Helsinki Metropolia University of Applied Sciences
Degree Programme in Information Technology

Osazee Agbonogieva

Performance of Efficient Digital Filters

Final Year Project. 21 April 2009

Instructor: Seppo Haltsonen, Principal Lecturer
Supervisor: Antti Piironen, Principal Lecturer
Language advisor: Taru Sotavalta, Senior Lecturer

Helsinki Metropolia University of Applied Sciences **Abstract**

Author	Osazee Agbonogieva
Title	Performance of efficient digital filters
Number of pages	31
Date	22 April 2009
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor Supervisor	Seppo Haltsonen, Principal Lecturer Antti Piironen, Principal Lecturer

An efficient digital signal uses an 8-bit embedded processor on a DSP platform, which was developed to design the technical specifications in this final year project.

The goal was to design a program that counts maximum bits time of 8, 16, 32 and 64 on a Matlab platform, specify the filter design and generate the C header as an export bit integer signed or unsigned, copy the filter coefficients to a platform called PSoC, and initialize with the coefficients generated by Matlab.

The properties were compared with standard and optimized FIR filter designs. How complementary FIR or IIR filter pairs could be utilized more generally in the front-end processing of digital filters was also studied.
In the project a program was designed that counts maximum bits on Matlab, and the filter coefficients were copied to PSoC as one data sample return time. The programming process takes a few seconds to display the result to the screen.

During the study on efficient digital filters it was discovered that a DSP processor uses a signal that comes from the real world.

Keywords	embedded system, DSP

Acknowledgments

Without the support of the many faces in my life, this work would not have been possible. I would like to acknowledge and thank each of the following: First and foremost, My God, Lord and Saviour Jesus Christ for the faithfulness, grace and mercy shown to me during my studies.

Additionally, I wish to express my gratitude to Dr. Antti Piironen for his supervision. Special thanks go to my wife Henna Agbonogieva for her helpfulness and support. This thesis would certainly not have been possible without her support.

Special thanks to my family, for providing for me physically, giving me a roof on my head and a meal every dinner time, and emotionally. They have really been there for me during some difficult times. Finally thanks to all my friends and my brothers and sister for their precious time, and efficient support.

Contents

1 Introduction

In order to acquire the knowledge behind efficient digital filters in which an input signal to a system contains extra unnecessary content or additional noise that can degrade the quality of the desired portion, in such cases one may remove or filter out the useless samples. For example, in the case of the telephone system, there is no reason to transmit very high frequencies since most speech falls within the band of 400 to 3,400 Hz.

Finite Impulse Response (FIR) filters are one of the primary types of filters used in Digital Signal Processing. FIR filters are said to be finite because they do not have any feedback. Therefore, if one sends an impulse through the system (a single spike), then the output will invariably become zero as soon as the impulse runs through the filter. [3;4]

The goal was to design a program that counts maximum bits time of 8, 16, 32 and 64 on a Matlab platform, specify the filter design and generate the C header as export bit integer signed or unsigned. Copy the Filter coefficients to platform called PSoC, to initialize with the coefficients generated by matlab.
A digital filter is programmable, i.e. its operation is determined by a program stored in the processor memory. This means the digital filter can easily be changed without affecting the circuitry (hardware).

An analog filter can only be changed by redesigning the filter circuit.
The characteristics of analog filter circuits (particularly those containing active components) are subject to drift and are dependent on temperature. Digital filters do not suffer from these problems and so they are extremely stable with respect to time and temperature. [2]

2 The Filter Design Technique

FIR filters are filters having a transfer function of a polynomial in *z*- and is an all-zero filter in the sense that the zeroes in the *z*-plane determine the frequency response magnitude characteristic. The *z* transform of a *N*-point FIR filter is given as follows:

$$H(z) = \sum_{n=0}^{N-1} h(n) z^{-n} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

FIR filters are particularly useful for applications where an exact linear phase response is required in suitable audio applications. The FIR filter is generally implemented in a non-recursive way which guarantees a stable filter. [4]

FIR filter design essentially consists of two parts:
(1) a approximation problem
(2) realization problem

The approximation stage takes the specification and gives a transfer function through four steps:

- A desired or ideal response is chosen, usually in the frequency domain.
- An allowed class of filters is chosen (e.g. the length *N* for a FIR filter).
- A measure of the quality of approximation is chosen.
- A method or algorithm is selected to find the best filter transfer function.

The realization part deals with choosing the structure to implement the transfer function which may be in the form of a circuit diagram or in the form of a program.

There are essentially three well-known methods for FIR filter design namely:

- The window method
- The frequency sampling technique
- Optimal filter design methods

2.1 The Window Method

The window method of FIR filter design is built on a more fundamental approach usually, called the Fourier series method. [1]

The time domain response of this ideal impulse response can then be used as coefficients for the filter. The window consists of multiply time domain coefficients by algorithm to smooth edges of the coefficients. There are several windows, each with a trade-off in transition width versus stop band attenuation:

- Rectangular – sharpest transition, least attenuation in the stop band (21dB)
- Hamming – over 3x transition width of rectangular, but 30dB attenuation
- Hamming – winder transition, but 40dB
- Blackman – 6x transition of rectangular, but 74dB
- Kaiser- Any (custom) window can be generated based on a stop band attenuation.[1]

When designing a filter using the window technique, the first step is to use response curves or trial and error and decide which window would be the right one use. Here is a code and frequency response for a filter with a humming window.

Matlab

```
lowpass filter design using 67 coefficient  hamming window
%design specifications
ws=.25;wp=.3;
N=67;
wc=(wp-ws) / 2+ws % calculate cut off frequency
% build filter coefficients ranges
```

```
n =-33:1:33;
hd =sin(2* n* pi *wc)./(pi* n); % ideal freq
hd (34) = 2 * pi * wc /pi;  % zero ideal freq
hm = hamming(N);          % calculate window coefficients
hf =hd.*hm';           % multiply window by ideal response
```

2.2 Design Technique

Some of the experiments carried out during this final year project was to show the basic technique on the design using matlab.

