Lab 2: Filter implementations and resource sharing

In this lab, we will observe a few different implementations of the same filter, and the effects of resource sharing on the area, power and speed of the implementations. We will use the Synopsys design_vision tool for this. Note that the number of licenses is limited to 5.

The files required for the lab are stored in /home/nitin/iep/lab2. You can use the command

```
cp -r /home/nitin/iep/lab2 .
```

Or use the file explorer to copy the whole directory.

Inside the directory, you will notice the following files:

1. filter_direct_21.v – Direct form I implementation of 21 tap FIR filter
2. filter_direct_21-random.v – Same as above, but with random coefficients
3. filter_broadcast_21.v – Broadcast implementation of the filter in (1) above
4. filter_broadcast_21-random.v – Broadcast implementation of the filter in (2) above
5. filter_serial_2.v – Serial implementation using only 2 multipliers
6. filter_serial_4.v – Serial implementation using only 4 multipliers
7. synth_fir.tcl – Synopsys synthesis script

Start the synopsys tools by typing

```
design_vision
```

in a terminal. This will bring up a GUI.

In the GUI, close to the bottom there is a space to type commands. Here type in the command

```
source synth_fir.tcl
```

This will run the synthesis script for the filter_direct_21 implementation. At the end, 5 files are generated:

1. filter_direct_21.area – area occupied by the implementation
2. filter_direct_21.time – timing report with slack (assuming a 20ns clock was the target)
3. filter_direct_21.power – power estimate (without considering activity)
4. filter_direct_21.hier – Hierarchy list (modules that are contained in the design)
5. filter_direct_21_syn.v – output Verilog file: gate level netlist

By editing the synth.tcl file, you can change the value of the TOP variable to one of the other names. Then re-run the script to generate the outputs for the new file.

In this lab, we are using the 180nm standard cell library from Faraday technologies (http://www.faraday-tech.com/). This is a freely downloadable library that can be used for your designs. One advantage of this library is that they also provide a memory compiler for efficient memory block implementations. In the present case, we are not using any memory blocks (only flip-flops), so this does not matter.

Notes on the designs

All the verilog code used in this design was generated automatically through a script written in the Python programming language, as part of an Mtech student project by Navneet Dutta in 2009. It is planned to release the code for free use, but at present it has not yet been released.

Coefficients

The coefficients correspond to a specific 21 tap low pass filter. You can observe that many of the coefficients are either 0 or +/- 1, or else have only a single 1 (power of 2 coefficients), which does not require a full multiplier for implementation. Because of this, Synopsys optimizes away the multipliers for these coefficients and replaces them with plain adders.

To avoid this, the *-random* filters have coefficients that have been altered in such a way that this trivial optimization is not possible. Hence the number of multipliers seen in these implementations should match the expected number.

filter_direct

This is just a direct-form-I implementation of the FIR filter. The overall structure of the code is quite easy to understand. By looking at the Verilog code, you can easily identify how the delay blocks, multipliers and adders have been implemented. In all the implementations, we have just used regular '*' operation for multiplication and '+' for addition, allowing the synthesis tool to choose suitable hardware. In this case, Synopsys chooses multipliers and adders from the DesignWare libraries.

filter_broadcast

Same coefficients as above, but now implemented using the Broadcast or Direct-Form-II realization. One thing you can notice here is that the size of the registers used here are all 19 bits wide, as they need to store the adder outputs. In the previous case, they were only storing the input, and hence were only 8 bits wide.

This can easily be optimized further, but the present implementation just uses a simple approach. Hence the broadcast design appears to be much larger. However, you can see that it is capable of higher speed of operation.

