1D- TIME-DOMAIN CONVOLUTION

Anusha Balaji *UFID-10514929*
Lalitha Geddapu *UFID-69211894*
Lalitha Mathi *UFID-29341966*
Steven Simon *UFID-10485449*

Convolution is the function of how the input signal (shown as the x array) is modified with respect to the kernel (shown as the h array) and is given by output signal (shown as y array), the elements of which are given by the sum of the products formed by multiplying all the elements of the kernel with appropriate elements of the input signal. The pseudo code for the convolution is as follows

```
for (i=0; i < outputSize; i++) {
    y[i] = 0;
for (j=0; j < kernelSize; j++) {
    y[i] += x[i - j] * h[j];
    }
    }
```

This pseudocode is not complete because the x array will be accessed outside of its bounds both at the beginning of execution (i-j is negative) and towards the end of execution, (i-j is larger than the x array). The output size is sum of the input size and the kernel minus 1 In this project we have implemented 16-bit unsigned integer operations, which need to be "clipped" to the maximum possible 16-bit value in case of overflow. The FPGA implementation will store the input signal in SRAM and will read in a kernel (limited to 96 elements due to resource limitations of FPGA which is the number of DSP units on the device) through the memory map. The FPGA will then execute using a go and size input from the memory map, while writing all results to another SRAM. The datapath will fully unroll the inner loop, and then we will be able to pipeline the outer loop.

The main challenge of interfacing with the SRAMs is dealing with both control and data signals that cross clock domains. SRAM RD and SRAM WR domains run on 200 MHz clocks but USER_APP domains run on clkA or 100MHz.

IMPLEMENTATION:

USER_APP:

It consists of three modules Signal buffer, Kernel buffer and Datapath

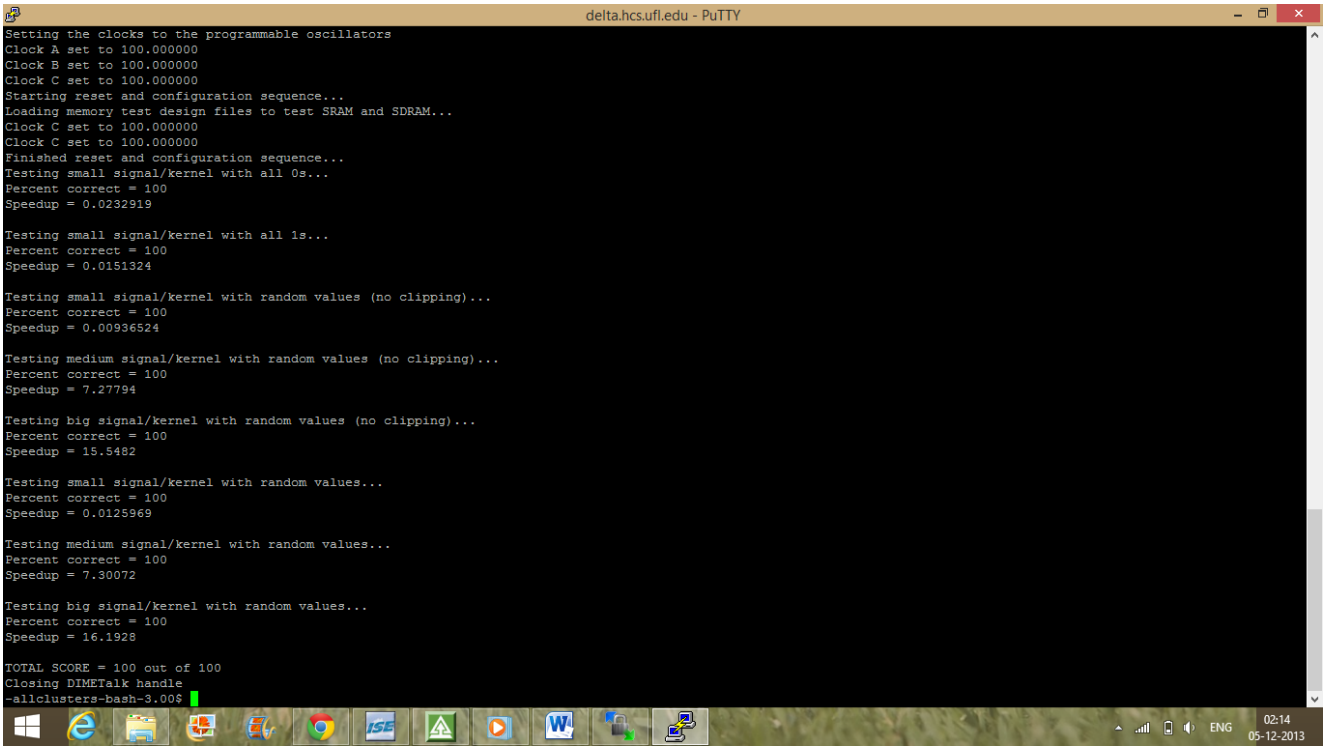
1. Signal Buffer: This is implemented using a FIFO-like interface that specifies if the buffer is empty/full. The data input to the FIFO should be 16 bits, and the output is an array of 96 16-bit elements. Output from FIFO is a 96-element window delivered to datapath stored in signal buffer. It is shifted left by one register each cycle so that it generates a new window every cycle.