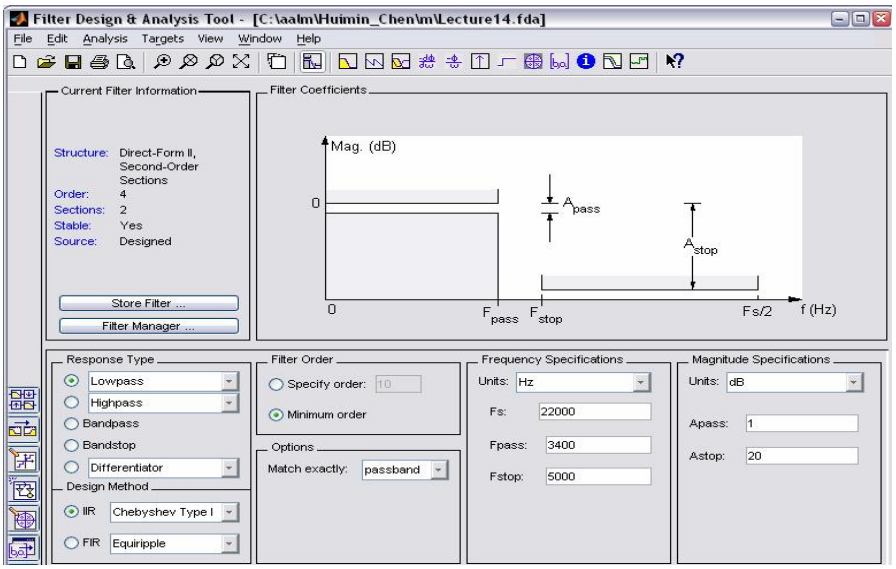

Figure 1. Filter Design Analysis [9]

Figure 1 shows the Filter Design Analysis Tool by simple typing fdatool on the Matlab command prompt and specifies the following unsigned parts: response type, filter order and frequency specifications, then design filter.

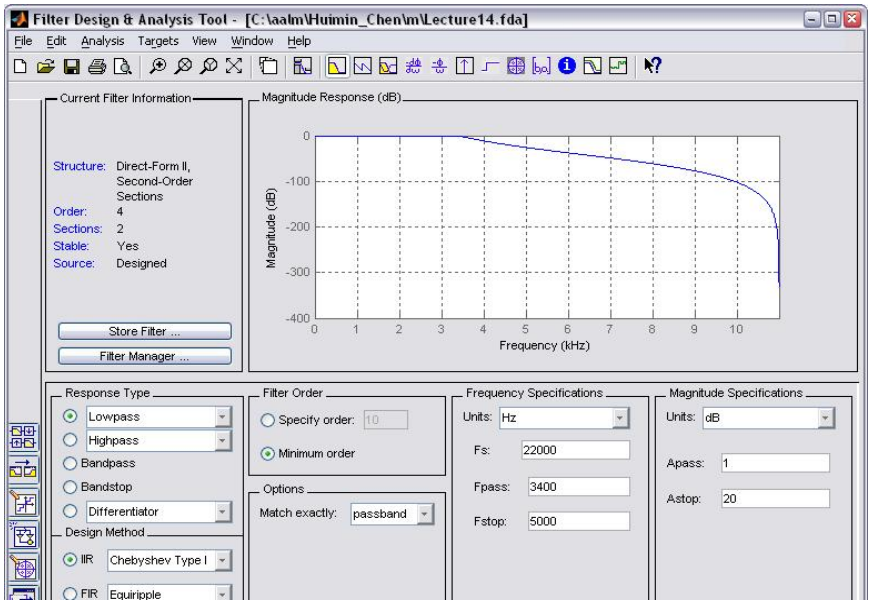

Figure 2.Magnitude Response [9]

Figure 2 shows the magnitude responses of low-pass Chebyshev filters design.

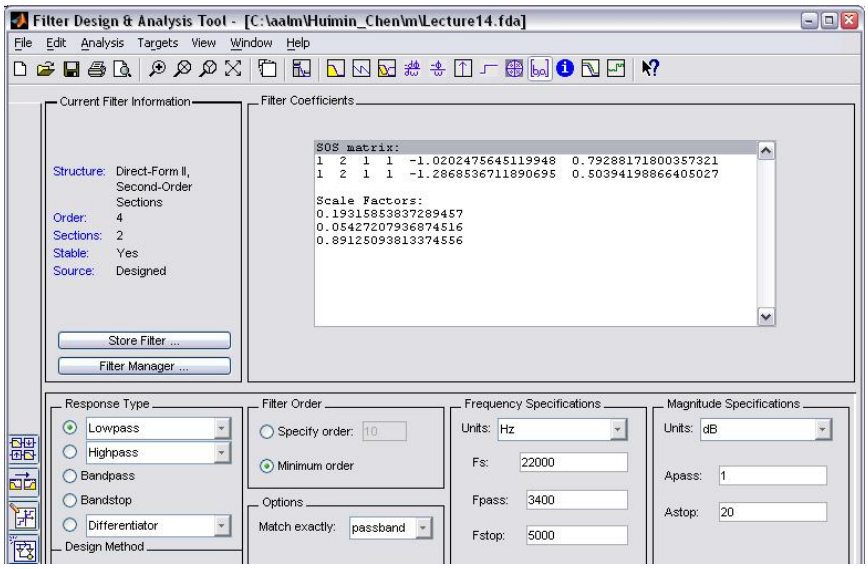

Figure 3. Export Coefficients [9]

Figure 3 represents the export coefficients which allow one to view the filter coefficients.