-random

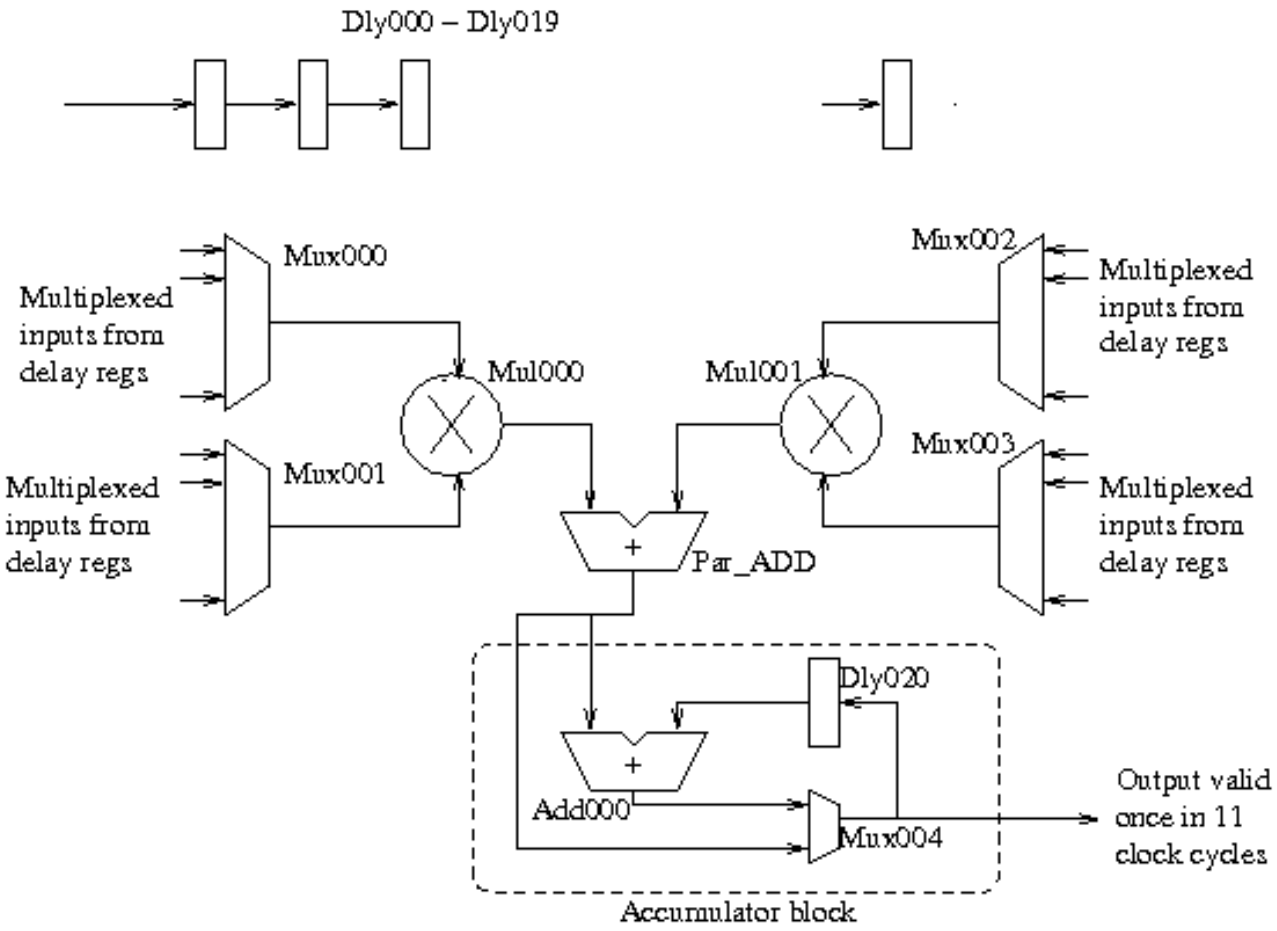
Same as above two filters, but with the coefficients replaced such that they are not trivial (0, +/-1 or power of two). The main difference is that the number of multipliers is more.

filter_direct_serial_2

This is a “serial” implementation, using the concept of resource sharing. The reason for resource sharing is that very often the hardware resource constraint does not permit use of many multipliers. If we can limit the number of multipliers, then we will need multiple clock cycles to complete the filtering operation. For example, with only 2 multipliers, we will need a minimum of 11 clock cycles, at the rate of 2 coefficients per clock cycle. The figure below shows a block diagram of the implementation.

This type of resource sharing is very important for large systems. Even for a small 21-tap FIR filter, you can notice that the required area (for random coefficients) can be brought down by nearly a factor of 2.5 or so. The penalty of course is that we need 11 clock cycles per output, meaning that with a 10ns clock, we will need 110ns per output. The direct FIR filter can generate one output every 10ns. When we are area constrained, this type of resource sharing is necessary.

Note how the Verilog code uses counters to decide the phase of the clock and multiplexers to give inputs to the resources at correct times. This is a general approach for resource sharing.



filter_direct_21_4

This is very similar to the previous code, except that it uses 4 multipliers and hence only 6 clock cycles are required for producing one output.