Department of Electrical and Electronic Engineering
Imperial College London

EE3-19

Real Time Digital Signal Processing

Course homepage: http://learn.imperial.ac.uk

Lab 5 – Real-time Implementation of IIR Filters

Paul D. Mitcheson
paul.mitcheson@imperial.ac.uk
Room 1112, EEE

Imperial College London

Objectives

- Learn to design IIR filters using Matlab.
- Implement the IIR filter using the C6713 DSK system in real-time.
- Measure the filter characteristics using a network or spectrum analyzer.

Setting up dynamic memory

Make a copy of the lab 4 project folder. Paste the folder uner the same root. Rename it to ***lab5.*** Power cycle the DSK hardware and ensure the USB cable is connected. Open the lab5 project in CCS.

Before you start you need to set up some memory on the DSK using the DSP BIOS tool. This is because rather than using statically defined arrays to hold data you will be using dynamic arrays that can be created on the fly (this applies to subsequent labs too). You need to so ensure you have this lab working and set up correctly as it will provide a template to the final project.

Open the C/C++ projects window. Double click the ***DSP_BIOS_.tcf*** file in this window to open the configuration tool (figure 1). Click on the + symbol next to **System**

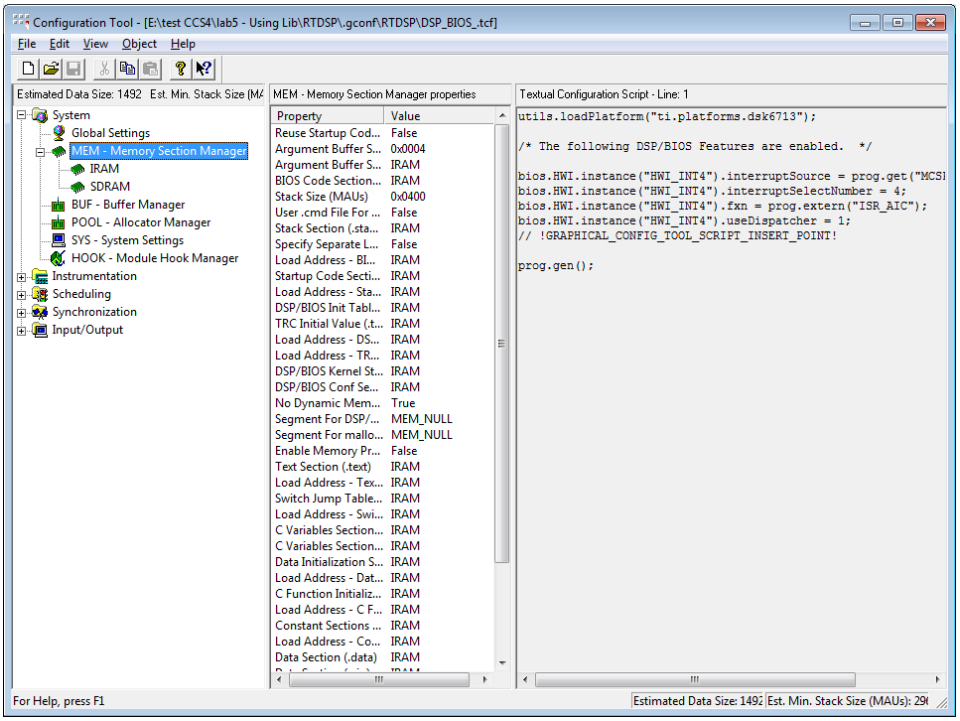


Figure 1 The configuration tool

Right click and select **Properties** from the **MEM – Memory Section Manager** option in the tree (shown selected in figure 1). In the **MEM** pop up window uncheck the box **No Dynamic Memory Heaps** as shown in figure 2. A warning pops up (we will do what it requires in the next step) so hit **Ok** to clear it. Hit **Apply** and then Hit **OK**

