Project 3: Analog to Digital Conversion
MAE 5483

Charles O’Neill
13 October 2004

Introduction

This project’s purpose is to become familiar with analog to digital conversion on a PIC micro-controller. Four programs demonstrate: assembly timing, A/D conversions, A/D conversion rates, and RC circuit time constants.

The micro-controller is a 28 pin DIP PIC16F876 manufactured by Microchip. The PIC voltage input is +5 volts DC via a $\mu$A7805 dc/dc voltage regulator. The compiler is the CCS C compiler (v. 3.207) for 14 bit PIC chips. C compilation occurs on a x86 based PC. Data transfer between the PIC and the PC is through a 9 pin serial cable. All programming and experiments were performed by Charles O’Neill.

Figure 1 shows the circuit schematic for this project. LED is active low and connected to pins B7. For all PIC inputs, 470 Ohm resistors restrict the current to a maximum of 10 mA. A 5K potentiometer is used to provide 0–5 Volt source into pin A0 of the A/D converter. Program 3 uses an RC circuit connected to pin B4 —variable resistor and capacitor components were used. The external button is connected to pin B0, the external interrupt pin. Pushing the single pole momentary button shorts pin B0 to ground. Otherwise, a hold-up resistor maintains about 5 volts on pin B0.

![Figure 1: Schematic](image-url)
Test equipment consisted of an Agilent 54621D 60 MHz Oscilloscope with Analog and Digital Input, and a RadioShack 22-805 multimeter. The Oscilloscope appeared to have some problems with zero offset voltages.

1 Calculation Timer

The objective of this program is to investigate calculation times with respect to the underlying assembly code. The C code is given in the adc-verify.c Code Listings section (10). Timer1 was used for added robustness since timer0 overflows too often for both good resolution and a large unique range.

The program performs addition, increments, subtraction, multiplication, and division for 8bit, 16bit, 32bit and floating point data types. A summary of the required times is:

<table>
<thead>
<tr>
<th>Calculation Timer:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer 8bit:</td>
<td></td>
</tr>
<tr>
<td>Add: 10 Ticks / 5 Calcs --&gt; .4 microseconds</td>
<td></td>
</tr>
<tr>
<td>Add++: 5 Ticks / 5 Calcs --&gt; .2 microseconds</td>
<td></td>
</tr>
<tr>
<td>Subtract: 10 Ticks / 5 Calcs --&gt; .4 microseconds</td>
<td></td>
</tr>
<tr>
<td>Multiply: 230 Ticks / 5 Calcs --&gt; 9.1 microseconds</td>
<td></td>
</tr>
<tr>
<td>Divide: 490 Ticks / 5 Calcs --&gt; 19.6 microseconds</td>
<td></td>
</tr>
<tr>
<td>Integer 16bit:</td>
<td></td>
</tr>
<tr>
<td>Add: 30 Ticks / 5 Calcs --&gt; 1.2 microseconds</td>
<td></td>
</tr>
<tr>
<td>Add++: 15 Ticks / 5 Calcs --&gt; .6 microseconds</td>
<td></td>
</tr>
<tr>
<td>Subtract: 30 Ticks / 5 Calcs --&gt; 1.2 microseconds</td>
<td></td>
</tr>
<tr>
<td>Multiply: 1115 Ticks / 5 Calcs --&gt; 44.6 microseconds</td>
<td></td>
</tr>
<tr>
<td>Divide: 1730 Ticks / 5 Calcs --&gt; 69.2 microseconds</td>
<td></td>
</tr>
<tr>
<td>Integer 32bit:</td>
<td></td>
</tr>
<tr>
<td>Add: 70 Ticks / 5 Calcs --&gt; 2.7 microseconds</td>
<td></td>
</tr>
<tr>
<td>Add++: 40 Ticks / 5 Calcs --&gt; 1.6 microseconds</td>
<td></td>
</tr>
<tr>
<td>Subtract: 70 Ticks / 5 Calcs --&gt; 2.7 microseconds</td>
<td></td>
</tr>
<tr>
<td>Multiply: 2745 Ticks / 5 Calcs --&gt; 109.8 microseconds</td>
<td></td>
</tr>
<tr>
<td>Divide: 5660 Ticks / 5 Calcs --&gt; 226.3 microseconds</td>
<td></td>
</tr>
<tr>
<td>Floating Point:</td>
<td></td>
</tr>
<tr>
<td>Add: 679 Ticks / 5 Calcs --&gt; 27.1 microseconds</td>
<td></td>
</tr>
<tr>
<td>Subtract: 761 Ticks / 5 Calcs --&gt; 30.4 microseconds</td>
<td></td>
</tr>
<tr>
<td>Multiply: 2330 Ticks / 5 Calcs --&gt; 93.2 microseconds</td>
<td></td>
</tr>
<tr>
<td>Divide: 6415 Ticks / 5 Calcs --&gt; 256.5 microseconds</td>
<td></td>
</tr>
<tr>
<td>e^x: 40895 Ticks / 5 Calcs --&gt; 1635.8 microseconds</td>
<td></td>
</tr>
<tr>
<td>log: 519 Ticks / 5 Calcs --&gt; 20.7 microseconds</td>
<td></td>
</tr>
<tr>
<td>sqrt: 7900 Ticks / 5 Calcs --&gt; 315.9 microseconds</td>
<td></td>
</tr>
<tr>
<td>cos: 46105 Ticks / 5 Calcs --&gt; 1844.1 microseconds</td>
<td></td>
</tr>
</tbody>
</table>

