices is conducted in a very similar way. The program shows how the timer registers are setup along with how the interrupts are enabled and managed. Also note how the timer is turned on by setting bit 4 of the respective timer control register. The timer can be turned off in a similar way by clearing the same bit. This example program sets the timer to be in the timer mode which uses the internal system clock as their clock source.

```
include HT86A72.inc
jmp begin
:
org 04h             ; external interrupt vectors
reti
org 08h
reti
org 0Ch
reti
org 10h             ; timer 2 interrupt vector
jmp tmr2int         ; jump here when timer 2 overflows
org 14h
reti
org 18h
reti
:
; internal timer 2 interrupt routine
tmr2int:
:
; timer 2 main program placed here
:
reti
:
begin:
; setup timer 2 registers
mov a,09bh          ; setup timer 2
mov tmr2,a
mov a,0097h         ; setup timer 2
mov tmr2c,a         ; setup timer 2
; setup interrupt register
mov a,01h           ; enable master interrupt
mov intc,a
mov a,01h           ; enable timer 2 interrupt
mov intch,a
:
```

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs.

Interrupt Register

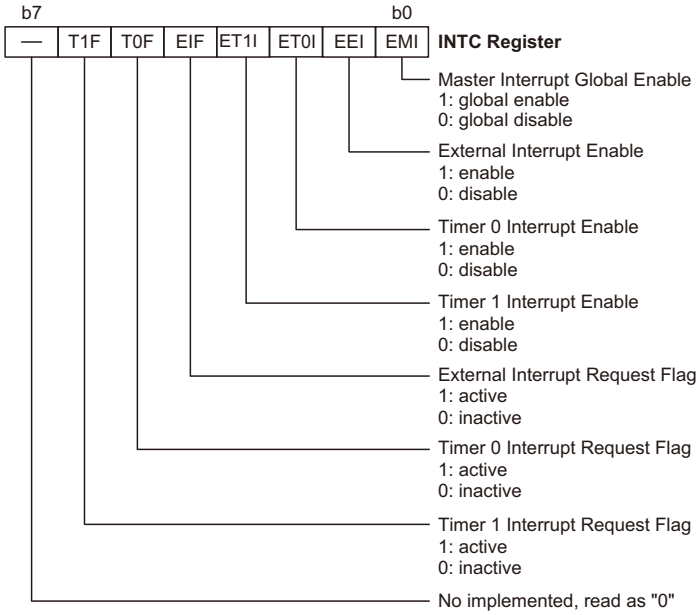
Overall interrupt control, which means interrupt enabling and flag setting, is controlled using two registers, known as INTC and INTCH, which are located in the Data Memory. By controlling the appropriate enable bits in these registers each individual interrupt can be enabled or disabled. Also when an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

Interrupt Operation

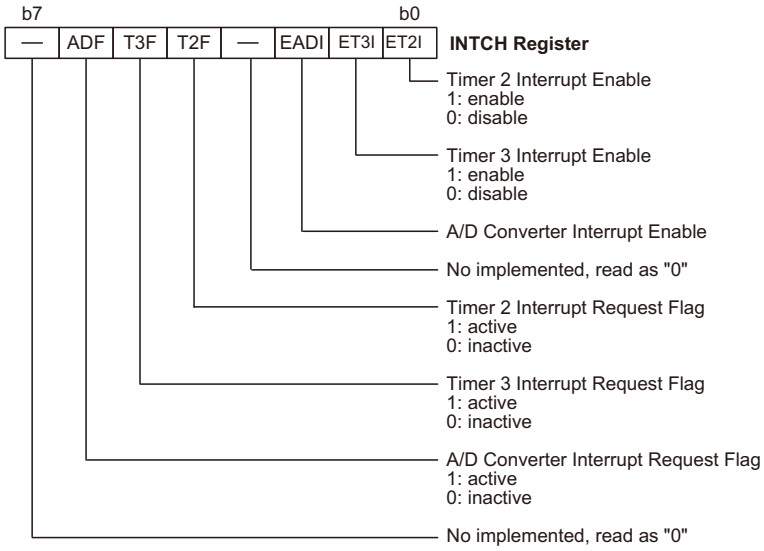
A timer overflow or the external interrupt line being pulled low will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, the Program Counter, which stores the address of the next instruction to be executed, 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 take program execution to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI statement, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

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

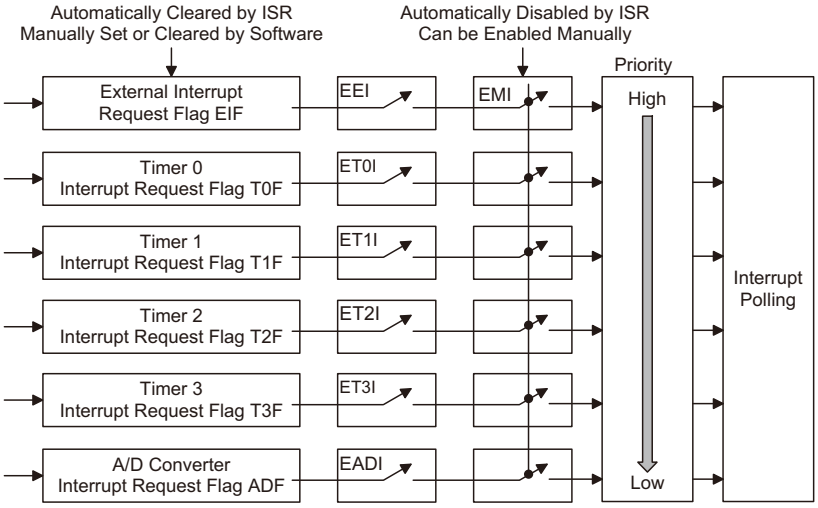
Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately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



Interrupt Control Register



INTCH 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 accompanying table shows the priority that is applied.

Interrupt Source	Interrupt Vector	Priority
External Interrupt	04H	1
Timer 0 Overflow	08H	2
Timer 1 Overflow	0CH	3
Timer 2 Overflow	10H	4
Timer 3 Overflow	14H	5
A/D Converter Overflow	18H	6

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

External Interrupt

Each device contains a single external interrupt function controlled by the external pin, $\overline{INT}$. For an external interrupt to occur, the corresponding external interrupt enable bit must be first set. This is bit 1 of the INTC register and known as EEI. An external interrupt is triggered by an external edge transition on the external interrupt pin $\overline{INT}$, after which the related interrupt request flag, EIF, which is bit 4 of INTC, will be set. A configuration option exists for the external interrupt pin to determine the type of external edge transition which will trigger an external interrupt. There are two options available, a low going edge or both high and low going edges. When the master interrupt and external interrupt bits are enabled, the stack is not full and an active edge transition, as setup in the configuration options, occurs on the $\overline{INT}$ pin, a subroutine call to the corresponding external interrupt vector, which is located at 04H, will occur. After entering the interrupt execution routine, the corresponding interrupt request flag, EIF, will be reset and the EMI bit will be cleared to disable other interrupts.

