Chapter 1

Lab 1 - Firmware

1.1 General description

Digital signal processing (DSP) is everywhere in our daily life. Meanwhile, DSP can be meant as quite different things ranging from `matlab` programming to hardware implementation. In a real-world project, engineers usually start from `matlab` modeling of the algorithm. However, this is not enough because `matlab` models are usually not real-time. Therefore, the algorithm needs to be implemented using for example DSP processors or fixed-functional hardware, which is one of the major tasks of embedded system design.

The purpose of LAB 1 is to go through the top-down flow from `matlab` modeling to assembly programming using the DSP processor **Senior** presented in this course. Although the lab is based on a simple example, the same flow can be applied to other more complicated DSP applications. Through this lab, you should be able to learn basic skills of assembly programming using the **Senior** instruction set. Furthermore, you should be aware of the finite-length error introduced in digital signal processing in the real-world implementation.

1.2 Task 1: Use `matlab` to create a lowpass filter

Download and decompress the source code for lab 1 from the course web page. Start `matlab` by typing `matlab`. Find the `matlab` file `lab1.m` in the uncompressed directory. This file is a demonstration of how 50 Hz line noise can be removed from an ECG signal by using a lowpass filter.

Change `lab1.m` so that the `fir1()` call creates the coefficients for a lowpass filter with a cut-off frequency of 15 Hz. To see exactly how `fir1()` works, type `help fir1` in `matlab`, although the following excerpt from the help page should be enough for this lab:

> *B = FIR1(N,Wn) designs an N'th order lowpass FIR digital filter and returns the filter coefficients in length N+1 vector B. The cut-off frequency Wn must be between 0 < Wn < 1.0, with 1.0 corresponding to half the sample rate.*

After you have changed the script, run the `lab1.m` script. If everything is correct, you should see two windows with a time domain and frequency domain analysis of the signals. In the time domain window you should be able to see that the ECG signal has now been filtered and the frequency domain analysis should also reflect the fact that virtually no energy remains at 50 Hz.

1.3 Task 2: Run and modify a simple assembly program

Find the assembly source file `helloworld.asm` which contains a small assembler program example. Use the assembler to convert the assembly source file to the binary code which can be understood and executed by the instruction set simulator.

Read through the example file, then try to use the `srasm` assembler (section 0.3) to convert the assembly source file to the binary code which can be understood and executed by the instruction set simulator (section 0.4).

Start `srsim` and run one instruction at a time until the program is finished. This can be done in the following way:

- Tell the simulator to run one cycle `r 1`
- Press enter to repeat the last command. (Hold down enter to single step faster.) Notice that some of the registers that have been printed to the screen have changed.
- If you want to see where the PC is located you can use the `l` command to list the source code surrounding the current program counter.
- You can see a few other commands that the simulator has by using the `h` command.
- If you want to run the simulator until it reaches the end of the program, use the `g` command.
- Take note of how many clock cycles that were used.
- Press q to exit the simulator when done

Open the `IOS0011` file and verify that they contain the expected output, that is, all numbers from 0 to 42.

Change the program so that a `repeat` based loop is used instead of a loop which uses normal conditional branches. When done correctly you should be able to run the same program in less than 100 clock cycles.

1.4 Task 3: Implement a single sample 32 tap FIR filter in assembler

In this task you are supposed to implement an interrupt handler friendly version of an FIR filter. Being interrupt friendly means that the FIR filter may not modify any register unless it can restore that register to its original value.

Open `lab1.asm` and try to understand what the file is doing:

1. Setup the stack pointer
2. Setup FIR kernel parameters
3. Initialize the sanity checker (which sets almost all registers to known values)
4. For 1000 samples, call the `fir_kernel` subroutine. This subroutine should read one sample from the input port (0x10), add this to the ring buffer, filter it using a 32 tap FIR filter, and write one sample to the output port (0x11).
5. Run the sanity checker that makes sure that (almost) all registers contain the same value they were set to in the beginning.

6. Quit the program (by writing to port 0x13)

In order to do this you need to perform at least the following tasks:

1. Convert the filter coefficients you created in `matlab` into suitable fixed point constants and enter them into `lab1.asm`
2. Finish the `fir_kernel` function.
3. Assemble and run `lab1.asm`
4. Verify that the output of the filter is correct. You can do this by running the `lab1.m` script in `matlab`. The relative error should be very small (less than 0.001) if everything works correctly.

