

## Subminiature Control Computer



### GENERAL FEATURES

- Small Size—only 1.5 square inches (0.375" × 1.75" × 0.875")
- Combination control computer and intelligent I/O coprocessor
- Operates on a wide power supply voltage range: 4.5–18 V
- Low power consumption—less than 75 mW
- Thermally protected onboard regulator supports high-current externally connected devices (see chart, p. 6)
- Programmable in PicBasic, C, and assembly language
- Use for standalone or networked data acquisition and control
- Use for standalone or networked sensor-to-computer interface
- Enclosed construction seals out harsh environment
- Industrial temperature range available (minimum quantities apply)

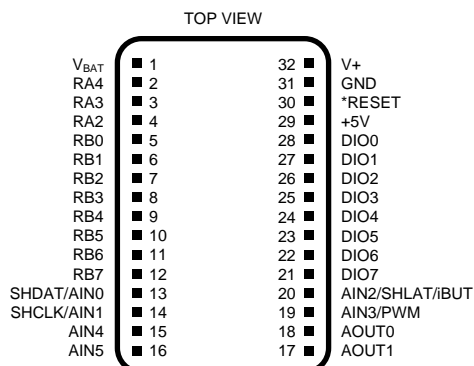
### CONTROLLER SECTION

- Uses EEPROM-programmable PIC16C84 processor: 1024 × 14 EEPROM program memory, 64 × 8 EEPROM data memory

- 11 bidirectional, bit-programmable high-current I/O lines: 25 mA sink per pin, 20 mA source per pin
- Software-provided serial communications routines: 300–9600 bps
- Communicates directly with I/O coprocessor to provide high-level processing and data management

### I/O COPROCESSOR SECTION

- 4-channel, 8-bit ADC, 0–5 V
- 2K × 8 EEPROM data storage memory, typically 1,000,000 erase/write cycles
- 2-channel 12-bit, 0–5-V ADC
- 2-channel 12-bit, 0–4.096-V DAC
- Battery-backable real-time clock/calendar
- Direct input frequency measurement
- PWM output: 2 Hz to 3.5 kHz, 5–95% duty cycle
- 8 bidirectional, bit-programmable, high-current I/O lines: 25 mA sink per pin, 20 mA source per pin
- 16-bit, 74LS595 (2) shift-register-controlled, TTL-level parallel output
- Automatic 4 × 6 keypad scanning
- Directly drives 4 × 20 LCD display, with auto backlight control
- Reads Dallas Semiconductor iButton serial numbers
- High-performance built-in functions: analog data averaging,  $V_{peak}$  and  $V_{min}$  recording, frequency measurement, event totalization



### DESCRIPTION

PicStic4 is a low-cost, industrial-oriented controller with a built-in intelligent I/O coprocessor all in a 1.5-square-inch enclosed DIP. Including options, PicStic4 incorporates digital inputs, digital outputs, analog inputs, analog outputs, real-time monitoring, extended data memory, power input regulation, and serial communication, along with a multitude of processing functions in a single module. PicStic4s can be used independently or networked together.

The architecture of PicStic4 is very straightforward. It consists of a user-programmable controller, similar in most respects to our original PicStic1, enhanced with an intelligent I/O coprocessor. The coprocessor provides extended data memory, automatically reads and sets 12-bit analog I/O, reads or sets the real-time clock, and supports up to 24 bits of additional parallel I/O. Beyond the physical attributes, the coprocessor also provides numerous processing and data management functions which can be accessed from the controller side.

Complicated program tasks like scanning keypads, driving an LCD display, recording  $V_{min}$ ,  $V_{peak}$ , and  $V_{avg}$  analog readings, reading and setting the real-time clock, recording important data in extended nonvolatile memory, totaling events, reading frequency, generating a constant PWM output, or reading Dallas Semiconductor iButton serial numbers, are all provided as simple callable functions from the I/O coprocessor. PicStic4 is designed to provide the user with a cost-effective and easily programmable control device without requiring the user to eat up valuable controller-side programming space with all the low-level code typically necessary to perform useful monitoring and control activities.

PicStic4's small 1.5-cubic-inch enclosed module and standard 32-pin DIP packaging allow it to be incorporated directly within a sensor or control device to add an analog-to-serial data interface or otherwise supplement the control intelligence at the sensor end of things. When using the PicBasic compiler, PicStic4 is software-compatible with Parallax BASIC Stamps but as much as 15 times faster. Of course, PicStic4 is not limited to only being programmed in PicBasic. Because PicStic4 uses a reprogrammable PIC16C84 processor and a customized compiler, PicStic4s can be programmed in compiled BASIC, C, or directly in PIC assembly language.

PicStic4 was designed as a controller that allows a user to program the detailed tasks necessary to achieve the objective while not wasting time and program space writing low-level support code. PicStic4 combines both a programmable processor and a special firmware-programmed coprocessor with a dedicated function repertoire. The two processors communicate via a high-speed serial connection, invisible to the user except for a simple `CALL` routine for sending a 3-byte command and receiving a 3-byte data reply.

PicStic4s come in three variations: PS4-P, PS4-Q, and PS4-X. All are physically the same size and pinout. All contain the I/O coprocessor with the extended firmware-controlled functions such as keypad scanning, input averaging, event counting, and so forth. The differences are strictly related to how much of the required hardware to do specific firmware functions is populated in a particular version.

All PS4s contain the control processor and an I/O coprocessor with base-level functions. The PS4-P is the base-level unit and contains the following I/O: 19 bits parallel (11 bits in the controller and 8 more in the I/O coprocessor), 4 channels of 8-bit ADC, and 16-Kb extended data memory. All coprocessor firmware functions, with the exception of those addressing the real-time clock, 12-bit ADC, and 12-bit DAC, are enabled.

The PS4-Q is equivalent to a PS4-P with the addition of the real-time clock and the 2-channel, 12-bit ADC (ADC only, not the DAC). The PS4-Q executes all PS4 I/O coprocessor firmware functions with the exception of those addressing the DAC.

Finally, the PS4-X is a fully configured unit equivalent to a PS4-P with real-time clock, and both 12-bit ADC and DAC hardware. It is capable of doing any of the firmware functions listed.

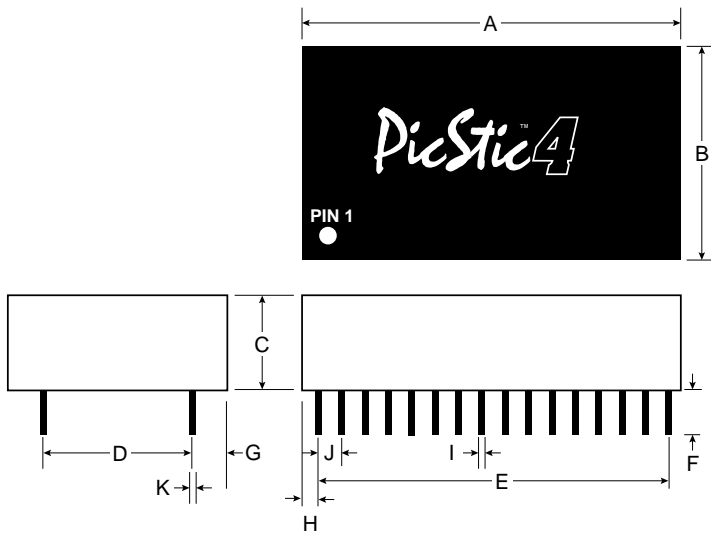
**The ability to perform “any” of the PicStic4’s “smart” coprocessor functions doesn’t mean that “all” of them are available at the same time.** For example, when an LCD and keypad are connected to a PS4, it uses all the parallel I/O lines and the 4-channel ADC inputs. While you still have 11 parallel I/O lines on the controller section, no 8-bit ADC is available. If the PS4 being used is the PS4-Q or PS4-X versions, however, the 2-channel, 12-bit ADC is available (plus the DAC in the -X version) and commands relating to it, such as read 12-bit analog min/max, could be exercised with the LCD and keypad. Compatible functions are explained in greater detail in the section on coprocessor configuration.

## ABSOLUTE MAXIMUM RATINGS

Supply Voltage	+16.0 V
Digital Input Voltage	0 V to +5.5 V
Analog Input Voltage	−0.5 V to +5.5 V
Storage Temperature	−25°C to +100°C
Lead Temperature	260°C
Operating Temperature	0°C to +70°C
	(−40°C to +85°C available by special order, minimum quantities apply)

Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

## MECHANICAL SPECIFICATIONS



Dim	Inches		Millimeters	
	min	max	min	max
A	1.735	1.765	44.06	44.83
B	0.865	0.885	21.97	22.47
C	0.365	0.385	9.27	9.77
D	0.590	0.610	14.98	15.49
E	1.490	1.510	37.84	38.35
F	0.240	0.300	6.09	7.62
G	0.128	0.148	3.25	3.76
H	0.113	0.133	2.87	3.38
I	0.019	0.021	0.48	0.53
J	0.095	0.105	2.41	2.66
K	0.007	0.009	0.17	0.22

## PIN DESCRIPTIONS

All three versions of PicStic4 are 32-pin DIPs with 0.6" spacing.

Pins with multiple designations indicate default and optional functions for the pin.

