ENGN2912E: Low Power VLSI System Design

Homework Assignment #3
Due **November 3, 2017, by 5pm**
This homework is to be done alone.

This homework assignment is meant to give you a better understanding of how changes at the algorithmic (behavioral) level of a finite impulse response (FIR) filter can reduce power dissipation without adversely affecting the accuracy of the algorithm itself by allowing a controlled introduction of error into the system. A FIR filter is the most basic type of filter in digital signal processing. With the FIR filter, we will be taking an impulse function input and convolving it with a 25-tap low-pass filter coefficient array.

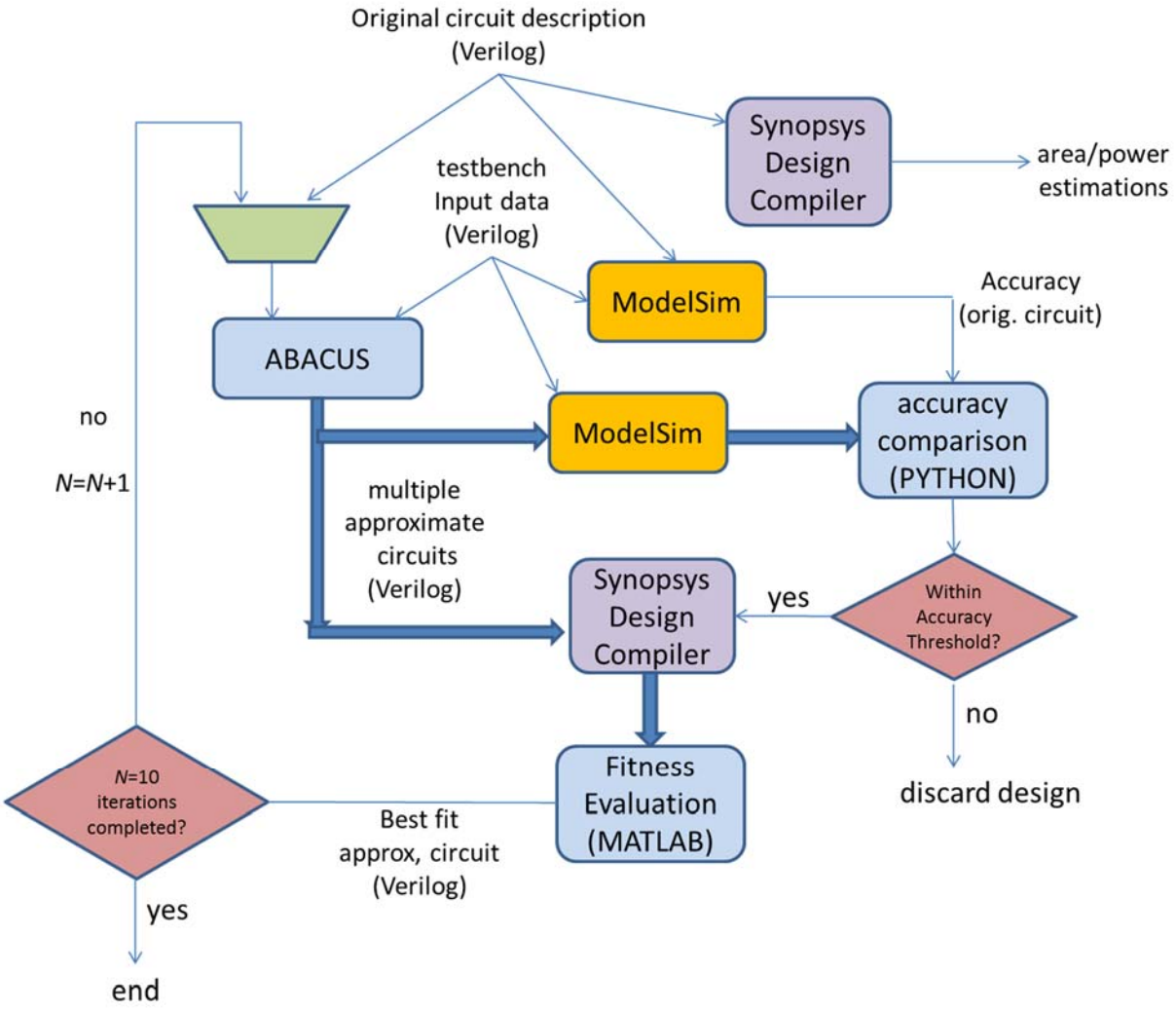
The assignment requires you start with a standard implementation of an FIR filter (as described in behavioral-level Verilog) and design a new approximate version of it that dissipates less power. In our case we will be using a 25-tap FIR filter, where the tap value indicates the number of delays in the filter. The tap value also determines the memory required, number of calculations, and amount of filtering.

