Chapter 13

Analog to Digital Conversion

Analog to digital (A/D) conversion circuits allow microcontrollers to interact with many types of analog sensors without elaborate external line conditioning circuits. An A/D conversion circuit translates analog voltage into a digital sequence of binary bits. By definition an analog voltage has an infinite number of representations while a digital voltage representation is finite. Hence, the analog to digital conversion process may result in data loss.

In order to minimize data loss the A/D converter must constrain the input voltage to a known range. The mathematical representation of the maximum voltage is V_R^+ and minimum voltage is V_R^- [1]. We also need to understand the concept of resolution and accuracy.

Resolution refers to the measurement of the finest detail that can be resolved by a device while accuracy deals with how close to reality is that representation. Applying these concepts to the A/D converter unit we define resolution as the finest input voltage sensitivity and accuracy as a digital translation deviation from the actual voltage. Mathematically the 18F4520 voltage resolution is given by

$$V_{RES} = \frac{V_R^+ - V_R^-}{2^N - 1} \tag{13.1}$$

where N is A/D converter bit allowance. The 18F4520 has by default of $V_R^+ = V_{DD}$, $V_R^- = V_{SS}$ and $N = 10$.

[1] When selecting these values always reference the maximum electrical characteristics on your products data sheet.

This means that assuming that $V_R^+ = 5V$ and $V_R^- = 0V$

$$V_{RES} = \frac{5V - 0V}{2^{10}} \tag{13.2}$$

$$= \frac{5V}{1024step} \tag{13.3}$$

$$= 4.88\frac{mV}{step} \tag{13.4}$$

This implies that a ten bit A/D unit is able to detect input voltage changes as small as $4.88mV$ when $V_R^+ = 5V$ and $V_R^- = 0V$. Any input voltage change within V_R^+ and $V_R^- V$ that is smaller than $4.88mV$ will not be detected by the A/D unit. However, if we change the maximum reference to $V_R^+ = 3V$ we get a $V_{RES} = 2.93mV$, which implies that a the A/D analog voltage range is inversely proportional to its resolution and it affects the way the A/D translates any given input voltage into its 10 digit sequence Q.

The formula to compute Q for any arbitrary analog input V_{in} and reference voltages is given by

$$Q = \frac{V_{in} - V_R^-}{V_{RES}} \tag{13.5}$$

Note that this formula does not apply for input voltages outside the reference range. If an input voltage larger than the maximum reference voltage is presented it will be translated to maximum binary sequence available 0x3FF. Similarly is an input voltage smaller than the minimum range voltage is presented it will be translated to the minimum binary sequence 0x000.

The 18F4520 has 13 A/D channels located in PORT A and PORT B. They are labeled ANT0, ANT1, ..., ANT12 on the data sheet, see Figure 13.1.

The A/D is controlled via three registers ADCON0, ADCON1 and ADCON2 and two interrupt control registers INTCON and PIE1. The control register ADCON0 is used to

- Enable/Disable the A/D unit
- Trigger a new A/D translation
- Monitor the A/D progress
- Choose the desired analog channel

ADCON0 Bit 0 is called the ADON or Enable bit, by setting this bit you turn ON the A/D converter. If you prefer to use bitmasking and pointers you may set ADON by writing

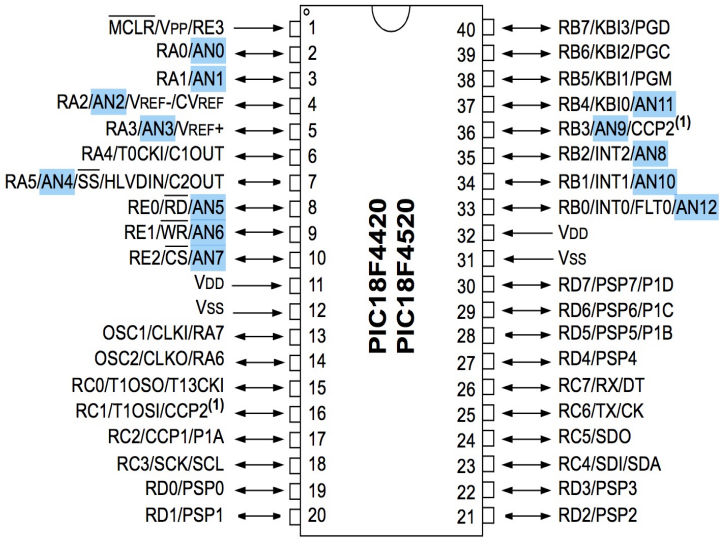

Figure 13.1: 18F4520 Microchip PIC

```
*ADCON0 |= 0x01;
```

The second bit is called the GO/DONE bit used to initiate an input voltage translation. Setting the GO/DONE bit is done by writing

```
*ADCON0 |= 0x02;
```

After the GODONE bit is set the system will reset GODONE after the input voltage has been properly translated. The next four bits CHS0, CHS1 ,CHS2 and CHS3 are used to select the desired A/D channel. Notice that the binary combination is what selects the channel. Therefore, selecting 0011 does not mean channel 1 and channel 0 but channel 3. Finally, the last two bits are unused. Remember that the 18F4520 has only one A/D with thirteen multiplexed channels, see Figure 13.2.