Pin	Signal	Description	Pin	Signal	Description
1	V <sub>BAT</sub>	Input for +3-V external battery for real-time clock backup. The battery negative terminal should be connected to ground.	21–28	DIO7–DIO0	I/O coprocessor general-purpose TTL-level programmable input/output pins. Each pin can sink 25 mA and source 20 mA (total current for this port should not exceed 50 mA sinking and 40 mA sourcing). DIO0 is also the frequency/event input .
2–4	RA4–RA2	Processor general-purpose TTL-level programmable input/output pins. Each pin can sink 25 mA and source 20 mA.	29	+5V	This is the 5-V internal 12-bit ADC voltage reference (nominally 5.0 V). This output may also be used to power minimal external circuitry or sensors. PicStic4 may be 5-V–only powered through this pin provided pin 32 is left unconnected.
5–12	RB0–RB7	Processor general-purpose TTL-level programmable input/output pins. Each pin can sink 25 mA and source 20 mA (total current for this port should not exceed 50 mA sinking and 40 mA sourcing) .	30	*RESET	When grounded, *RESET provides a master clear of the system. All outputs are cleared.
13	AIN0/SHDAT	<b>Def.:</b> Analog input, channel 0, 8 bits, 0–5 V. <b>Opt.:</b> Shift register data output .	31	GND	Combined digital and analog ground.
14	AIN1/SHCLK	<b>Def.:</b> Analog input, channel 1, 8 bits, 0–5 V. <b>Opt.:</b> Shift register clock output.	32	V+	PicStic4 power supply input. V+ is nominally 8–12 V. If pin 32 is open, PicStic4 can be 5-V powered directly to pin 29.
15,16	AIN4,AIN5	Analog inputs, channels 4 and 5, 12 bits, 0–5 V.			
17,18	AOUT1,AOUT0	Analog outputs, channels 0 and 1, 12 bits, 0–4.096 V.			
19	AIN3/PWM	<b>Def.:</b> Analog input, channel 3, 8 bits, 0–5 V. <b>Opt.:</b> PWM output.			
20	AIN2/SHLAT/iBUT	<b>Def.:</b> Analog input, channel 2, 8 bits, 0–5 V. <b>Opt.:</b> Shift register latch output. <b>Opt.:</b> iButton input.			

## DC Electrical Characteristics

Operating Temperature  $T_a = 0^\circ\text{C}$  to  $70^\circ\text{C}$   
 Operating Voltage  $V_+ = 9\text{ V}$ ,  $\text{GND} = 0\text{ V}$

Characteristic	Min	Typ	Max	Units	Condition
Supply Voltage ( $V_+$ )	7	9	16	V	$V_{CC} = 5.5\text{ V}$
Supply Current ( $I_{CC}$ )		5	9.5	mA	
Memory:					
Data EEPROM Endurance		1,000,000			
Program EEPROM Endurance		1,000			E/W Cycles
Digital I/O:					
Input Low Voltage ( $V_{il}$ )	0		0.9	V	$I_{ol} = 8.5\text{ mA}$ $I_{ol} = -3.0\text{ mA}$
Input High Voltage ( $V_{ih}$ )	1.9		5.5	V	
Output Low Voltage ( $V_{ol}$ )		0.45	0.6	V	
Output High Voltage ( $V_{oh}$ )	4.3			V	

## A/D Converter Characteristics

### 8-bit ADC (all versions)

Characteristic	Min	Typ	Max	Units	Condition
Resolution	8			bit	$V_{SS} \leq AIN \leq V_{REF}$
Linearity Error		$\pm 1$		bit	
Offset and Gain Error		$\pm 1$		bit	
Voltage Reference	3.0	5.0	5.3	V	
Analog Input Range		$-0.3$ to $V_{REF}+0.3$		V	See note 1
Analog Input Impedance		10k		$\Omega$	
Conversion Time		1		ms	

### 12-bit ADC (PS4-Q and -X versions only)

Characteristic	Min	Typical	Max	Units	Condition
Resolution	12			bit	$V_{REF}$ is $V_{CC}$ , See note 2
Linearity Error		$\pm 3/4$		bit	
Offset and Gain Error		$\pm 2$		bit	
Voltage Reference	4.5	5.0	5.5	V	
Analog Input Range		$-0.5$ to $V_{REF}+0.5$		V	See note 3
Analog Input Impedance		250k		$\Omega$	See note 1
Conversion Time		1		ms	

## D/A Converter Characteristics

### 12-bit DAC (PS4-X version only)

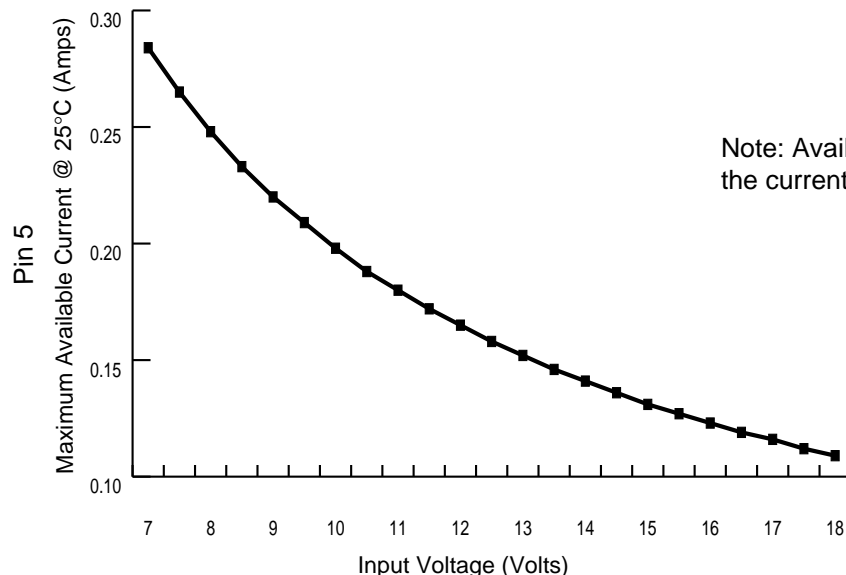
Characteristic	Min	Typ	Max	Units	Condition
Resolution	12			bit	
Full-Scale Error		$\frac{1}{2}$		bit	
Offset and Zero Error		$\pm 2$	$\pm 18$	mV	
Supply Voltage Required	4.5		5.5	V	
Analog Output Range	4.065	4.095	4.125	V	at 25°C
Analog Output Impedance		40	120	$\Omega$	measured to ground
Update Time		1		ms	See note 1

Note 1: Conversion and update times are determined by the time for a command and response from the coprocessor rather than any timing specific with the conversion or update itself.

Note 2: If the onboard voltage regulator reference is inadequate for the application, an alternative is to power the PicStic 4 via pin 29 with a supply of 5.000 V.

Note 3: Two diodes are tied to each analog input which will conduct when the input voltage is one drop below  $V_{SS}$  or one drop above  $V_{REF}/V_{CC}$ .

### Maximum available current draw from pin 5 with input power applied to pin 1.



Note: Available current does not include the current used by any I/O pins.

### Example

Input voltage = 10 V + 2 I/O pins sinking 15 mA each

Current available = Max. Current (from chart) – Current used by I/O pins

$$I = 195 \text{ mA} - 30 \text{ mA} \\ = 165 \text{ mA}$$

Typically, you shouldn't use more than  $\frac{2}{3}$  of the maximum available current. Therefore,

$$I_{MAX} = 165 \times \frac{2}{3} \\ = \sim 110 \text{ mA}$$

## 1.0 QUICK START

You probably want to hook up the PicStic4 and try it out without having to read this whole datasheet. Use the following steps to connect and configure the PicStic4 with a simple PicBasic program. Be sure to thoroughly read the remaining sections before trying to use any of the features not described here.

Copy the PicBasic files to a new directory on your PC's harddrive (i.e., copy a:\*.\* C:\PICSTIC). Use DOS EDIT (or your favorite text editing program) and type in the following program. (Remember to save it in ASCII text only mode to the new working directory under the name QUICK.BAS).

From the DOS prompt (C:\PICSTIC) compile the program using the command: PBC QUICK. The PicBasic program QUICK.BAS will be automatically compiled and then assembled into QUICK.HEX.

Plug the PicStic programmer into a parallel port on the PC. Apply power to the board. (Make sure the 5-jumpers on the programmer are in the DIP position for programming the PicStic4). From the DOS prompt run the

programming software using the command: EPIC.

When the program starts click the Open File box and choose the QUICK.HEX file for loading. (Check to make sure the boxes on the right side of the screen indicate: device 8X, size 1K, osc XT, Watchdog OFF, code protect OFF, and power up timer HIGH). You can now place the PicStic into the programmer. Note the direction of Pin1. Click on Erase to clear the PicStic's memory. Click on Program to program the new file into the PicStic.