Timer Interrupt

For a timer generated interrupt to occur, the corresponding timer interrupt enable bit must be first set. Each device contains four 8-bit timers whose corresponding interrupt enable bits are known as ET0I, ET1I, ET2I and ET3I and are located in the INTC and INTCH registers.

Each timer also has a corresponding timer interrupt request flag, which are known as T0F, T1F, T2F and T3F, also located in the INTC and INTCH registers. When the master interrupt and corresponding timer interrupt enable bits are enabled, the stack is not full, and when the corresponding timer overflows a subroutine call to the corresponding timer interrupt vector will occur. The corresponding Program Memory vector locations for Timer 0, Timer1, Timer 2 and Timer 3 are 08H, 0CH, 10H and 14H. After entering the interrupt execution routine, the corresponding interrupt request flags, T0F, T1F, T2F or T3F will be reset and the EMI bit will be cleared to disable other interrupts.

A/D Converter Interrupt

The internal A/D Converter interrupt is initialised by setting the A/D interrupt request flag (ADF:bit6 of INTCH). When the interrupt is enabled, and the stack is not full and the ADF bit is set, a subroutine call to location ″18H″ will occur. The related interrupt request flag, ADF, will be reset and the EMI bit cleared to disable further interrupts.

Programming Considerations

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

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

All of these interrupts have the capability of waking up the MCU 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.

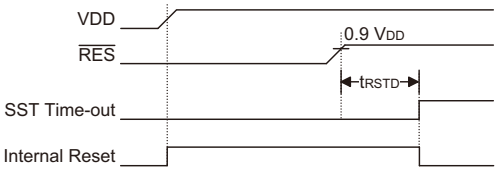
In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is running. One example of this is where after power has been applied and the microcontroller is already running, the $\overline{\text{RES}}$ line is forcefully 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

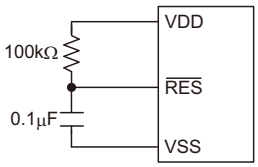
There are five ways in which a microcontroller reset can occur, through events occurring both internally and externally:

- 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.



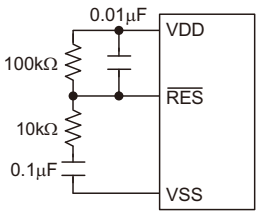
Power-On Reset Timing Chart

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

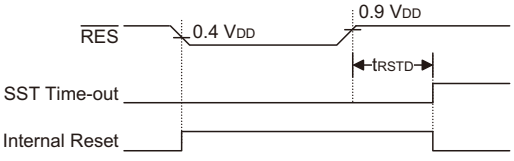
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Enhanced Reset Circuit

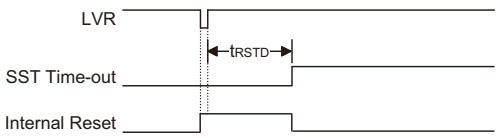
More information regarding external reset circuits is located in Application Note HA0075E on the Holtek website.

- $\overline{\text{RES}}$ Pin Reset
  This type of reset occurs when the microcontroller is already running and the $\overline{\text{RES}}$ 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.



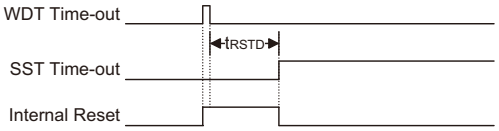
$\overline{\text{RES}}$ Reset Timing Chart

- 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.



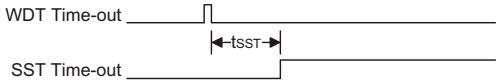
Low Voltage Reset Timing Chart

- 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	All Timer will be turned off
Prescaler	The Timer Prescaler will be cleared
Input/Output Ports	I/O ports will be setup as inputs
Stack Pointer	Stack Pointer will point to the top of the stack

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

Register	Reset (Power-on)	WDT Time-out (Normal Operation)	$\overline{RES}$ Reset (Normal Operation)	$\overline{RES}$ Reset (HALT)	WDT Time-out from HALT
MP0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
BP	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
WDTS	0000 0111	0000 0111	0000 0111	0000 0111	uuuu uuuu
STATUS	--00 xxxx	--1u uuuu	--uu uuuu	--01 uuuu	--11 uuuu
INTC	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TMR0C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
TMR1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TMR1C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PD	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PE	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PEC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
TMR2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TMR2C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
TMR3	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TMR3C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
INTCH	-000 -000	-000 -000	-000 -000	-000 -000	-uuu -uuu
TBHP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
DAL	xxxx ----	uuuu ----	uuuu ----	uuuu ----	uuuu ----
DAH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
VOL	xxx- ----	xxx- ----	xxx- ----	xxx- ----	uuu- ----
VOICEC	0000 -00-	0000 -00-	0000 -00-	0000 -00-	uuuu -uu-

Register	Reset (Power-on)	WDT Time-out (Normal Operation)	$\overline{\text{RES}}$ Reset (Normal Operation)	$\overline{\text{RES}}$ Reset (HALT)	WDT Time-out from HALT
LATCH0H	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
LATCH0M	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
LATCH0L	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
LATCH1H	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
LATCH1M	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
LATCH1L	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
LATCHD	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
ADRL	xxxx ----	xxxx ----	xxxx ----	xxxx ----	uuuu ----
ADRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR	0100 0000	0100 0000	0100 0000	0100 0000	uuuu uuuu
ACSR	----- --00	----- --00	----- --00	----- --00	----- --uu
LVDC	000x 0-00	000x 0-00	000x 0-00	000x 0-00	uuux u-uu
SBCR	0110 0000	0110 0000	0110 0000	0110 0000	uuuu uuuu
SBDR	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu

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

Oscillator

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

The two methods of generating the system clock are:

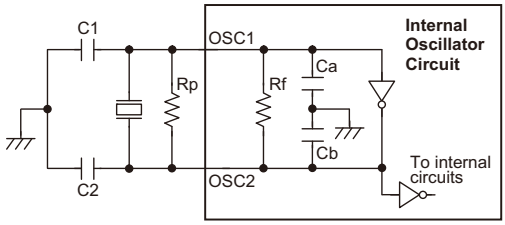
- External crystal/resonator oscillator
- External RC oscillator

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

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

External Crystal/Resonator Oscillator

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



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

Crystal/Resonator Oscillator

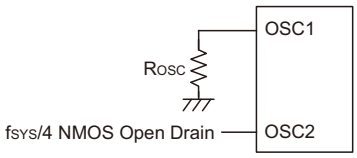
with the crystal or resonator manufacturer's specification. The external parallel feedback resistor, Rp, is normally not required but in some cases may be needed to assist with oscillation start up.

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

Oscillator Internal Component Values

External RC Oscillator

Using the external system RC oscillator requires that an external resistor is connected. For the Mask version device a value of between 120kΩ and 280kΩ is required For the OPT device a value of between 40kΩ and 90kΩ is required. This external resistor is connected between OSC1 and VSS. A clock signal, with a frequency of the generated system clock divided by 4, will be provided on OSC2 as an output which can be used for external synchronisation 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. Note that it is the only microcontroller internal circuitry together with the external resistor, that determines the frequency of the oscillator.



