Application Specific Signal Processors: 2019 Course work

The course work consists of a set of smaller assignments. Each group can choose whatever set of assignments they want to complete. More exercises completed gives a better grade.

The exercises are:

1. Fixed Point and Float16 Implementation of N*M in M*K matrix Multiplication (2 points)
2. The FIR filter (2p or 4p)
3. Optimizing the processor interconnect (1p or 2p)
4. Farrow filter (2p)
5. Reducing processor word length (1p)
6. Creating a custom functional unit (3p)
7. FPGA power measurement / using ModelSim (1p)
8. Audio signal processing over external I/O (2p)
9. Personal assignment

Grading

Groups of 1 or 2 persons start with 0 points, groups of 3 persons start with **-1** points. Accepted completion of the **lab** exercises also gives points. If the course teacher has approved your answers for 4 labs, you get **1** additional point. If you complete all exercises your get another 1 point. Doing assignments are prerequisite for grading the projects.

Course grades

3 points:	1
5 points:	2
7 points:	3
9 points:	4
11 or more points:	5

Deadline for finishing exercises is end of November 2019, after that **-1** point for each 10 days of delay. No assignments or project work will be accepted after December.

Application Specific Signal Processors

Useful commands

Some useful information for every

assignment: *Program compilation*

```
tcecc -O3 -o program.tpef source.c -a processor.adf --unroll-threshold=10000
--inlinethreshold=10000
```

Simulate program and display consumed clock cycles

```
ttasim -a processor.adf -p program.tpef -e"run; puts [info proc cycles];quit;"
```

Simulation with a graphical interface

```
proxim processor.adf program.tpef
```

Simulating your design produces the `output.out` file. To verify that you did not break the design, use *diff*. **Notice that when loop unrolling is used, a few samples from the end of the files will differ.**

```
diff output.out reference.out
```

There is also a tool name *typeout* that you can use to print the contents of the stream files:

```
typeout output.out 34 2
```

(34 is the number of samples you wish to see, from the beginning)

How to work with WAV files

In the FIR filter and Farrow filter exercises the processed data is audio, which is in the WAV file format, but converted as a text file for easier interpretation. The WAV files used have 8 bit samples (1 sample is 1 byte) and there is a 44 byte long header. When you process the audio file with your filter, you must copy the 44 header bytes *without modification* to the output stream and only filter the samples that come after the 44 bytes. **The WAV files are in unsigned 8-bit data format, which must generally be transformed to signed-8 bit for filtering.** If you operate with the "int" data type, this can be achieved by reducing 128 from each audio input sample before computations, and by again adding 128 before outputting the filtered signal.

How to measure performance?

In the exercises you must also know the processing time of your filter, which you get by calculating the length of the main filtering loop. A way to do this is to use Proxim “Source” à “Profile data” à “Highlight top execution counts” and see how long many clock cycles the main loop of the filter takes. Alternatively, this can be done by dividing the total execution time by the number of samples, which for example in the FIR case is 16340 (together with the header, there are 16384 samples).

How to estimate processor size?

The size of a processor is given with different measures depending on the implementation technology. The most common way to express the size of silicon circuits (such as ASICs) is to use *number of NAND gate equivalents* N_{ge}.

Another way to express the size of an ASIC is mm^2 (square millimeters), which depends heavily on the technology library.

On FPGAs the circuit size description is more difficult, as the basic blocks of FPGAs depend on the manufacturer and the FPGA product. Moreover, FPGAs contain different kinds of blocks such as *logic cells*, *memory cells* and *DSP elements*. The ASIC size equivalent for each of these block types can be roughly estimated statistically, but the conversion coefficients are FPGA-dependent.

In this course the size of a processor is measured using Altera Quartus II, and a transformation equation is provided to acquire approximate ASIC GEs.

In Altera Quartus II, after *compilation* has successfully finished, the *Project Navigator* shows the resource usage of different components.

