Continuous Battery Check Using a PIC18F4520

By: Rodney Singleton II
April 2, 2010

Executive Summary
Every electrical appliance needs power in order to perform. In the case where systems run solely off of battery power or solar energy, power efficiency is the primary factor affecting the system’s conservation. Once the battery life goes low, everything will power down, leaving the user oblivious and without use of the system. Thus, it is necessary to obtain the battery life levels, even when the system is performing other functions. At every instance of functioning, the system will be checking its battery life, allowing the user to know when the system is running on low-power and adjust accordingly. Implementation of the continuous battery check can be written directly in the code for the microcontroller.

Keywords
PIC18F4520, ADC Converter, Battery Life, Low-power, Interrupt, μC
Objective

The purpose of this application note is to discuss the best way to code for battery life testing, specifically using a PIC18F4520. This includes how to continually check the battery life, even during the process of other functions. Lastly, it will cover how to interrupt said functions and potentially warn the user about an imminent shut down.

Introduction

There are several ways to determine the battery life of a system, resulting in the output being interfaced with the user. However, the most efficient and effortless way is by using the Analog-to-Digital converter function on the PIC18F4520. An Analog-to-Digital converter takes a continuous signal, such as the life of a battery, and converts it into a discrete one, making it computer-friendly. Using the microcontroller, the discrete signal can now be interpreted and placed into coding in which you can define its use.

The PIC18F4520 has 13 different inputs you can use for your analog signal, and 5 registers that control them: ADRESH, ADRESL, ADCON0, ADCON1, and ADCON2. For the purpose of battery testing, the Analog-Digital Control registers are used.

Implementation

Step 1: ADC Initialization

The first step in implementing a battery check using the PIC18F4520 is to initialize the ADC registers on the PIC device. In order to do this, the ADC library file is needed. At the top of the code, write this line:

```
#include <ADC.h>
```

Next, you must open the necessary ports in order to input the analog signal. This is done by a series of codes, starting with opening the ADC ports to read. The reference voltages must then be set to their specific settings. For a more accurate result, use Channel 3 as the positive reference voltage and ground for the negative. The entire code for opening the necessary ports is found below:

```
OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_12_TAD, 
        ADC_CH0 & ADC_INT_OFF, 0); //open adc port for reading
ADCON1 =0x10; //set VREF+ to CH3 and VREF- to GND (VSS)
```

All of the ports are now open and ready to be read.
Step 2: Reading the Analog Signal
In the main function of your program, you must initialize what port is specifically being read and manipulated. The µC will then read it, and you will give the command for the conversion. While processing, you do not want to do anything else, so a “waiting” command is instilled. Finally, the last command will take the digital signal and it can then be manipulated in the code. This is done in four lines, which are found below:

```c
SetChanADC(ADC_CH0);
ConvertADC(); //perform ADC conversion
while(BusyADC()); //wait for result
adc_result = ReadADC(); //get ADC result
```

Step 3: Continuous Reading During Other Functions
As stated before, one of the issues in battery check is continuously testing for a low voltage. This can be defeated by adding the same line of code during the for loop, or while loop, for a sub-function. In this example, the sub function is a program which lights up LED.

```c
loop=50;

while(loop>0)
{
    //Battery Check
    SetChanADC(ADC_CH0);
    ConvertADC();
    while(BusyADC());
    adc_result = ReadADC();

    for(count = 1; count < 40000; count++);
    PORTDbits.RD3 = !PORTDbits.RD3; //Lights up the LED
    loop--;
}
```

This program will continuously check the battery even during the flashing of the LED. For every iteration of the while loop, the Battery Check commands will be called, in the midst of the program.

Step 4: Creating an Interrupt in the Event of Low Battery
In step 3, we learned how to read the battery voltage at all instants of a code. Now, we will take that a step further and create interrupts for when the battery life decreases below the optimum threshold. As stated before, the battery life is now being read as a digital
value. In order to test whether the battery is low, we can compare this number to a hexadecimal value. This number is determined by the number of bits that represent your reference voltage. In the case of the PIC18F4520, the voltage is spread over 1024 bits. Therefore, if you had 5 Volts as your Vref+, you would have 5/1024 volts per bit, or 4.88mVolts per bit. Knowing this number, you can turn any voltage limit into a binary number. For instance, if you wanted 2 volts as the cutoff limit, you would multiply it with the reciprocal of 4.88m.

\[
\text{2 volts} \times \frac{1 \text{ bit}}{4.88m \text{ Volts}} = \text{409 bits}
\]

Now, you can take whatever value you got in your ADC conversion and compare it to 409, or 19A in hexadecimal. From there, you can write statements for the program to interrupt whatever’s functioning and power down. An example code for this type of interrupt is found below, using the same LED function as in step 3. Notice there is a place where you can alert the user of an imminent shutdown.

```c
loop=50;

while(loop>0)
{
    //Battery Check
    SetChanADC(ADC_CH0);
    ConvertADC();
    while(BusyADC());
    adc_result = ReadADC();

    //Voltage Comparison
    if(adc_result<0x019A){
        //Output “Low Battery”
        break;
    }

    for(count = 1; count < 40000; count++);
    PORTDbits.RD3 = !PORTDbits.RD3; //Lights up the LED
    loop--;
}
```

Instead of continuing to run the program, the µC will break the function and the main program to power everything off. We can also allow the battery to charge back up to the optimum battery level, while the program lies dormant. We can do this by using more comparative statements, and periodically re-checking the battery. Using the same
example, let’s code the program to start running again after the battery is charged to 3 volts.

\[
\begin{align*}
3 \text{ volts} \times \frac{1 \text{ bit}}{4.88 \text{ m Volts}} &= 615 \text{bits} \\
615 \text{ decimal} &= 267 \text{ hexadecimal}
\end{align*}
\]

```
loop=50;

while(loop>0)
{
    //Battery Check
    SetChanADC(ADC_CH0);
    ConvertADC();
    while(BusyADC());
    adc_result = ReadADC();

    //Switch from ‘if’ to ‘while’
    while(adc_result<0x019A){
        //Output “Low Battery”
        //Re-check battery
        SetChanADC(ADC_CH0);
        ConvertADC();
        while(BusyADC());
        adc_result = ReadADC();

        //Check if battery is still less than 3 volts
        if(adc_result<0x0267){
            adc_result=0x00;
        }
    }

    for(count = 1; count < 40000; count++);
    PORTDbits.RD3 = !PORTDbits.RD3;
    loop--;
}
```

By switching the if statement to a while statement in the figure above, the program will come to a halt as long as soon as the battery falls lower than 2 volts. There is also a spot in the code in which you can alert the user of the low battery life. The advantage of this new code is that the program will stay halted until the battery increases above 3 volts. Thus, the program will run again after we’ve reached our optimum voltage.
Conclusion
This application note discusses the least difficult way to implement a battery check during a program. The main topics covered were initializing the ADC ports, reading the signal, and manipulating your input using comparisons. The best way to allow the interface between battery life and the user is to notify the user when the battery life is low while saving the user’s program. This way, the user can pick off where he/she left off once the battery is charged to a desired level.
References

PIC18F4520 Data Sheet, Microchip

MPLAB C18 Libraries

MPLAB C18 User’s Guide