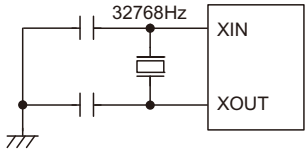
External RC Oscillator

Watchdog Timer Oscillator

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

RTC Oscillator

A 32KHz crystal can be connected to pins XIN and XOUT to implement an RTC oscillator. The RTC oscillator is used as a clock source for Timer/Event Counter 3 but must be first enabled using a configuration option. If the configuration option enables the RTC oscillator then it will automatically become the clock source for the Timer/Event Counter 3. The RTC oscillator will continue to operate even if the HALT instruction is executed and the device is powered down. It may be necessary to connect two small capacitors between XIN, XOUT and ground for correct operation of the RTC.



RTC Oscillator

Power Down Mode and Wake-up

Power Down Mode

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

Entering the Power Down Mode

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

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

Standby Current Considerations

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

Wake-up

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

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

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

Each pin on Port A can be setup via an individual configuration option to permit a negative transition on the pin to wake-up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction.

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

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

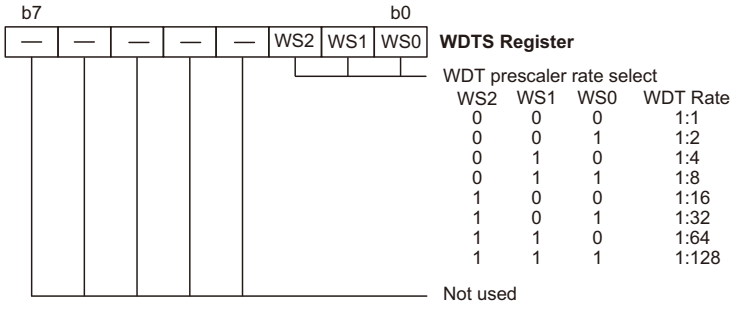
The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by one of two sources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

The internal WDT oscillator has an approximate period of 65μs at a supply voltage of 5V. If selected, it is first divided by 256 via an 8-stage counter to give a nominal period of 17ms. Note that this period can vary with VDD, temperature and process variations. For longer WDT time-out periods the WDT prescaler can be utilized. By writing the required value to bits 0, 1 and 2 of the WDTS register, known as WS0, WS1 and WS2, longer time-out periods can be achieved. With WS0, WS1 and WS2 all equal to 1, the division ratio is 1:128 which gives a maximum time-out period of about 2.1s.

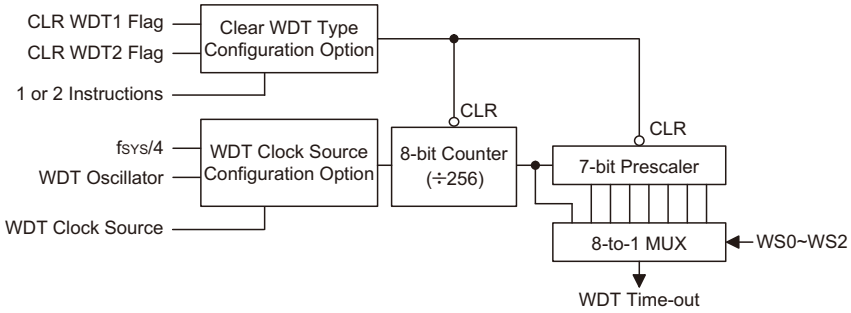
A configuration option can select the instruction clock, which is the system clock divided by 4, as the WDT clock source instead of the internal WDT oscillator. If the instruction clock is used as the clock source, it must be noted that when the system enters the Power Down Mode, as the system clock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

Under normal program operation, a WDT time-out will initialise a device reset and set the status bit TO. However, if the system is in the Power Down Mode, when a WDT time-out occurs, only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the WDT and the WDT prescaler. The first is an external hardware reset, which means a low level on the $\overline{RES}$ 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 Register



Watchdog Timer

Voice Output

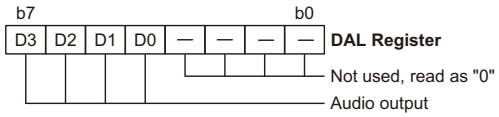
Voice Control

The voice control register controls the voice ROM circuit and the DAC circuit and selects the Voice ROM latch counter. If the DAC circuit is not enabled, any DAH/DAL outputs will be invalid. Writing a "1" to the DAC bit will enable the DAC circuit, while writing a "0" to the DAC bit will disable the DAC circuit. If the voice ROM circuit is not enabled, then voice ROM data cannot be accessed. Writing a "1" to the VROMC bit will enable the voice ROM circuit, while writing a "0" to the VROMC bit will disable the voice ROM circuit. The LATCH bit determines which voice ROM address latch counter will be used as the voice ROM address latch counter.

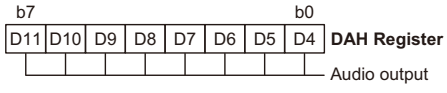
Audio Output and Volume Control – DAL, DAH, VOL

The audio output is 12-bits wide whose highest 8-bits are written into the DAH register and whose lowest four bits are written into the highest four bits of the DAL register. Bits 0~3 of the DAL register are always read as zero.

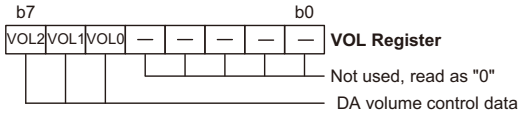
There are 8 levels of volume which are setup using the VOL register. Only the highest 3-bits of this register are used for volume control, the other bits are not used and read as zero.



