

Using a PIC for Analog to Digital Conversion

Luke LaPointe

November 12, 2009

Executive Summary: Analog to Digital Conversion or ADC is mandatory when dealing with analog signals that must be stored and analyzed by a computer. This application note will discuss how A/D conversion works, programming the design onto a PIC, and then transmitting it serially to a computer.

Keywords: DSPIC30F2020, ADC, Analog to Digital Conversion, PIC

Table of Contents

Introduction	Error! Bookmark not defined.
Objective	3
Issues	3
Accuracy	3
Configure A/D Module	3
• ADCON	3
○ ADON	3
○ ADSIDL	3
○ FORM	3
○ EIE	3
○ ORDER	4
○ SEQSAMP	4
○ ADCS<2:0>	4
• ADSTAT	4
• ADPCFG	5
• ADCPC0	5
○ IRQEN0	5
○ PEND0	5
○ SWTRG0	5
○ TRGSRC0<4:0>	5
Configuring the Timer	5
Baud Rate	5
• U1BRG	5
Design	6
Results & Conclusion	8
References	9

Introduction

Analog to Digital Conversion or ADC essentially takes a continuous signal and turns it into a discrete signal that a computer can use and manipulate. The digitizing aspect is based on the sample rate (determined by the clock cycle frequency of the PIC) and number discrete levels that the A/D conversion module is capable of. The higher the sample rate, the more data points can be collected in a smaller amount of time and thus making the digital signal appear less choppy. Similarly, the more discrete levels, the more accurate each step in amplitude of the digital signal will be. A good ADC will have both a high sample rate and lot of discrete levels.

Objective

With our project, we will be using a programming a Peripheral Interface Controller or PIC to convert the analog signals into something digital that our computer interface will be able to use. Specifically, we need the ability to be able to capture a DC step input, triangle wave, and AC sine wave on our PIC and then transfer that data to a computer interface for analysis.

Issues

Accuracy:

We are using the DSPIC30F2020 which has a 10 bit addressing, which means we can represent a signal with 2^{10} or 1024 discrete levels. So with a 5 volt signal, we can accurately store the data in between 0 volts and 5 volts with $5/(1024) = 4.88 \text{ mV}$ steps. This results in a loss in accuracy when performing A/D conversion. The higher the bit addressing is or the smaller the signal is can improve the accuracy of the output.

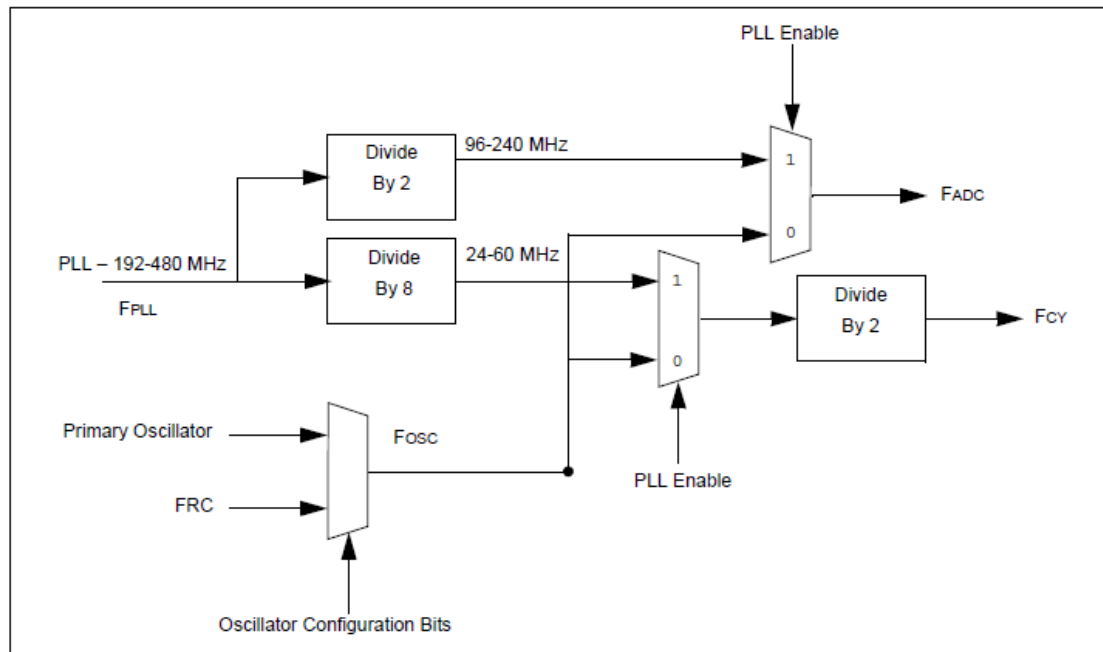
Configure A/D Module:

The DSPIC30F2020 has an A/D module already in place but in order to use it the necessary control registers have to be set.

- **ADCON** or the A/D Control Register, which consists of 16 bits.
 - **ADON:** A/D Operating Mode bit. Setting this bit to a 1 means that the A/D converting module is enabled and 0 disables it. This bit is usually turned on after all the other configuration bits are set.
 - **ADSIDL:** Stop in Idle Mode bit. 0 means it will operate in idle mode and 1 means that A/D conversion will stop once idle mode has been entered.
 - **FORM:** Data Output Format bit. Setting this to 1 means that the output will be displayed as a fraction and 0 will display as an integer.
 - **EIE:** Early Interrupt Enable bit. Setting this to 1 will send an interrupt after the first A/D conversion is completed, whereas 0 will interrupt after the second interrupt. We will be using 0 in our project, because we want to convert two signals at the same time before sending them to the computer. This bit can only be changed when ADON is set to 0.

- **ORDER:** Conversion Order bit. This specifies whether the even or odd analog input is converted first. Setting to a 0 means that the even input will be converted first then the odd and 1 is vice versa.
- **SEQSAMP:** Sequential Sample Enable. Setting this bit to a 0 will mean that the shared S&H is sampled concurrently with the dedicated S&H, unless the shared S&H is busy. In this case, then the shared S&H will sample once a new conversion is started. We will be using 0 for our project for this reason.
- **ADCS<2:0>:** A/D Conversion Clock Divider Select bits. This is based on the Frequency that the A/D module is set to and whether or not PLL is enabled. We have PLL enabled and wished to have a frequency divider of $F_{ADC}/14$, so we set this bit to 5 or 101 in binary. Assuming the PIC is setup to use FRC with Hi-Range and have an un-tuned frequency of 14.55 MHz, then $F_{CY} = 14.55 \text{ MHz} * 2 = 29.1 \text{ MHz}$ or 29.1 Million Instructions Per Second (MIPS). This means that F_{ADC} should be $F_{CY} * 8 = 232.8 \text{ MHz}$. Thus the resulting frequency divider $F_{ADC}/14 = 232.8 \text{ MHz}/14 = 16.6 \text{ MHz}$ or 16.6 MIPS. This will be sufficient for our purposes. Figure 18-2 shows a derivation of the frequency of the system clock or F_{CY} . An alternative way to determine F_{CY} is to output the PIC's clock and measure its frequency on the oscilloscope.

FIGURE 18-2: SYSTEM CLOCK AND FADC DERIVATION



1

- **ADSTAT** or the A/D Status Register which also has 16 bits, but only the 6 least significant bits (LSB) are usable. These bits essentially specify which pairs are ready for conversion or display the status of the conversion process. These must be cleared in the ADC interrupt after the conversion is complete.

