FFT Frequency Detection on the dsPIC

by Jac Kersing Nov 2012

Abstract

In this article Jac explains how he solved a practical challenge using powerful dsPIC devices and Fast Fourier Transform algorithms. He accomplishes this using Flowcode for dsPIC and PIC24s. Read more to discover how! An example is included to accompany this article.

Requirements

Software:

- Flowcode for dsPIC & PIC24

Problem

A little while ago I found myself faced with the challenge to transport signals of two sensors over a two wire cable. The sensors require power to operate and produce an analog signal requiring some interpretation to translate it to an binary (on/off) value. There being no power available at the 'remote' location adds to the challenge.

Several solutions were considered to solve the challenge. One was to use the wires to supply (low voltage) power and use some kind of radio transmitter/receiver solution to transfer the data. Two 30 cm layers of reinforced concrete caused the signals to be too unreliable for this to work.
An other solution was to use a PIC attached to the sensors with some kind of protocol to transmit the status of the sensors. It would be nice if this solution could operate without any battery power as the sensors require up to 30 mA, a considerable drain on a battery given the 24 hour operation of the sensors. In the ideal world the cable would provide power to the remote location and transport the sensor status back.

Hardware

As I am working on a project requiring dsPICs and FFT I could not resist the temptation and designed the solution based on frequency generation and FFT. The high level design can be found in illustration 1.

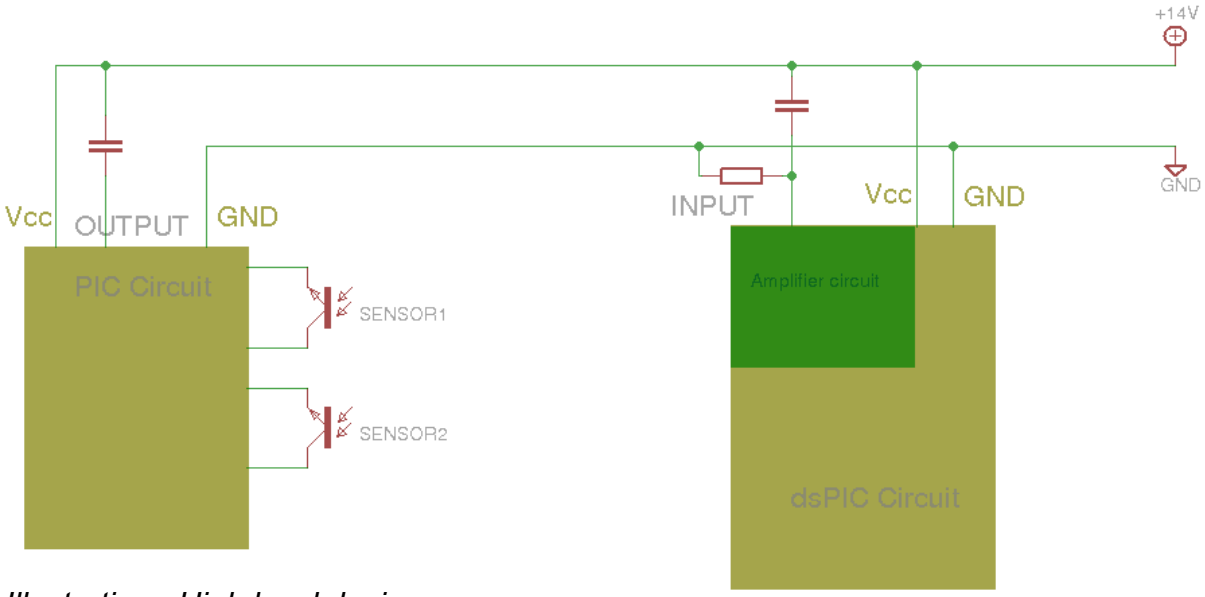

Illustration : High level design

The PIC circuit has the sensors attached and converts the input values read to a true/false state. If the state of a sensor is true a sine of a certain frequency is output. The frequency is 1000 Hz for sensor 1 and 1500 Hz for sensor 2. At any given time the output might be idle, have a frequency of 1000 Hz, 1500 Hz or both 1000 Hz and 1500 Hz. The output signal is 'fed' to the two wire connection to the dsPIC.