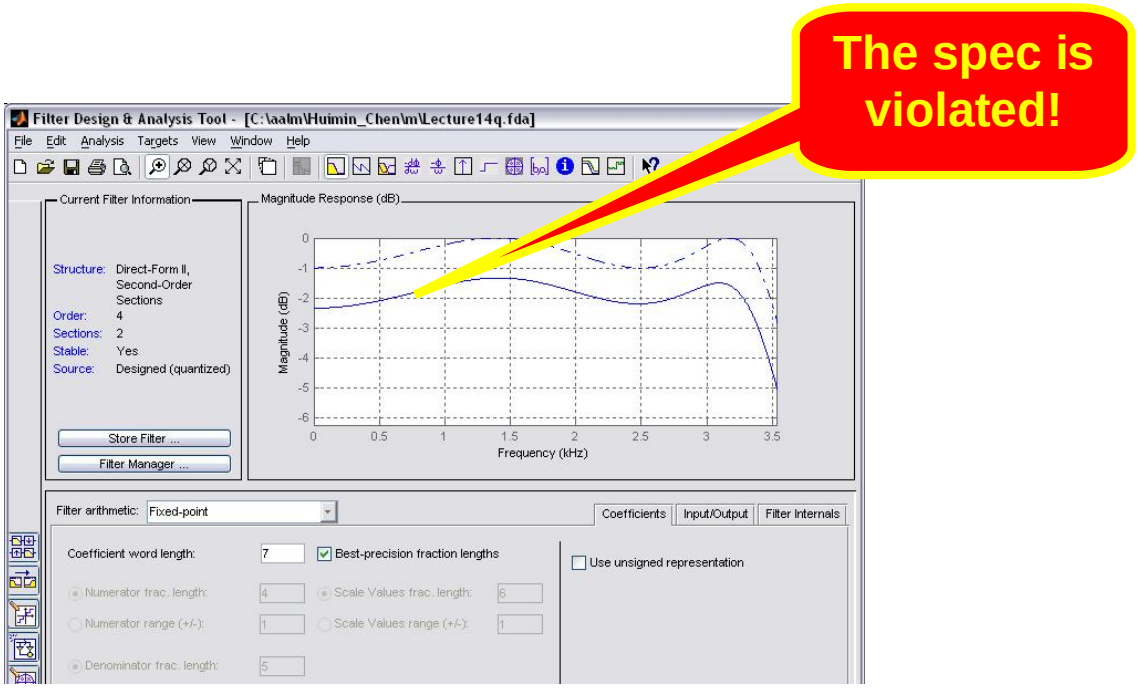

Figure 4. Quantized Filter [9]

Figure 4 represents quantized filters of the fourth operating mode for FDATool properties for the filter coefficients that make up a quantized filter which allows performing numeral tasks. The order of a digital filter is the number of previous inputs (stored in the processor's memory) used to calculate the current output. [5; 6]

3 Designs of FIR Filters

This means different mathematical expressions for the transfer functions of FIR (Finite Impulse Response) and IIR (Infinite Impulse Response). Filters also result in very different design methods for both filter types, while a given frequency response is approximated in case of an IIR filter by a rational fractional transfer function of z, with poles and zero. [1]

I decided to include here the general digital filter design process which can be broken up into four steps for the reader to understand the process:

- Approximation
- Synthesis and realization
- Performance analysis
- Implementation

3.1 Approximation

Approximation toward a signal is the position and the force is the mean of that signal over the entire time interval. In the Fourier series approximation of the mean is referred to as zero order approximation and given the coefficient as a_0 for example, $X(t) = a_0$, where t represents time. [2]

3.2 Synthesis and Realization

A synthesis and realization is based on a discrete-time liner network, which stands for the equivalent to the filter realization process for analog filters. [2]

3.3 Performance Analysis

In performance analysis the filter coefficient is determined near an upper degree of precision in approximation steps, which help in the tools measurement behaviour of a program when executed. The designer makes sure that errors introduced by finite precision will not be violation of the filter specifications. When performing an analysis, it is best to take a long-term approach to ensure that the performance improvement is initiative. [2]

3.4 Implementation

Implementation of a digital filter occurs either in software or hardware. In software implementation there is a decision on what type of computer or microprocessor the software will run on using DSP chips, which are designed mainly for DSP operations. [2]

3.5 Power Efficient DSP Applications

From cell phones to personal digital assistants, all dependent on the batteries that power them on. I have a Nokia 6131 which I am using at the moment it is user friendly and has a standard battery, Li-Ion 820 mAh (BL-4C) which lasts up to 9 hours of calls.

Another type of a digital filter is the analog IIR filter. Analog filters utilize analog components with filter continuing signals and IIRs use numerical processing to filter sampled data signals. This leaves both based on pole and zero on yielding Butterworth, Chebyshef, Bessel on filter response curves. An IIR filter offers a computational efficiency and a relatively short time delay, especially when the linear phase is of lesser importance.[4]

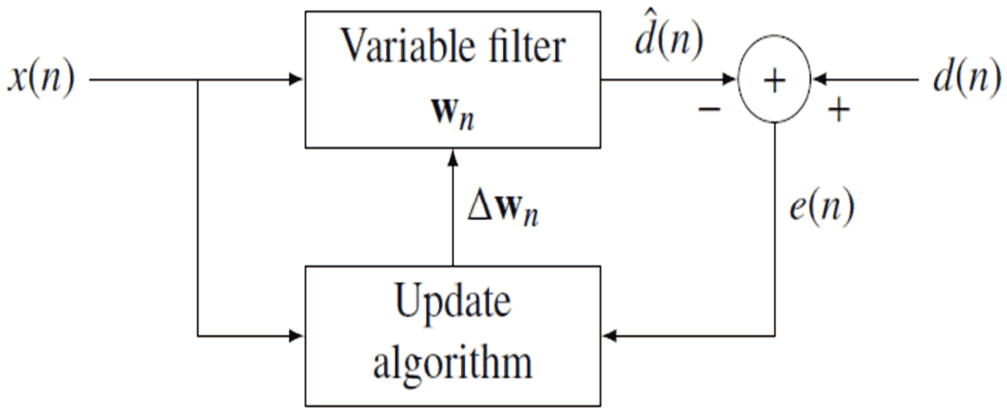

Figure 5. Filter flowing [8]

Figure 5 illustrates the main attraction of an FIR filter flowing through a perfect linear phase response, hence the constant group delay across the frequency band. [1]

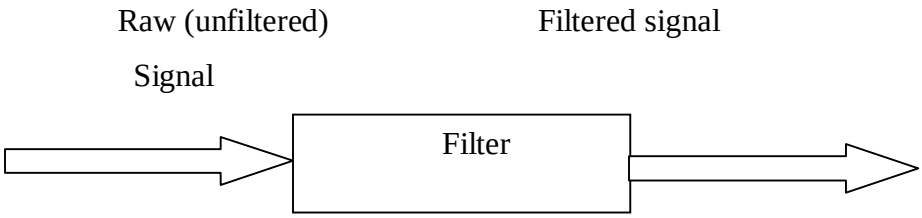

Figure 6. Analog and digital filters [8]

The block diagram in figure 6 illustrates the function of a filter to remove unwanted parts of the signal, such as random noise, or to extract useful parts of the signal, such as the components lying within a certain frequency range. They are different in analog and digital physical makeups.