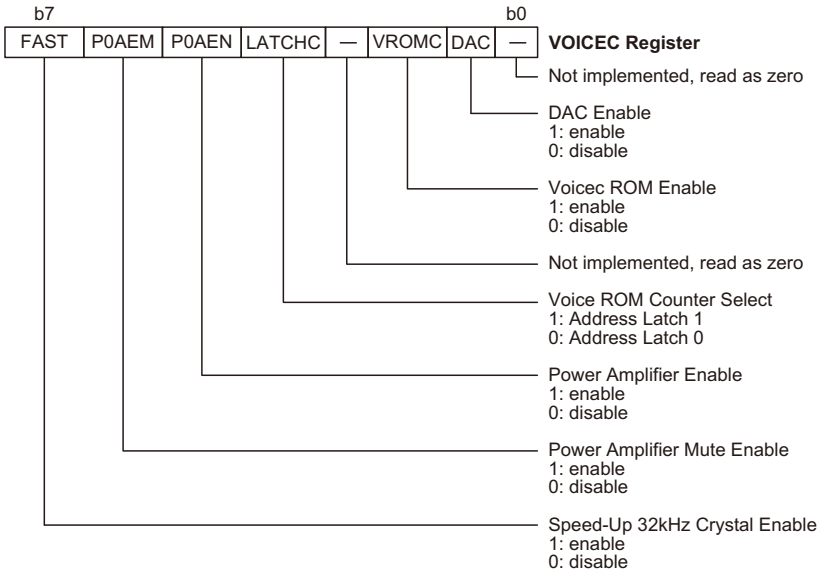
Digital to Analog Data Low Register



Digital to Analog Data High Register



Volume Control Register



VOICE Control Register

Voice ROM Data Address Latch Counter

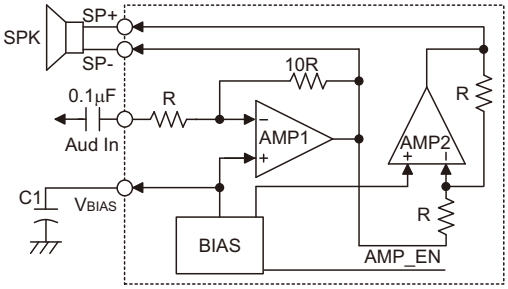
The Voice ROM address is 21-bits wide and therefore requires three registers to store the address. There are two sets of three registers to store this address, which are LATCH0H/LATCH0M/LATCH0L and LATCH1H/LATCH1M/LATCH1L. The 21-bit address stored in one set of these three registers is used to access the 8-bit voice code data in the Voice ROM. After the 8-bit Voice ROM data is addressed, a few instruction cycles, of at least 4us duration, are needed to latch the Voice ROM data. After this the microcontroller can read the voice data from the LATCHD register.

Example: Read an 8-bit voice ROM data which is located at address 000007H by address latch 0

```
Set         [26H].2             ; Enable voice ROM circuit
mov         A, 07H              ;
mov         LATCH0L, A          ; Set LATCH0L to 07H
mov         A, 00H              ;
mov         LATCH0M, A          ; Set LATCH0M to 00H
mov         A, 00H              ;
mov         LATCH0H, A          ; Set LATCH0H to 00H
call        Delay               ; Delay a short period of time
mov         A, LATCHD           ; Get voice data at 000007H
```

Power Amplifier

Each device contains an audio power amplifier which is an integrated class AB monophonic type speaker driver. It has the properties of high S/N ratio, high slew rate, low distortion, large output voltage swing, excellent power supply ripple rejection, low power consumption, low standby current and power off control etc.



Aud In: Audio input

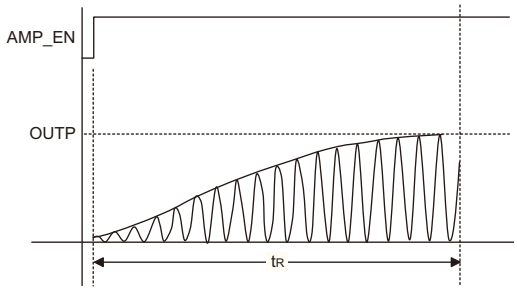
V_{BIAS}: Speaker non-inverting input voltage reference

SP+:Audio Positive output

SP-: Audio Negative output

OUTP Rising Time (t_R)

When AMP_EN enables the Power Amplifer, note that it requires a certain time before it can output fully on the OUTP pin. However, this delay time depends on the value of C1. The C1 capacitor is connected between VBIAS and VSS.



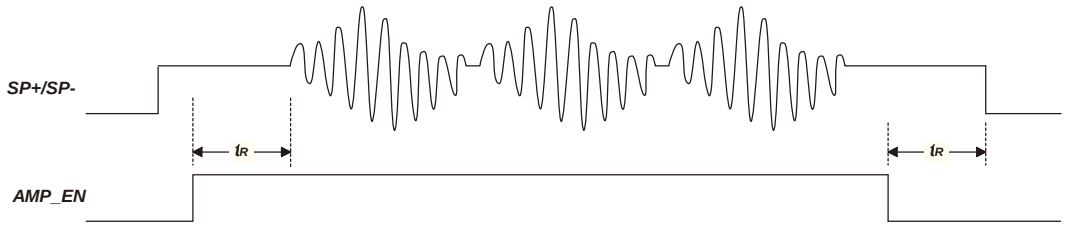
Capacitor / t_R / Voltage	0.1μF	1μF	4.7μF	10μF
2.2V	15ms	30ms	90ms	185ms
3V	15ms	30ms	90ms	185ms
4	15ms	30ms	90ms	185ms

For battery based applications, power consumption is a key issue, therefore the amplifier should be turned off when in the standby state. In order to eliminate any speaker sound bursts while turning the amplifier on, the application circuit, which will incorporate a capacitance value of C1, should be adjusted in accordance with the speaker s audio frequency response. A greater value of C1 will improve the noise burst while turning on the amplifier. The recommended operation sequence is:

Turn On: audio signal standby (1/2VDD) → enable amplifier → wait t_R for amplifier ready → audio output

Turn Off: audio signal finished → disable amplifier → wait t_R for amplifier off → audio signal off

If the application is not powered by batteries and there is no problem with amplifier On/Off issues, a capacitor value of 0.1μF for C1 is recommended.



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.

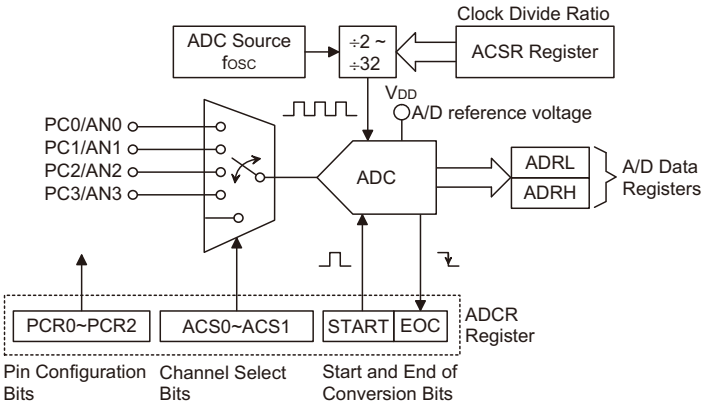
A/D Overview

The devices contain 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 a 12-bit digital value.

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

A/D Converter Data Registers – ADRL, ADRH

The devices have a 12-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 digitized conversion value. They 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 ADRL utilises only 4 of its 8-bit contents as it contains only the lower 4 bit of the 12-bit converted value.



A/D Converter Structure

In the following table, D0~D11 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
ADRL	D3	D2	D1	D0	—	—	—	—
ADRH	D11	D10	D9	D8	D7	D6	D5	D4

A/D Data Register

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 ACS1~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 ACS1~ACS0 bits in the ADCR register to determine which analog channel is actually connected to the internal A/D converter.

The ADCR control register also contains the PCR2~PCR0 bits which determine which pins on Port C are used as analog inputs for the A/D converter and which pins are to be used as normal I/O pins. Note that if the PCR2~PCR0 bits are all set to zero, then all the Port C 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 EOC bit in the ADCR register will be set high 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 EOC bit in the ADCR register is used to indicate when the analog to digital conversion process is complete. This bit will be automatically cleared to zero 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 EOC 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.