1.5 Task 4: Scaling the coefficients

Once you have gotten the FIR filter working you should ensure that you get as small relative error as possible. Two things are required for this: You need to scale the coefficients so that you utilize as much as possible of the available 16 bits. You also need to round the coefficients correctly.

1.6 Task 5: Performance measurement

You now need to measure the performance of your FIR filter under the assumption that it is running as an interrupt handler. In other words, the clock cycles used by the top level loop and the sanitycheck code is unimportant. You must thus separate how much time this code takes by running `lab1.asm` with an empty `handle_sample` which only contains a `ret` instruction. If the processor is running at 10 MHz[1], how much of the CPU time is used by the interrupt handler?

You should also discuss the latency of your signal processing system. That is, approximately how many clock cycles does it take from an interrupt is generated to the time when a new sample is presented on the output of the D/A converter under the assumption that the Senior Processor is connected as shown in figure 1.1.

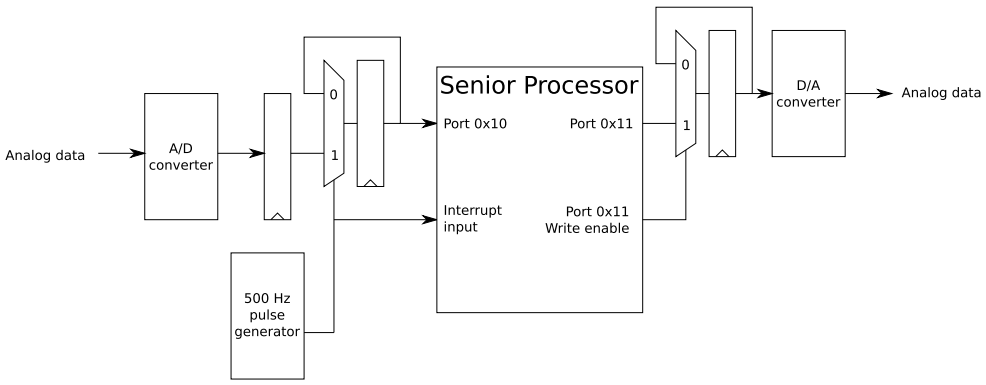

Figure 1.1: The Senior processor as connected to an A/D and D/A converter.

Is the latency always the same in your implementation? Is there anything you can do to reduce the latency?

[1]The reason for the slow clock frequency is probably that we want to save power in a battery operated environment.

1.7 Task 6: Write a 10 sample 32 tap Frame FIR implementation

Modify the filter kernel so that the interrupt handler is responsible for processing 10 samples during the same interrupt. You can assume that the system is connected as shown in figure 1.2. That is, the A/D converter part of the hardware contains a FIFO which stores up to 10 incoming samples. (The D/A converter part contains a similar FIFO.)

In order to test your filter kernel you also need to modify the top level loop so that only 100 iterations are run instead of 1000.

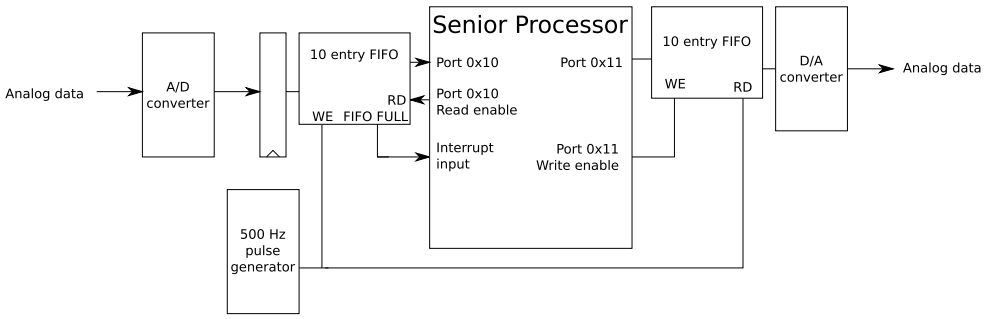

Figure 1.2: The Senior processor as connected to a buffered A/D and D/A converter.

Measure how much CPU time that is used by the interrupt handler. You should also discuss the latency of this implementation.

1.8 Task 7: Reality check

Discuss the following:

- Under what circumstances would you want to use the single sample FIR filter?
- Under what circumstances would you want to use the frame FIR filter?
- Under what circumstances would you want to run these in an interrupt handler?
- Based on your experience in this lab, are there any improvements you would want to make to the Senior processor?