An analog filter uses analog electronic circuits made up of components such as resistors, capacitors and opamps to produce the required filtering effect. Such filter circuits are widely used in applications such as noise reduction and video signal enhancement. There are well established standard techniques for designing an analog filter circuit for a given requirement.

At all stages, the signal being filtered is an electrical voltage or current which is the direct analogue of the physical quantity for example a sound or video signal or transducer output involved.

A digital filter uses a digital processor to perform numerical calculations on the sampled values of the signal. The Processor may be a general-purpose computer, such as a PC, or a specialised DSP (Digital Signal Processor) chip resulting in binary numbers. Successive sampled values of the input signal are transferred to the processor, which carries out the numerical calculations on them.

The following diagram in figure 7 shows the basic setup of such a system.

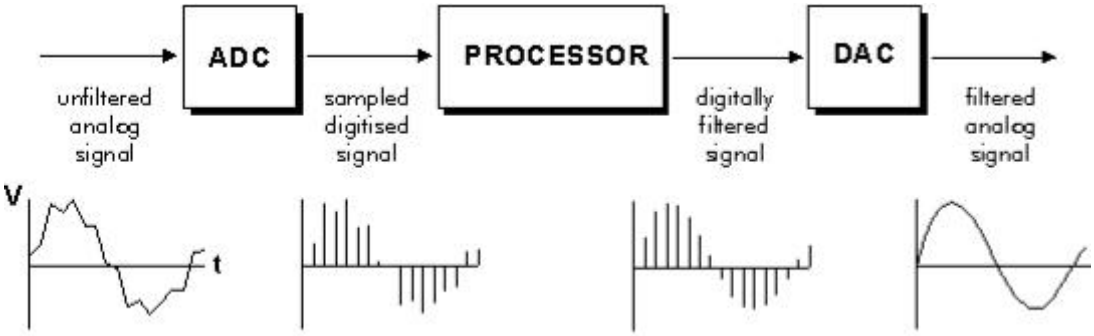

Figure 7. Analog Filters for Data Conversion [4, 48]

Figure 7 explains how the analog input signal is sampled and digitised using an ADC (analog to digital converter), which now represents the sampled values of the filtered signal passing through a DAC (digital to analog converter).When converting the signal back to analog, in a digital filter the signal is represented by a sequence of numbers rather than a voltage or current [4].

4 Electromagnetic Interferences (EMI)

EMI (Electromagnetic Interference) is the disruption of the operation of an electronic device when it is in the vicinity of an electromagnetic field radio frequency (RF) spectrum that is caused by another electronic device. As the high technology field advances, so do the problems from electromagnetic interference (EMI).

EMI issues are increasingly problematic for the system designer as semiconductors in general become faster, more integrated, and unfortunately noisier. Engineers who design layouts with little regard for EMI issues are finding that their designs are not performing to specification or are not working at all. However, most EMI issues can be avoided in advance by using an appropriate system design approach coupled with proper printed circuit board (PCB) layout techniques. [7]

4.1 EMI Transmission

Understanding how noise is transmitted can help identify potential EMI problems in a circuit. For transmission to occur, noise has to be sourced, coupled, and received in a system. Figure 8 illustrates how EMI enters a system. All three elements must be present for any EMI problem to exist. Therefore, if any one of the three is minimized or taken out of the system, the interference is reduced or eliminated.

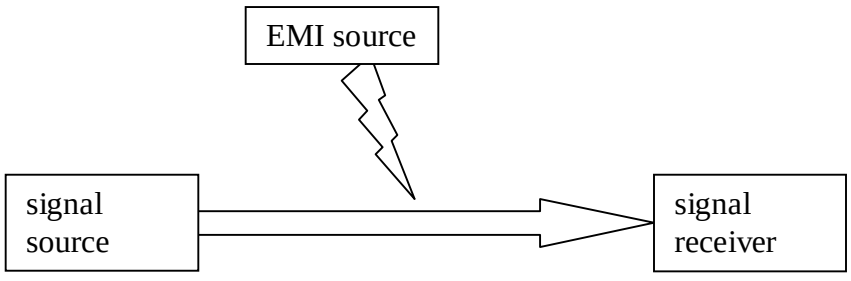

Figure 8. Coupling path EMI pathway [7]

4.2 EMI Sources

The sources of EMI include microprocessors, microcontrollers, electrostatic discharges, transmitters, transient power components, AC supplies, and lightning. Within a microcontroller system the digital clock circuitry is usually the biggest generator of wide-band noise, which is noise that is distributed throughout the frequency spectrum. With the increase of faster semiconductors and faster rates, these circuits can produce harmonic disturbances up to 300 MHz, which should be filtered out. [7]

4.3 Coupling Paths

My study showed that one or more obvious ways the noise can be coupled into a circuit is through conductors. If a wire runs through a noisy environment, the wire will pick up the noise inductively and pass it into the rest of the circuit.

Coupling can also occur in circuits that share common impedances and with radiated electric and magnetic fields which are common to all electrical circuits. Whenever the current changes, electromagnetic waves are generated. These waves couple over to nearby conductors and interfere with other signals within the circuits. [7]

5 Results and Discussion

The main goal was to design a program that counts maximum bits time of 8, 16, 32 and 64 on a Matlab platform, specify the filter design and generate the C header as export bit integer signed or unsigned. The filter coefficients were copied to platform called PSoC, and initialized with the coefficients generated by matlab.

The code designed corresponded to the sample code which was modified to count as one data sample stop timer and return elapsed time in milliseconds. I built and programmed on a PSoC platform and displayed the values on an LCD board, each with a maximum loop of 10, 20, 30, 40, and 50. The output values were recorded in milliseconds with a straight line graph, and the maximum loop of 70 and 140 was carried out in a rough sketch which matches the straight line.