- **ADPCFG** or the A/D Port Configuration Register is 16 bits and specifies which channels will be used as analog inputs. The DSPIC30F2020 has 12 available channels which is represented by the 12 LSB of the ADPCFG register. Setting a specific bit to 1 implies that the port pin is in Digital mode, port read input is enabled, and the A/D input multiplexor is connected to ground. A 0 implies that the port pin is in analog mode, port read input is disabled, and the A/D samples pin voltage. For our system, we want AN0 and AN1 as analog inputs, so we would set this register 1111 1100.
- **ADCPC0** or the Convert Pair Control Register #0 is a 16 bit register that controls the triggering of the A/D conversions.
 - **IRQEN0**: Interrupt Request Enable 1 bit. Setting this to 1 means that an IRQ will be generated when the requested A/D conversion from channels AN1 and AN0 is complete. For our project, we are using the AN0 and AN1 bits as analog inputs and would therefore set this bit high.
 - **PEND0**: Pending conversion of status 0 bit. When this bit is set high, that means the channels AN1 and AN0 are currently being converted. When the bit is low, conversion in these channels has completed.
 - **SWTRG0**: Software Trigger 0 bit. Setting this bit high essentially acts as a start button for A/D conversion in channels AN0 and AN1, given there are the resources available. This bit will be reset low when PEND bit is set high.
 - **TRGSRC0<4:0>**: Trigger 0 Source Selection bits. This essentially lets the A/D module know where to expect the trigger to come from for channels AN1 and AN0. For our project, we will trigger off a timer 1 period match. This corresponds to 01100, but there are many other options for triggering.

Configuring the Timer:

In order to use timer 1 to trigger the A/D conversion process on a timer1 period match, we have to set the period to something. If our $F_{CY}=29.1\text{MHz}$ or 29.1 MIPS, then our corresponding period $T_{CY}=1/F_{CY}=34.36\text{nsec}$. In order to ensure accurate A/D conversion we should set our A/D period or $T_{AD} = 21 * T_{CY} = 21 * 34.36\text{nsec} = 721.64 \text{ nsec}$ and we should set the period of timer 1 to be 100 times larger than T_{CY} . $T_{T1} = T_{CY} * 100 = 3.34 \text{ usec}$. In order to use the timer, some configuration bits need to be set.

Baud Rate:

In order for the PIC to communicate with a computer interface, both systems need to know the rate at which characters will be transferred known as the baud rate and it has its own register on the PIC.

- **U1BRG** or the UART Baud Rate Generator is a 16 bit register that can be calculated using the following formula. $U1BRG = (F_{CY} / [16 * \text{BaudRate}]) - 1$. For our project, we wanted a baud rate of 9600 and we are operating FRC_HI_RANGE which means that our clock cycle frequency is 19.7 MHz. $U1BRG = (19.7 \text{ MHz} / [16 * 9600]) - 1 = 127.25$. Therefore, we should assign the integer value 127 to this register in order to communicate with the computer interface with a baud rate of 9600.