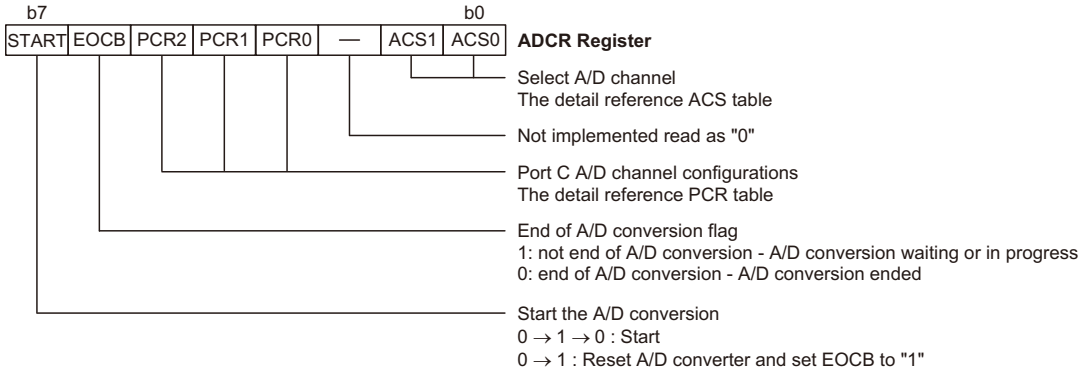
A/D Converter Clock Source Register – ACSR

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

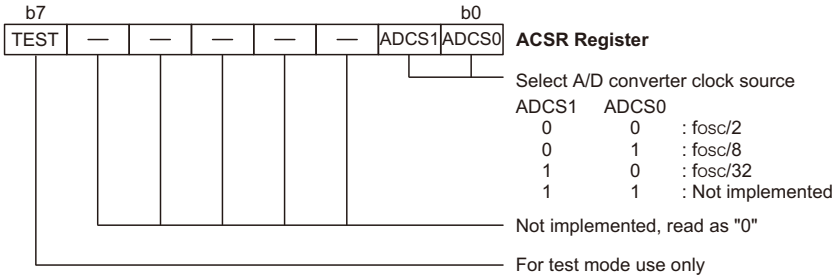
Although the A/D clock source is determined by the system clock f_{OSC}, and by bits ADCS1 and ADCS0, there are some limitations on the maximum A/D clock source speed that can be selected. Refer to the following table.

ACS1	ACS0	Analog Channel
0	0	AN0
0	1	AN1
1	0	AN2
1	1	AN3

ACS Table: A/D Channel Select Table



ADCR Register



ACSR Register

PCR2	PCR1	PCR0	3	2	1	0
0	0	0	PC3	PC2	PC1	PC0
0	0	1	PC3	PC2	PC1	AN0
0	1	0	PC3	PC2	AN1	AN0
0	1	1	PC3	AN2	AN1	AN0
1	0	0	AN3	AN2	AN1	AN0

PCR Table: Port A/D Channel Configuration Table

f_{OSC}	A/D Clock Period (t_{AD})			
	ADCS1, ADCS0=01 (f_{OSC}/2)	ADCS1, ADCS0=10 (f_{OSC}/8)	ADCS1, ADCS0=00 (f_{OSC}/32)	ADCS1, ADCS0=11
4MHz	500ns	2μs	8μs	—
6MHz	333ns	1.3μs	5.3μs	—
8MHz	250ns	1μs	4μs	—

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 C. Bits PCR2~PCR0 in the ACSR registers, not configuration options, determine whether the input pins are setup as normal Port C 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 C 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 initialized 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 initialize 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.

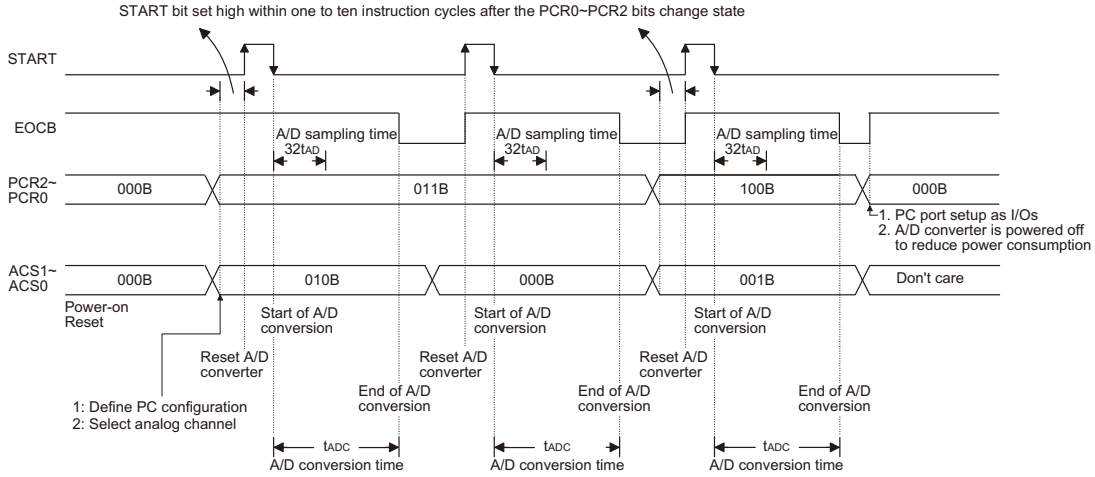
Summary of A/D Conversion Steps

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 ACS1~ACS0 bits which are also contained in the ADCR register.
- Step 3
  Select which pins on Port C 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 ADCR registers programming operation.
- Step 4
  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 5
  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.

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



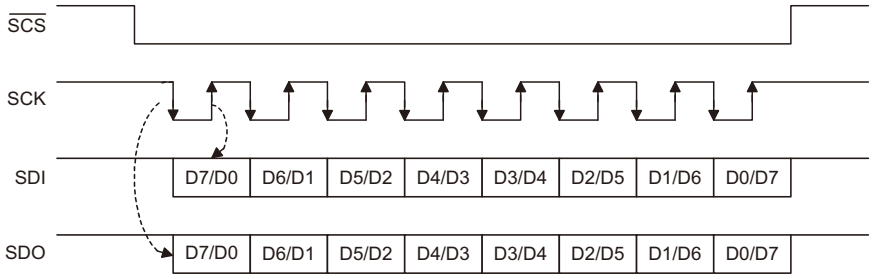
A/D Conversion Timing

SPI Serial Interface

The device includes a single SPI Serial Interfaces. The SPI interface is a full duplex serial data link, originally designed by Motorola, which allows multiple devices connected to the same SPI bus to communicate with each other. The devices communicate using a master/slave technique where only the single master device can initiate a data transfer. A simple four line signal bus is used for all communication. The four SPI lines are shared with I/O pins PD0~PD3, the function of which is chosen using a configuration option.