You may now place the programmed PicStic into your application. For this application you will need to apply 9–12 VDC between Pin 31 (ground) and Pin 32 (+V) or 5 VDC between Pin 31 (ground) and Pin 29 (+5). NOTE: Applying power backwards or to the wrong pins will certainly damage the PicStic (and void the warranty). An oscilloscope or logic probe will show the pins RB0–7 and IO0–7 counting in binary fashion. If you don't have access to one of these, use an LED with a series resistor tied to either ground or +5. Touch the other lead of the LED to any of the outputs to see the bit toggling.

```

Start: Dirs=$FF          ' port B pins as outputs
      B19=$03 : B20=0 : B21=0 ' set I/O configuration to just DI0
      call PASS           ' go do it

      B19=$05 : B20=0 : B21=0 ' set DI00-7 as all outputs
      call PASS           ' go do it

Loop:  for B0=0 to 255      ' select all possible combinations of bits

      Pins=B0              ' set outputs of RB0-7 (port B)

      B19=$07 : B20=B0 : B21=0 ' set outputs of DI00-7 (I/O port)
      call PASS           ' go do it

      next B0              ' go get next combination

      goto Loop           ' done with all combinations, now go do again
  
```

**Figure 1.1:** This quick-start program performs a binary counting output on all I/O pins.

## 2.0 PICSTIC4 OVERVIEW

PicStic4 is a low-cost, industrially oriented controller in a 1.5-square-inch 32-pin DIP enclosed package. It is a combination of a Microchip Technology PIC16C84 RISC processor and a separate I/O coprocessor which manages a number of onboard I/O peripherals. Together, they provide an extraordinarily powerful tool for accomplishing control applications.

The PIC16C84 (or whichever variation is presently in use) includes onchip RAM, EEPROM, timers, and parallel I/O. An adjacent I/O coprocessor manages PicStic4's real-time clock, ADC, DAC, and expanded data memory hardware peripherals as well as providing a variety of firmware operations using those peripherals.

## 2.1 PIC16C84 PROCESSOR

The PIC16C84 is a high-performance, CMOS, fully static 8-bit microcontroller with  $1K \times 14$  EEPROM program memory and 64 bytes of EEPROM data memory. Its high performance is due to instructions that are all single word (14-bit wide), which execute in a single cycle (1  $\mu$ s at 4-MHz clock), except for program-branches which take two cycles (2  $\mu$ s).

The PIC16C84 has four interrupt sources and an 8-level hardware stack. The peripherals include an 8-bit timer/counter with an 8-bit prescaler (effectively a 16-bit timer) and 13 bidirectional I/O pins. The high-current drive (25 mA maximum sink, 20 mA maximum source) of the I/O pins often reduces the requirement for external drivers.

The PIC16C84 is the same processor used on our PicStic1, 2, and 3 devices. Think of PicStic4 as simply a PicStic1 with some intelligent I/O attached.

## 2.2 PICSTIC4 I/O COPROCESSOR

The PIC16C84 processor uses two processor lines (RA0 and RA1) to serially communicate commands and data to and from the I/O coprocessor. The data rate of this serial connection is 62.5 kbps. Communicating with it simply requires the use of the CALL command. When invoked, it transmits a 3-byte command/parameter value to the I/O coprocessor and returns with a 3-byte data value to the user program. This procedure is explained in detail in Section 6.1.

The I/O coprocessor has a 4-channel 8-bit ADC, 2-channel 12-bit ADC, 2-channel 12-bit DAC, 16 Kb ( $2K \times 8$ ) EEPROM data memory, and a battery-backable (external, via  $V_{BAT}$  pin 1) real-time clock/calendar. How many of these options are populated on a PicStic4 is determined by the -P, -Q, and -X designations.

All units contain the PIC16C84 processor, I/O coprocessor, 8-bit ADC, and expanded data memory. This base unit is designated as the PicStic4-P.

The PicStic4-Q contains these same attributes plus the 2-channel 12-bit ADC and real-time clock. PicStic4-X adds the 2-channel DAC to round out the family.

The most important factor in PicStic4 is that, unlike connecting an ADC or real-time clock directly to the PIC16C84 processor and having separate callable routines taking up valuable program space, a single call routine is used to obtain all information from the I/O coprocessor. Depending upon the command and parameters sent when calling PASS, the I/O coprocessor executes its preprogrammed functions to carry out that command. Because the I/O coprocessor also contains significantly more firmware memory than the user-pro-

grammable memory in the PIC16C84, **many distinct and powerful functions are included. However, because of the limited user program space, it is not our intention that you should expect to use all of these functions in a single application program.**

## 2.3 COPROCESSOR-CONTROLLED PERIPHERALS

The I/O coprocessor has a 4-channel 8-bit ADC, 2-channel 12-bit ADC, 2-channel 12-bit DAC, 16 Kb ( $2K \times 8$ ) EEPROM data memory, and a battery-backable real-time clock/calendar as direct controlled peripherals. Using these peripherals requires first that they physically be populated in the PicStic4 in use and that the I/O coprocessor has been sent the configuration parameters which enables these particular attributes.

While all the hardware attributes can be enabled at the same time, only certain firmware functions can run concurrently. Because the firmware configuration also defines the specific electrical characteristics of PicStic4's analog and digital I/O lines, care should be observed to make sure that the configuration is correct for whatever is to be attached to these pins.

If the external physical connections are not in conflict, it is possible to configure the I/O coprocessor for one function during one part of your program and then reconfigure it for something else at another place. For example, using both LCD display and 8-bit ADC at the same time is an invalid configuration. You could, however, first configure and use the 8-bit ADC and then reconfigure it to allow LCD display.

The 4-channel 8-bit ADC has an input range of 0–5 V and an input impedance of 10 k $\Omega$ . The 2-channel 12-bit ADC also has an input range of 0–5 V, but the input impedance is 250 k $\Omega$ . While the PicStic4 uses high-speed successive-approximation ADCs, their conversion rates are limited by the time it takes to request a conversion and receive the data. In a typical application, the DAC update and ADC conversion rate is about 1000 per second (1 ms).

The 2-channel 12-bit DAC has an output range of 0–4.096 V and a maximum output impedance of 140  $\Omega$ . Care should be taken not to ground this line or it will allow excessive current to flow which may damage the PicStic.

The quartz-crystal-controlled real-time clock/calendar has a resolution ranging from month of the year down to seconds. Its operating voltage is 2.0–5.5 V (as measured at pin 29), however the clock can only be read or written when this voltage is  $\geq 4.75$  V. When the PicStic4 is unpowered, the time can be maintained by attaching a 3-V lithium battery connected to  $V_{BAT}$  pin 1.



Since current consumption during backup is less than 3  $\mu$ A, only a small battery is required.

Finally, the I/O coprocessor also manages a 16-Kb (2K  $\times$  8) EEPROM which can be used as extended data memory (not program memory). The memory is directly addressable by location and all but the last 32 bytes of it is available to the user. While there are many ways it could be used, two typical methods come to mind: data logging and tables of data constants.

In the former, a program simply reads or computes values which are directly stored for later retrieval or

transmission. In the latter, long strings for LCD menuing or large lookup tables could be stored in this memory rather than displace valuable program space in the processor section. Placing this data in the extended memory first requires executing a small program to "upload" them. Once entered in the nonvolatile extended memory, it will remain forever until rewritten or erased. Any program can be run that uses the stored data by simply addressing specific string or table locations.

---

## 3.0 PICSTIC4 SOFTWARE

When it comes from the factory, the PicStic4 has no software in the user program space. Code is developed using cross-development tools running on a desktop PC and programmed into the PicStic4 for execution. There are several development environments from which to choose. Regardless of the program language used, communication with the I/O coprocessor section is still via the same callable subroutine.

### 3.1 ASSEMBLY LANGUAGE

Any cross-assembler capable of creating code for the PIC16C84 processor can be used to write assembly language programs for the PicStic4. Microchip's assembler is available at no cost from their BBS and Web site. More information about the PIC instruction set and how to connect to the Microchip BBS may be found in the *Microchip Data Book*. Their Web site may be contacted at <<http://www.microchip.com/>>.

Other cross-assemblers are available including one from various vendors that enhance the PIC instruction set with one more familiar to 8051 programmers.

A cross-assembler also comes with the Micromint PicBasic development package.

### 3.2 PICBASIC

Micromint's PicBasic compiler allows the use of BASIC Stamp-compatible programs on the PicStic4, but with much higher execution speed. PicBasic also provides the capability to include custom assembly language routines for time-critical tasks.

The PicBasic compiler was originally designed to be Stamp I compatible. However, because it is a compiler, it can also have additional commands and enhancements without changing the hardware. Current enhancements include PEEK, POKE, 9600-bps console communications, and I<sup>2</sup>C communication. The PicBasic command list should, therefore, be presumed to be the minimum command list at any one time.

Contact Micromint for more information about current enhancements to PicBasic.

### 3.3 C

While our documentation centers on the use of PicBasic with PicStic4, a PIC C cross-compiler is available. The integrated C development environment gives developers the capability to quickly produce efficient code from an easily maintainable high-level language. The compiler includes built-in functions to access the PIC hardware such as INPUT and OUTPUT\_HIGH. Variables including structures may be directly mapped to memory such as I/O ports to best represent the hardware structure in C.

Functions may be implemented inline or separate. Function parameters are passed in reusable registers. Inline functions with reference parameters are implemented efficiently with no memory overhead.

It is possible that various C compilers will be available. Call Micromint for information on vendor and price.

## 4.0 PROGRAMMING THE PICSTIC4

The user-programmable part of PicStic4 uses Microchip Technology's PIC16C84 EEPROM microcontroller which can be reprogrammed hundreds of times. These programs can be created using a number of resources, as described above.