Addition and subtraction are cheap; multiplication and division are more expensive; special function are exorbitant. In general, doubling the bits for integer
operations requires approximately double the time. Floating point is an order of magnitude slower than integer operations.

Snippets of 8bit assembly listing produced by the compiler is shown below. Addition requires 2 clock cycles, which is 0.4 microseconds\(^1\) —exactly corresponding to the previous timing experiment. Increments are 1 cycle. Multiplication and division require CALL operations.

```
.............. a = a + b; // 5x Additions
O50F: MOVF 32, W
0510: ADDWF 31, F
.............. a++; // 5x Additions
0548: INCF 31, F
.............. a = a - b; // 5x Subtractions
057C: MOVF 32, W
057D: SUBWF 31, F
```

For 16bit data types, the assembly for addition and subtraction are shown below. Addition requires 6 cycles; Incrementing requires 3 cycles. Referring to the timing experiment shows an exact match.

```
.............. a = a + b; // 5x Additions
0816: MOVF 33, W
0817: ADDWF 31, F
0818: BTFSC 03.0
0819: INCF 32, F
081A: MOVF 34, W
081B: ADDWF 32, F
.............. a++; // 5x Additions
0867: INCF 31, F
0868: BTFSC 03.2
0869: INCF 32, F
```

32 bit operations are similar. Floating point requires significantly more time because the PIC does not have an on-board floating point processor. For simple integer operations, the assembly code and the actual timing match.

## 2 Analog to Digital Verification

This program performs and evaluates an analog to digital conversion(ADC). The C code is given in the adc-verify.c Code Listings section (17). The objective is to demonstrate the PIC’s onboard A/D converter.

The program’s heart is reading and converting the analog input to a 16bit digital representation. A loop continuously performs the conversions.

```c
for (;;) {
    adc_int = read_adc();
    value = (float) adc_int * (float) ADC.RES;
}
```

\(^1\)0.2 microseconds per cycle
Figure 2 shows an 8bit visual representation of the 16bit A/D conversion output for a somewhat linearly increasing input voltage. The lowest bit changes for every change in digital voltage representation. Each successively higher bit changes half as often.

Proper reference voltages appear critically important. The reference voltage pins AN2 and AN3 were connected to 0 and 5 volts. The following 16F876.h header file shows some of the ADC connection options.

```
#define NO_ANALOGS 7 // None
#define ALL_ANALOG 0 // A0 A1 A2 A3 A5
#define AN0_AN1_AN2_AN4_VSS_VREF 1 // A0 A1 A2 A5 VRefh=A3
#define AN0_AN1_AN3 4 // A0 A1 A3
#define AN0_AN1_VSS_VREF 5 // A0 A1 VRefh=A3
#define AN0_AN1_AN4_VREF_VREF 0x08 // A0 A1 A5 VRefh=A3 VRefl=A2
#define AN0_AN1_VREF_VREF 0x0D // A0 A1 VRefh=A3 VRefl=A2
#define AN0 0x0E // A0
#define AN0_VREF_VREF 0x0F // A0 VRefh=A3 VRefl=A2
```

This project uses AN0_AN1_VREF_VREF for analog inputs AN0 and AN1 with references AN2 and AN3.