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M9Ks	DSP Elements
Cyclone III: EP3C25F324C6						
adaptive	8247 (1)	2930 (0)	0 (0)	202240	26	12
processor:b2v_inst	8128 (0)	2848 (0)	0 (0)	0	0	12
processor_decompressor:decomp						
fu_add_always_1:fu_add_1	131 (99)	96 (96)	0 (0)	0	0	0
fu_add_always_1:fu_add_2	131 (99)	96 (96)	0 (0)	0	0	0
fu_add_eq_gt_gtu_sub_always_1:fu_alu_comp_1	256 (102)	99 (99)	0 (0)	0	0	0
fifo_stream_in_1:fu_input	9 (9)	8 (8)	0 (0)	0	0	0
fu_and_ior_xor_always_1:fu_logic_1	131 (101)	98 (98)	0 (0)	0	0	0
fu_lsu_with_bytemask_always_3:fu_LSU	259 (259)	118 (118)	0 (0)	0	0	0
fu_mul_always_2:fu_mul_1	115 (101)	97 (97)	0 (0)	0	0	6
fu_mul_always_2:fu_mul_2	115 (101)	97 (97)	0 (0)	0	0	6
fifo_stream_out_1:fu_output	7 (7)	5 (5)	0 (0)	0	0	0
fu_shl_shr_shru_always_1:fu_shifter_1	300 (75)	71 (71)	0 (0)	0	0	0
processor_interconn:ic	4059 (1998)	0 (0)	0 (0)	0	0	0
processor_decoder:inst_decoder	1153 (1153)	468 (468)	0 (0)	0	0	0
processor_ifetch:inst_fetch	131 (131)	57 (57)	0 (0)	0	0	0
rf_1wr_1rd_always_1_guarded_0:rf_bool	10 (10)	2 (2)	0 (0)	0	0	0
rf_1wr_1rd_always_1_guarded_0:rf_RF_1	297 (297)	256 (256)	0 (0)	0	0	0
rf_1wr_1rd_always_1_guarded_0:rf_RF_2	297 (297)	256 (256)	0 (0)	0	0	0
rf_1wr_1rd_always_1_guarded_0:rf_RF_3	297 (297)	256 (256)	0 (0)	0	0	0
rf_1wr_1rd_always_1_guarded_0:rf_RF_4	266 (266)	256 (256)	0 (0)	0	0	0
rf_1wr_1rd_always_1_guarded_0:rf_RF_5	297 (297)	256 (256)	0 (0)	0	0	0
rf_1wr_1rd_always_1_guarded_0:rf_RF_6	296 (296)	256 (256)	0 (0)	0	0	0

Figure 1.

The equation to calculate the number of gate equivalents from the Project Navigator is

N_{ge} = 3.73 * (L + D*107), where

L is the number of Logic Cells in the processor and D is the number of DSP elements. Figure 1 shows where to find these numbers. In other words, for an accurate result, the number of Logic Cells must be manually added up from every processor component.

Note: there is a sum of logic cells of the whole processor (8128 in Figure 1), but this value is *not* equal to the added LC count of subcomponents.

In the example of Figure 1, N_{ge} would be 3.73 * (8557 + 12*107) = 36707.

Projects List and details

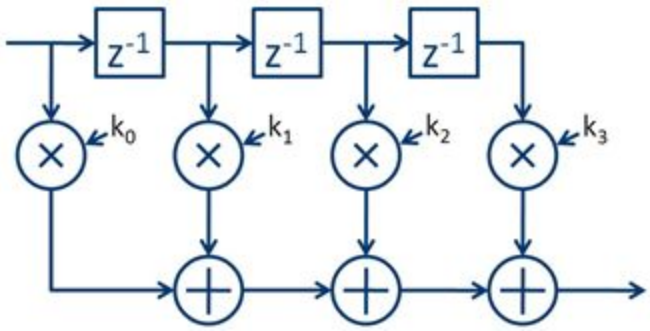
1. Fixed Point Implementation of N*M in M*K matrix Multiplication (2 point)

This program is just a nested loop. The only difference is that it should be implemented as a fixed point program, i.e. it's inputs and outputs are all FLOAT32 but the middle computations all will be fixed point and half floating point and the final results again must be FLOAT32 either printed in screen or stored in a float32 array. Please use the following function format so I can use it in my test code to check the results:

*int Matrxi_Mul(float * A, float * B, float *Result, int rows_of_A, int columns_of_A ,int rows_of_B, int columns_of_B, bool Fix_Or_Float16)*
// A is first matrix, B is the second, Result is the resultant matrix which obviously would be of size rows_of_A in
//Fix_Or_Float16 determines if the middle computations are fixed or float 16
// return 1 ; //if multiplication can be done
// return 0 ; //if multiplication cannot be done

2. The FIR filter (2p or 4p)

In this assignment you will write the C code of a FIR filter from scratch and put the filter on a TTA processor to the FPGA board. Depending on how much you optimize the filter, you can gain 2 or 4 points from this assignment.