Programming a PicStic4 is done serially, involving only five signal connections. The five signals are power (+5V), ground (GND), and \*RESET (which must be pulled to 12 V), and port pins PB7 (serial data) and PB6 (serial clock).

The PicStic4 PicBasic Development package includes a programmer which is plug-compatible with PicStic4 and needs no programming adapters. (Note: the Parallax BASIC Stamp programmer cannot be used for programming PicStic4.) The PicStic4 can be programmed

with most other PIC16C84 programmers by making a simple five-wire adapter to connect these five signals appropriately.

Most assemblers and compilers enable you to take advantage of a variety of 16C84 configuration options. These include code protection, powerup timer, watchdog timer, oscillator designation (PicStic4 uses the XT), and user programming of the 64 bytes of additional EEPROM data memory. Be aware that you may have to set these conditions on the programmer menu for proper PicStic4 programming.

For a more involved description of the PIC16C84 programming algorithm and technology, refer to the *Microchip Data Book*.

## 5.0 PICSTIC4 HARDWARE

We provide sample code for many of PicStic4's hardware features in the Appendix. It is beyond the scope of this document to provide full details about each device. However, we have tried to include enough information to allow the programmer to develop code beyond that used for the sample.

### 5.1 SERIAL CONNECTIONS

PicStic4s have no specific pins for serial I/O. Various compilers allow the user to designate the physical pin locations of serial in and serial out. The circuits in Figure 5 in the Appendix should be used to interface the PicStic4 to an RS-232C serial port. While we recom-

```
Start:    peek $85,B0      ' read PortA direction control port into B0
          bit2=0           ' make RA2 an output bit
                               ' (leave other bits alone)
          poke $85,B0      ' write PortA direction control port from B0

Loop:     peek $05,B0      ' read PortA output latch register into B0
          bit2=1           ' set RA2 output data as logic '1'
                               ' (leave other bits alone)
          poke $05,B0      ' write PortA output data from B0

          peek $05,B0      ' read PortA output latch register into B0
          bit2=0           ' set RA2 output data as logic '0'
                               ' (leave other bits alone)
          poke $05,B0      ' write PortA output data from B0

          goto Loop
```

**Figure 5.1:** This sample program toggles PA2 using PEEK and POKE instructions.

mend the use of a proper level-shifting serial interface, PicStic4 also works in other BASIC Stamp serial configurations.

## 5.2 PA2, PA3, AND PA4

Until the PA2, PA3, and PA4 I/O lines are directly accessible as port addresses in PicBasic, they are

available using the PEEK and POKE instructions. They are available directly from assembly language and C.

The PicBasic program in Figure 5.1 initializes PA2 as an output bit and then sets and resets it continuously. Care must be taken in selecting the correct bit and changing only that bit, both in the direction control port and in the output port.

## 6.0 I/O COPROCESSOR

PicStic4's expanded capabilities become available through its I/O coprocessor. Your PicBasic (and assembly language) programs have access to the additional I/O via the CALL statement. Three variables are used to pass and receive all data to and from the I/O coprocessor. The communication between processors runs at 62.5 kbps. To send a command and receive a reply requires less than 1 ms.

### 6.1 SENDING A COMMAND

Commands are always three bytes in length. The first byte is the command byte. The second and third bytes are used to pass a 16-bit parameter to the coprocessor. Command bytes which require no parameter are padded with null (don't care) data.

Variable

<u>Name</u>	<u>Contents</u>
B19	Command
B20	Parameter Low (low byte of a 16-bit value)
B21	Parameter High (high byte of a 16-bit value)

Once the three command variables are set, the user calls the I/O coprocessor. The coprocessor performs the command task and returns a reply in the same three variables. If the coprocessor has completed the task successfully, the first byte will contain the same command byte it was sent, with the following two bytes holding reply data. If the coprocessor cannot perform the command task as sent, it will return three 00 bytes.

Variable

<u>Name</u>	<u>Contents</u>
B19	Command Echo
B20	Reply Low (low byte of a 16-bit value)
B21	Reply High (high byte of a 16-bit value)

This Call routine, named PASS, is included in the software library on the diskette included with the PicStic4 development system.

### 6.2 EXAMPLE

Figure 6.1 shows an example of setting the configuration byte to support a keypad and LCD.

```

START:  B19 = $03: REM Command 3 = Set Configuration
        B20 = $00: REM LSByte of configuration word 2000h
        B21 = $20: REM MSByte of configuration word 2000h

        CALL PASS

        IF B19 <> $03 THEN ERROR: REM Good reply?

        SEROUT 7,N9600,("Done")
        END

ERROR:  SEROUT 7,N9600,("Error")
        END

```

**Figure 6.1:** This program shows an example of setting the configuration byte to support a keypad and LCD.

## 7.0 I/O COPROCESSOR COMMAND SET SUMMARY

The main PicStic4 processor and the I/O coprocessor communicate over a full-duplex asynchronous serial connection at 62.5 kbps. The I/O coprocessor uses a very simple binary protocol that maximizes throughput without sacrificing clarity.

All commands consist of the command byte and two

parameter bytes. All commands generate a reply consisting of an echo of the command byte followed by two data bytes. All parameters are passed in low-byte, high-byte order.

The leftmost column in the table labeled "Decimal" represents the decimal equivalent of the hexadecimal values found in the "Command" and "Reply" columns.

Decimal	Command	Parameter	Reply	Parameter	Function
0	00	xx xx	00	FF FF	Clear the communications channel
1	01	xx xx	01	vv vv	Read the firmware version number
2	02	xx xx	02	FF FF	Erase the entire EEPROM
3	03	cc dd	03	FF FF	Set configuration word to ddcc
4	04	xx xx	04	cc dd	Read configuration word
5	05	bb xx	05	FF FF	Set I/O port bit directions
6	06	xx xx	06	bb 00	Read I/O port bit directions
7	07	nn xx	07	FF FF	Set digital outputs
8	08	xx xx	08	nn 00	Read digital outputs
9	09	xx xx	09	nn 00	Read digital inputs
10	0A	cc xx	0A	nn mm	Read analog input channel cc
11	0B	cc xx	0B	FF FF	Set analog output channel pointer to cc
12	0C	nn mm	0C	FF FF	Set analog output channel to mmnn and increment pointer
13	0D	cc xx	0D	nn mm	Read analog output channel cc
14	0E	cc xx	0E	nn mm	Read analog input channel cc minimum value
15	0F	cc xx	0F	nn mm	Read analog input channel cc maximum value
16	10	cc xx	10	FF FF	Clear min and max values for analog input cc
17	11	xx xx	11	ff gg	Read the current input frequency
18	12	dd xx	12	FF FF	Set the totalizer debounce period
19	13	xx xx	13	tt uu	Read the totalizer count
20	14	xx xx	14	FF FF	Clear the totalizer count
21	15	tt uu	15	FF FF	Set the PWM total period value
22	16	hh ii	16	FF FF	Set the PWM high period value
23	17	xx xx	17	cc ii	Read the last decoded iButton serial number one byte at a time
24	18	xx xx	18	FF FF	Clear the LCD display
25	19	rr cc	19	FF FF	Position the LCD cursor on row rr and column cc

26	1A	cc xx	1A	FF FF	Turn the LCD cursor on (cc=1) or off (cc=0)
27	1B	cc dd	1B	FF FF	Send 1 or 2 characters to the LCD display
28	1C	xx xx	1C	kk 00	Read next keypad button pressed from buffer. If none, kk=FF
29	1D	xx xx	1D	FF FF	Clear the keypad buffer
30	1E	aa bb	1E	FF FF	Set EEPROM address pointer
31	1F	nn mm	1F	FF FF	Set next location in EEPROM
32	20	xx xx	20	nn mm	Read next location in EEPROM
33	21	xx xx	21	FF FF	Erase the EEPROM storage area
34	22	ss xx	22	FF FF	Set the shift register to ss
35	23	xx xx	23	ss 00	Read the current shift register setting
36	24	xx xx	24	mm yy	Read clock month and year
37	25	xx xx	25	ww dd	Read clock day of week and day of month
38	26	xx xx	26	mm hh	Read clock minute and hour
39	27	xx xx	27	ss 00	Read clock second
40	28	mm yy	28	FF FF	Set clock month and year
41	29	ww dd	29	FF FF	Set clock day of week and day of month
42	2A	mm hh	2A	FF FF	Set clock minute and hour

## 8.0 I/O COPROCESSOR COMMAND SET DETAILS

All commands consist of a command byte followed by a 16-bit parameter. The 16-bit parameter value is passed low byte first, followed by the high byte.

All replies consist of an echo of the command byte, followed by a 16-bit reply value. Again, the low byte is passed first, followed by the high byte. In commands that don't have any data to return, the reply value is FFFF to indicate no error. When an error does occur, the I/O coprocessor returns three 00 bytes (the command byte is not echoed in this case).

### Clear the communications channel (0) 00

It is important that the PicStic4 processor stay in sync with the I/O coprocessor. When in doubt, simply send zero bytes to the coprocessor until it replies with a valid reply. When a reply is received, the processors are in sync and communications may commence.