2. Kernel Buffer: It is an array that consists of 96 16-bit elements, which shifts in 16 bits every time that the memory map writes data to an address that corresponds to the kernel provided by the C-Code. After 96 memory map transfers, the entire kernel will be loaded.

3. Datapath: Convolution operation is performed in this part using a 16 stage of pipeline and 96 DSP multipliers. The first row of the datapath consists of 96 16-bit multipliers, each of which will multiply corresponding elements from the signal and kernel. We used two registers after each multiplier and adder to store the result in one and perform clipping in one.

4. Controller: Controller module will interface the memory maps with the SRAMs. The controller gives the control signals go, size, start address to SRAM read and SRAM write.

USER_APP Hardware Result:



SRAM RD:

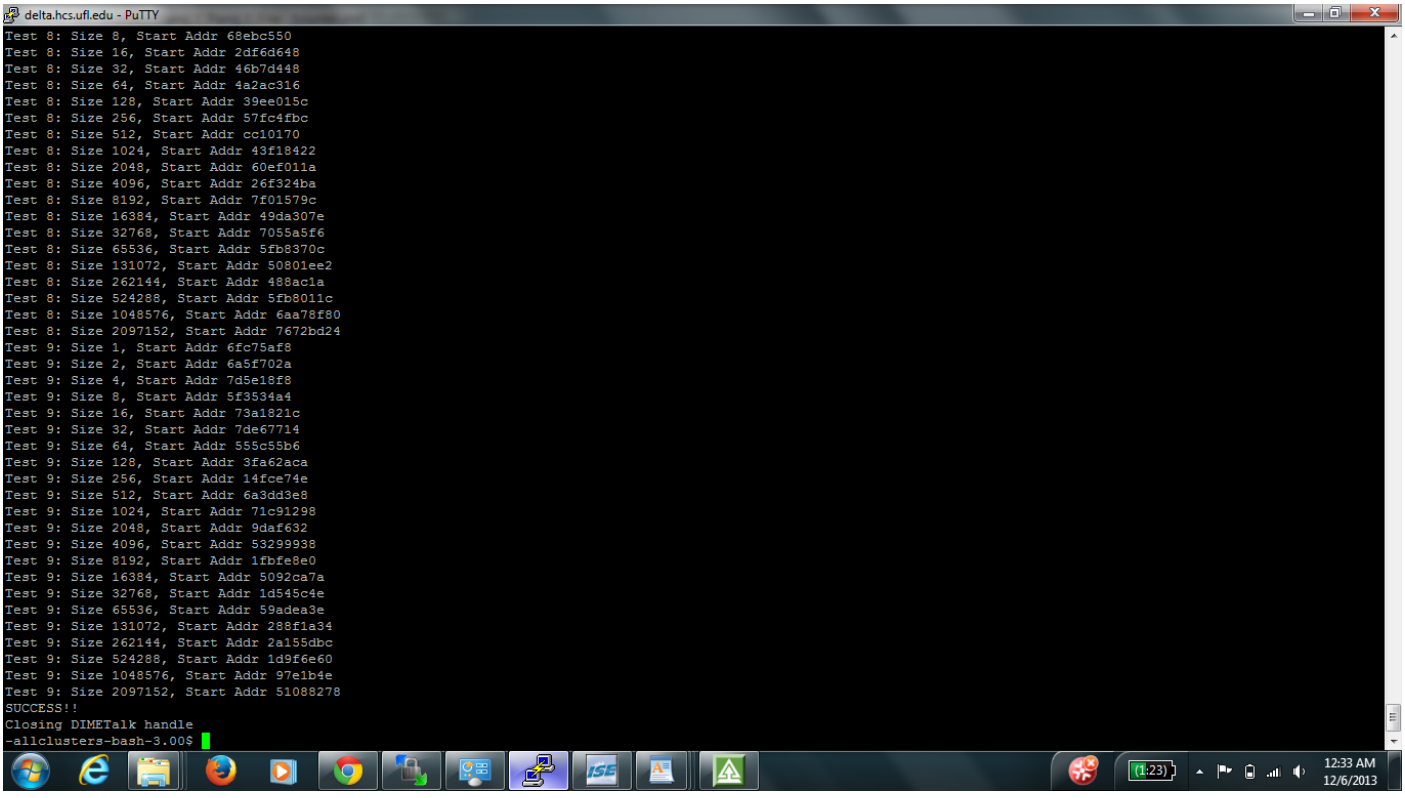
It consists of the following modules. SRAM_RD, Address Generator, Count, FIFO 64

1. SRAM_RD: SRAM reads the data from SRAM element of the board and delivers it to the FIFO. The input FIFO is of 64 bits which is read as 16 bits out of the FIFO.
2. Address Generator: This module generates the addresses that SRAM data is stored in. SRAM addresses need to be incremented by 2 for every other data. Address generator runs on the same clock as the SRAM (200 MHz). The address generator stops generating addresses as soon as the counter done is set i.e., when size number of elements are read from the FIFO. It gets the start address from the counter when the handshake sends the receive signal.