At the dsPIC the input signal needs to be isolated from the power feed. A high pass filter consisting of a capacitor and resistor take care of this. The output of this very simple filter is a weaker (due to cable losses and the filter) version of the wave form generated by the PIC. This wave form needs to be amplified and converted to the input range of the dsPIC. A simple circuit with an OPAMP, some resistors and some capacitors solves that problem (see illustration 2)

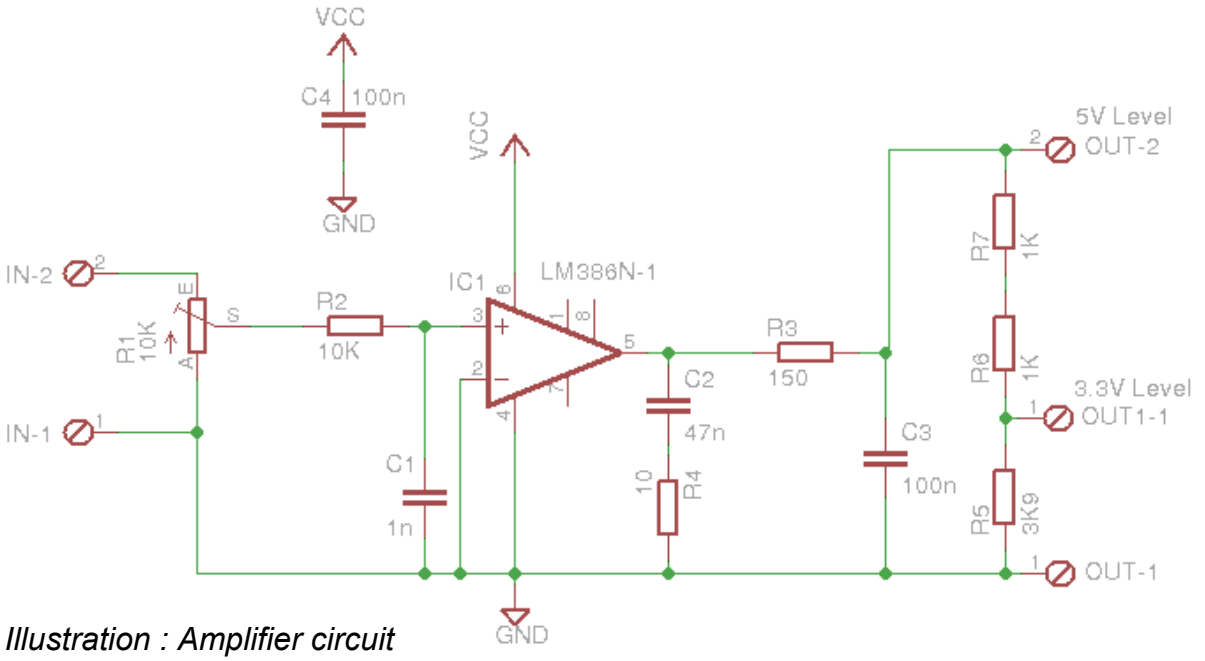

Illustration : Amplifier circuit

The 3.3 volt level output of the amplifier circuit is used as input for an analog input (AN0) of the dsPIC.

The prototype is build using the circuit in illustration 2, an EB64 – 16 bit PIC / dsPIC multiprogrammer with a dsPIC33FJ128GP802 fitted and an EB004 – LED board to display the relayed signal.

Theory

Before we start working on the software a (tiny) bit of theory is needed. Fast Fourier Transform decomposes a series of samples into 'bins' of different frequencies. Each bin represents the extend the frequencies contained in that bin are present in the input signal. To be able to distinguish between two frequencies we need them to be in different bins.

Microchip provides a library of FFT routines for their dsPIC range of micro controllers. These routines take an input array sized power-of-two (64, 128, 256...) and provide the output in an array sized half that of the input array.
The size of the frequency band contained within each bin in the output is equal to the sample frequency divided by the number of samples.

An example:
When using a sample frequency of 20 kHz and a sample size of 256 samples, the size of each frequency band is 78.125 Hz. With a sample size of 128 samples the band would be 156.25 Hz.

Something to take into account when determining the samples frequency is the requirement of the sample rate to be at least twice the highest possible input frequency. (Google for Nyquist for background)

Software
The hardware connects the signal to be sampled to an input of the Analog-to-Digital Converter (ADC). Flowcode provides us with an ADC component to convert input signals to digital values. However, due to the requirements of the FFT used we are unable to use this component.

Due to the requirements of the Microchip FFT library and of the PIC the buffers used for FFT and DMA need to be created with special compiler attributes. Flowcode does not allow us to specify these attributes, as a result these buffers are created in the 'Supplementary Code' definitions and function declarations.

In addition to the memory requirements, the FFT library defines the format of the input to be in 1.15 data format. This format has 1 sign bit and 15 fractional bits and can (only) be used to represent decimal numbers in the range -1.0 to (nearly) +1.0.