Cmd: **00 xx xx** where: xx = don't care

Reply: **00 FF FF** where: FF indicates no error

Hardware: All versions

Example: C: **00 00 00** R: **00 FF FF**

The communications channel between the PicStic4 processor and the I/O coprocessor is cleared and acknowledged.

### Set the configuration word (3) 03

bit 12: reserved  
 bit 13: enable LCD/keypad support  
 bit 14: reserved  
 bit 15: reserved

Reply: **03 FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **03 03 00** R: **03 FF FF**

The I/O coprocessor is configured for both 8- and 12-bit analog I/O.

---

## Read the module configuration (4) 04

While the I/O coprocessor supports many different kinds of I/O, only certain options may be used simultaneously. The PicStic4 may use command 04 to query the coprocessor to find out what options it has enabled. See command 03 for a description of the format of the data returned.

Cmd: **04 xx xx** where: *xx* = don't care  
 Reply: **04 cc dd** where: *cc* = option bitmap (low byte)  
                                   *dd* = option bitmap (high byte)

Hardware: All versions

Example: C: **04 00 00** R: **04 00 02**

The I/O coprocessor is configured for totalizer operation.

---

## Set the I/O port bit directions (5) 05

The I/O coprocessor's eight digital I/O lines (DIO0–DIO7) may be configured for either input or output on a bit-by-bit basis. An 8-bit value determines the bit directions. Putting a 1 in a bit position sets the corresponding I/O bit for input. A 0 sets the bit for output. Bit directions may be changed at any time.

Cmd: **05 bb xx** where: *bb* = bit directions (00–FF)  
                                   *xx* = don't care  
 Reply: **05 FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **05 F0 00** R: **05 FF FF**

Bits 0–3 of the I/O coprocessor's digital I/O port are configured as outputs while bits 4–7 are set up as inputs.

---

## Read the I/O port bit directions (6) 06

The I/O coprocessor's eight digital I/O lines (DIO0–DIO7) may be configured for either input or output on a bit-by-bit basis. This command returns the current settings of the eight I/O bit directions. A 1 in a bit position indicates the corresponding I/O bit is set for input. A 0 indicates the bit is set for output.

Cmd: **06 xx xx** where: *xx* = don't care  
 Reply: **06 bb nn** where: *bb* = bit directions  
                                   *nn* = 00

Hardware: All versions

Example: C: **06 00 00** R: **06 0F 00**

Bits 0–3 of the I/O coprocessor's digital I/O port are currently configured as inputs while bits 4–7 are outputs.

---

## Set the digital outputs (7) 07

The I/O coprocessor's eight digital outputs (DIO0–DIO7) are set to the given value. Any port bits configured as inputs are unaffected by this command.

Cmd: **07 nn xx** where: *nn* = output byte (00–FF)  
                                   *xx* = don't care  
 Reply: **07 FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **07 25 00** R: **07 FF FF**

Bits 0, 2, and 5 of the I/O coprocessor's digital I/O port are set high while the rest of the bits are set low.

**Read digital outputs** (8) 08

The current settings of the eight digital outputs (DIO0–DIO7) are returned as a single 8-bit value.

Cmd: **08 xx xx** where: *xx* = don't care  
 Reply: **08 mm nn** where: *mm* = current output  
                                   byte  
                                   *nn* = 00

Hardware: All versions

Example: C: 08 00 00      R: 08 34 00

Bits 2, 4, and 5 of the I/O coprocessor's digital I/O port are currently set high while the rest of the bits are low.

**Read digital inputs** (9) 09

The I/O coprocessor's eight digital inputs (DIO0–DIO7) are read and returned as a single 8-bit value. Any port bits configured as outputs return their current settings.

Cmd: **09 xx xx** where: *xx* = don't care  
 Reply: **09 mm nn** where: *mm* = current input  
                                   byte  
                                   *nn* = 00

Hardware: All versions

Example: C: 09 00 00      R: 09 C1 00

Bits 0, 6, and 7 of the I/O coprocessor's digital I/O port are currently high while the rest of the bits are low.

**Read analog input channel** (10) 0A

The I/O coprocessor supports four 8-bit (AIN0–AIN3) and two 12-bit (AIN4 and AIN55) analog inputs. The readings are returned as 16-bit values (low byte first), with any unused high-order bits set to zero. Both kinds of analog inputs support just 0–5 V on their inputs.

Cmd: **0A cc xx**    where:    *cc* = analog channel to  
                                       read (00–05)  
                                       *xx* = don't care

Reply: **0A nn nn** where: *nn* = analog input value

Hardware: All versions (AIN0–AIN3)  
PS4-Q, PS4-X (AIN4 and AIN5)

Example: C: 0A 04 00      R: 0A 2C 09

Analog input channel 4 currently has a voltage of 2.866 V applied to it (92C hex or 2348 decimal).

**Set the analog output channel pointer** (11) 0B

The I/O coprocessor supports two 12-bit digital-to-analog converters (AOUT0 and AOUT1). Since it's not possible to pass both the channel number and the data in a single command, a pointer is used to keep track of which DAC is to be referenced by the next DAC command. This command sets the pointer to the channel number of the DAC to be referenced by the next DAC command.

Cmd: **0B cc xx** where: *cc* = analog output  
channel (00–01)  
*xx* = don't care

Reply: **0B FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: 0B 01 00      R: 0B FF FF

The analog output channel pointer is set to 1 so the next command that references a DAC will access analog output channel 1.

**Set the next analog output channel** (12) 0C

Using the channel pointer set with command 0B, this command sets the target DAC to a specified value. The channel pointer is incremented after the target DAC is set, and the pointer wraps around to channel zero when the maximum channel number is reached. The low byte of the specified value should be sent first.

Cmd: **0C nn mm** where: *mmnn* = value to put in target DAC (000–FFF)

Reply: **0C FF FF** where: *FF* indicates no error

Hardware: PS4-X only

Example: C: 0C 7B 08 R: 0C FF FF

The analog output channel pointed to by the analog output pointer (set with command 0B) is set to 2.171 V (087B hex or 2171 decimal).





## Read the current input frequency (17) 11

A square wave with a frequency in the range of 2 Hz–2 kHz may be read on the DIO0 input. The value returned is a 16-bit hexadecimal value representing the period of the signal. To convert to a frequency, use the following:

$$\text{Frequency} = \frac{125000}{\text{Period}}$$

Cmd: **11 xx xx** where: *xx* = don't care  
 Reply: **11 ff gg** where: *ggff* = period of input frequency

Hardware: All versions

Example: C: **11 00 00** R: **11 73 4D**

A frequency of 5.04 Hz (4D73 hex or 19827 decimal) is being applied to the DIO0 pin.

## Set the totalizer debounce period (18) 12

When totaling incoming pulses, the signal edges can be quite ragged if something like a switch is producing them. The result would be several counts of every edge if the input wasn't debounced. A debounce period of up to 255 ms may be set to clean up those ragged edges. Note that the longer the period, the lower the effective frequency that can be handled on the input.

Cmd: **12 dd xx** where: *dd* = debounce period in milliseconds (00–FF)  
*xx* = don't care  
 Reply: **12 FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **12 1C 00** R: **12 FF FF**

The debounce period is set for 28 ms.

## Read the totalizer count (19) 13

The totalizer keeps a running count of the number of pulses that have been received on the DIO0 input. The counter can be queried at any time using this command. The value returned is a 16-bit value representing the number of pulses received since the last time the totalizer was cleared. Command 14 must be used to clear the count.

Cmd: **13 xx xx** where: *xx* = don't care  
 Reply: **13 cc dd** where: *ddcc* = number of pulses received

Hardware: All versions

Example: C: **13 00 00** R: **13 2A 00**

The I/O coprocessor has received 42 pulses since the count was last cleared.

## Clear the totalizer count (20) 14

When enabled, the totalizer continues counting incoming pulses even after the totalizer has been read, eventually rolling over after FFFF. This command clears the totalizer count.

Cmd: **14 xx xx** where: *xx* = don't care  
 Reply: **14 FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **14 00 00** R: **14 FF FF**

The totalizer count is cleared.

## Set the PWM total period value (21) 15

When the PWM output is enabled and a high period has already been set, this command causes the PWM output (AIN3) to become active and generate a waveform with the given total period and high period. If a high period hasn't been set yet, the total period value will be stored and the PWM output will become active when the high period is set with command 16.

Convert between period, frequency, and duty cycle using the equations below. Approximate range is 2 Hz–3.5 kHz, 5–95% duty cycle.

$$tp = \frac{125000}{\text{Frequency}} \quad hp = \text{Duty Cycle} \times tp$$

$$\text{Frequency} = \frac{125000}{tp} \quad \text{Duty Cycle} = \frac{hp}{tp}$$

Cmd: **15 tt uu** where: *uutt* = total period (tp)  
Reply: **15 FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **15 00 7D** R: **15 FF FF**  
Assuming the high period has been set, a 3.9-Hz (7D00 hex or 32000 decimal) waveform is produced on the PWM output pin.

## Set the PWM high period value (22) 16

When the PWM output is enabled and a total period has already been set, this command causes the PWM output (AIN3) to become active and generate a waveform with the given total period and high period. If a total period hasn't been set yet, the high period value will be stored and the PWM output will become active when the total period is set with command 15.

Convert between period, frequency, and duty cycle using the equations shown under command 15. Approximate range is 2 Hz–3.5 kHz, 5–95% duty cycle.