The multimeter and oscilloscope voltages are plotted against the A/D converter’s voltage in Fig. 3. The oscilloscope’s measurements are disappointing...the far cheaper multimeter has a performance which is more ‘linear’ with respect to the A/D conversion. Comparing the A/D converter and the multimeter gives a maximum error of 0.03 volts, far larger than the $5.0/1024 = 0.004$ Volt A/D resolution. A quick check of Figure 2 shows that all of the lower bits are working. Also, the raw integer 16bit number can be smoothly varied without gaps. Providing a different reference voltage didn’t improve the bias. Artificially adjusting the `#define ADC_RES 5.0/1024.0` again can not adjust the bias. The A/D converter appears precise with respect to the 10 bits but not accurate with respect to other measurements devices.
3 Analog to Digital Timing

The program investigates A/D timing. The A/D converter triggers off either an internal RTCC clock or directly off the main 20MHz oscillator. The objective is to determine conversion properties with various A/D settings. Two timing methods are used: internal software timers and an oscilloscope.

For the internal software timing method, timer1 is used to determine the time interval between a block of code.

```c
  set_timer1(0);
  adc_int=read_adc();  //10x ADC
  ...
  adc_int=read_adc();
  timer1=get_timer1();
```

The A/D timing output for the 10 bit conversion with an rtcc interrupt every 256 ticks is:

- 23.2 microseconds 1.376 Volts RTCC_DIV_1
- 23.4 microseconds 1.376 Volts RTCC_DIV_2
- 21.8 microseconds 1.376 Volts RTCC_DIV_3
- 21.5 microseconds 1.376 Volts RTCC_DIV_4
- 38.8 microseconds 1.376 Volts ADC_CLOCK_DIV_2
- 4.7 microseconds 2.500 Volts ADC_CLOCK_DIV_8
- 7.8 microseconds 1.376 Volts ADC_CLOCK_DIV_32

The surprise is the 4.7 microsecond sample timing causes an incorrect A/D conversion. From the datasheet, the internal RC timer source has a 2 to 6 microsecond sample time. The minimum time appears to be approximately 6 microseconds per conversion or 170KHz.
Changing to 8bit conversion yields significantly faster rate.

2.7 microseconds 1.367 Volts RTCC_DIV_1
3.0 microseconds 1.367 Volts RTCC_DIV_2
1.3 microseconds 1.367 Volts RTCC_DIV_3
1.2 microseconds 1.367 Volts RTCC_DIV_4
4.0 microseconds 1.367 Volts ADC_CLOCK_DIV_2
3.7 microseconds 2.500 Volts ADC_CLOCK_DIV_8
2.6 microseconds 1.367 Volts ADC_CLOCK_DIV_32

This gives a maximum rate of about 700KHz.

The oscilloscope method is next. This method determines the period between a port’s low and high outputs. Figure 4 shows the oscilloscope output of 40 microseconds for RTCC_DIV_1 with 10 bit conversion. For 16 bit conversion,

![Figure 4: Oscilloscope Method](image)

the oscilloscope timings are:

40 microseconds RTCC_DIV_1
40 microseconds RTCC_DIV_2
38 microseconds RTCC_DIV_3
40 microseconds RTCC_DIV_4
40 microseconds ADC_CLOCK_DIV_2
6 microseconds ADC_CLOCK_DIV_8
10 microseconds ADC_CLOCK_DIV_32

The 8 bit oscilloscope timings are:

40 microseconds RTCC_DIV_1
38 microseconds RTCC_DIV_2
38 microseconds RTCC_DIV_3
38 microseconds RTCC_DIV_4
40 microseconds ADC_CLOCK_DIV_2
4 microseconds ADC_CLOCK_DIV_8
8 microseconds ADC_CLOCK_DIV_32

The software timing appears lower than the oscilloscope timing.
4 Simple Time Constant Identification with an ADC

This program investigates the identification of a 1st order system’s time constant. The C code is given in the adc-id.c Code Listings section (22). The objective is to determine the time constant for an RC circuit.

4.1 Governing Equations

The governing equation for the resistor/capacitor circuit shown in Figure 1 is found by summing currents around the circuit. The governing equation is:

\[ C \frac{dv}{dt} + \frac{v}{R} = 0 \]

This 1st order equation has a simple solution for the natural response—an initial voltage of \( V_0 \).

\[ v(t) = V_0 e^{-t/\tau} \quad \text{with} \quad \tau = RC \]

Rearranging this equation yields \( \tau \), the time constant.

\[ \tau = \frac{t}{\ln \frac{v(0)}{v(t)}} \]

Waiting \( T \) seconds for the voltage to drop from \( V_0 \) to \( V_S \) gives a time constant of:

\[ \tau = T \left( \ln \frac{v(0)}{v(t)} \right)^{-1} \]

Since \( v(0)/v(t) \) tends to zero, a shorter—but not too short—switch time \( T \) is desired.