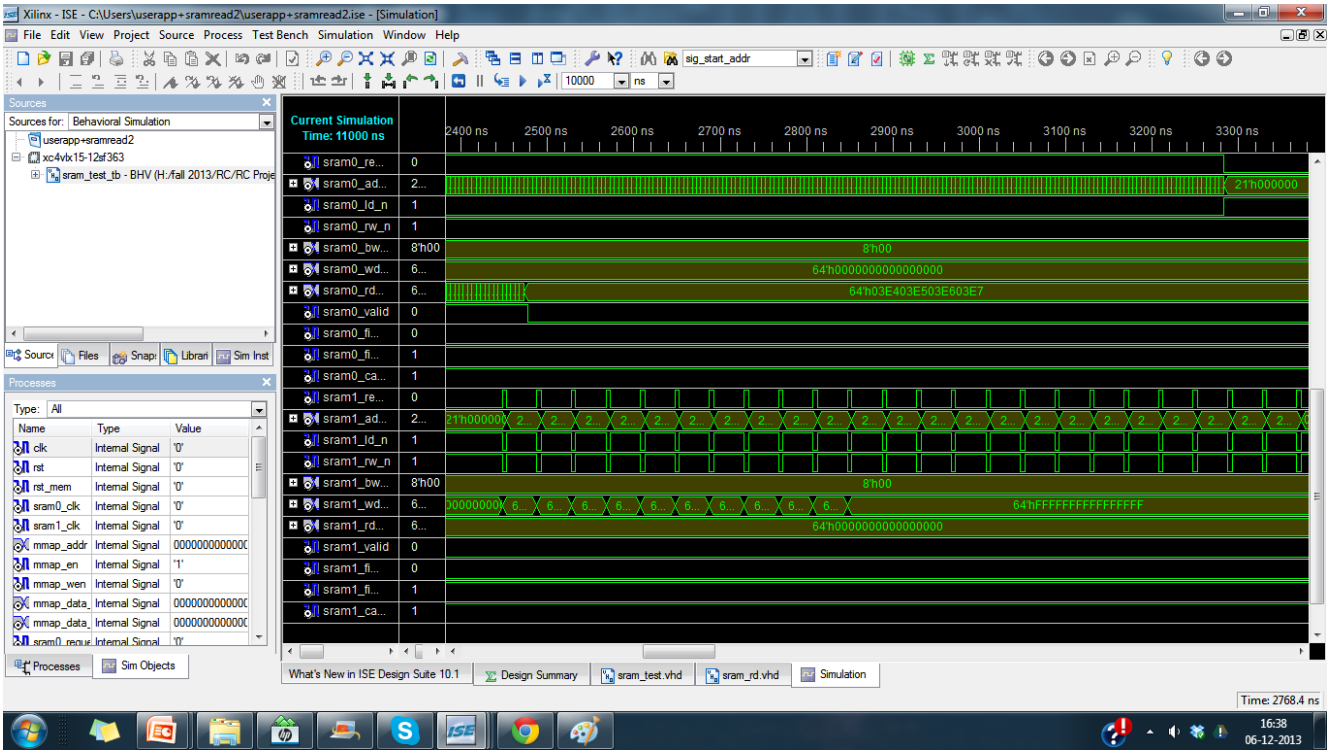
3. Count: Count Module receives a go signal from the controller and generates a signal send to start the handshake and also counts the number of 16 bit width words read from the FIFO comparing it to the size. When it is equal to size, a done signal is sent to user_app controller module which stops reading input from the FIFO.

4. FIFO_PROG_FULL: This FIFO has a 64 bit input and gives out a 16 bit output. In addition to having full and empty flags, it also has a programmable full flag which is set when the fifo is half full. This flag is used by the address generator.