Cmd: **16 hh ii** where: *iihh* = high period (hp)  
Reply: **16 FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **16 53 2C** R: **16 FF FF**

Assuming the total period has been set to 7D00 hex (32000 decimal), a 3.9-Hz waveform with a 35.5% duty cycle is produced on the PWM output pin.

## Read iButton serial number (23) 17

When an iButton device is touched to the reader, PicStic4 reads the serial number from the device and saves it. Command 17 returns the last read serial number one byte at a time, along with a counter that tells what byte number of the serial number it is and whether it's new or old data. If multiple devices are read between queries, only the last is retained.

An iButton serial number consists of a device family byte, six serial number bytes, and a checksum, for a total of eight bytes. The I/O coprocessor numbers the bytes 1–8. A specific byte may be requested by passing the byte number as the command parameter. If byte 0 is requested, the next byte in sequence is returned.

The first byte returned contains the byte number being returned. The second byte contains the actual data. If the high-order bit of the byte number is clear, the data is new and hasn't been read before. If the bit is set, the data has already been read and is considered old.

Cmd: **17 bb xx** where: *bb* = byte number (00–07)  
*xx* = don't care

Reply: **17 bb dd** where: *bb* = byte number (00–07)  
*dd* = data byte

Hardware: All versions

Example: C: **17 01 00** R: **17 01 02**

The iButton serial number family code is requested and a new family code value of 02 hex is returned.

Example: C: **17 00 00** R: **17 02 C6**

The next byte of the iButton serial number is requested and a new second-byte value of C6 hex is returned.

Example: C: **17 01 00** R: **17 81 02**

The family code is requested again and an old family code value of 02 hex is returned.

---

## Clear the LCD display (24) 18

When optional LCD display hardware is connected to the I/O coprocessor, this command clears the display.

Cmd: **18 xx xx** where: *xx* = don't care  
Reply: **18 FF FF** where: *FF* indicates no error

Hardware: All versions with optional LCD display hardware

Example: C: **18 00 00** R: **18 FF FF**  
The LCD display is cleared.

---

## Position the LCD cursor (25) 19

The LCD cursor may be positioned anywhere on the display to allow updating of just certain areas of the display.

Cmd: **19 rr cc** where: *rr* = new cursor row position (00–03)  
*cc* = new cursor column position (00–13)

Reply: **19 FF FF** where: *FF* indicates no error

Hardware: All versions with optional LCD display hardware

Example: C: **19 02 0E** R: **19 FF FF**  
The LCD cursor is placed on row 2 (the third line) and column 14 (0E hex).

---

## Turn the LCD cursor on or off (26) 1A

There are times when you may not want the LCD cursor to be displayed. It is easy to turn the cursor on and off whenever necessary. Characters sent to the display are still displayed at the next cursor position whether or not it's currently on.

Cmd: **1A cc xx** where: *cc* = 00 to turn the cursor off or 01 to turn the cursor on

Reply: **1A FF FF** where: *FF* indicates no error

Hardware: All versions with optional LCD display hardware

Example: C: **1A 01 00** R: **1A FF FF**  
The LCD display cursor is turned on.

---

## Display one or two characters (27) 1B on the LCD display

Display one or two ASCII characters on an LCD display (if the module is configured for LCD support). If the second character is 0, only the first character is put on the display.

Cmd: **1B cc dd** where: *cc* = first character to be displayed  
*dd* = second character to be displayed

Reply: **1B FF FF** where: *FF* indicates no error

Hardware: All versions with optional LCD display hardware

Example: C: **1B 48 69** R: **1B FF FF**  
The string "Hi" is displayed on the LCD display at the current cursor position. Note that 48 hex (72 decimal) is an ASCII "H" and 69 hex (105 decimal) is an ASCII "i".

---

## Read the next keypad button (28) 1C

Up to eight keypad presses are buffered. This command retrieves the next button from the buffer. If no key has been pressed, a value of FF hex is returned. See Figure A.1 in the Appendix regarding keycodes returned.

Cmd: **1C xx xx** where: *xx* = don't care  
Reply: **1C kk 00** where: *kk* = code for key pressed (00–17, FF=no keypress)

Hardware: All versions with optional keypad hardware

Example: C: **1C 00 00** R: **1C 03 00**  
The "3" button on the keypad was the next key pressed.

---

## Clear the keypad buffer (29) 1D

Up to eight keypad presses are buffered by the I/O coprocessor. This command will clear the keypad buffer of any unread presses.

Cmd: **1B xx xx** where: *xx* = don't care  
 Reply: **1B FF FF** where: *FF* indicates no error

Hardware: All versions with optional keypad hardware

Example: C: **1D 00 00** R: **1D FF FF**  
 The keypad buffer is cleared.

---

## Set the EEPROM address pointer (30) 1E

Since it's not possible to specify an address and data in the same command when writing to the I/O coprocessor EEPROM, an address pointer is used. This command is used to set the EEPROM address pointer. The new address pointer value is used on all subsequent EEPROM reads and writes.

Cmd: **1E aa bb** where: *baaa* = new address pointer value (0000–03FF)  
 Reply: **1E FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **1E 1B 02** R: **1E FF FF**  
 The EEPROM address pointer is set to 021B hex (539 decimal). The next EEPROM read or write will access that address.

---

## Write to the EEPROM (31) 1F

Write a 16-bit data value to the next EEPROM location and increment the EEPROM address pointer. The target location must already have been erased with either command 02 or command 21.

Cmd: **1F nn mm** where: *mmnn* = data to be written to EEPROM (0000–FFFF)  
 Reply: **1F FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **1F E8 69** R: **1F FF FF**

A 16-bit data value of 69E8 hex (27112 decimal) is written to the current EEPROM location and the EEPROM address pointer is incremented.

---

## Read from the EEPROM (32) 20

Read a 16-bit data value from the next EEPROM location and increment the EEPROM address pointer.

Cmd: **20 xx xx** where: *xx* = don't care  
 Reply: **20 nn mm** where: *mmnn* = data read from EEPROM (0000–FFFF)

Hardware: All versions

Example: C: **20 00 00** R: **20 82 1F**

A 16-bit data value of 1F82 hex (8066 decimal) is read from the current EEPROM location and the EEPROM address pointer is incremented.

---

## Erase the user EEPROM area (33) 21

While most of the EEPROM is available for user storage, a portion is used by the I/O coprocessor to store configuration information. This command erases just the user area of the EEPROM, leaving the configuration information alone. See command 02 for information on erasing the entire EEPROM.

Cmd: **21 xx xx** where: *xx* = don't care  
 Reply: **21 FF FF** where: *FF* indicates no error

Hardware: All versions

Example: C: **21 00 00** R: **21 FF FF**  
 The user storage area of the EEPROM is erased.





## 9.0 COMPATIBLE FEATURES

While the PicStic4 supports many different features, not all those features may be used simultaneously. The module has limited resources and in many cases uses a single pin for more than one function.

When issuing the set configuration (command 03) command, you're free to specify any or all options. However, not all option selections are compatible with each other. Fortunately, the PicStic4 is smart enough not to allow you to enable two mutually exclusive functions. When in doubt, configure the module as you want it, then read the configuration back using the 04 (read configuration) command. The result will be the actual module configuration.

PicStic4 comes in three flavors: P, Q, and X. Please note that when we speak of configuration options on any device, it only refers to configuration of the coprocessor. Configuring the coprocessor has no effect on the processor section. It is also not necessary to configure the coprocessor at all if none of its functions are to be used. The default configuration of all PicStic4 modules is for parallel I/O and 8-bit analog to be enabled. The real-time clock (Q and X versions) is always available and not considered a configuration option. Reading and setting it, however, uses the same PASS program as all other coprocessor communications.

### 9.1 PICSTIC4 P

This minimal version of PicStic4 supports 19 bidirectional digital lines and four 8-bit analog inputs. All digital and analog I/O may be used simultaneously.

When you select other configurations such as touch memory or PWM, the 8-bit analog input lines are used for these functions and 8-bit analog capability is sacrificed.

#### Compatible Configurations:

- Digital Inputs and Digital Outputs
- Digital Inputs, Digital Outputs, and 8-bit Analog Inputs
- Digital Inputs, Digital Outputs, and Touch Memory
- Digital Inputs, Digital Outputs, Digital Shift Register
- LCD/Keypad
- LCD/Keypad and Touch Memory
- Digital Inputs, Digital Outputs, and Frequency Input
- Digital Inputs, Digital Outputs, and PWM Output

### 9.2 PICSTIC4 Q AND X

Giving up the 8-bit analog I/O to use the firmware function built into PicStic4 is offset by the addition of 12-bit analog inputs and a real-time clock. The PicStic4 X has 12-bit analog outputs in addition to all the functions of the Q unit. The real-time clock is not set on modules as received from the factory. References to 12-bit analog below refer to 12 analog inputs on the Q version and both analog input and outputs on the X version.