4.2 Implementation

The adc-id.c implements the time constant estimate derived above. The process is to charge the capacitor and measure the time response. When the voltage drops below a digital input pin’s critical trigger voltage, the PIC reports the time elapsed. Then the time constant is estimated from above.

First, the program sets up the A/D converter. If needed, a chance to calibrate the system is provided. Otherwise, the program uses a hard coded trigger voltage. Timer1 provides a 24 bit range with the addition of an overflow interrupt service routine. Time is scaled back to dimensional seconds with both timer and overflow information.

```c
#include <avr/io.h>

#define PIN 3

void timer1_int (void)
{
    timer1_overflows++;
}
```
The heart of the program is below. Time Calculations

```c
/* Float pin and start timing */
timer1_overflows=0;
set_timer1(0);
output_low(PIN_B4);
while(input(PIN_B4)); // Wait for Pin to go low
timer1 = get_timer1();
Time = (float) timer1 + TIMER1_MAX * (float) timer1_overflows) * TIME_SCALE;
tau = Time/log(Volt_Zero/Volt_Switch);
```

4.3 Results

The program returned reasonable results. The first test is with a 1uF capacitor and a 10.0 Ohm resistor for a time constant of 0.01 seconds. The program reports:

\[
\text{Tau} = .009836 \text{ seconds}
\]

With a 1uF capacitor and a 98.2 Ohm resistor, the time constant is 0.0982 seconds —requiring timer1 overflow.

\[
\text{adc-id --- Determines the time constant of a RC circuit}
\]
\[
\text{Calibrate ? (y/n)}
\]
\[
\text{Ready to test.... Waiting for trigger}
\]
\[
\text{Tau} = .098679 \text{ seconds}
\]

A larger 100uF capacitor and a 10.0 KOhm resistor gives a time constant of 0.98 seconds. The program reports:

\[
\text{Tau} = .848622 \text{ seconds}
\]

The difference is probably in the capacitor’s internal resistance since the decay required almost 1 second.

This program appeared robust enough for additional experiments. One example is determining the effective capacitor leakage resistance. With only the capacitor in the circuit, a 1uF capacitor’s RC ‘time constant’ is:

\[
\text{Tau} = 1.634420 \text{ seconds}
\]

This gives an effective capacitor resistance of 1.6 MOhm. Returning to the 100uF capacitor discussed above, the experimentally determined internal resistance is too low to determine with this setup.
4.4 Another Method

Another method for determining the time constant is to directly output many voltage and time measurements and perform a time domain system identification routine. Code is included in the adc-id.c source for time domain identification with a series of time and voltages. However, the theory is not as trivial as the time constant method.

Conclusions

Four programs were created to test timing and Analog to Digital conversions. The programs experimented with: PIC assembly code, A/D conversion, conversion rates, and RC circuit time constants. The A/D converter works, but has some accuracy problems when compared to other voltage measurement devices.
/*
 * calc_timer.c ——- Times various operations
 *
 * Charles O’Neill
 * MAE 5483
 * Project 3.1
 */

/* Default PIC Initialization */
#include <16F876.h>
#include <stdlib.h>
#include <math.h>
#define delay (clock=2000000)
#define use delay (clock=2000000)
#define fuses HS, NOWDT
#define use rs232 (baud=19200, parity=N, xmit=PIN_C6, rcv=PIN_C7)

/* Global Variables */
#define TIMEROVERHEAD 2 // Ticks to Move Timer to a Register
#define TIME_SCALE 5 // 2E6/4 ticks per microsecond

/* Global Variables */
char type[10];

/* Function Prototypes */
void printout (int16 time, int calcs, char* type);
void integer8 (void);
void integer16 (void);
void integer32 (void);
void floats (void);

/* Main Program for calculation timer */
void main()
{
    /* Inform the user what is happening. */
    printf ("\n\r\n\rCalculation Timer: ");
/* Setup Timer */
setup_timer_1(T1INTERNAL | T1_DIV_BY_1);

/* 8 bit Integer */
integer8();

/* 16 bit Integer */
integer16();

/* 32 bit Integer */
integer32();

/* Floating Point */
floats();