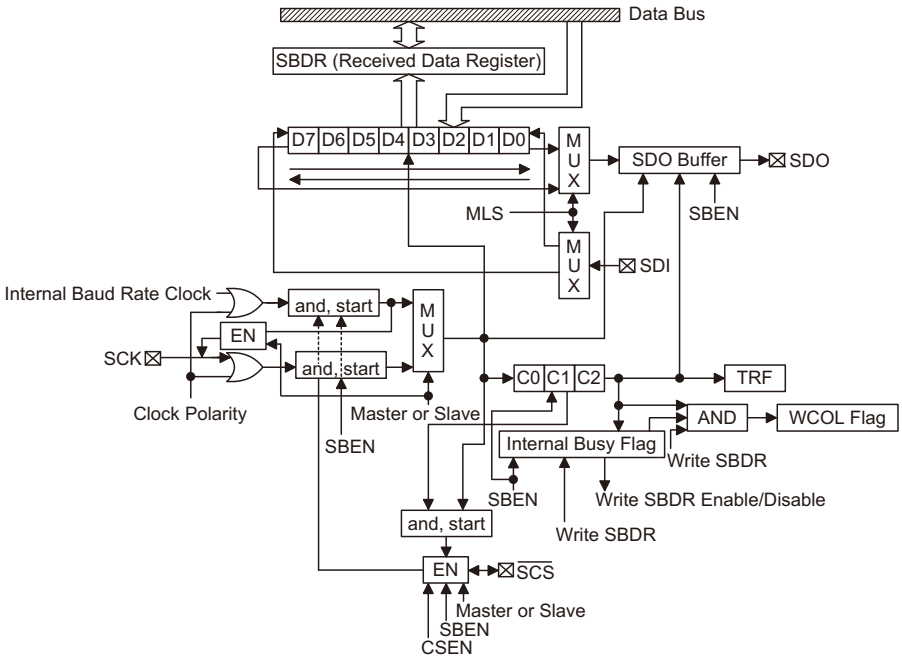
SPI Interface Communication

Four lines are used for SPI communication known as SDI - Serial Data Input, SDO - Serial Data Output, SCK - Serial Clock and SCS - Slave Select. Note that the condition of the Slave Select line is conditioned by the CSEN bit in the SBCR control register. If the CSEN bit is high then the SCS line is active while if the bit is low then the SCS line will be in a floating condition. The following timing diagram depicts the basic timing protocol of the SPI bus.



	D7	D6	D5	D4	D3	D2	D1	D0	
SBCR	CKS	M1	M0	SBEN	MLS	CSEN	WCOL	TRF	SBCR : SERIAL BUS
DEFAULT	0	1	1	0	0	0	0	0	CONTROL REGISTER
SBDR	D7	D6	D5	D4	D3	D2	D1	D0	SBDR : SERIAL BUS
DEFAULT	U	U	U	U	U	U	U	U	DATA REGISTER

Note: "U" means unchanged.



SPI Block Diagram

SPI Registers

There are two registers associated with the SPI Interface. These are the SBCR register which is the control register and the SBDR which is the data register. The SBCR register is used to setup the required setup parameters for the SPI bus and also used to store associated operating flags, while the SBDR register is used for data storage.

After Power on, the contents of the SBDR register will be in an unknown condition while the SBCR register will default to the condition below:

CKS	M1	M0	SBEN	MLS	CSEN	WCOL	TRF
0	1	1	0	0	0	0	0

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

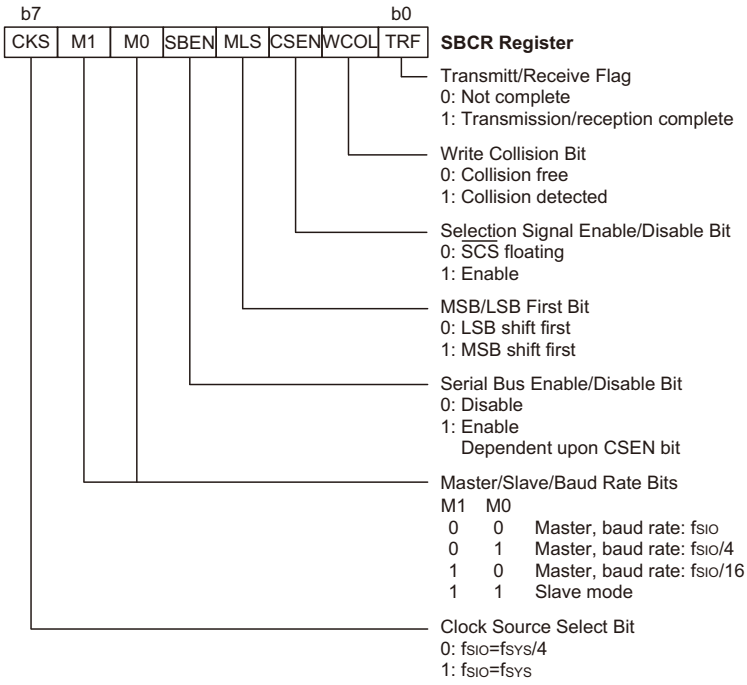
SPI Bus Enable/Disable

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

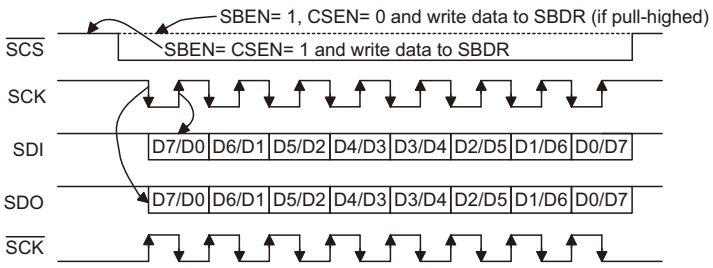
To Disable the SPI bus SCK, SDI, SDO, $\overline{SCS}$ floating.

SPI Operation

All communication is carried out using the 4-line interface for both Master or Slave Mode. The timing diagram shows the basic operation of the bus.



SPI Interface Control Register



SPI Bus Timing

The CSEN bit in the SBCR register controls the overall function of the SPI interface. Setting this bit high, will enable the SPI interface by allowing the SCS line to be active, which can then be used to control the SPI interface. If the CSEN bit is low, the SPI interface will be disabled and the SCS line will be in a floating condition and can therefore not be used for control of the SPI interface. The SBEN bit in the SBCR register must also be high which will place the SDI line in a floating condition and the SDO line high. If in Master Mode the SCK line will be either high or low depending upon the clock polarity configuration option. If in Slave Mode the SCK line will be in a floating condition. If SBEN is low then the bus will be disabled and SCS, SDI, SDO and SCK will all be in a floating condition.

In the Master Mode the Master will always generate the clock signal. The clock and data transmission will be initiated after data has been written to the SBDR register. In the Slave Mode, the clock signal will be received from an external master device for both data transmission or reception. The following sequences show the order to be followed for data transfer in both Master and Slave Mode:

- Master Mode
  Step 1. Select the clock source using the CKS bit in the SBCR control register
  Step 2. Setup the M0 and M1 bits in the SBCR control register to select the Master Mode and the required Baud rate. Values of 00, 01 or 10 can be selected.
  Step 3. Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Slave device.
  Step 4. Setup the SBEN bit in the SBCR control register to enable the SPI interface.
  Step 5. For write operations: write the data to the SBDR register, which will actually place the data into the TXRX buffer. Then use the SCK and SCS lines to output the data.
  Goto to step6.For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
  Step 6. Check the WCOL bit, if set high then a collision error has occurred so return to step5. If equal to zero then go to the following step.
  Step 7. Check the TRF bit or wait for an SBI serial bus.
  Step 8. Read data from the SBDR register.
  Step 9. Clear TRF.
  Step10. Goto step 5.