In ADCON1 the four least significant bits called PCFG0, PCFG1, PCFG2, and PCFG3 are used to select which channels in the microcontrollers are set as analog versus digital system. We also have bit 4 named VCFG0 used to enable the maximum voltage Reference and bit 5 VCFG1 used to enable the minimum voltage reference. If you enable VCFG0 and/or VCFG1 you must supply external power supply to their corresponding input pins. The last two bits 6 and 7 in ADCON1 are unused.

In ADCON2 the MSB ADFM (bit 7) is called the Format Select Bit. This bit selects a right versus left justification of the translated bits. Justification is necessary since the the 18F4520 as it is an 8 bit processor while the A/D is a 10

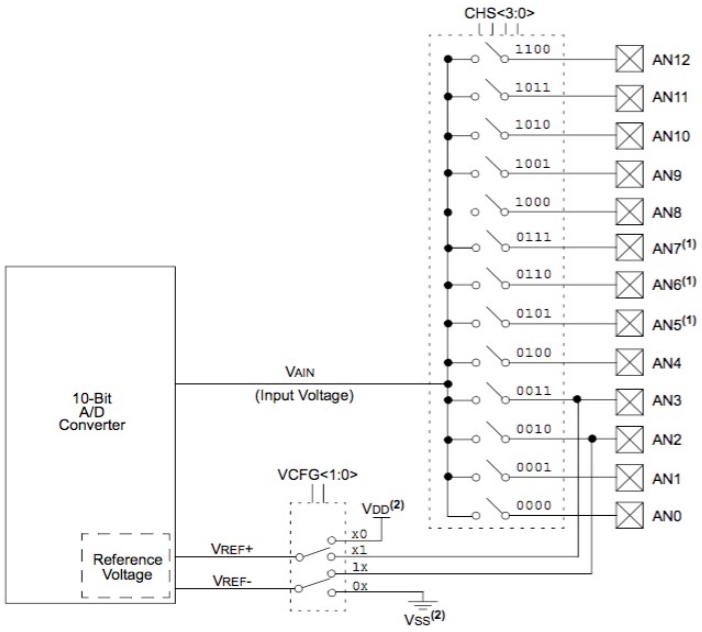

Figure 13.2: A/D Control Register.

bit circuit. This means that the data generated by the A/D does not fit a single register and hence two registers are required to deposit the result. The final storing registers are called ADRESL and ADRESH where justification plays a major role in understanding the final result.

Right justification delivers the first 8 least significant bits from the A/D translation on ADRESL while the remaining two bits are placed in the LS location of ADRESH, the rest of the bits are zero padded, i.e. assuming that an unknown 10 bit number was generated from the A/D with right justification we would end up with

```
ADRESH = 0000 00XX
ADRESL = XXXX XXXX
```

Left justification delivers the first 8 most significant bits from the A/D translation on ADREH while the remaining two bits are placed in the MS location of ADRESL while the rest of the bits are zero padded, i.e. assuming that an unknown 10 bit number was generated from the A/D with left justification we would end up with

```
ADRESH = XXXX XXXX
ADRESL = XX00 0000
```

The easiest way to assemble ADRESH and ADRESL into a single variable is to create a sixteen bit pointer pointed to ADRESL,

```
 int16 *Q = 0xFC3;
```

and set the A/D as a Right Justified system. If you do not desire to use a sixteen bit pointer then you must assemble the data onto a sixteen bit variable utilizing bitmasking. On one hand if you have a Right Justified A/D the data must be assembled by shift ADRESH to its rightful position. For example,

```
int16 Q = 0;
Q = ((int16)ADRESH << 8) | (int16)ADRESL;
```

On the other hand if you have a Left Justified A/D the data must assembled by shifting both ADRESH and ADRESL to its rightful place. For example,

```
int16 Q = 0;
Q = ((int16)ADRESL>>6) | ((int16)ADRESH<<8);
```