/* Printout Function ---- saves RAM/ROM */

void printout(int16 time , int calcs , char* type){
    printf("\n\nrt%s :\t%lu Ticks / %d Calcs -- > %5.1f microseconds", type, time , calcs , (float) time / (float) TIME_SCALE / (float) calcs);
}

/* Integer 8 bit */

void integer8(void){
    int16 timer1;
    int a = 1;
    int b = 1;

    /* Print Calculation's Data Type */
    printf("\n\nrtInteger 8 bit: ");

    /* Addition */
    set_timer1(0);
    a = a + b;  // 5x Additions
    a = a + b;
    a = a + b;
    a = a + b;
    timer1 = get_timer1() - TIMER_OVERHEAD;
    strcpy(type , "Add");
    printout(timer1 , 5 , type);

    /* Addition+ + */
```c
set_timer1(0);
a++;  // 5x Additions
a++;
a++;
a++;
timer1=get_timer1() - TIMEROVERHEAD;
strcpy(type, "Add++");
printout(timer1, 5, type);

/* Subtract */
set_timer1(0);
a = a - b;  // 5x Subtractions
a = a - b;
a = a - b;
a = a - b;
a = a - b;
timer1=get_timer1() - TIMEROVERHEAD;
strcpy(type, "Subtract");
printout(timer1, 5, type);

/* Multiply */
set_timer1(0);
a = a * b;  // 5x Multiply
a = a * b;
a = a * b;
a = a * b;
a = a * b;
timer1=get_timer1() - TIMEROVERHEAD;
strcpy(type, "Multiply");
printout(timer1, 5, type);

/* Divide */
set_timer1(0);  // 5x Divide
a = a / b;
a = a / b;
a = a / b;
a = a / b;
a = a / b;
timer1=get_timer1() - TIMEROVERHEAD;
strcpy(type, "Divide");
printout(timer1, 5, type);
}

/**************************************************************************
* Integer 16 bit
**************************************************************************/

void integer16(void){
    int16 timer1;
}
```c
int16  a = 1;
int16  b = 1;

/* Print Calculation’s Data Type */
printf("\n\nInteger 16 bit: ");

/* Addition */
set_timer1(0);
a = a + b;    // 5x Additions
a = a + b;
a = a + b;
a = a + b;
timer1=get_timer1()-TIMER_OVERHEAD;
strcpy(type, "Add");
printout(timer1, 5, type);

/* Addition++ */
set_timer1(0);
a++;   // 5x Additions
a++;
a++;
a++;
a++;
timer1=get_timer1()-TIMER_OVERHEAD;
strcpy(type, "Add++");
printout(timer1, 5, type);

/* Subtract */
set_timer1(0);
a = a - b;    // 5x Subtractions
a = a - b;
a = a - b;
a = a - b;
a = a - b;
timer1=get_timer1()-TIMER_OVERHEAD;
strcpy(type, "Subtract");
printout(timer1, 5, type);

