Tahmid

Acquisition Time

Before an analogue to digital conversion is to be started, a small amount of time must be allowed for the holding capacitor of the PIC ADC module to fully charge to the input level. This is how the ADC works and unless this minimum time is allowed, the ADC will give an incorrect result. This time is known as the acquisition time.

The datasheet provides the equation for calculating the minimum acquisition time. If you want, you may check it out, but the datasheet also provides the value of the minimum acquisition time, which is 19.72µs. For now, this is all you need to know. You don't need to know how it comes, but if you want to, you may check it out in the datasheet. So, when the ADC is on, a minimum of 19.72µs must be allowed before the conversion can be started. Since, this is the minimum value, you can use a larger value. I will use around 50µs. There is no special reason to choose this and it is a rather arbitrary value just chosen as it is quite a bit larger than the minimum required time more than twice as much.

For the analogue input, the maximum recommended (by Microchip) input impedance is 2.5kΩ. While larger input impedances may be used, it is best to stick to Microchip's recommendation. If the input does have a larger input impedance, an analogue buffer (voltage follower) employing an operational amplifier may be used.

There are a few more things to consider here that are covered in part 6.

ADC Conversion Clock (TAD)

The analogue to digital conversion time per bit is defined as TAD. The analogue to digital conversion requires a minimum 12 TAD per 10-bit conversion. The source of the analogue to digital conversion clock is software selected. The seven possible options for TAD are:
2 TOSC
4 TOSC
8 TOSC
16 TOSC
32 TOSC
64 TOSC
Internal ADC module RC oscillator (2-6 µs)

TOSC = 1/FOSC, where FOSC is the frequency of the oscillator, off which the PIC is running. For a PIC16F877A running off a 20MHz crystal, FOSC = 20MHz.

For correct analogue to digital conversions, the analogue to digital conversion clock (TAD) must be selected to ensure a minimum TAD of 1.6 µs.

Example:

For correct analogue to digital conversion, what are the possible TAD options, when a 20MHz crystal oscillator is used?

Solution:
FOSC = 20MHz
TOSC = (1/20MHz) = 0.05µs

For 1.6μs, TAD = (1.6μs/0.05μs) TOSC = 32 TOSC

So, when TAD = 32 TOSC, 1 TAD = 1.6μs (minimum required time).
Since 1.6μs is the minimum value, let's give it a little more time. As 32 TOSC, 64 TOSC and the internal ADC module RC oscillator can all be used and 32 TOSC meets only the minimum required time, we can use the other settings. When 1 TAD = 64 TOSC, 1 TAD = 64(0.05μs) = 3.2μs

The internal ADC module RC oscillator has a typical TAD time of 4μs, that is, 1 TAD = 4μs. But the actual value may vary from 2μs to 6μs the entire range meets the required minimum time of 1 TAD = 1.6μs.

The analogue to digital conversion takes 12 TAD. So, in the above example, if we used 64 TOSC, 1 TAD = 64(0.05μs) = 3.2μs. The conversion will take 12TAD = 12*3.2μs = 38.4μs. The entire conversion will comprise of the initial acquisition delay plus the conversion time plus any delay in the middle, such as time taken for setting/configuring registers, etc. So, if we had used 40μs for the acquisition delay, the minimum total analogue to digital conversion time would be (40+38.4)μs = 78.4μs. When TAD is to be selected, it is better to select a choice that is significantly larger than the minimum specified value of 1.6μs.

Note: When a 20MHz crystal oscillator is used, the frequency is internally divided by 4 by the PIC to 5MHz. But FOSC = 20MHz, not 5MHz. 5MHz is FCY.

Microchip does not recommend the use of the internal ADC module RC oscillator for device frequencies greater than 1MHz (unless in sleep mode but we're not concerned with that now). While this does not mean that it can not be used, it is better not to use it as it is not recommended and we can simply select a setting for TAD from one of the other settings.

Right / Left Justification

The ADC conversion gives a 10-bit result. We know that the PIC is an 8-bit microcontroller and the registers are all 8-bit. For anything greater, multiple registers must be used. Thus, for storing a 10-bit result, two registers are used: ADRESH and ADRESL. Now, if a 10-bit value is stored in 2 bytes, there is more than one way the value may be stored. In the PIC, the result may be stored in one of two ways as set in software. The result may be left-justified or right-justified.

Now, what is justification, you may ask. It's actually pretty simple.

Let's start with the more commonly used setting: right-justification. We all know of right-justification in word-processors. It's when you start typing from the absolute right of the page / typing area and whatever you type is shifted to the left to accommodate. The right-justification of the ADC result storage is similar to this. The LSB (least significant bit) is stored in the right most bit of the ADRESL register bit 0. Each of the other bits holds the other bits of the result. So, in ADRESL, bit 0 stores bit 0 of the ADC conversion result, bit 1 stores bit 1, bit 2 stores bit 2, and so on. So, bit 7 of ADRESL stores bit 7 of the ADC conversion result. That leaves us with 2 more bits bits 8 and 9. These are stored in ADRESH. Bit 8 of the ADC conversion result is stored in bit 0 of ADRESH and bit 9 of the ADC conversion result is stored in bit 1 of ADRESH. So, the MSB (most significant bit) is stored in the left most bit of ADRESH bit 1. Bits 2 and onwards till bit 7 store 0. This is pretty much all there is to right-justification. Simple, isn't it?