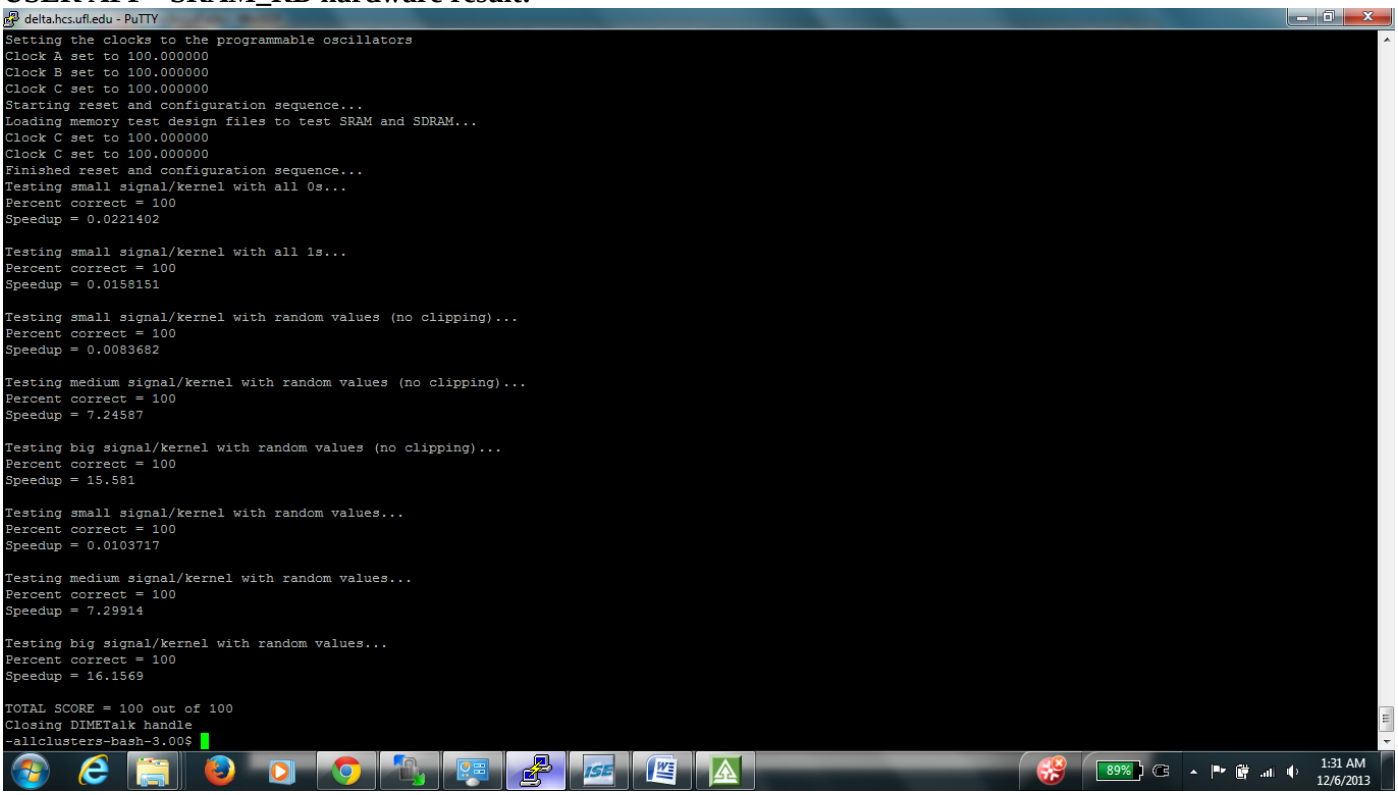
SRAM_RD hardware result:



USER_APP+ SRAM_RD SIMULATION RESULT:



USER APP+ SRAM_RD hardware result:



SRAM WR:

It consists of the following modules. SRAM_WR, Address Generator, Count, FIFO 64, COUNT_Z

1. Address Generator: Address generator runs on the same clock as the SRAM (200 MHz) and starts once it receives valid signal from USER_APP controller with a handshake. This module generates the addresses where SRAM data is stored. SRAM addresses need to be incremented by 2 for every other data.