The quality of an approximate version of this design is assessed using Mean Squared Error (MSE) and Normalized Cross-Correlation (NCC). The frequency response of the approximate versions generated will be compared against the "golden" implementation (implemented in software with floating point inputs for impulse input data and coefficients) and its closeness is measured using both MSE and NCC. Note, the original Verilog implementation of the FIR filter will also have some MSE, given its 8-bit fixed point implementation compared to the ideal floating point computations done in software.

The comparison between the original and approximate versions of the FIR filter will be done in custom designed hardware. That is, it's not the Verilog code you will be comparing, but rather, the mapping of the two Verilog descriptions to actual hardware implementations. All the tools you need will be available on the CCV machines.

For this assignment, we will be using a mix of commercial tools as well as "in house" tools developed as part of research projects in our lab. Unlike the last homework where you were required to invest some time and effort understanding how to use the Cadence tools, in this homework we will be treating these tools as "black boxes" so you will only need to learn how to run a series of scripts that call these tools to evaluate your circuits. The design flow for this homework assignment is shown on the next page.

The ABACUS tool is at the heart of this design flow. The idea is to use ABACUS to generate multiple approximate versions of your original circuit and guide it into selecting an appropriate "best" design among all the circuits generated. Note that any approximate circuit generated by ABACUS that is determined to have a change in accuracy beyond a certain threshold is immediately discarded and no longer considered for selection. The new "best" design at the end of the iteration is then fed back to ABACUS n times to generate new approximate circuits, in the hope that the best design can be successively improved upon. We will choose to set n=10 for this assignment.



The Synopsys Design Compiler tool is used to generate power and area estimations for your circuit, while a python script is used to compare the change in accuracy of the approximate circuit relative to the original design. As mentioned on the previous page, accuracy is measured by comparing the MSE of the original design with an approximate version. Note that only if the change in accuracy is within a user-defined threshold will the new approximate circuit be considered as a viable solution. By checking accuracy first, we avoid estimating the power and area of these unusable designs.

The fitness evaluation will help you guide the process. This is a short program written in MATLAB that evaluates a cost function weighing accuracy, power, and area such that

$$fitness = \alpha * accuracy + \beta * power + \gamma * area$$

It is up to you to determine what values to use for the weight parameters α, β, and γ.

The final results you obtain for your approximate FIR circuits will vary greatly depending on the values you choose for accuracy threshold, fitness threshold and α, β, γ parameters. You will need to

spend some time tweaking these values to understand how they affect the results and how they may be set to generate the "best" approximate design possible. To accomplish this goal, follow these steps:

1. On the course webpage, you can find a tar file called *abacus_archive.tar* containing a folder with everything you need for this assignment. Copy this tar file onto your home directory on the CCV machines. There are several ways to transfer files to the CCV machines. If you want to use a GUI, WinSCP works well. For the host address use `transfer.ccv.brown.edu`.

2. Extract the files from the tar file using:
   `tar –xvf abacus_archive.tar`
   Once extracted, you will find a folder for the ABACUS tool named *ABACUS_ENGN2912*. This folder will contain several folders called *DataAnalyze, Modified, Original, Population, trunk* and two scripts to run the entire flow --- *launch_ABACUS.sh,* and *run_ABACUS_flow*.

3. The script file *run_ABACUS_flow* is the one you want to open and edit. Open it using your favorite linux text editor(e.g., Vim, Gedit, etc.). Once open, you will see accuracy thresholds and fitness weight parameters defined from line 31 onto line 38. You will not need to change anything besides these lines.

4. Select an accuracy threshold for the MSE. Keep the NCC threshold as it is. Note that the hardware implementation of the filter already has an MSE of 76 compared to the software floating point implementation. It is advised that accuracy threshold values range from 76-200 for the approximate variants.