Figure 9 shows the performance of the computing time and loop for one data sample. The point indicates the different word lengths in bits, and the loop value.

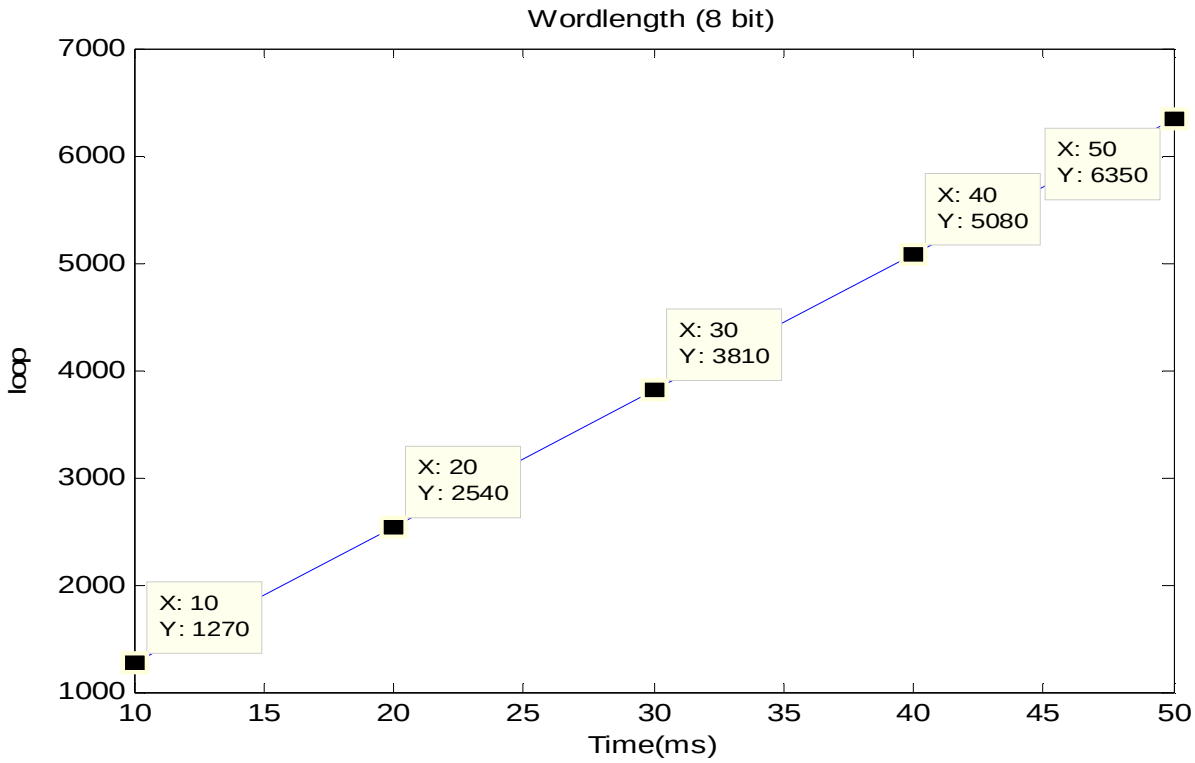

Figure 9. An 8-bit convolutions FIR filter graph

In figure 9, the symbols indicate the following:

y = mx + b where m is the slop and b is intercept

m = slop

b = y intercept

where X is time and Y is the Loop

(X, Y)

(20, 2540)

(30, 3810)

$$m = \frac{y2 - y1}{x2 - x1}$$

$$m = \frac{3810 - 2540}{30 - 20}$$

$$m = \frac{1270}{10} = 127ms$$

y =127 * + b

2540 = 127 *20 +b

2540 -127 * 20 = b

where b = 0 intercept

Table 1. Word length of 8 bits

Word length 8 bit	**Time**	**Loop**
8	1270 ms	10
8	2540 ms	20
8	3810 ms	30
8	5080 ms	40
8	6350 ms	50

Figure 9 and table 1 show the word length of 8 bits (char), 16 bits (int), 32 bits (long int) and 64 as float on performance using PSoC platform. To process each loop (x) and the different in time was recorded in (y) during programming it takes few seconds to display, the times values as the loop increase the output was almost twice for each case. I used the result to plot a graph on Matlab showing the point of x and y values, the mat lab generate C header which was signed as bit integer.