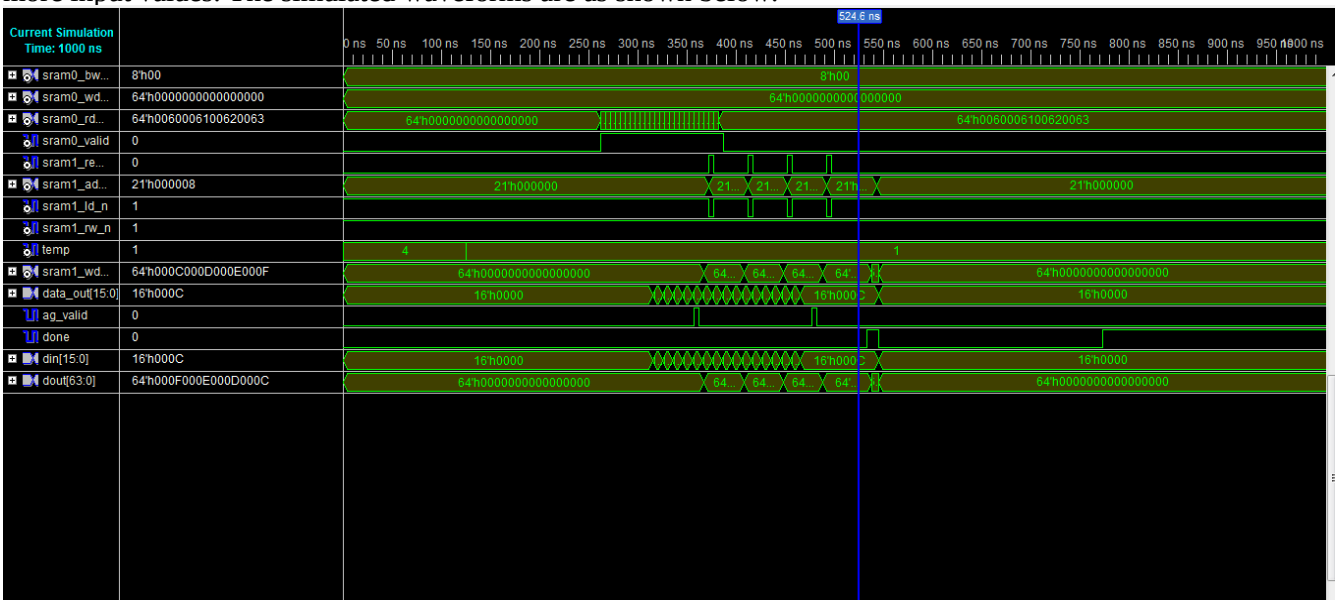
2. Count: Count Module receives a go signal from controller and generates a signal send to start the handshake and also counts the number of 16 bit width words being written into the FIFO and comparing it to the size. When it is equal to size, a done signal is generated.

3. FIFO_64: A 16 bit input 64 output FIFO has been implemented which takes the data from datapath as input. It gives the output to the SRAM_WR interface to be written onto the memory.

4. COUNT_Z : This counter is initiated by the dine signal of the Count module. It checks if size is a multiple of four using the mod function and sets the select line to the mux correspondingly to write those many junk values.

5. Mux: A mux is used to select the data coming from the datapath or the junk value value we intend to write in case the size is not a multiple of four.

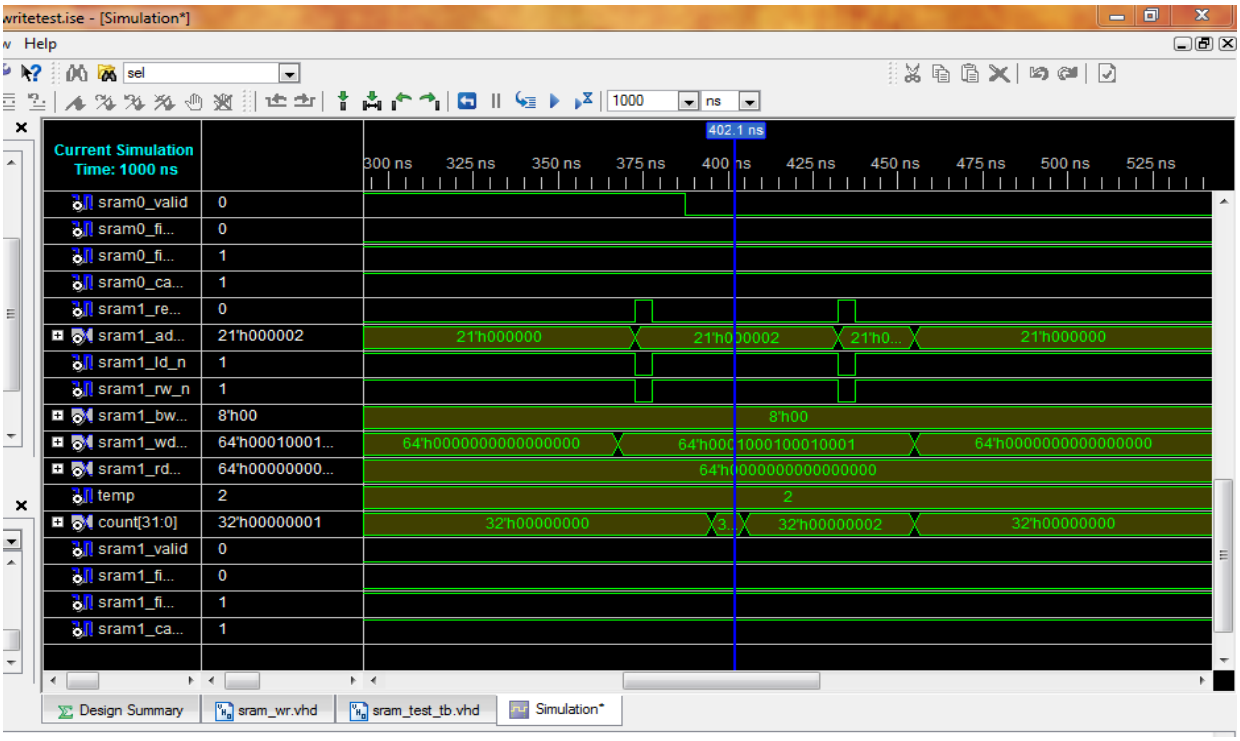
SRAM_WR module works in the simulation. We implemented the same counter as in SRAM_RD. One more counter that checks if the size is a multiple of 4 is implemented. The address generator has been slightly modified to suit the SRAM write functionality. The test bench has been modified to check for more input values. The simulated waveforms are as shown below.