/* Multiply */
set_timer1(0);
a = a * b;    // 5x Multiply
a = a * b;
a = a * b;
a = a * b;
a = a * b;
timer1=get_timer1()-TIMER_OVERHEAD;
strcpy(type, "Multiply");
printout(timer1, 5, type);
```
/* Divide */
set_timer1(0);    // 5x Divide
a = a / b;
a = a / b;
a = a / b;
a = a / b;
a = a / b;
timer1 = get_timer1() - TIMEROVERHEAD;
strcpy(type, "Divide");
printout(timer1, 5, type);

……………………………………………………………………………………………………
……………………………………………………………………………………………………
*/ Integer 32 bit */
……………………………………………………………………………………………………
……………………………………………………………………………………………………
/* void integer32(void) */
……………………………………………………………………………………………………
……………………………………………………………………………………………………
int16 timer1;
int32 a = 1;
int32 b = 1;
……………………………………………………………………………………………………
……………………………………………………………………………………………………
/* Print Calculation's Data Type */
……………………………………………………………………………………………………
……………………………………………………………………………………………………
printf("%n%rInteger 32 bit");
……………………………………………………………………………………………………
……………………………………………………………………………………………………
/* Addition */
set_timer1(0);
a = a + b;    // 5x Additions
a = a + b;
a = a + b;
a = a + b;
a = a + b;
timer1 = get_timer1() - TIMEROVERHEAD;
strcpy(type, "Add");
printout(timer1, 5, type);
……………………………………………………………………………………………………
……………………………………………………………………………………………………
/* Addition++ */
set_timer1(0);
a++;    // 5x Additions
a++;
a++;
a++;
a++;
timer1 = get_timer1() - TIMEROVERHEAD;
strcpy(type, "Add++");
printout(timer1, 5, type);
……………………………………………………………………………………………………
……………………………………………………………………………………………………
/* Subtract */
set_timer1(0);
a = a - b;    // 5x Subtractions
a = a - b;
a = a - b;
a = a - b;
a = a - b;
timer1 = get_timer1() - TIMER_OVERHEAD;
strcpy(type, "Subtract");
printout(timer1, 5, type);

void floats(void){
    int16 timer1;
    float a = 1;
    float b = 1;

    /* Print Calculation's Data Type */
    printf("\nFloating Point: ");

    /* Addition */
    set_timer1(0);
    a = a + b;   // 5x Additions
    a = a + b;
    a = a + b;
    a = a + b;
}
timer1 = get_timer1() - TIMEROVERHEAD;
strcpy(type, "Add");
printout(timer1, 5, type);

/* Subtract */
set_timer1(0);
a = a - b;   // 5x Subtractions
a = a - b;
a = a - b;
a = a - b;
a = a - b;
timer1 = get_timer1() - TIMEROVERHEAD;
strcpy(type, "Subtract");
printout(timer1, 5, type);

/* Multiply */
set_timer1(0);
a = a * b;   // 5x Multiply
a = a * b;
a = a * b;
a = a * b;
a = a * b;
timer1 = get_timer1() - TIMEROVERHEAD;
strcpy(type, "Multiply");
printout(timer1, 5, type);

/* Divide */
set_timer1(0);   // 5x Divide
a = a / b;
a = a / b;
a = a / b;
a = a / b;
a = a / b;
timer1 = get_timer1() - TIMEROVERHEAD;
strcpy(type, "Divide");
printout(timer1, 5, type);

/* exp() */
set_timer1(0);   // 5x exp
a = exp(b);
a = exp(b);
a = exp(b);
a = exp(b);
a = exp(b);
timer1 = get_timer1() - TIMEROVERHEAD;
strcpy(type, "e^x");
printout(timer1, 5, type);

/* log() */
```c
set_timer1(0); // 5x exp
a = log(b);
a = log(b);
a = log(b);
a = log(b);
a = log(b);
timer1=get_timer1()-TIMER_OVERHEAD;
strcpy(type, "log");
printout(timer1, 5, type);

/* sqrt() */
set_timer1(0); // 5x sqrt
a = sqrt(b);
a = sqrt(b);
a = sqrt(b);
a = sqrt(b);
a = sqrt(b);
timer1=get_timer1()-TIMER_OVERHEAD;
strcpy(type, "sqrt");
printout(timer1, 5, type);

/* cos() */
set_timer1(0); // 5x cos
a = cos(b);
a = cos(b);
a = cos(b);
a = cos(b);
a = cos(b);
timer1=get_timer1()-TIMER_OVERHEAD;
strcpy(type, "cos");
printout(timer1, 5, type);
```

```c
adC-verify.c

/*
 *      adC-verify   Verify the ADC is working correctly
 *      
 *      Charles O’Neill
 *      MAE 5483
 *      Project 3.2
 * /

/*----------------------------------------------------------
 * Default PIC Initialization
 *----------------------------------------------------------*/
#include <16F876.h>
device AIX=10
use delay(clock=2000000)
```
```c
#define ADC_RES (5.0/1024.0)

void main()
{
    int16 adc_int;
    float value;

    setup_adc_ports( AN0, AN1, VREF, VREF );
    setup_adc( ADC_CLOCK_INTERNAL );
    setup_counters( RTCC_INTERNAL, RTCC_DIV_2);

    set_adc_channel( 0 );
    delay_us(5);
    set_tris_c(0x7f);

    for (;;)
    {
        output_low(PIN_B7);
        adc_int=read_adc();
        delay_us(250);
        output_high(PIN_B7);
        delay_us(250);