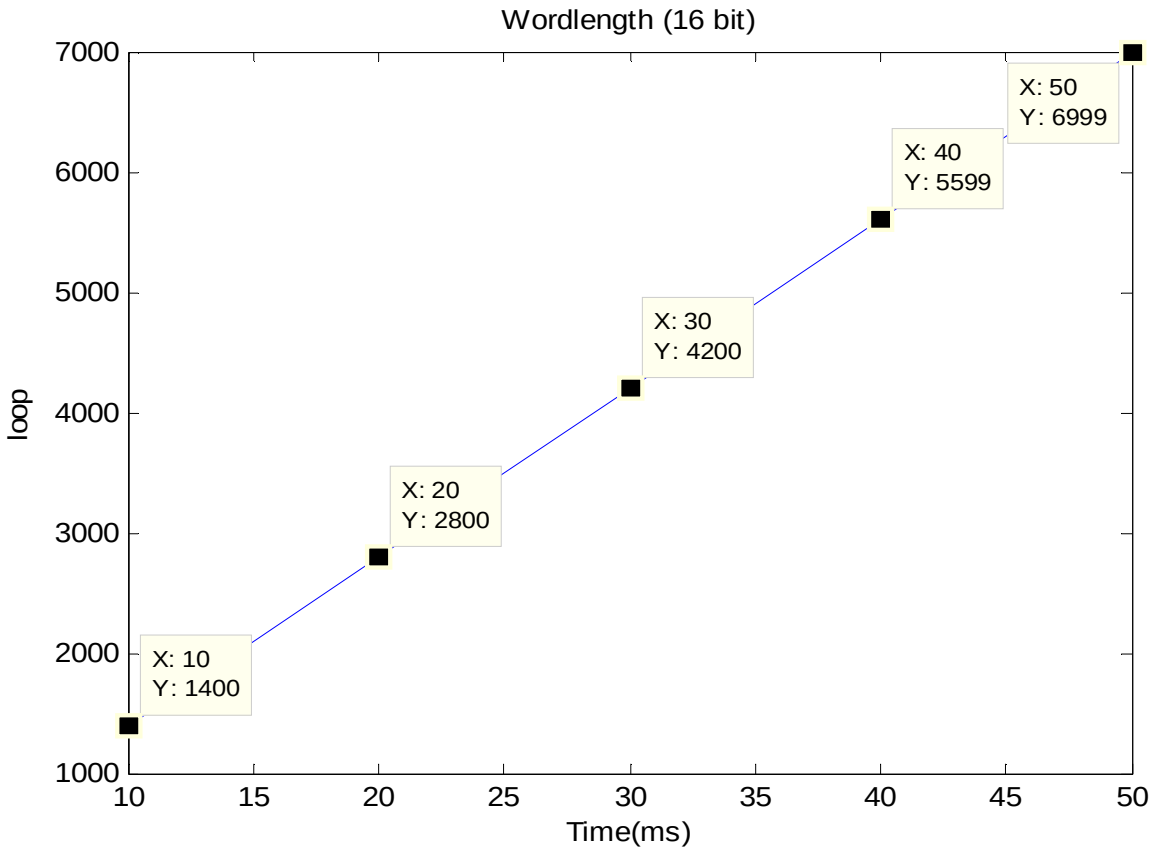

Figure 10. 16-bit convolutions FIR filter graph

Calculation of the slope and intercept of a 16 bit performance shows the values of x and y increased with half of the previous. The point of interception did not fall at zero, but when I expanded the graph, it dropped to zero.

y = mx + b where m is the slop and b is intercept

m = slope

b = y intercept

where x is time and y is the loop

(X, Y)

(20, 2800)

(30, 4200)

$$m = \frac{y2 - y1}{x2 - x1}$$

$$\text{m} = \frac{4200 - 2800}{30 - 20}$$

$m = \frac{1400}{10} = 140ms$

y =140 * + b

2800 = 140 *20 +b

2800 -140 * 20 = b

where b= 0 intercept

Table 2. Word length of 16bits

Word length 16 bit	**Time(y)**	**Loop(x)**
16	1400 ms	10
16	2800 ms	20
16	4200 ms	30
16	5599 ms	40
16	6999 ms	50

Figure 10 and table 2 above show the word length of 16-bits using the Filter Design & Analysis Tool to design the accurate 16 int filter coefficients on the PSoC platform.

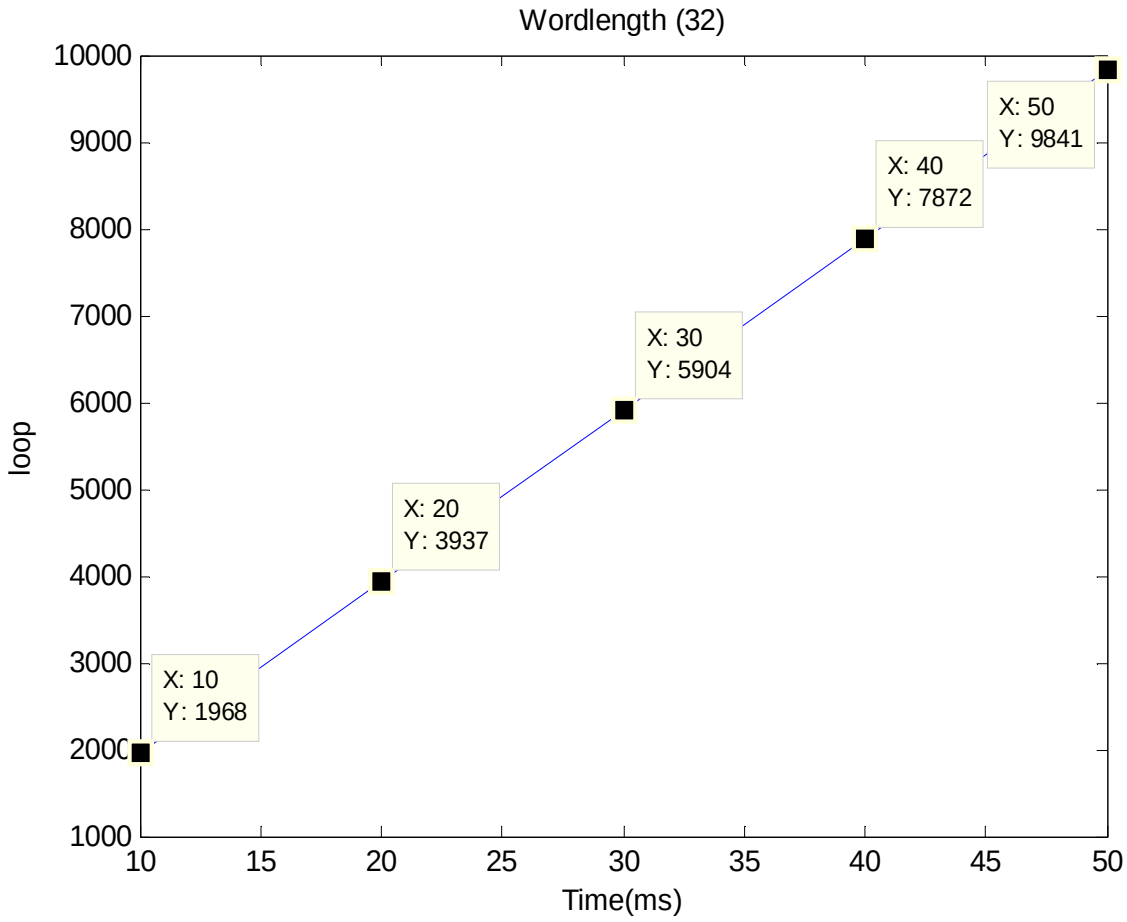

Figure 11. 32-bit convolutions FIR filter graph

Figure 11 and table 3 show the word length of 32-bits using the filter coefficients generated from Matlab to the PSoC, to process the computing time taking.

y = mx + b where m is the slop and b is intercept

m= slope

b = y intercept

where x is time and y is the loop

(X, Y)

(20, 3937)

(30, 5904)

$$m = \frac{y2 - y1}{x2 - x1}$$

$$\text{m} = \frac{5904 - 3937}{30 - 20}$$

$m = \frac{1967}{10} = 196.7ms$

y = 196.7 * + b

3937 = 196.7 *20 +b

3937 - 196.7 * 20 = b

where b = 3 intercept

Table 3. Word length of 32 bits

Word length 32 bit	**Time(y)**	**Loop(x)**
32	1968 ms	10
32	3937 ms	20
32	5904 ms	30
32	7872 ms	40
32	9841 ms	50

Figure 11 and table 3 show the word length of 32 bits on performance using PSoC platform. To process each loop (x) and the different in time was recorded in (y) during programming it takes few seconds to display.