Sampling the input

A sample frequency of 20 kHz is used (R3/C3 in the circuit diagram form a very simple 10 kHz low pass filter). The sampling is driven by Timer 3, the results are transferred to a 32 sample input buffer using DMA. Once the input buffer is full an interrupt will start code to transfer the samples to their final destination. (The PIC actually alternates between two input buffers to allow the interrupt code to copy samples while new samples are acquired without the risk of overwriting the samples being copied)

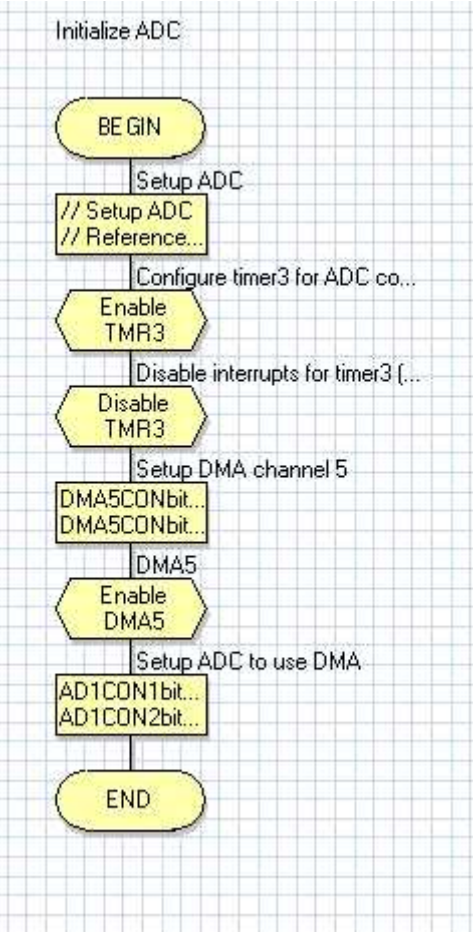

Illustration : InitADC Macro

The macro to setup the ADC, Timer 3 and DMA contains a number of C code blocks. These take care of the 'low level' setup of the ADC channel, the DMA channel and enable the ADC. The Interrupt blocks setup Timer 3 with the correct frequency. The next step is to immediately disable the interrupt as we do not want to use it directly and setup DMA5 with a custom interrupt for DMA5.

For the DMA5 code to work some additional C code is required, a helper routine performing the actual data copy out of the DMA buffer to the FFT buffer. The code can be found in the 'Supplementary Code' function implementations.

FFT

Fast Fourier Transform is performed on complex numbers. This means the input values need to be transformed to complex values. (This is very simple as the imaginary component of the input signal is zero) The c-code in the StartADC macro zeros the imaginary components and (re)set the pointers used to transfer samples. The DMA5 code copies the real values to the sample array.

The FFT code requires a number of constants (twiddle factors), these are set up in the C code blocks in InitFFT.

Once the samples have been read the signal strength of the input signal needs to be processed. The ProcesFFT macro takes care of this. It also translated the complex values to integers and copies them to an array accessible by 'normal' flow chart elements.

Main program

The main program is rather simple. It starts with setting the oscillator, calls the macros to initialize FFT and ADC and enters an endless loop. In the loop samples are acquired (StartADC) and processed (ProcessFFT). Depending on the contents of the buckets for 1000Hz and 1500Hz outputs B15-B12 and B11-B8 are set (signal present) or cleared (no signal).

Minor Challenges

While developing the code a number of minor challenges presented themselves:

- Flowcode for dsPIC is missing the include file for the FFT library. This can be solved in two ways, the first is to copy it from the Microchip distribution, the second option (which has been implemented) is to add the relevant definitions to the 'Supplementary Code' definitions and function declarations.
- The use of C code and a library preclude simulation of the code. To debug the code I revered to the proven method of displaying information on a display (I used the EB058, allowing a number of lines of text to be displayed.)
- The batch file used to link programs that is provided with FlowCode does not use the DSP library.

To be able to use DSP functions the file 'pic16_C30_link.bat' (which can be found in: c:\program files\Matrix Multimedia\Flowcode PIC24&dsPIC v4\Tools\MX_bats) needs to be changed. The line starting with 'pic30-gcc" should be altered to read :

```
pic30-gcc -Wl,%1.o,-L"%~dp0..\C_tools\lib",-ldsp,--heap=256,--report-mem,--script="%~dp0..\C_tools\Support\%3\gld\p%2.gld",-o%1.cof
```

(this is all one line where the bold text has been added to the original text)

Further reading

Below are some links to other resources and articles on related subjects, and technical documentation relating to the hardware used for this project...

Flowcode:	http://www.matrixmultimedia.com/flowcode.php
Learning Centre:	http://www.matrixmultimedia.com/lc_index.php
User Forums:	http://www.matrixmultimedia.com/mmforums
Product Support:	http://www.matrixmultimedia.com/sup_menu.php