However, the hardware output showed signs of metastability initially. We checked our sram_wr entity for signals that are not synchronized. The done logic for the total circuit included a signal sram_fifo_empty

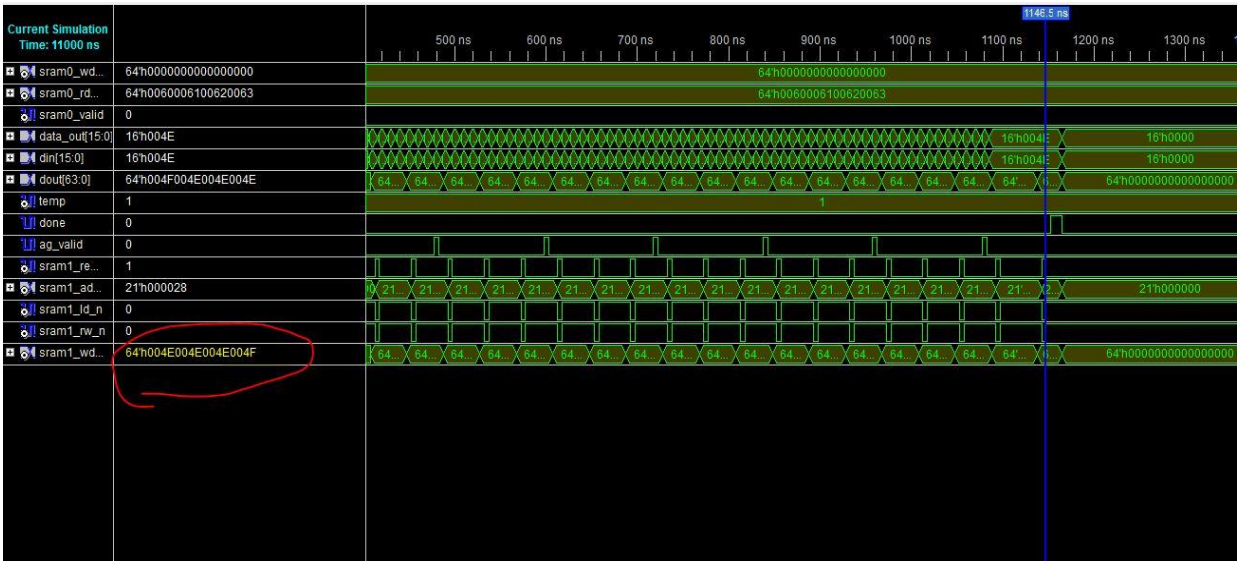
that is coming from sram_clk region. We synchronized the signal using a dual flip flop. The metastability issue was eliminated in the hardware.

We checked if our second counter i.e., the one generating the done after checking size is a multiple of four is working. We checked if the modulus operation is giving the correct value to the temp variable and it was fine. The counting operation of the entity was also in sync with the value of the temp. It counted for the right number of times as shown in the below simulation waveform.