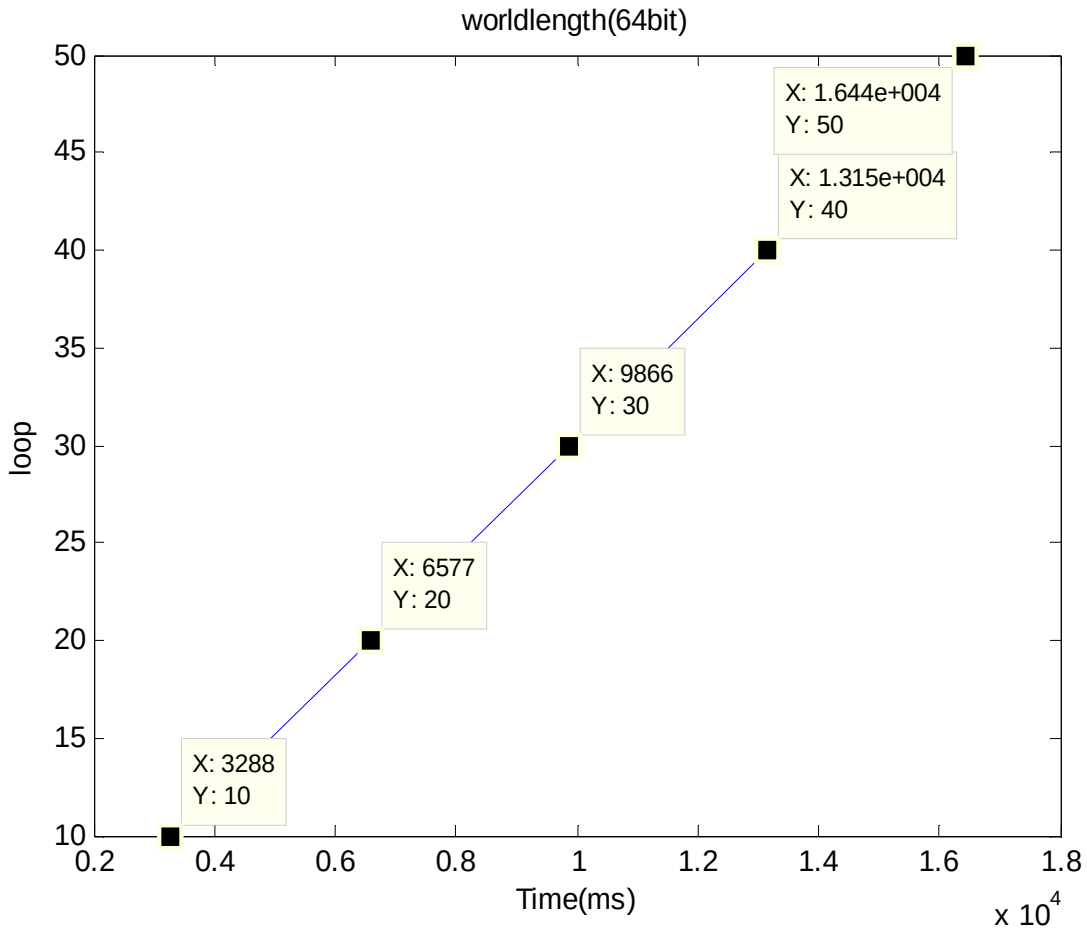

Figure 12. 64 bit convolutions FIR filter graph

Figure 12 and table 4 show the word length of 64-bits floating, using the filter coefficients generated by Matlab, which one transferred to the PSoC platform in order to process the computing time taking for one data sampling.

y= mx + b where m is the slop and b is intercept.

m= slope

b= y intercept

where x is time and y is the loop

(X, Y)

(20, 6577)

(30, 9866)

$$m = \frac{y2 - y1}{x2 - x1}$$

$$m=\frac{9866-6577}{30-20}$$

$$m=\frac{3289}{10}=328.9m\ s$$

y =328.9 * + b

6577= 328.9 *20 +b

6577 -328.9 * 20 = b

where b = -1 intercept

Table 4. Word length of 64 bits

Word length 64 bit	**Time(y)**	**Loop(x)**
64	3288 ms	10
64	6577 ms	20
64	9866 ms	30
64	13154 ms	40
64	16442 ms	50

Figure 12 and table 4 above show the word length of 64 bits floating using the Filter Design & Analysis Tool.

6 Conclusions

The goal was to design a program that counts maximum bits time of 8, 16, 32 and 64 on a Matlab platform, specify the filter design and generate the C header as export bit integer signed or unsigned. The Filter coefficients were copied to a platform called PSoC, and initialized with the coefficients generated by matlab.

The testing was carried out with different length filter structure in order to determine the time and loop taking for each process to display on an LCD screen. The lack of verification does represent a weakness in this project, however, and further research and testing could be performed in this area.

During the study on efficient digital filters I discovered that a DSP processor uses a signal that comes from the real world. It must be capable of changing behaviour based on what it sees in the real world. We live in an analog world in which the information around us changes sometimes very quickly. A DSP system must be able to process these analog signals and respond in a timely manner.