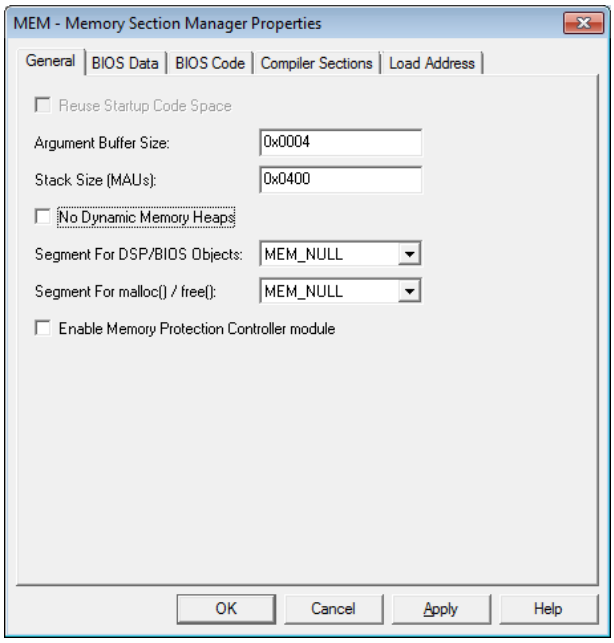


Figure 2 Assigning Dynamic Memory

Back in the Configuration tool clikc the + symbol next to **MEM** then right click **IRAM** and select properties. Check the option **create a heap in this memory** (figure 3). Hit **Apply** then **Ok**

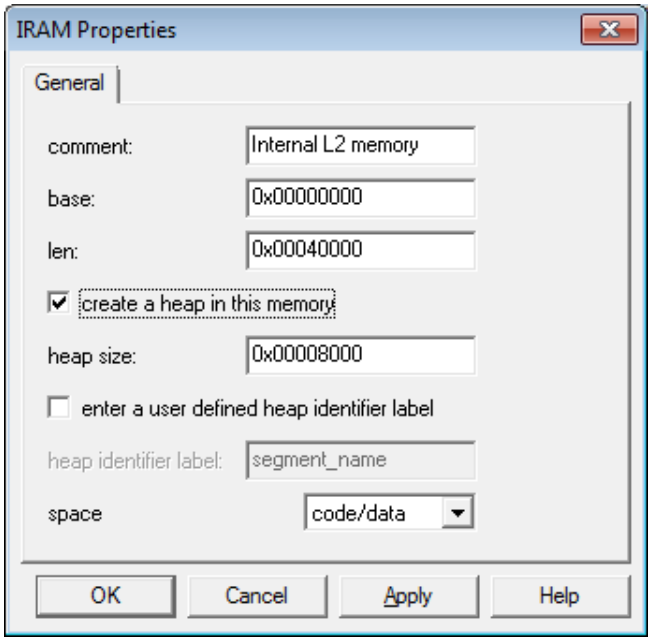


Figure 3 Creating a Heap

Now go back and right click and select **Properties** from the **MEM – Memory Section Manager** and ensure that the memory segments for both **DSP/BIOS Objects** and **malloc()/free** are both set to IRAM as shown in figure 4. Hit **Apply** then **OK.**

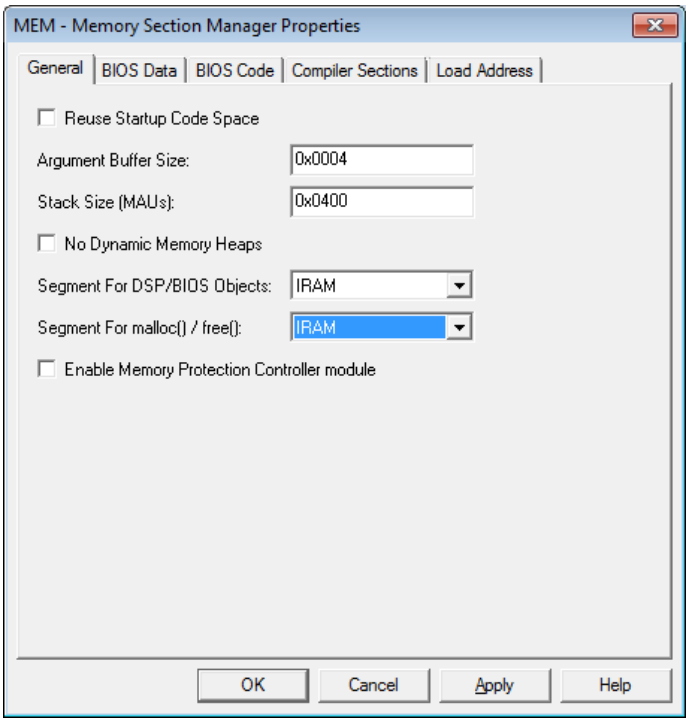


Figure 4 Setting malloc() function calls to use IRAM memory

Later on in the lab you will be using a function called Calloc() to declare dynamic arrays (calloc() is a similar function to malloc()). The above steps have ensured that the arrays declared will be written into Internal RAM (IRAM).