Your task is to design a TTA processor that can perform 4-tap FIR filtering and fills one of the following requirements:

Level 1 requirements (more demanding) (4p)

- FIR filtering must take less than or equal to 4 clock cycles per sample

- Estimated chip area must be less than 38 000 gates

Level 2 requirements (less demanding) (2p)

- FIR filtering must take less than or equal to 9 clock cycles per sample - Estimated chip area must be less than 38 000 gates

The FIR coefficients are: $k_0 = 37$, $k_1 = 109$, $k_2 = 109$, $k_3 = 37$ The delay line is supposed to be initialized to 0's. The files that you work with are located in the "ASSP"-folder on the desktop. Make a copy of the files to another folder to work with them. You should create a *C* language program that reads the WAV-audio file named `input.in` and produces a filtered file `output.out`. In the input file there is an irritating noise, which is filtered out. Finally, you must synthesize the FIR filter on the FPGA and make sure that it computes the same result as in the simulation.

3. Optimizing the processor interconnect (1p or 2p)

Reducing the number of connections between functional units (FUs) and buses is an important optimization step. It reduces the area occupied by the processor implementation and increases the clock frequency (MHz) that the processor can achieve.

Apply this exercise to either the farrow filter or FIR filter processor that you have designed.

1. Inside your working folder, create a new processor database file by using the command: (don't care about the *No explorer plugin given* message)

```
explore procdb.dsdb
```

2. Add a starting point processor design file to the database:

```
explore -a [filename].adf procdb.dsdb
```

3. Create a new folder for design exploration, e.g. *work*/explore `mkdir explore`
`cd explore`

4. Inside the design exploration folder, execute the following command to get the LLVM intermediate representation of your program:

```
tcecc -O3 --unroll-threshold=10000 --inline-threshold=10000 -o
program.bc
--emit-llvm [path/to/your/program].c
```
use this file name!

5. In design exploration folder create an empty file named

'correct_simulation_output'. (if our program would print output, we would put it here) touch correct_simulation_output

6. Add the test application to the database:

```
explore -d . ../procdb.dsdb
```

7. Copy the input file to the exploration folder cp ../input.in .

8. Start the interconnect optimization by the command:

```
explore -v -e ConnectionSweeper -u cc_worsening_threshold=2 -s 1
../procdb.dsdb
```

NOTE: -u cc_worsening_threshold parameter defines how many percent

cyclecount is allowed to drop from starting point architecture

-v prints information about optimization progress and best configurations

9. Write out best configuration using command: explore -w [CONF_ID] ../procdb.dsdb

Replace [CONF_ID] with the number given as output of step 8.

10. The reduced-interconnect processor is now in the file [CONF_ID].adf

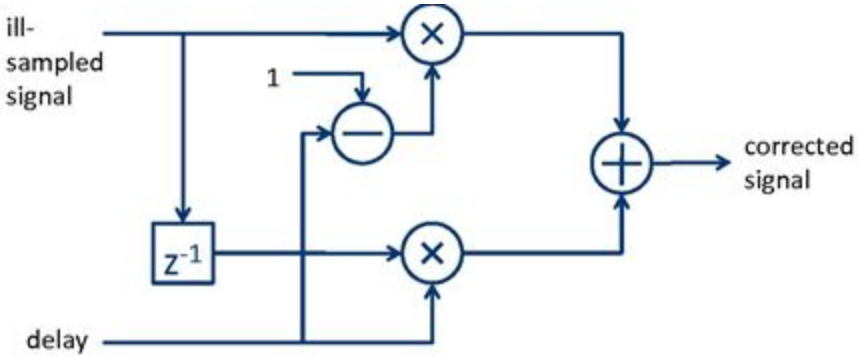
To get **1** point from this exercise, present the ADF file in Prode to the course teacher. To get **2** points, put the reduced processor to FPGA and report how much smaller the reduced processor was than the original one.

4. Farrow filter (2p)

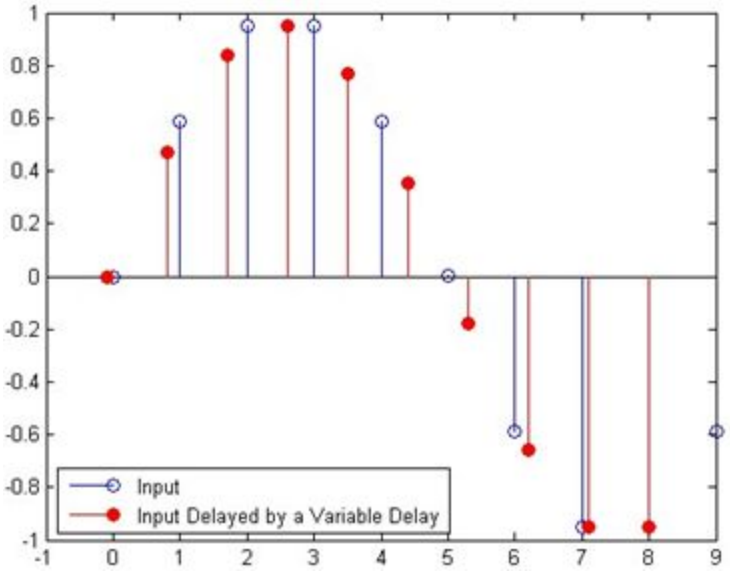
y(n) = [1-d(n)]*x(n) + d(n)*x(n-1)

where x is the ill-sampled signal,

d is the delay signal, and y is the corrected signal



Your assignment is to design a linear Farrow filter from scratch and put the processor on the FPGA. A Farrow filter is used to perform small, variable sampling rate changes. Practically the filter works like an interpolator.