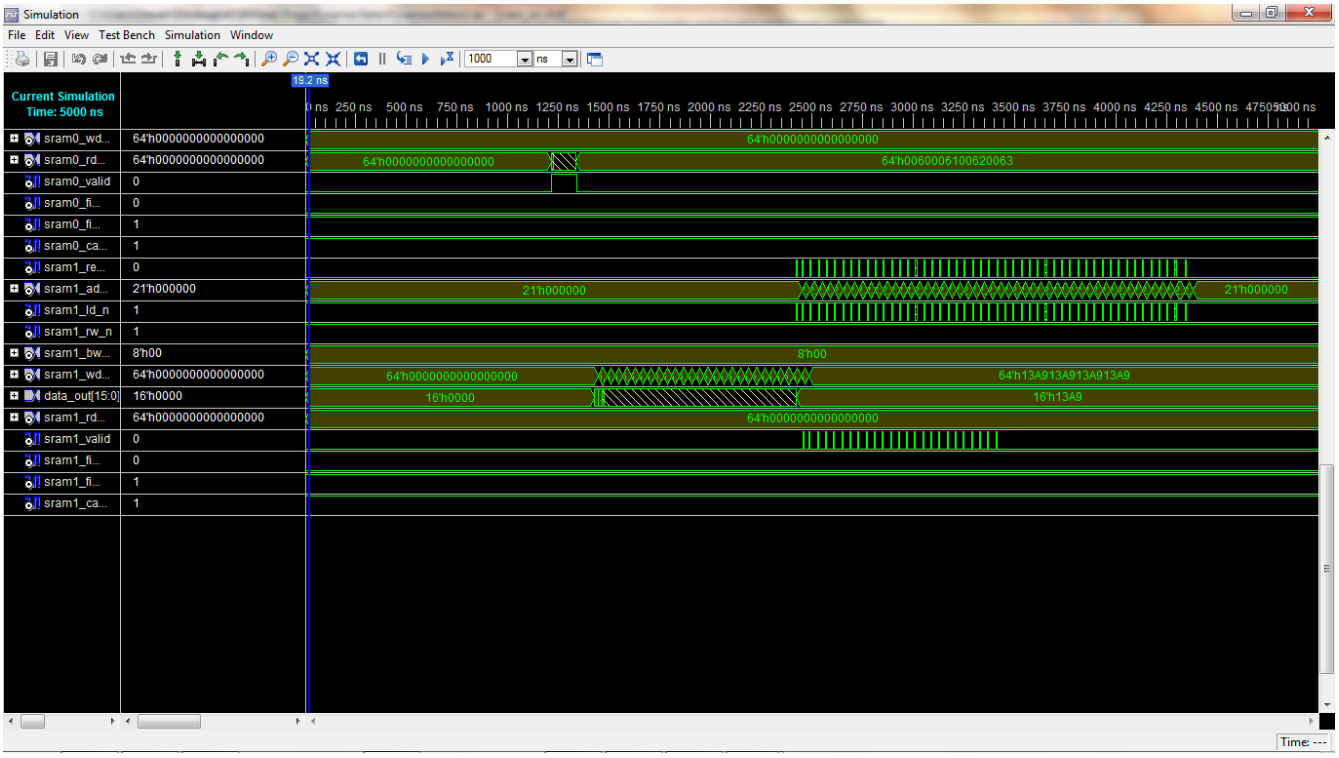
We then checked the address generator's temp value. It is getting incremented by two until it is reset by the done of the whole circuit**. Done is asserted** as soon as count is greater than 2 and the select of the mux is high. The select signal just writes the last data input as the junk.

We checked the data going into the fifo, the fifo data out and the sram_wdata signal. We delayed the done signal by some cycles so that sufficient number of addresses would be produced to check the data coming out. Our multiplexer has both inputs as data_in(data coming from datapath). So, the junk value being written into the FIFO would be the last data_in. That has been circled in red in the below simulation screenshot. The size we checked for is 77. So the **junk value is being written properly** into the FIFO through the MUX logic. It can be seen in the screenshot.



We cross checked the address generator's functionality with the provided sram_wr functionality and it didn't deviate much. The **sram_request gets set at the right place** i.e., at the start of each address generated. The first counter is the same entity we used for sram_rd.

FULL SIMULATION (SRAM_RD, SRAM_WR, USER_APP)



PROBLEMS ENCOUNTERED AND SOLUTIONS:

While working with the USER_APP module, we struggled to get the logic right for the counter inside the signal buffer. We zeroed in on the problem after many simulation waveform checks and the counter was modified to implement on clock reset mode. The datapath also took a lot of simulations and bit file generations till we took care of the timing constraints by using two registers after each multiplier and adder to store and clip the output. The email that suggested the above modification was helpful.

In SRAM_RD module, we couldn't get the counter logic right till the end. We tried many ways of starting the handshake and sending the start address to the address generator. We tried giving 1) go to the handshake and sending the address to the address generator directly from counter, then 2) go signal to the handshake from the counter and start address directly from the controller, both of which didn't work on the board despite of working in the simulation. The problem was with the assertion of done. The test 0 hanged every time we tried to run the bit files. Finally, when we implemented the counter in such a way that it is initiated on go signal from the controller and sends a start signal to the handshake and start address to the address generator, it worked on the board.

FINAL RESULTS:

1. User app works both on the board and simulation. The bit file has been attached.

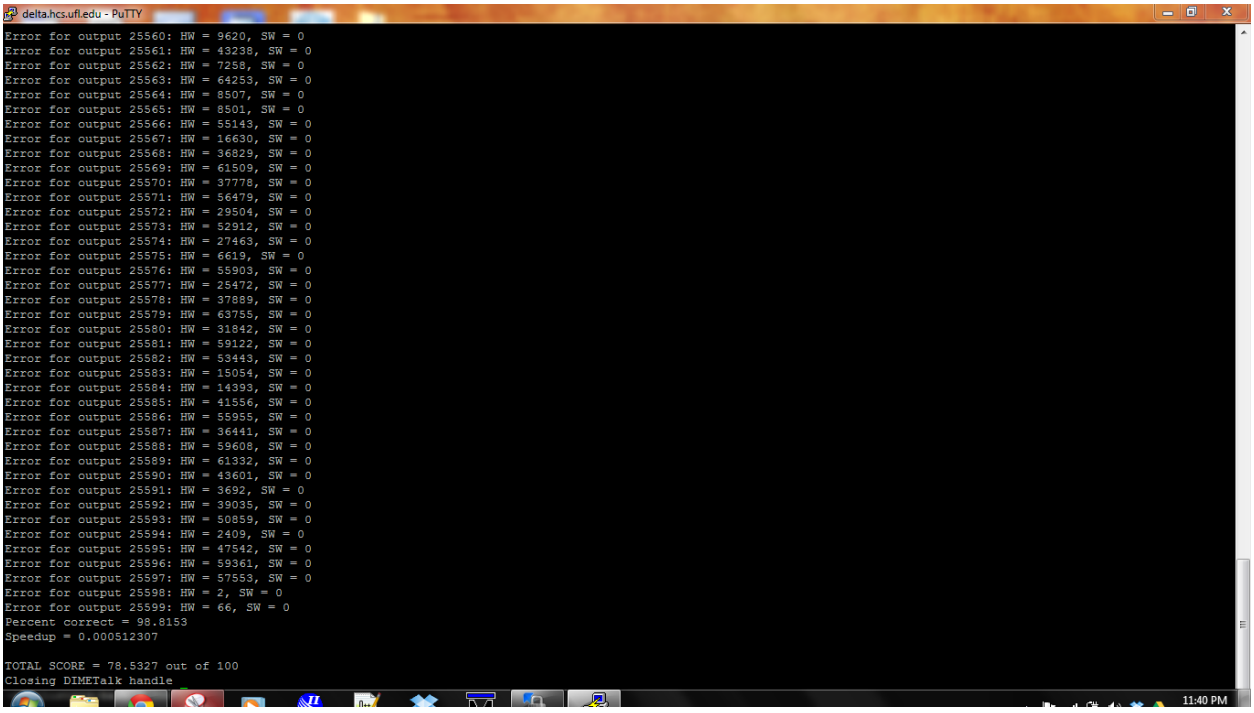
2. SRAM_RD entity works both on the board and simulation. The bit file has been attached.

3. Both SRAM_RD, USER_APP have been tested with the given sram_wr.ngc file. It works on the board and simulation. The bit file has been attached.

4. SRAM_WR works in simulation.

5. No modifications to C code were done.

6. The whole project works in the simulation. Partially on board.


delta.hcs.ufl.edu - PuTTY
Error for output 25560: HW = 9620, SW = 0
Error for output 25561: HW = 43238, SW = 0
Error for output 25562: HW = 7258, SW = 0
Error for output 25563: HW = 64253, SW = 0
Error for output 25564: HW = 8507, SW = 0
Error for output 25565: HW = 8501, SW = 0
Error for output 25566: HW = 55143, SW = 0
Error for output 25567: HW = 16630, SW = 0
Error for output 25568: HW = 36829, SW = 0
Error for output 25569: HW = 61509, SW = 0
Error for output 25570: HW = 37778, SW = 0
Error for output 25571: HW = 56479, SW = 0
Error for output 25572: HW = 29504, SW = 0
Error for output 25573: HW = 52912, SW = 0
Error for output 25574: HW = 27463, SW = 0
Error for output 25575: HW = 6619, SW = 0
Error for output 25576: HW = 55903, SW = 0
Error for output 25577: HW = 25472, SW = 0
Error for output 25578: HW = 37889, SW = 0
Error for output 25579: HW = 63755, SW = 0
Error for output 25580: HW = 31842, SW = 0
Error for output 25581: HW = 59122, SW = 0
Error for output 25582: HW = 53443, SW = 0
Error for output 25583: HW = 15054, SW = 0
Error for output 25584: HW = 14393, SW = 0
Error for output 25585: HW = 41556, SW = 0
Error for output 25586: HW = 55955, SW = 0
Error for output 25587: HW = 36441, SW = 0
Error for output 25588: HW = 59608, SW = 0
Error for output 25589: HW = 61332, SW = 0
Error for output 25590: HW = 43601, SW = 0
Error for output 25591: HW = 3692, SW = 0
Error for output 25592: HW = 39035, SW = 0
Error for output 25593: HW = 50859, SW = 0
Error for output 25594: HW = 2409, SW = 0
Error for output 25595: HW = 47542, SW = 0
Error for output 25596: HW = 59361, SW = 0
Error for output 25597: HW = 57553, SW = 0
Error for output 25598: HW = 2, SW = 0
Error for output 25599: HW = 66, SW = 0
Percent correct = 98.8153
Speedup = 0.000512307
TOTAL SCORE = 78.5327 out of 100
Closing DIMETalk handle
11:40 PM