- Slave Mode
  Step 1. The CKS bit has a don't care value in the slave mode.
  Step 2. Setup the M0 and M1 bits to 00 to select the Slave Mode. The CKS bit is don't care.
  Step 3. Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Master device.
  Step 4. Setup the SBEN bit in the SBCR control register to enable the SPI interface.
  Step 5. For write operations: write data to the SBCR register, which will actually place the data into the TXRX register, then wait for the master clock and SCS signal. After this goto step 6. For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
  Step 6. Check the WCOL bit, if set high then a collision error has occurred so return to step5. If equal to zero then go to the following step.
  Step 7. Check the TRF bit or wait for an SBI serial bus.
  Step 8. Read data from the SBDR register.
  Step 9. Clear TRF
  Step10. Goto step 5

SPI Configuration Options

Several configuration options exist for the SPI Interface function which must be setup during device programming. The first is a configuration to select the PD0~PD3 pins to be used as the SPI interface pins. Another option is to enable the operation of the WCOL, write collision bit, in the SBCR register. Another option exists to select the clock polarity of the SCK line. A configuration option also exists to disable or enable the operation of the CSEN bit in the SBCR register. If the configuration option disables the CSEN bit then this bit cannot be used to affect overall control of the SPI Interface.

Error Detection

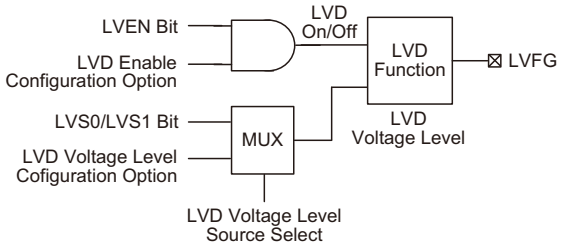
The WCOL bit in the SBCR register is provided to indicate errors during data transfer. The bit is set by the Serial Interface but must be cleared by the application program. This bit indicates a data collision has occurred which happens if a write to the SBDR register takes place during a data transfer operation and will prevent the write operation from continuing. The bit will be set high by the Serial Interface but has to be cleared by the user application program. The overall function of the WCOL bit can be disabled or enabled by a configuration option.

Programming Considerations

When the device is placed into the Power Down Mode note that data reception and transmission will continue. The TRF bit is used to generate an interrupt when the data has been transferred or received.

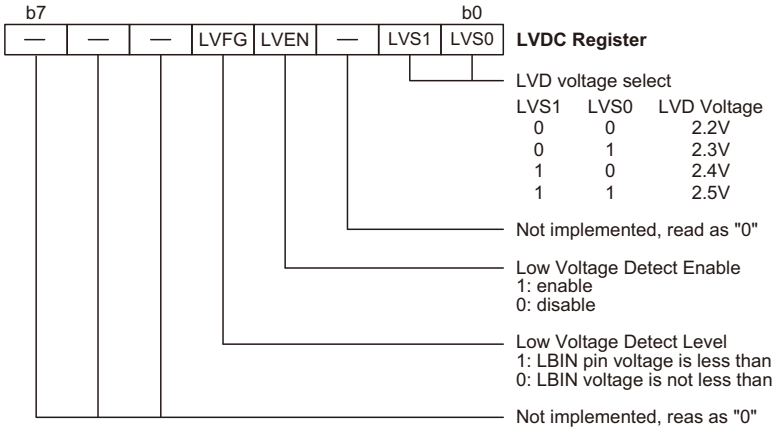
Low Voltage Detect Function

A low voltage detect function is implemented within the microcontroller. The LVD function is controlled using the LVDC register and configuration options. To enable the LVD function, both the LVD configuration option should be enabled and the LVEN bit should be set high. After setting the LVEN bit high, the circuit requires about 100μs to stabilise. After this time has elapsed, the LVFG bit can be monitored to look for low voltage conditions.



LVD Block Diagram

The LVS0 and LVS1 bits are used to define the LVD voltage threshold level, however configuration options can also be used to define this voltage. A configuration option is used to decide whether these two register bits or the configuration option is used to define the LVD voltage threshold level. As the low voltage detector circuitry will consume a certain amount of power, the LVEN bit can be reset to zero to turn off the LVD internal circuitry to reduce power consumption.



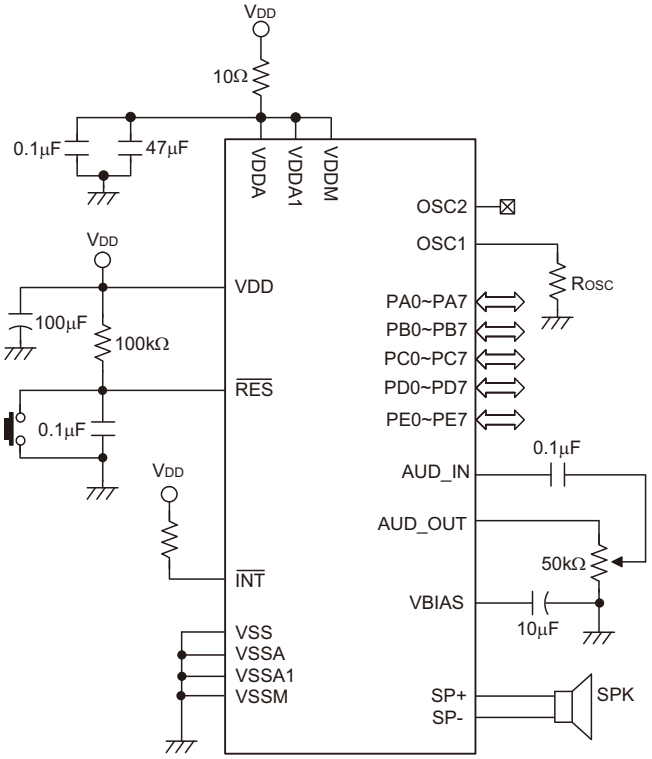
Low Voltage Detect Control Register – LVDC

Configuration Options

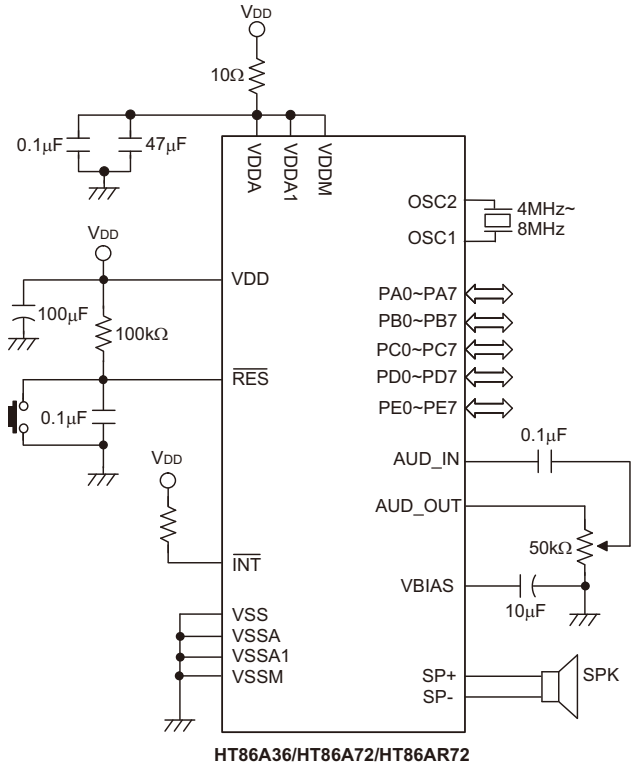
Configuration options refer to certain options within the MCU that are programmed into the device during the programming process. During the development process, these options are selected using the HT-IDE software development tools. As these options are programmed into the device using the hardware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