The picture above shows a *Matlab* example of Farrow filtering: the original (ill-sampled) signal is depicted in blue, and the Farrow filter computes the red (corrected) samples for different time instants that are given in the *delay* signal.

Requirements

- The filtering time of one sample must be 6 clock cycles or less

- Estimated chip area must be less than 25 000 gates [19 000 gates if *estimate* was used]

The processor has two signal inputs and one output. To enable two input signals, you need two signal input function units to your processor. If you name them INPUT_1 and INPUT_2, you invoke them from the code by

```
_TCEFU_FIFO_U8_STREAM_IN("INPUT_1", 0, signal1,
status1);
```
and
```
_TCEFU_FIFO_U8_STREAM_IN("INPUT_2", 0, signal2, status2);
```

Notes

The *delay* data is scaled to the left by 4, which means that "16" in the data is 1 and it has a 44 byte header consisting of zeros. The signal data file is a WAV file that can be listened to (and has a 44 byte WAV header). The lengths of both signals are 5669 bytes.

The filtered output signal should be WAV as well. You might not hear a big difference with your ears, but measurements would reveal that the delay error in the signal is reduced by more than a factor 2 though the filtering.

5. Reducing processor wordlength (1p)

In this assignment you reduce the processor's data width from 32 bits to a smaller number, such as 16 bits. You can perform this optimization on any application (FIR or Farrow). Finally, perform an FPGA synthesis and see how much the processor size is reduced. Perform this exercise either on your FIR or Farrow solution.

The processor wordlength is reduced by modifying each *bus* and *port* to the chosen wordlength. The minimum wordlength is application-specific. In practice the wordlength reduction can be done in Prode, which is quite slow, or by directly using find-and-replace in a text editor that has the ADF file open.

Determining the minimum wordlength of the processor is not straightforward; trial-and-error works relatively well as the success of wordlength reduction can be detected already in the simulator.

The procedure is as follows:

1. *Compile program as a TPEF-file for the 32-bit processor*
2. *Make a copy of the 32-bit processor ADF file and reduce the wordlength of the processor copy*
3. *Use ttasim or proxim to run the TPEF –file generated in step 1 on the reduced processor*
4. *If the output is identical, the wordlength reduction did no harm*
5. *If the wordlength is small enough, proceed to FPGA synthesis, otherwise try to reduce further*

Notice that for the LSU unit there is only the 32-bit implementation. Therefore, leave the port width for the LSU to 32 bits in every case.

6. Creating a custom functional unit (3p)

In this exercise you will create a custom functional unit to one of the filters you have designed in the exercises before. Completing this exercise requires that you have some skills in writing VHDL.

The purpose of the custom functional unit is to improve the performance of the signal processor. Practically, the new functional unit replaces a set of filtering instructions with one powerful instruction and thus reduces the cycle count of the filter. A special instruction like this is multiply-accumulate (MAC) that can be found in every commercial signal processor, or a SIMD instruction such as the one that was used in lab 4.

The task of designing a custom FU is not overtly hard, but requires the use of quite a few programs. Instructions for getting started are in the coursework ZIP file under the folder custom_instruction.

7. FPGA power measurement / using ModelSim (1p)

In this additional exercise you are given instructions for measuring the power consumption of your processor (FIR or Farrow). The processor functionality is simulated with Mentor Graphics ModelSim, after which the detailed gatelevel signal activities are handed over to Altera PowerPlay, which has FPGA-model specific power consumption coefficients and can provide accurate estimates of the processor power efficiency.

8. Audio signal processing over external I/O (2p)

In this assignment you receive hardware blocks for attaching a TTA processor to audio I/O connectors of an FPGA board. Audio processing is performed in real-time for an audio signal arriving from an external source (e.g. MP3 player or such). The result is demonstrated in the lab by showing that the audio is filtered by the FPGA.

9. Personal assignment

If you would like to make an exercise of some other topic, it can be proposed. The topic can involve for example multiprocessors such as NIOS II, accessing external I/O (mainly a CCD camera). Compact and clear ideas are preferred.