Design

The following code was developed by Design Team 2 and demonstrates A/D conversion on channel AN0 of the PIC triggered by a timer period 1 match and sent serially through the UART port to a computer interface.

```
#include "p30f2020.h"
#define TIMER_PERIOD 0x0064          // Set the timer period for 3.43 usec
_FOSC(CSW_FSCM_OFF & FRC_HI_RANGE & OSC2_IO); //Oscillator Config
_FOSCSEL(FRC_PLL);                  //Oscillator Selection
_FWDT(FWDTEN_OFF);                  //Turn off WatchDog Timer
_FGS(CODE_PROT_OFF);                //Turn off code protect
_FPOR( PWRT_OFF );                  //Turn off power up timer
_FBS(BSS_NO_FLASH)

int main()
{
    //Setting up Interrupts
    U1MODEbits.UARTEN = 1; //enable UART
    U1MODEbits.STSEL = 0; //1 stop bit to end transmission
    U1MODEbits.PDSEL = 000; //0 parity bits
    U1MODEbits.BRGH = 0; //High Baud Rate Enable bit(set standard)
    U1MODEbits.RXINV = 0; // 1 is idle bit
    U1MODEbits.LPBACK = 0; //Disable loopback mode
    U1MODEbits.USIDL = 0; //Continue UART in idle mode
    U1MODEbits.ALTIO = 0; //Use U1TX and U1RX not U1ATX and U1ARX
    U1MODEbits.IREN = 0; //Disable IrDA encoder and decoder
    U1BRG = 127; //9600 baud rate
    U1MODEbits.ABAUD = 0;
    U1STAbits.UTXEN = 1; //Enable transmissions
    IEC0bits.U1TXIE = 1; //Enable transmit interrupts
    U1STAbits.UTXBRK = 0; //Disabling break sequence
    U1STAbits.UTXISEL1 = 0; //Interrupt generated when char is written to buffer
    U1STAbits.UTXISEL0 = 0;
    IEC0bits.U1RXIE = 1; //UART interrupt receiving is enabled
    IFS0bits.U1RXIF = 0; //initialize receiver to 0
    IFS0bits.U1TXIF = 0; //initialize transmitter to 0
    U1STAbits.URXISEL = 000; //Interrupt whenever a character is received
    IPC2bits.U1TXIP = 4; //set transmit interrupt priority
    IFS0bits.ADIF = 0; // Clear AD Interrupt Flag
    IFS0bits.T1IF = 0; // Clear Timer1 Interrupt Flag
    IEC0bits.T1IE = 0; // Timer Interrupt is not needed
    IPC2bits.ADIP = 4; // Set ADC Interrupt Priority
    IEC0bits.ADIE = 1; // Enable the ADC Interrupt
```

```

TRISB = 0x01; //AN0 is an analog input
TRISA = 0;
TRISD = 0;
TRISE = 0;
TRISF = 0;

//Configure ADC settings
ADCONbits.ADSIDL = 0;    // Operate in Idle Mode
ADCONbits.FORM = 0;      // Output in Integer Format
ADCONbits.EIE = 0;       // No Early Interrupts
ADCONbits.ORDER= 0;      // Even channel first
ADCONbits.SEQSAMP= 1;    // Sequential Sampling Enabled
ADCONbits.ADCS= 5;       // Clock Divider is set up for Fadc/14
ADPCFG = 0xFFFE;        // AN0 is an analog input
ADSTAT = 0;              // Clear the ADSTAT register
ADCPC0bits.TRGSRC0= 0xC; // Trigger conversion on TMR1 Prd Match
ADCPC0bits.IRQEN0= 1;    // Enable the interrupt

// Set up Timer1
T1CON = 0;                // Timer with 0 prescale
TMR1 = 0;                 // Clear the Timer counter
PR1= TIMER_PERIOD;       // Load the period register
// Set up the Interrupts
IFS0bits.ADIF = 0;        // Clear AD Interrupt Flag
IFS0bits.T1IF = 0;        // Clear Timer1 Interrupt Flag
IEC0bits.T1IE = 0;        // Timer Interrupt is not needed
IPC2bits.ADIP= 4;         // Set ADC Interrupt Priority
IEC0bits.ADIE= 1;         // Enable the ADC Interrupt

while(1)
{
    ADCONbits.ADON = 1;    // Start the ADC module
    T1CONbits.TON = 1;     // Start the Timer
    long c;
    for(c=0;c<4000000;c++); //delay in between writes
    T1CONbits.TON= 0;       //Stop Timer
    ADCONbits.ADON = 0;     //Stop ADC module
    TMR1 = 0;               // Clear the Timer counter
}
}

//ADC Interrupt handler
void __attribute__((interrupt, no_auto_psv)) _ADCInterrupt(void)
{
    int AN0Result;
    IFS0bits.ADIF = 0;      //Clear ADC Interrupt Flag
    AN0Result = ADCBUF0;    //Get the conversion result
}