        value = (float) adc_int * (float) ADC_RES;
        printf( "/n\r_id=%f", adc_int, value);
    }
}
```

adc-time.c
/* Default PIC Initialization */
#include <16F876.h>
#define device ADC=10
#define use delay(clock=20000000)
#define fuses HS,NOWDT
#define use rs232(baud=19200, parity=N, xmit=PIN_C6, rcv=PIN_C7)

/* Global Defines */
#define ADC_RES 5.00/256.0
#define TIME_SCALE 0.2
#define TIMER1_MAX 65536.0

void statechange(int timer_state);
int timer1_overflows = 0;
int B_button_trigger = 0;

/* RTCC Interrupt */

#define rtcc
void timer0_int(void){
    set rtcc(128);
}

/* Timer1 Interrupt */

#define timer1
void timer1_int(void){
    timer1_overflows++;
}

/* External Interrupt */

#define ext
void button_int(void){
    B_button_trigger ^= 1; // Change Button State
}

/* Main Program */
void main(){
    /* Variables */
    int State = 1;
```c
int adc_int;
int timer1;
float Voltage;
float Time;

/* Initialize Timers */
setup_timer1(T1_INTERNAL | T1_DIV_BY_1);
setup_counters( RTCC_INTERNAL , RTCC_DIV_1);

/* Analog to Digital Conversion */
setup_adc_ports( AN0_AN1_VREF_VREF );
setup_adc( ADC_CLOCK_INTERNAL );
read_adc(ADC_START_AND_READ);
set_adc_channel( 0 );
delay_us(5);

/* Initialize Interrupts */
enable_interrupts(INT_EXT);
enable_interrupts(INT_RTCC);
enable_interrupts(INT_TIMER1);
enable_interrupts(GLOBAL);

/* Continuous Loop */
printf("\n\r\nStarting Loop");
while(State < 10){
    // Test Block
    output_low(PIN_B7);
    adc_int=read_adc(); //10x ADC
    adc_int=read_adc();
    adc_int=read_adc();
    adc_int=read_adc();
    adc_int=read_adc();
    adc_int=read_adc();
    adc_int=read_adc();
    adc_int=read_adc();
    adc_int=read_adc();
    output_high(PIN_B7);
    // Switching
    if(B_button_trigger) {
        delay_ms(20);
        B_button_trigger=0;
        statechange(State++);
        printf("\n\r");
    }
}
for(State=1;State <10;State++){
    // Test Block
```
timer1_overflows = 0;
set_timer1(0);
adc_int=read_adc();  //10x ADC
adc_int=read_adc();
adc_int=read_adc();
adc_int=read_adc();
adc_int=read_adc();
adc_int=read_adc();
adc_int=read_adc();
timer1=get_timer1();

// Switching and Evaluation
Time = ((float)timer1 + TIMER1_MAX * (float)timer1_overflows) * TIME_SCALE;
Voltage = (float)adc_int * ADC_RES;
printf("\n\r%5.1f microseconds %5.3f Volts",Time/10.0,Voltage);
statechange(State);
}

/* ADC State Changer/*/ 

void statechange(int timer_state){
  // Change the Timer Enroute
  switch(timer_state){
    case 1: setup_counts(RTCCINTERNAL, RTCCDIV1); 
      printf("RTCCDIV1");
      break;
    case 2: setup_counts(RTCCINTERNAL, RTCCDIV2);
      printf("RTCCDIV2");
      break;
    case 3: setup_counts(RTCCINTERNAL, RTCCDIV4);
      printf("RTCCDIV4");
      break;
    case 4: setup_counts(RTCCINTERNAL, RTCCDIV8);
      printf("RTCCDIV8");
      break;
    case 5: setup_counts(RTCCINTERNAL, RTCCDIV128);
      printf("RTCCDIV128");
      break;
    case 6: setup_counts(RTCCINTERNAL, RTCCDIV2);
      setup_adc(ADC_CLOCKINTERNAL);
      printf("ADC_CLOCKINTERNAL");
      break;
    case 7: setup_adc(ADC_CLOCKDIV2);
      printf("ADC_CLOCKDIV2");
  }
```c
break;
case 8: setup_adc(ADC_CLOCK_DIV_8);
    printf("ADC_CLOCK_DIV_8\n");
    break;
case 9: setup_adc(ADC_CLOCK_DIV_32);
    printf("ADC_CLOCK_DIV_32\n");
    break;
case 10: for (;;);
    break;
}
```

```c
ADC
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
B_button_trigger ^= 1;  // Change Button State

delay_ms(50);

/∗ Timer1 Interrupt ∗/  
#include	imer1

void timer1_int(void){
    timer1_overflows++;
}

float test(float Volt_Switch, int Pin);
void time_series(float Tguess, int Pin);
float calibrate(int Pin);