Close the Configuration Tool window (make sure you save the changes when asked to do so in the pop up).

There is no additional C file top obtain from blackboard for this project. Use the existing C file and modify its contents to do the following exercises.

Introduction

In this lab you will use Matlab to design some IIR filters and then implement them in real time on the DSP system.

The transfer function of an IIR filter is given by:

$$H(z) = \frac{b_0 + b_1 Z^{-1} + b_2 Z^{-2} + ... + b_M Z^{-M}}{1 + a_1 Z^{-1} + a_2 Z^{-2} + ... + a_N Z^{-N}}$$

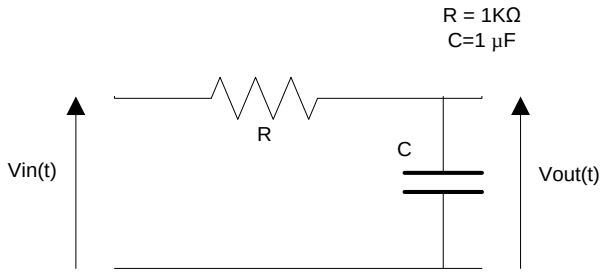
The corresponding time-domain difference equation is:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + ... + b_M x(n-M) - a_1 y(n-1) - a_2 y(n-2) - ... - a_N y(n-N)$$

Single-pole filter

Your first task is to map an analogue filter into a discrete time version (as an IIR filter) using the Tustin transform.

The analogue filer is given below:



Since the cut-off frequency in this circuit is low in comparison to the sampling frequency, a good approximation can be achieved without applying frequency warping.

- Find the transfer function of the circuit as a Laplace transform.
- The sampling frequency of the discrete filter will be 8KHz. Use the Tustin transform to convert the Laplace domain transfer function into a Z-domain transfer function.
- Manipulate the equation until you have it in the form of an IIR filter.
- Implement this filter on the DSK.

By driving the input with a low frequency square wave[1] find the time constant of the filter. There is also a high pass filter (HPF) on the input to the board so choose the frequency of the square wave carefully so this HPF has little effect on the shape of the waveform and thus you can measure the correct time constant. Verify the frequency response is as expected and find the corner frequency. How does the corner frequency and the time constant compare to what you expect from the analogue circuit?

Bandpass filter: Direct form II

For the next example we want to program an elliptic bandpass filter with the following specifications:

Order: 4th
Passband: 180-450 Hz
Passband ripple: 0.4 dB
Stopband attenuation: 23 dB

This can be designed using the Matlab function ellip. Assume the sampling frequency is set to 8KHZ. You should write a Matlab function that calculates the coefficients for such a filter and writes them in a text file called coef.txt in a format suitable for inclusion into a C program. For example, for a third order filter file might contain:

```
double a[] = {1, -1.76,  1.1829,  -0.2781, };
double b[] = {0.0181, 0.0543, 0.0543, 0.0181, };
```

The comma following the final value is optional, but makes your Matlab routine simpler. Note that the function ellip requires you to specify **frequencies normalized to the Nyquist frequency**.

