Lab 8
IIR Filter Design

November 4, 2015

1 Introduction

There are specifications regarding passband, and stopband edges, passband ripple and stopband attenuation when designing filters. There are also several common filter types such as Butterworth, Chebychev and Elliptic filters, each having its advantages and disadvantages. In this lab you will design a high order filter based on the given specification. Furthermore, you will learn more about the boards and how to setup hardware interrupts.

2 Lab

2.1 Code implemention using interrupts and DSP/BIOS ver 5

Interrupts are used to stop the current process in the CPU so that a certain task can be serviced. There are two different types of interrupts. The first kind is hardware interrupt, and the second is software interrupt. The problem with hardware interrupts, is that the CPU cant service multiple interrupts at the same time. This will cause events to be missed and interrupts to be ignored and real-time may be missed. Using software interrupts, different interrupts will be serviced based on the priority level which when designed correctly will help achieve real-time.

The C6713 comes with a variety of the visualization tools. Using the DSP/BIOS real-time analysis tools the performance of the DSP may be monitored, and hence provide a tool for optimization. Some of these tools include:

- CPU load graph,
- Execution graph, and
- Message log.

2.2 Hardware interrupts

1. Start a new project. This time only include the files:

   a `c6713dskinit.c`

   b `c6713dskinit.h`

   c `dsk6713_aic23.h`

   d `dsk6713.gel`

2. Create a `DSP/BIOS` file and add it to the project. Here are the steps needed:

   (a) right click on your project,

   (b) click `new`,

   (c) click `other`,

   (d) select `rtsc`,

   (e) select `dsp/bios v5.x`,

   (f) select `platform ti.platforms.dsk6713`

3. Create a hardware interrupt that is triggered by the data received from the MCBSP1. In the hardware service routine, read the data and retransmit it.

   (a) Open the DSP/BIOS FILE and click on `Scheduling`,

   (b) Click on `HWI-Hardware Interrupt Service Routine Manager`,

   (c) In the `interrupt source field` select `MCSP 1 Receive`

   (d) In the `function field` enter the name of the hardware interrupt function that you want to create. *This field needs the assembly name of the function which is the name that you use in C preceded by an underscore*,

   (e) Click on the `Dispatcher` tab and select `Use Dispatcher`.

4. Put your code in the hardware interrupt function.

5. In your main, the only thing you need is the `comm intr()` function.

6. Test your hardware interrupt by connecting the input to either the function generator or to any other source and make sure that you are getting the same thing out.

7. Turn on the `CPU Load Graph` from the `DSP/BIOS` menu,

   (a) click on `tools`

   (b) click on `RTOS Analyzer`

   (c) click on `RTA/Legacy`

   (d) click on `cpu load`

2.3 Fixed-point vs floating-point

Floating-point arithmetic requires more clock cycles than fixed-point. Depending on your application, using fixed-point might be desirable. To design a filter function that uses fixed-point arithmetic, you will need to use `int` (signed 32-bits) or `short` (signed 16-bits) for all your data types. But, the filter coefficients are of type float usually between -1 and 1. Therefore, you need to scale your filter coefficients by 2^{n-1} where n is the desired number of bits including a sign bit, then round the numbers to an integer. Further scaling is required if the coefficients are not between -1 and 1.

1. Design an elliptic filter with the following specification
   - $f_{pass} = 4kHz$,
   - $A_{pass} = 0.1dB$,
   - $f_{stop} = 4.5kHz$,
   - $A_{stop} = 50dB$,
   - $f_s = 48kHz$.
2. Turn on the `CPU Load Graph` from the `DSP/BIOS` menu,
3. Scale the coefficients to obtain them in 16-bit format and store them.
4. Design a filter that uses fixed-point operation.
5. There are several ways to group the poles and zeros of the filters. Implement the filter twice: once by pairing the poles closest to the unit circle and go inward, and the other by pairing the poles furthest from the unit circle and go outward. Each time pair the poles with the zeros closest to it.
6. Compare the performance of the two implementations. run your code and record the CPU load.
7. Compare the CPU load of the fixed point approach with the floating point approach of implementing a filter.

2.4 Extra Credit

2.4.1 Using log events

In order to observe the operation of the DSP, the `printf` may be used to display the values of certain variables. This will work, but in the implementation, the `printf` function is not optimized and will load the CPU and eventually will prevent it from achieving real-time. Another way is to create a log-event and use the `LOG printf` function instead.

1. Add to your source file: `#include <stdio.h>`, and modify your program to use the `printf` function to display the data that you are reading.
2. What is the CPU load? Explain.
3. Now we will create a *log event*.
   (a) Open the `DSP/BIOS FILE` and click on `Instrumentation`.
   (b) Right-Click on `LOG -Event Manager` and select `Insert LOG`.
   (c) Rename the log event to something meaningful,

(d) View the properties of the log event you just created and select `printf` to be the `datatype`.

(e) Now, in your code, instead of using the `printf` use `LOG printf`. *It is your responsibility to find out the required arguments for this function.*

(f) Turn on the `Message Log` from the `DSP/BIOS` menu.

(g) Run your code and make sure it works.

2.4.2 Software interrupts

Now we will try to create a software interrupt that will take advantage of the scheduling capability of the board. Here is how you can create a software interrupt.

1. Open the `DSP/BIOS FILE` and click on `Scheduling`,
2. Right-Click on `SWI-Software Interrupt Manager` and select `Insert SWI`,
3. Rename the `SWI` to something meaningful.
4. View the properties of the created SWI and insert the name of the function that you intend to create. *Again, do not forget to precede the name of the software interrupt function by an underscore*
5. In your source file create the software interrupt function that will be executed when an interrupt occurs.
6. You can `cut and paste` the same code that you had in your hardware interrupt function (`HWI`),
7. Now in your `HWI` interrupt function use the following function: `SWI post(&swi handle)`. Where `swi handle` is the name of your software interrupt.
8. Test your code and make sure it works.