#### Compatible Configurations:

- Digital Inputs and Digital Outputs
- Digital Inputs, Digital Outputs, and 8-bit Analog Inputs
- Digital Inputs, Digital Outputs, and Touch Memory
- Digital Inputs, Digital Outputs, Digital Shift Register
- LCD/Keypad
- LCD/Keypad and Touch Memory
- Digital Inputs, Digital Outputs, and Frequency Input
- Digital Inputs, Digital Outputs, and PWM Output
- Digital Inputs, Digital Outputs, 8-bit Analog Inputs, and 12-bit Analog
- Digital Inputs, Digital Outputs, Touch Memory, and 12-bit Analog
- Digital Inputs, Digital Outputs, Digital Shift Register, and 12-bit Analog
- LCD/Keypad, 12-bit Analog
- LCD/Keypad, Touch Memory, and 12-bit Analog
- Digital Inputs, Digital Outputs, Frequency Input, and 12-bit Analog
- Digital Inputs, Digital Outputs, PWM Output, and 12-bit Analog



## 10.0 DIGITAL AND ANALOG I/O

All PicStic4s support TTL level bidirectional digital I/O lines. There are 11 bidirectional I/O lines from the processor and 8 bidirectional I/O lines from the coprocessor. The PicStic4 also supports an external shift register attached to pins 13, 14, and 20 which accommodate either 8 or 16 additional output bits. All the bidirectional I/O lines are available for use during most modes of operation. The shift register outputs are available in lieu of coprocessor 8-bit analog operation.

All PicStic4s have 4 channels of 8-bit ADC. PicStic4Q also includes a 2-channel 12-bit ADC and PicStic4X has both a 2-channel 12-bit ADC and DAC. Many coprocessor functions require the 8-bit ADC lines to be reconfigured as digital I/O.

### 10.1 HARDWARE

The bidirectional digital I/O lines are TTL input-compatible. As outputs they have limited high and low drive capability for direct connections to such things as LEDs and relays. The PicStic4 has a total package rating for maximum available output current. This value should not be exceeded or the module may be damaged.

The shift register outputs are TTL-compatible. The user must include drivers on these outputs when interfacing to devices that require higher currents or voltages than a TTL line can handle. Damage caused by applying a high voltage to an input or drawing too much current from an output is **not** covered by warranty.

The 8- and 12-bit analog inputs all accept voltages in the range of 0–5 V. A smaller range may be used, but you'll be giving up some resolution by not scaling the signal to use the full 0–5-V range. Take care not to exceed 5 V on any analog input. Damage caused by applying a high voltage to an input is **not** covered by warranty.

The 12-bit analog outputs produce a voltage in the range of 0–4.096 V. Be sure to use a buffer between the

DAC output and the device being controlled if that device has a lower input impedance than 600  $\Omega$  or needs more current than the DAC can provide. Damage caused by drawing too much current from an output is **not** covered by warranty.

### 10.2 SOFTWARE

The PicStic4 digital I/O operates at different speeds depending on whether the I/O is on the processor or coprocessor section. The highest speed is attained with a program directly controlling the processor I/O bits. Because the the I/O coprocessor communicates serially with the processor and requires a 3-byte command to set or reset any coprocessor bit, or read the ADC, any coprocessor related function will be delayed by the time associated for that communication. PicStic4's analog I/O is all directed by the coprocessor. The processor and coprocessor communicate at 62.5 kbps. Typical ADC or DAC program execution speed is less than 1 ms per reading or output update.

Typically, the inputs or outputs may be written to or read at any time. However, when using specific configurations that use the I/O port pins (e.g., LCD mode) or the 8-bit ADC lines (such as PWM or iButton), reading or writing the ports directly may not be relevant. See Section 9 for more details about when the digital I/O ports or analog functions may be used concurrently with other features.

The 12-bit ADC and DAC can be used with virtually all other firmware functions. Enabling 12-bit operation, however, still requires that the coprocessor receive the proper configuration parameters.

For the 12-bit analog I/O, values consist of 16-bit hexadecimal numbers where the upper 4 bits are zero. 8-bit analog inputs use the same 16-bit hexadecimal representation with the upper 8-bits being zero.

## 11.0 PWM OUTPUT

The PicStic4 can be used to generate a variable-frequency, variable-duty-cycle square wave, often called a pulse-width modulated waveform. By specifying the total waveform period and the duration of the high portion of the period, any frequency from 2 Hz to 3.5 kHz with a duty cycle of 5–95% may be synthesized.

### 11.1 HARDWARE

The PWM waveform comes from the AIN3 line (pin 19) of a PicStic4. No additional circuitry is required.

To enable the PWM output, set bit 2 of the configuration word. Until waveform parameters have been set

(see section 8), the PWM output will remain idle, even after being enabled.

### 11.2 SOFTWARE

To specify the waveform parameters, use command 21 to set the total period value and command 22 to set the high period value. See Section 8 for more details.

The parameters set using these commands only stay in effect until power is removed from the PicStic4. When power is reapplied, the PWM output will remain idle, even if it's enabled in the configuration word

---

## 12.0 FREQUENCY INPUT

PicStic4 can be used to measure the frequency of a square wave. Any frequency from 2 Hz to 2 kHz may be measured.

### 12.1 HARDWARE

The waveform should be applied to the DIN0 line (pin 28) of a PicStic4. All other digital input lines remain available. No other circuitry is required.

### 12.2 CONFIGURATION

The frequency input feature is enabled by setting bit 3 of the configuration word.

### 12.3 SOFTWARE

Command 17 is used to query the current frequency input measurement. The value returned is actually the period of the waveform. See Section 8 for more details.

---

## 13.0 TOTALIZER INPUT

Rather than measuring the time between pulses as is done with the frequency feature, the totalizer feature can be used to count the number of pulses received over a known time period. Up to  $2^{16}$  (65,536) pulses can be counted at rates up to 500 Hz.

### 13.1 HARDWARE

The waveform should be applied to the DIO0 line (pin 17). All other digital I/O lines remain available. No other circuitry is required.

### 13.2 CONFIGURATION

The totalizer input feature is enabled by setting bit 9 of the configuration word.

### 13.3 SOFTWARE

Command 18, 19, and 20 are used to set the debounce period, clear the count, and query the current totalizer count. The count is *not* automatically cleared when read.

## 14.0 TOUCH MEMORY AND iBUTTON

Dallas Semiconductor makes a line of small devices the size of button-cell batteries that can be used to hold data. Referred to as iButton or Touch Memory, every one of these devices contains a unique 48-bit serial number laser etched into its silicon. The PicStic4 can read the serial number from most members of the Touch Memory family.

Touch Memory uses a unique one-wire interface that makes inserting the device into a reader quick and foolproof. An example application is a door-access controller. Every person authorized for entry into a particular area has a Touch Memory device attached to their ID badge, on their keychain, or in a ring worn on their finger. To unlock the door, they press the Touch Memory device into the reader mounted next to the door. The PicStic4 reads the device and reports the serial number back to the main computer. The computer looks up the number to find out who the person is and issues a command to the PicStic4 to unlock the door if a match is found.

For more information on Touch Memory and the various devices available, contact Dallas Semiconduc-

tor, 4401 S. Beltwood Pkwy., Dallas, TX 75244-3292, and ask for the *Automatic Identification Data Book*.

### 14.1 HARDWARE

The PicStic4's Touch Memory interface requires the addition of just a Touch Memory receptacle and a single pull-up resistor (on pin 20).

### 14.2 SOFTWARE

Command 17 retrieves the data read from the last Touch Memory device to the Touch Memory input. See section 8 for details on the procedure for reading the device.

Specific members of the iTouch Memory family supported include: DS1990A, DS1991, DS1992, DS1993, DS1994, DS1995, DS1996, DS1982, DS1985, DS1986, and DS1920. The original DS1990 is *not* supported.

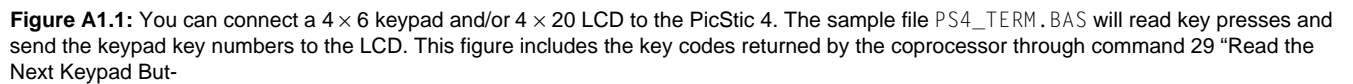
Note that many of the above devices include features besides the serial number, such as RAM and temperature sensing. None of these additional features may be accessed using the PicStic4.