If you're still not clear, this example may help clear any doubts.

Example:

A circuit for analogue to digital conversion is set up, to utilize the internal ADC of the PIC16F877A. VREF+ = 5.0V, VREF- = 0V. The value of TAD is selected such that TAD > 1.6µs. An input voltage of 2.0V is measured by the ADC and the corresponding value is stored in the registers ADRESH and ADRESL. What are the values of ADRESH and ADRESL, if the result is right-justified ?

Solution:
ADC conversion result = (2.0 / 5.0) * 1023 = 409
ADRESL stores the lower 8 bits. 409 in binary is 01 1001 1001. The lower 8 bits are stored in ADRESL and the upper 2 bits are stored in ADRESH. So, ADRESL stores 1001 1001. ADRESH stores 0000 0001.

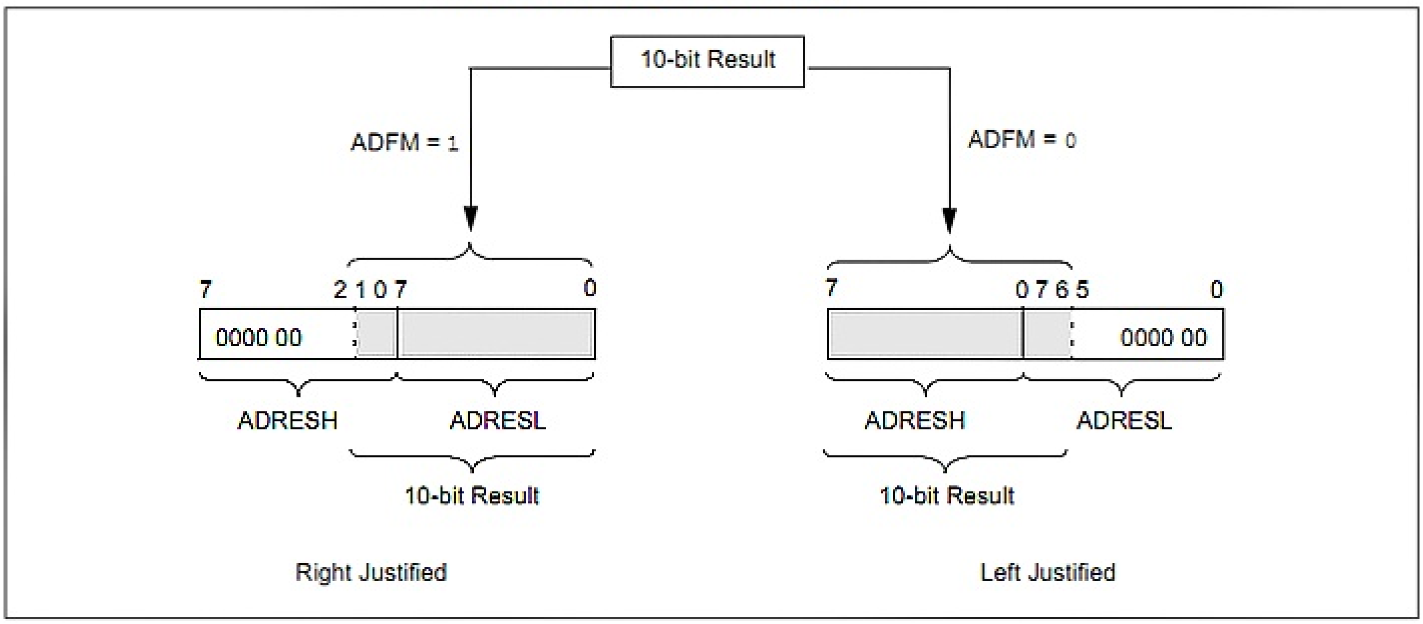
Now, let's talk about left-justification. While this is less commonly used, it is quite useful as you will find out. You must know the left-justification in word-processors. You start typing from the left and everything you write is shifted to the right. Left-justification is also like this. The 10-bit result is again stored in ADRESH and ADRESL but in a different way. While the result is left-justified, the upper 8 bits are stored in ADRESH and the lower 2 bits are stored in ADRESL. So, bits 9, 8, 7, 6, 5, 4, 3, 2 of the ADC conversion result are stored in ADRESH bits 7, 6, 5, 4, 3, 2, 1, 0 in that order. So, bits 1 and 0 of the ADC conversion result are stored in ADRESL bits 7 and 6 in that order. Bits 5 onwards to bit 0 store 0.

Example:

A circuit for analogue to digital conversion is set up, to utilize the internal ADC of the PIC16F877A. VREF+ = 5.0V, VREF- = 0V. The value of TAD is selected such that TAD > 1.6µs. An input voltage of 2.0V is measured by the ADC and the corresponding value is stored in the registers ADRESH and ADRESL. What are the values of ADRESH and ADRESL, if the result is left-justified ?

Solution:
ADC conversion result = (2.0/5.0)*1023 = 409
ADRESH stores the upper 8 bits. 409 in binary is 01 1001 1001. The lower 2 bits are stored in ADRESL So, ADRESL stores 0100 0000. ADRESH stores 0110 0110.

A/D RESULT JUSTIFICATION



Continued in part 3.............