13.0.1 Pointer to Structures Approach

The use of pointers to structures simplify the software minimizing the possibility of mistakes with the added bonus of increasing software readability. The first step is to create a structure for every control register. Here are the three necessary structures to control the A/D system

```
struct _ADCON0{
    int ADON:1;
    int GODONE:1;
    int CHSx:4;
    int unused:2;
};
struct _ADCON0 *ADCON0 = 0xFC2;

struct _ADCON1{
   int PCFGx:4;
   int VCFG0:1;
   int VCFG1:1;
   int unused:2;
};
struct _ADCON1 *ADCON1 = 0xFC1;
```

```
struct _ADCON2{
   int ADCSx:3;
   int ACQTx:3;
   int unused:1;
   int ADFM:1;
};
struct _ADCON2 *ADCON2 = 0xFC0;
```

Every structure presented above is named after the register is represents in the data sheet and every structure has elements that can be called using the following pattern

```
POINTER_NAME -> ELEMENT  = VALUE;
```

For example, you may turn ON the A/D converter by setting the ADON bit inside ADCON0 using

```
ADCON0->ADON = 1;
```

In the case of a multi-bit selection as in the channel selection simplifies to

```
ADCON0->CHSx = 0;
```

where the previous example selects channel 0 as the input to the A/D conversion unit.

13.1 Software Examples

We can test the A/D converter by connecting a battery or variable power supply to Channel A0. You can then compile the following C program on PCW-CCS.

```
#include <18f4520.h>
#use delay (clock = 20000000)
#fuses HS, NOWDT, NOLVP
#include "..\Library\myPtrLibrary.h"
#include "..\Library\modifiedlcd.h"

float Vres = 5.0 / 1023.0;
```

```
main(){
   *TRISA = 0x01;     // Pin A0 Input
   ADCON0->ADON = 1;  // Turn ON channel
   ADCON0->CHSx = 0;  // Channel 0
   ADCON1->PCFGx = 7; // Port A Analog
   ADCON2->ADFM = 1;  // Right Justified
   lcd_init();
   while(1){
      ADCON0->GODONE =  1;     // Go get one Analog Input
      while(ADCON0->GODONE){}  // Waiting for translation
      printf(lcd_putc,"\f Vin = %f", Vres * (*Q));
      delay_ms(500);
   }
}
```

If you desire to read multiple channels you have to read their data one at a time. Here is an example that reads channels 0 and 1 and presents the data onto the LCD panel

```
#include <18f4520.h>
#use delay (clock = 20000000)
#fuses HS, NOWDT, NOLVP
#include "..\Library\myPtrLibrary.h"
#include "..\Library\modifiedlcd.h"

float Vres = 5.0 / 1023.0;

main(){
   int16 ch0, ch1;
   *TRISA = 0x03;     // Pin A0 & A1 Input
   ADCON0->ADON = 1;  // Turn ON channel
   ADCON0->CHSx = 0;  // Channel 0
   ADCON1->PCFGx = 7; // Port A Analog
   ADCON2->ADFM = 1;  // Right Justified
   lcd_init();
   while(1){
      ADCON0->CHSx = 0;
      ADCON0->GODONE = 1;
      while(*ADCON0 & 0x02);
      ch0 = *Q;
```

```
      ADCON0->CHSx = 1;
      ADCON0->GODONE = 1;
      while(*ADCON0 & 0x02);
      ch1 = *Q;

      printf(lcd_putc,"\f V0=%f \n V1=%f ", Vres * ch0, Vres * ch1);
      delay_ms(500);
   }
}
```

13.2 A/D Interrupts

Most microcontrollers A/D have configurable interrupts triggered by a successful A/D conversion, i.e. a reset of bit ADIF on register PIR1.

```
#include <18f4520.h>
#use delay (clock = 20000000)
#fuses HS, NOWDT, NOLVP
#include "..\Library\myPtrLibrary.h"
#include "..\Library\modifiedlcd.h"

unsigned int16 ch1, ch2;
float Vres = 5.0 / 1023.0;

#int_AD
void AD_isr(){
  if( ADCON0->CHSx == 0){
     ch1 = *Q * Vres;
     ADCON0->CHSx = 1; }    // Make it channel 2
  else if( ADCON0->CHSx == 1){
     ch2 = *Q * Vres;
     ADCON0->CHSx = 0; }    // Make it channel 3
}

main(){
  unsigned int16 z=0x0000;
  lcd_init();
  *TRISA = 0x07;
  ADCON0->ADON = 1;                 // /64 Ch1 AD-ON
  ADCON0->CHSx = 0;                 // All analog, Right Justified
```

```
  PIE1->ADIE = 1;
  INTCON->PEIE = 1;                  // Periphera int Enable, Global
  INTCON->GIE = 1;
  ADCON0->GODONE = 1;
  while(1){
     printf(lcd_putc,"\f C1=%f \nC2=%f", ch1, ch2);
     delay_ms(500);
     ADCON0->GODONE = 1;
  }
}
```