5. Select values for the weight parameters α, β, and γ. As a guide, it is suggested that you set these parameters such that they sum up to 1. The relative difference between these parameters is perhaps more important that their absolute value. Explain and justify your rationale in choosing these values.

6. Verify the permissions on the following executable files: *launch_ABACUS.sh, run_ABACUS_flow, odin_II.exe* (found in ./trunk/ODIN_II/ODIN_II) by typing `ls –l` to see permission status. Make sure all 3 files are owner executable and change permission as needed using `chmod`).

7. Test out the ABACUS flow by directly executing the script *run_ABACUS_flow* to start the circuit synthesis process to generate an approximate circuit. Before you do this, is it a good idea to change lines 25–26 of the script to produce fewer generations (just for your test run). Also, make sure ModelSim and Design Compiler are loaded before you execute the script or you will get error messages about vsim or dc_shell no being found. These tools can be loaded with the commands:
   ```
   module load synopsys
   module load modelsim/10.1a
   ```

   Note that you will get some error messages that LD_PRELOAD cannot be preloaded. You can ignore this. The script will print out the area, power, and accuracy information after each call to ABACUS (i.e., after each of the n iterations). Record area, power, and accuracy results for all designs (i.e., after the n^{th} generation).

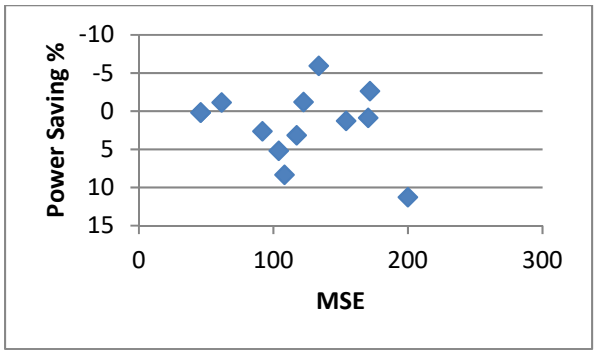
- Area and Power reports for the original design should be stored under *Original/output/* as *area_rpt* and *pwr_rpt*
- Information for approximate designs will be stored in the folder *DataAnalyze/Data/FilesInfo_G<Generation>.txt.* There will be a file for each generation with reports about the MSE, NCC, Area Saving% and Power Saving%

8. Given the results you got for part 7, adjust your accuracy threshold and repeat step 4. You may need to do this a few times to get a good understanding of how you can best guide the process toward a good solution. Please understand, each run of the ABACUS flow with 10 generations can take up to several hours (2–3), so plan your runs accordingly. It may be a good idea to run the tool for fewer generations (again, by adjusting lines 25–26 in the script) to just check how your accuracy thresholds and fitness weights are affecting the results. Note that for your final results you must set the number of generations back to 10.

   IMPORTANT: You will need to allocate resources and time on the CCV machine for your script. These configurations are already managed in the *launch_ABACUS.sh* script. Therefore, to launch the entire flow, run the *launch_ABACUS.sh* script by typing
   `sbatch launch_ABACUS.sh`

9. Given the design points you got so far, try adjusting the fitness parameters, α, β, and γ (as well as the fitness threshold) to see if you can improve upon your design further by repeating steps 4 and 5.

10. Generate a table showing threshold values used, α, β, and γ values and power, area, and accuracy results obtained with these values. Graph power savings vs. accuracy for all the designs you obtained from the tool as shown in the example below:



11. The ABACUS toolflow automatically produces pdf images of the impulse response and filter response of the approximate designs compared to the ideal floating point implementation. Print out these results for what you consider the five best designs. These pdf files are saved in the *DataAnalyze* folder as *figure_G<Generation>F<fileno.>.pdf*

12. Include a comprehensive discussion on your results. How did adjustments in parameter values and thresholds change your results? What was your rationale for the adjustments throughout the process? What design would you consider best and why?

More about the ABACUS tool can be found in the following publications. Their content may be helpful in completing this assignment:

[1] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits," ACM/*IEEE Design Automation and Test in Europe*, March 2014.

[2] Kumud Nepal, Soheil Hashemi, Hokchhay Tann, R. Iris Bahar, Sherief Reda, "Automated High-Level Generation of Low-Power Approximate Computing Circuits," *IEEE Transactions on Emerging Topics in Computing (TETCSI)*, Aug. 2016.