/∗ Main Program ∗/  

void main(){

    /* Variables */
    float
    Volt_Switch, tau;

    /* Inform the User */
    printf("adc_id−−−Determines the time constant of an RC circuit");

    /* Setup ADC port */
    setup_adc_ports( AN0, AN1, VSS, VREF );
    setup_adc( ADC_CLOCK_INTERNAL );
    setup_counters( RTCC_INTERNAL, RTCC_DIV_1 );
    set_adc_channel( PIN_A0 );
    delay_us(5);

    /* Initialize Interrupts */
    enable_interruptions(INT_EXT);  // Port B Interrupt
    enable_interruptions(INT_TIMER1);
    enable_interruptions(GLOBAL);  // Turn on Interrupts

    /* Initialize Timers */
    setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);

    /* Set the PIC's TriState I/O */
    set_tris_b(0x7f);

    /* Calibrate Pin Switching Voltage */
    printf("Calibrate? (y/n)");
    if(getchar()=='y'){

23
```c

Volt_Switch=calibrate(5);
} else {
    Volt_Switch=VOLTAGE_SWITCH;
}

/* Test circuit */
tau=test(Volt_Switch, 7);
printf("\n\n\rTau=%f seconds\n", tau);

/* Method 2 */
time_series(tau, 7);
printf("Done!!\n");

}*/

/* Test

float test(float Volt_Switch, int Pin){

    int16   adc_int;
    float   Volt_Zero;
    float   tau, Time;
    int16   timer1;

    /* Allow user to reconfigure circuit */
    printf("\n\rReady to test .... Waiting for trigger");
    while(!B_button_trigger);

    /* Charge Capacitor */
    output_high(PIN_B4); // Charge Capacitor
delay_ms(1000); // for a sufficient time

    /* Get initial Voltage */
    Volt_Zero= (float)read_adc() * ADC_RES;

    /* Float pin and start timing */
    timer1_overflows=0;
    set_timer1(0);
    output_low(PIN_B4);
    while(input(PIN_B4)); // Wait for Pin to go low
    timer1 = get_timer1();
    Time   = ((float) timer1 + TIMER1_MAX * (float) timer1_overflows) *
              TIME_SCALE;
    tau    = Time/log(Volt_Zero/Volt_Switch);
    return(tau);

*/
```
/ * Calibrate */
float calibrate (int Pin) {
    float Volt_Switch;


    /* Pin Numbers*/
    printf("n
r Calibrating.... Adjust voltage and press button.");

    /* Calibrate Pin */
    while(!B_button_triggered){
        if(input(PIN_B4)) {
            output_low(PIN_B7); // Light LED
        } else {
            output_high(PIN_B7); // Dark LED
        }
    }
    B_button_triggered = 0;
    Volt_Switch = (float)read_adc() * ADC_RES;
    printf("n
r Finished Calibrating %f Volts", Volt_Switch);
    return (Volt_Switch);
}

/* Test Series Method */
void time_series (float Tguess, int Pin) {
    int16 adc_int;
    float Time, Voltage;
    int16 Voltage_integer [TIME_SAMPLES];
    int Delay;
    int16 timer1 [TIME_SAMPLES];
    int timer1_over [TIME_SAMPLES];
    int index = 0;

    /* Allow user to reconfigure circuit */
    printf("n
r Ready to test .... Waiting for trigger");
    B_button_triggered = 0;
    while(!B_button_triggered);

    /* Charge Capacitor */
    output_high(PIN_B4); // Charge Capacitor


```c
delay_ms(1000); // for a sufficient time

/* Timing */
Delay=(int)(Tguess/(TIME_SCALE*(TIME_SAMPLES*100.0)));
printf("\n\n\n\Using\n\n\sample\delta\nt of: \nd\microseconds\n\n", Delay);

/* Float pin and start timing */
timer1_overflows=0;
set_timer1(0);
output_low(PIN_B4);

/* Float pin and stop timing */
for(index=0; index<=TIME_SAMPLES−1; index++){
    delay_us(Delay);
    timer1[index] = get_timer1();
    Voltage_integer[index] = read_adc();
    timer1_over[index] = timer1_overflows;
}
for(index=0; index<=TIME_SAMPLES−1; index++){
    Time= (timer1[index] + TIMER1_MAX * timer1_over[index]) * TIME_SCALE;
    Voltage = Voltage_integer[index] * ADC_RES;
    printf("\n\n\n\Time: %f\n\nVoltage: %f", Time, Voltage);
}
```