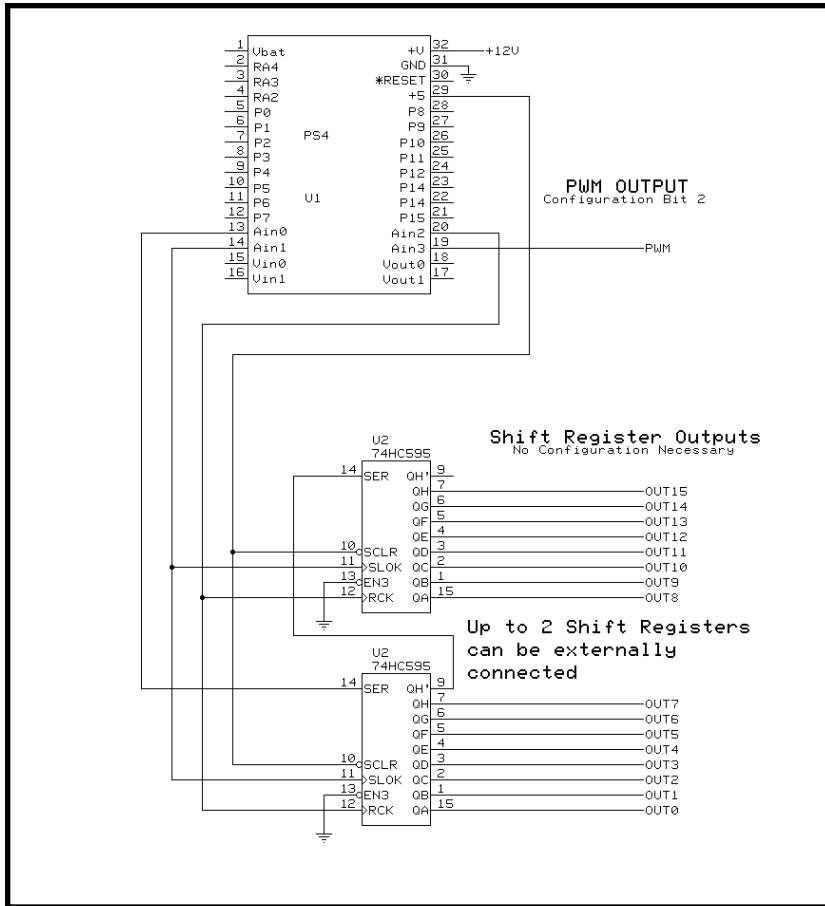
Devices sold by Micromint are covered by the warranty and patent indemnification provisions appearing in its Terms of Sale only. Micromint makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. Micromint makes no warranty of merchantability or fitness for any purposes. Micromint reserves the right to discontinue production and change specifications and prices any time and without notice. This product is intended for use in normal commercial applications. Applications requiring extended temperature and unusual environmental requirements, or applications requiring high reliability, such as military, medical life support or life-sustaining equipment, are specifically **not** recommended without additional processing by Micromint for such application.

# Micromint Chips

Micromint, Inc.  
4 Park St.  
Vernon, CT 06066  
(860) 871-6170  
Fax: (860) 872-2204  
[www.micromint.com](http://www.micromint.com)

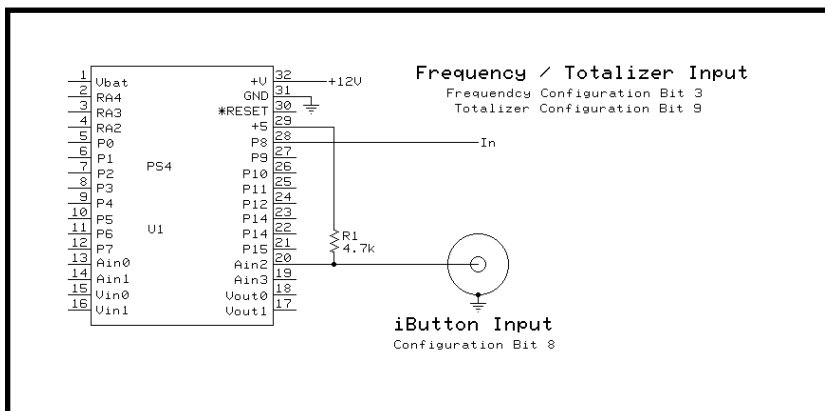
The following schematics demonstrate how to connect to the PicStic 4 to make use of the additional functions provided by the I/O coprocessor. The program listings are provided on the development system diskette.





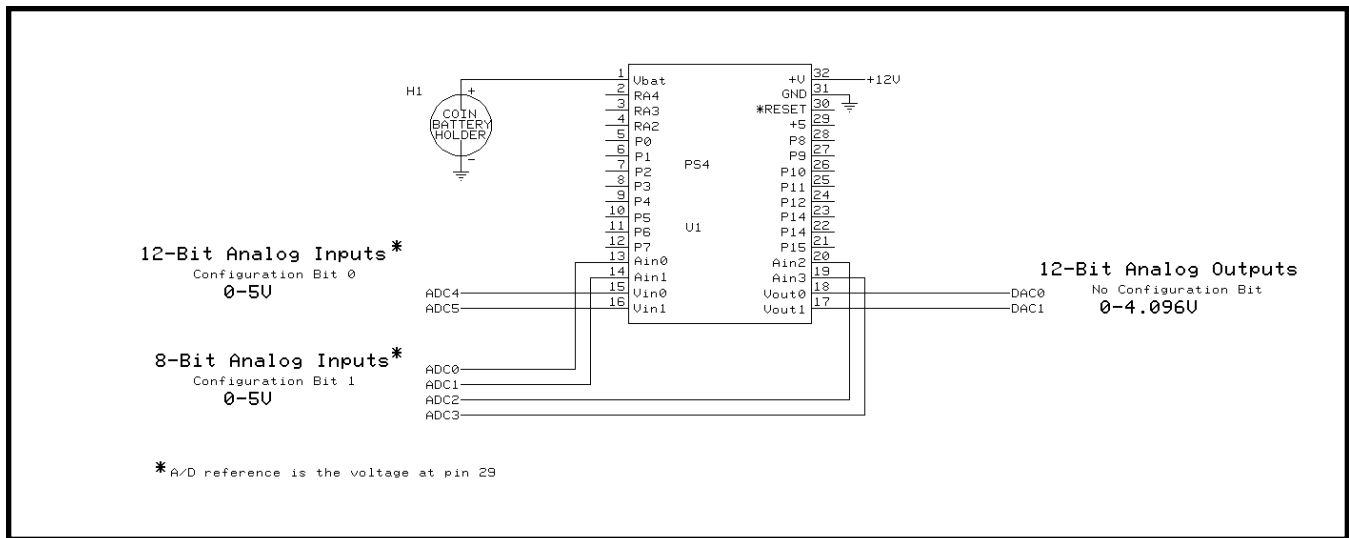
**Figure A1.2:** Here you see where to obtain the PWM output. Also, you see how to attach up to two external shift registers for an additional 16 output bits. (Shift register outputs are available only when the shift register out function is selected.)

The sample file `PS4_OUTS.BAS` allows you to test these output functions. PWM is set to continuously vary the pulse width of a set frequency. 16-bit data is continuously written out the shift register port.

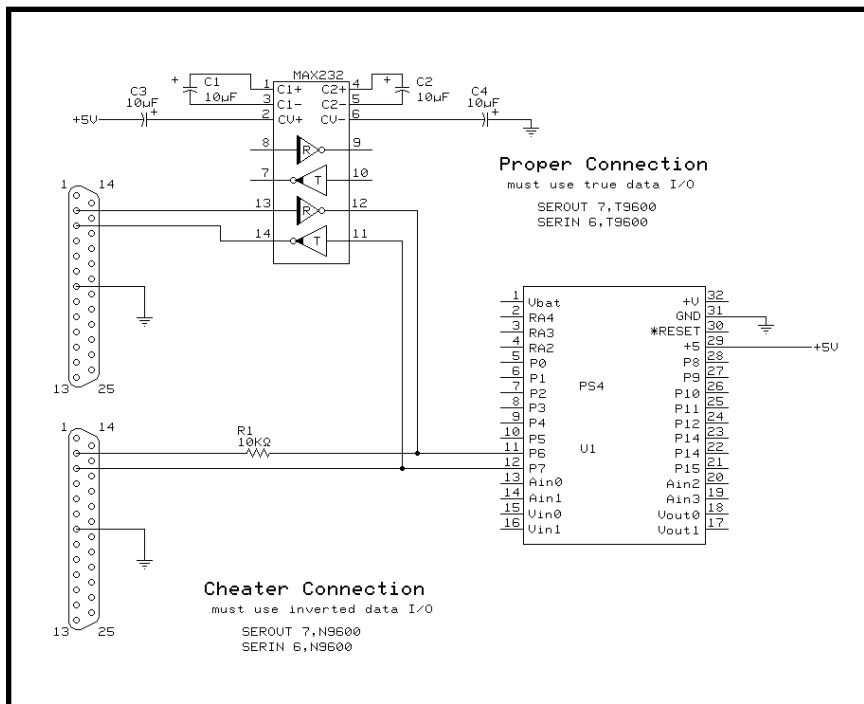


**Figure A1.3:** This schematic indicates the connections for frequency and totalizer input. iButton input requires an additional pull-up resistor. (iButton is available only when 8-bit A/D inputs are disabled.)

The sample file `PS4_INS.BAS` shows how to use the input functions. Frequency is calculated on request, while the totalizer function tallies input changes between requests. iButton input is reported whenever a new button is presented to the iButton socket.



**Figure A1.4:** This schematic shows external battery connection necessary for the real-time clock/calendar. External battery voltage equals 3 V (lithium or alkaline cells). Additionally, analog 8- and 12-bit inputs are indicated as well as 12-bit analog outputs (8-/12-bit inputs 0-5 ( $V_{CC}$ ) V, 12-bit outputs 0-4.096 V). The sample program PS4\_ADDA.BAS samples and displays voltage levels presented at the A/D inputs. PS4\_DAC.BAS sets up the DACs with continuous ramping outputs.



**Figure A1.5:** This figure demonstrates how serial communication connection can be made to your PC. The proper connection uses a MAX232 to derive  $\pm 10$  V for RS-232C communications. The cheater connection can be used when only temporary or short distance RS-232A communications is used.

NOTE: This connection may not work with every PC.

NOTE: SEROUT and SERIN routines are different for each connection type. DEMO programs are written for the cheater connection.