References

1 Steven W. Smith. The scientist and engineer's guide to digital signal processing 1997

2 Edmund Lai. Practical Digital Signal processing for engineers and technicians 2004

3 Dietrich Schlichthärle. Digital filters basics and design, Bosch Telecom Frankfurt 2000

4 M. D. Lutovac, D. V. Tosic, B. L. Evans. Filter Design for Signal Processing using MATLAB and Mathematics

5 Miroslav .D. Lutovac, Dejan.V. Tosic, Brian L. Evans. Filter design for signal process using MATLAB and Mathematical. 2000.

6 John G. Proakis and Dimitris G. Manolakis. Digital signal processing principles, algorithms, and applications, 3rd edition 1996

7 Saeed V. Vaseghi. Advanced digital signal processing and noise reduction 3rd ed. 2006

8 Simon Haykin. Adaptive Filter Theory Prentice Hall, 2002

9 Richard Lee Ozenbaugh. EMI Filter Design. 2nd edition revised and expanded. 2001

Appendix 1, Timer maximum counts

```
#include <m8c.h>        // part specific constants and macros
#include "fdacoefs.h"   // coefficients generated by mat lab
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#include "PRS8.h"       // The cycling timer type PRS 8 generates a control
pulse
                        // which, after the preset blowdown interval.

#define MaxPeriod 0xffff // 16 bit timer maximum count
#define MaxLoop 10       // Number of loops to perform
#define NUM_SAMPLES 20

// Function prototypes
void InitAll(void);
void Tic(void);
WORD Toc(void);
void LCD_PrDecimalWord(WORD xx);
void fil(unsigned int samples_count);
DTYPE input(void);
void output(DTYPE_DOUBLE y);

// Global variables
DTYPE Input[FILTER_LENGTH];
unsigned int Input_oldest = 0;
DTYPE_DOUBLE Output[FILTER_LENGTH];
unsigned int Output_oldest = 0;

// Sample array
DTYPE Samples[NUM_SAMPLES] = {
           43, 57, 12, 54, 11,
           32, 57, 23, 44, 1,
           3, 68, 13, 54, 21,
           43, 61, 3, 4, 16
};

// main routine

void main()
{
   char xx,yy,zz;     // test variables
   unsigned int ii;   // loop counter
   WORD wElapsedTime; // duh...

 // initialize all resources
 InitAll();

 // loop forever testing the execution time
  while(1)
```

```
{
   // start execution timer
Tic();

 // Process 10 times per 32 data samples
 // for (ii = 0; ii < MaxLoop; ii++){
    fil(1);
//}

// put your test above this line-------------------

// get elapsed time
wElapsedTime = Toc(); // Toc() returns the elapsed time in milliseconds,

// Display the result
LCD_Position(0,6);    // first line, column 6 of LCD
LCD_PrDecimalWord(wElapsedTime);  // print result as decimal number

 } // while

}

DTYPE input(void) {
static int i = 0;

if (i < NUM_SAMPLES) {
return Samples[i++];
} else
    {

i = 0;

return Samples[i++];
}
    }

void output(DTYPE_DOUBLE y) {
Output[Output_oldest] = y;
Output_oldest = (Output_oldest + 1) % FILTER_LENGTH;
}

/*
 * FIR Netrino.com
 */

void fil(unsigned int samples_count)
{
           DTYPE sample = 0;
           unsigned int i = 0, j = 0;
           DTYPE_DOUBLE y = 0;

           for (i = 0; i < samples_count; i++) {
          /*
           * Sample the input signal (perhaps via A/D).
           */
           sample = (DTYPE) PRS8_bReadPRS(); // random data from hardware
PRS-generator
```

```
                        /*
                        * Insert the newest sample into an N-sample circular
buffer.
                        * The oldest sample in the circular buffer is
overwritten.
                        */
                        Input[Input_oldest] = sample;

                        /*
                        * Multiply the last H inputs by the appropriate
coefficients.
                        * Their sum is the current output.
                        */
                        y = 0;

                        for (j = 0; j < FILTER_LENGTH; j++)
                        {
                        y += H[j] * Input[(Input_oldest + j) % FILTER_LENGTH];
                        }

                        Input_oldest = (Input_oldest + 1) % FILTER_LENGTH;

                        /*
                        * Output the result.
                        */
                        // output(y); // just extra overhead...
                }
}

// InitAll() initializes everything
void InitAll(void){
                char theStr[] = "Time:       ms";    // couple of strings for
legend
                char otherStr[] = "Loops:" ;                            //

   M8C_EnableGInt;    // duh...

   // initialize sleep timer
   SleepTimer_Start();
   SleepTimer_SetInterval(SleepTimer_8_HZ);   // Set interrupt to a  7 Hz
   SleepTimer_EnableInt();

                // Initialize LCD and print legend
   LCD_Start();
   LCD_Position(0,0);
   LCD_PrString(theStr);
   LCD_Position(1,0);
   LCD_PrString(otherStr);

   // print loop counter value
   LCD_Position(1,6);
   LCD_PrDecimalWord((WORD) MaxLoop);

   // initialize 16 bit timer
   Timer16_WritePeriod(MaxPeriod);
   Timer16_WriteCompareValue(0x0001);
   Timer16_EnableInt();
   Timer8_DisableInt();
```

```
   Timer8_Start();

   // PSR8
   PRS8_Start();

} // InitAll()

// Tic() starts 16 bit timer
void Tic(void){
              Timer16_WritePeriod(MaxPeriod);
              Timer16_Start();
} // Tic()

// Toc() reads the 16 bit timer count, stops the timer,
// and return elapsed time in milliseconds
WORD Toc(void){
           WORD wElapsedTime ;

           wElapsedTime = MaxPeriod - Timer16_wReadTimer();
           Timer16_Stop();

           return wElapsedTime;
} // Toc()

// LCD_PrDecimalWord() prints an unsigned integer value
// in deimal format
void LCD_PrDecimalWord(WORD xx){
           char ii;
           signed char cnt;
           WORD tmp;
           char str[] = "     ";

           tmp = xx;
           cnt = 4;
           while(tmp){
                     ii = tmp%10;
                     str[cnt--] = '0'+ii;
                     tmp -= ii;
                     tmp /= 10;
                     } // while

           for(; cnt >= 0; cnt--)
                     str[cnt] = ' ';

           LCD_PrString(str);
} // LCD_PrDecimalWord()
```