[1] Remember the line input is rated at $2V_{RMS}$ max

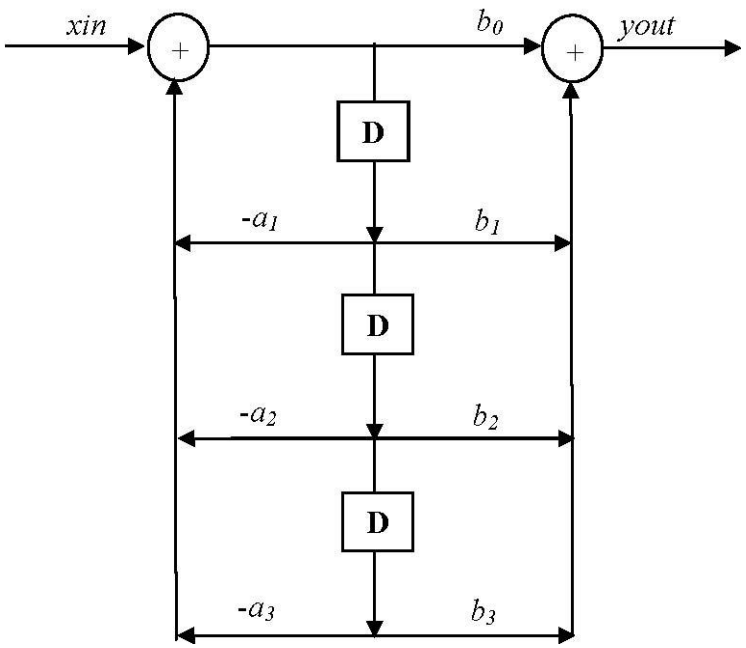


Figure 5: Direct form II structure

Write a C program to implement the IIR filter in direct form II as shown in Figure 5. Your program should work with any filter order, but you can assume that a[] and b[] are the same length.

You can determine the filter order (which is one less than the length of a[]) and allocate the required temporary storage (on the fly) with the following statements:

```
N = sizeof(a)/sizeof(a[0])-1; /*  Find the order of the filter. */

x = (double *)calloc(N+1, sizeof(double));
```

You may require the x[] array to be length N or N+1 according to the nature of your algorithm.

Verify the filter frequency response agrees with the Matlab prediction.

Use the profiling clock method (described in lab1) to determine how many instruction cycles per sample are needed for a filter of order *n* in the form *A* + *Bn*. Wrap the filter code within a function so that the count includes only the instructions between the calls to mono_read_16Bit() and mono_write_16Bit().

Now recompile your program but use the -02 compiler option to optimize the program for speed. See what difference this makes to the number of instruction cycles required.

Bandpass filter: Direct form II transposed

Rewrite your program so that it implements a Direct Form II transposed structure, as shown in Figure 6. Verify the filter response is unchanged. Obtain a similar plot of performance in the form of $A+nB$ as you did in the non-transposed case.

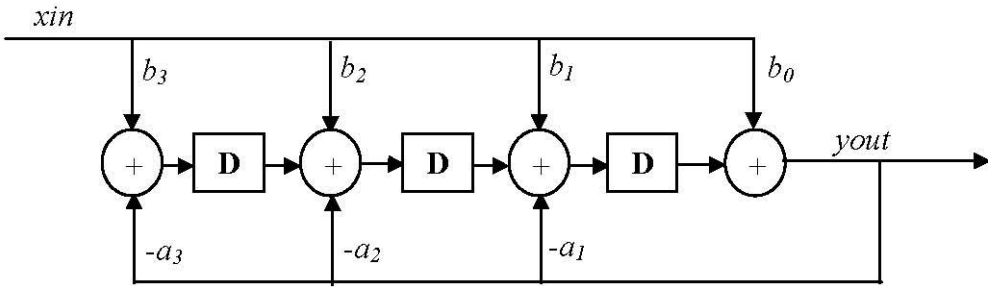


Figure 6 Direct form II transposed structure

Deliverables

As before, you do not have to write a formal report including abstract, conclusions etc for this lab. You are, however, required to write a report which provides program listings and evidence (e.g. graphs) that you have done the exercises. Ensure you cover the points made below.

Marks are awarded as follows:

- Design of simple filter using Tustin transform and derivation of filter coefficients [8]
- Implementation of that filter on DSP [5]
- Measurement of digital response and comparison to response of analogue filter [3]
- Direct form II filter design, implementation, explanation of code, frequency response plot and comparison to Matlab frequency plot. [12]
- Direct form II transposed filter implementation and frequency response plot [6]
- Performance comparison between the transposed and non-transposed filter implementations with justification [5]

There is no need to include all your code as an appendix for this report – just include the core parts of your filters as you describe the implementations.

Revision History

18th Nov 2010 – Rewritten for CCS v4 and Win 7
3rd Feb 2011 – modifications to mark allocation