```

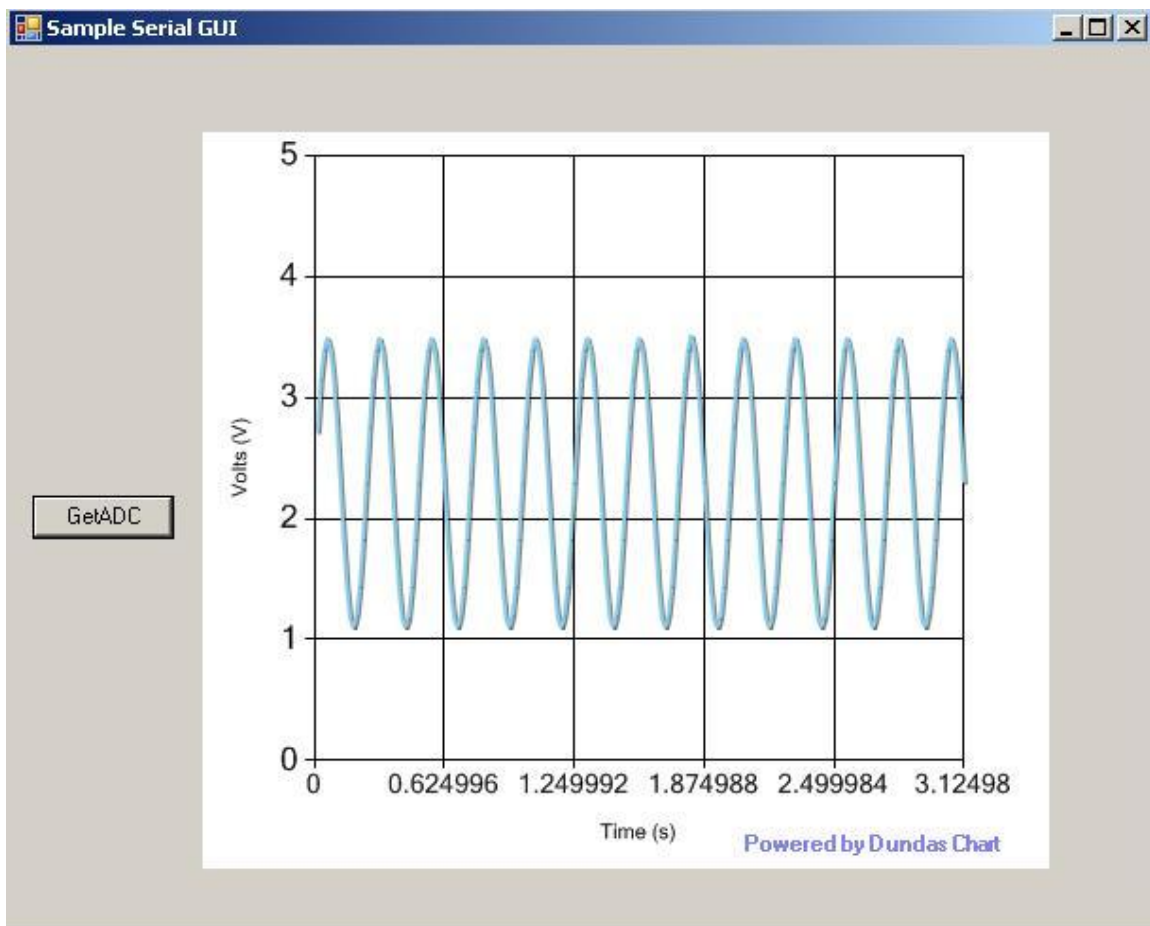
```

    AN0Result >> 2;
    while( U1STAbits.UTXBF);           //Wait while transmit buffer is full
    U1TXREG = channel0Result;           //Transmit data through UART port
    ADSTATbits.P0RDY = 0;               //Clear ADSTAT bits
}
//Transmit Interrupt handler
void __attribute__((interrupt, no_auto_psv)) _U1TXInterrupt(void)
{
    IFS0bits.U1TXIF = 0;
}

```

Results & Conclusion

The above code was tested with a simple visual basic interface. If we input a 2 Vpp 4 Hz sine wave with 2 volt DC offset into channel AN0 on the PIC, then we expect this analog signal to be sampled and digitized with the same characteristics. Below, the corresponding digital output signal is displayed using visual basic. This verifies that A/D conversion is indeed working as expected and we can proceed to develop code for the DC step input, triangle wave, and AC sine wave voltage signal tests for our project.



References

¹ *DsPIC30F1010/202X Data Sheet*. Microchip, 06. Web. 13 Nov. 09.
<<http://ww1.microchip.com/downloads/en/DeviceDoc/70178C.pdf>>.

² *Serial Communications Using the dsPIC30F UART*. Microchip. Web. 13 Nov. 09.
<http://dkc1.digikey.com/us/en/tod/Microchip/dsPIC30F_SerialCommUART_NoAudio/Serial_Communications_using_the_dsPIC30F_UART_noaudio.swf>.