No.	Options
I/O Options	
1	PA0~PA7: wake-up enable or disable (bit option)
2	PA0~PA7: pull-high enable or disable (bit option)
3	PB0~PB7: pull-high enable or disable (bit option)
4	PC0~PC7: pull-high enable or disable (bit option)
5	PD0~PD7: pull-high enable or disable (bit option)
6	PE0~PE7: pull-high enable or disable (bit option)
7	PD pin shared function select: select PD3~PD0 as I/O pins or as serial interface function
Oscillation Option	
8	OSC type selection: RC or crystal
9	RTC: enable or disable
Interrupt Option	
10	INT Triggering edge: Falling or both
Watchdog Options	
11	WDT: enable or disable
12	WDT clock source: WDROSC or T1
13	CLRWDT instructions: 1 or 2 instructions
Low Voltage Reset Option	
14	LVR select: enable or disable
Low Voltage Detect Option	
15	LVD voltage: 2.2V~2.5V
16	LVD select: enable or disable
17	LVD voltage level control select: Register bits or configuration option
Serial interface Option	
18	Serial interface CPOL: falling edge or rising edge
19	Serial interface WCOL: enable or disable
20	Serial interface CSEN: enable or disable

Application Circuits



HT86A36/HT86A72/HT86AR72



HT86A36/HT86A72/HT86AR72

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 $\neq$ 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

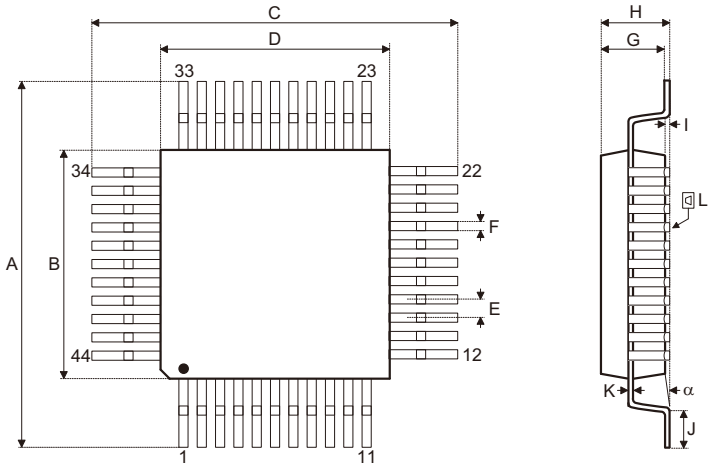
XORM A,[m]	Logical XOR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″XOR″ [m]
Affected flag(s)	Z

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 (http://www.holtek.com.tw/english/literature/package.pdf) for the latest version of the package information.

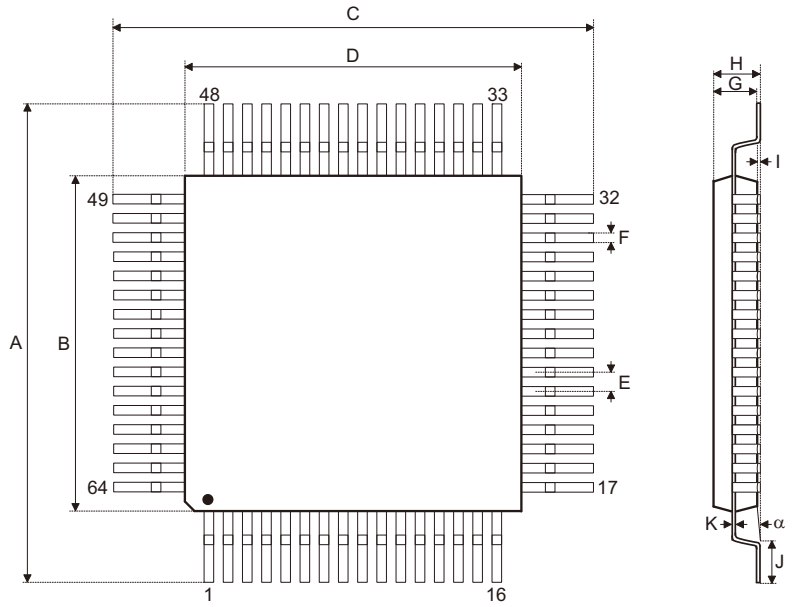
44-pin LQFP (10mm×10mm) (FP2.0mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.469	—	0.476
B	0.390	—	0.398
C	0.469	—	0.476
D	0.390	—	0.398
E	—	0.031	—
F	—	0.012	—
G	0.053	—	0.057
H	—	—	0.063
I	—	0.004	—
J	0.018	—	0.030
K	0.004	—	0.008
α	0°	—	7°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	11.90	—	12.10
B	9.90	—	10.10
C	11.90	—	12.10
D	9.90	—	10.10
E	—	0.80	—
F	—	0.30	—
G	1.35	—	1.45
H	—	—	1.60
I	—	0.10	—
J	0.45	—	0.75
K	0.10	—	0.20
α	0°	—	7°

64-pin LQFP (10mm×10mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.469	—	0.476
B	0.390	—	0.398
C	0.469	—	0.476
D	0.390	—	0.398
E	—	0.020	—
F	—	0.008	—
G	0.053	—	0.057
H	—	—	0.063
I	—	0.004	—
J	0.018	—	0.030
K	0.004	—	0.008
α	0°	—	7°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	11.90	—	12.10
B	9.90	—	10.10
C	11.90	—	12.10
D	9.90	—	10.10
E	—	0.50	—
F	—	0.20	—
G	1.35	—	1.45
H	—	—	1.60
I	—	0.10	—
J	0.45	—	0.75
K	0.1	—	0.2
α	0°	—	7°

Holtek Semiconductor Inc. (Headquarters)
No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
http://www.holtek.com.tw

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

Holtek Semiconductor (China) Inc.
Building No. 10, Xinzhu Court, (No. 1 Headquarters), 4 Cuizhu Road, Songshan Lake, Dongguan, China 523808
Tel: 86-769-2626-1300
Fax: 86-769-2626-1311

Holtek Semiconductor (USA), Inc. (North America Sales Office)
46729 Fremont Blvd., Fremont, CA 94538
Tel: 1-510-252-9880
Fax: 1-510-252-9885
http://www.holtek.com