Final Project Report

Reconfigurable Computing

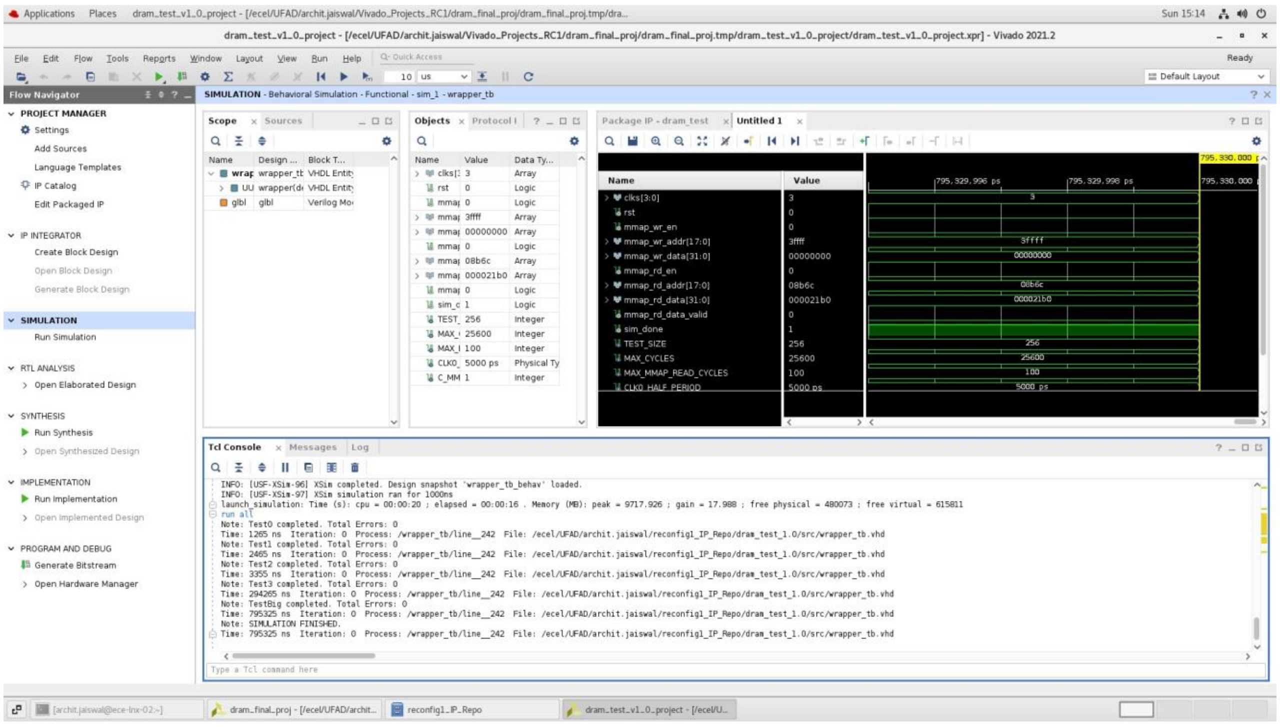
EEL 5721

Group members:

Archit Jaiswal

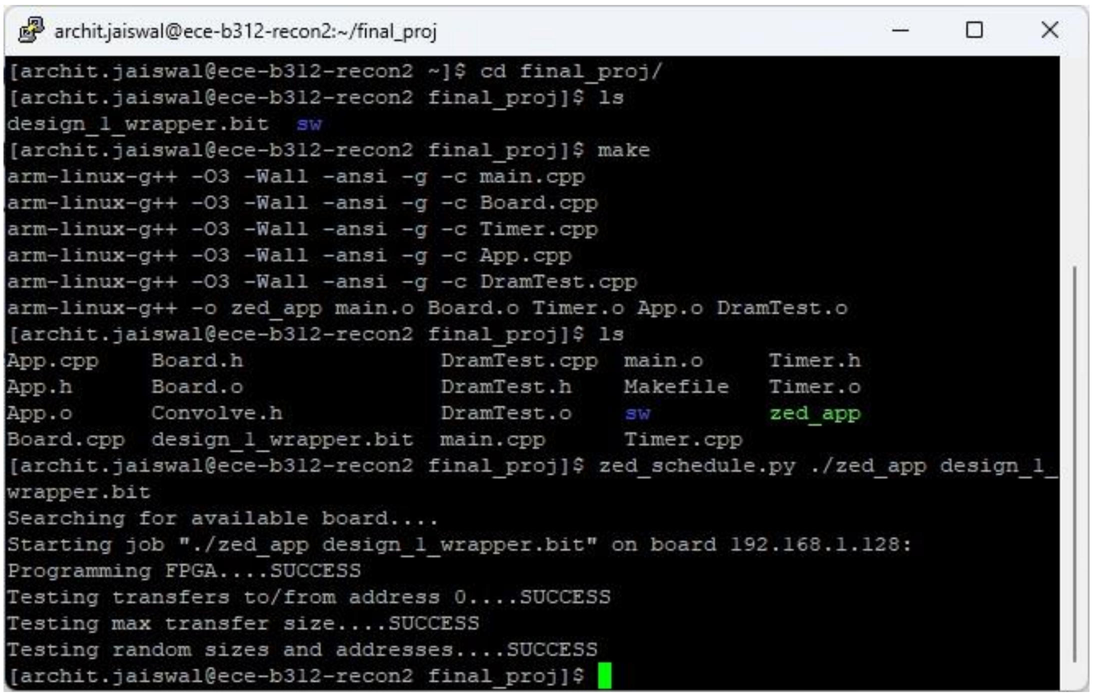
Junfu Cheng

1. We successfully compiled the dram_test project to ensure the Vivado setup works correctly. The screenshot below shows that we successfully simulated and generated the dram_test_v1_0.
   Note: To make it work correctly, we had to upgrade all the IPs used in this project and always add the dma_fifo every time the project is closed and reopened.



```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'wrapper_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:20 ; elapsed = 00:00:16 . Memory (MB): peak = 9717.926 ; gain = 17.988 ; free physical = 480073 ; free virtual = 615811
run all
Note: Test0 completed. Total Errors: 0
Time: 1265 ns  Iteration: 0  Process: /wrapper_tb/line__242  File: /ecel/UFAD/archit.jaiswal/reconfig1_IP_Repo/dram_test_1.0/src/wrapper_tb.vhd
Note: Test1 completed. Total Errors: 0
Time: 2465 ns  Iteration: 0  Process: /wrapper_tb/line__242  File: /ecel/UFAD/archit.jaiswal/reconfig1_IP_Repo/dram_test_1.0/src/wrapper_tb.vhd
Note: Test2 completed. Total Errors: 0
Time: 3355 ns  Iteration: 0  Process: /wrapper_tb/line__242  File: /ecel/UFAD/archit.jaiswal/reconfig1_IP_Repo/dram_test_1.0/src/wrapper_tb.vhd
Note: Test3 completed. Total Errors: 0
Time: 294265 ns  Iteration: 0  Process: /wrapper_tb/line__242  File: /ecel/UFAD/archit.jaiswal/reconfig1_IP_Repo/dram_test_1.0/src/wrapper_tb.vhd
Note: TestBig completed. Total Errors: 0
Time: 795325 ns  Iteration: 0  Process: /wrapper_tb/line__242  File: /ecel/UFAD/archit.jaiswal/reconfig1_IP_Repo/dram_test_1.0/src/wrapper_tb.vhd
Note: SIMULATION FINISHED.
Time: 795325 ns  Iteration: 0  Process: /wrapper_tb/line__242  File: /ecel/UFAD/archit.jaiswal/reconfig1_IP_Repo/dram_test_1.0/src/wrapper_tb.vhd
```

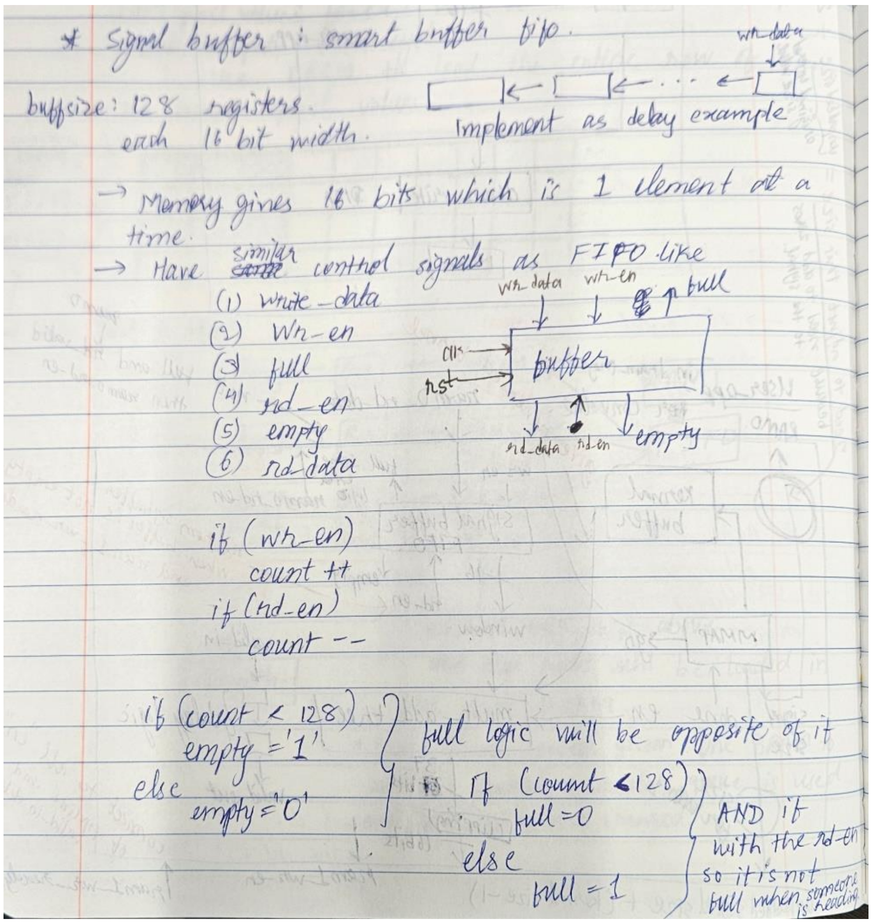
Figure 1: Vivado Simulation of dram_test IP



```
archit.jaiswal@ece-b312-recon2:~/final_proj
[archit.jaiswal@ece-b312-recon2 ~]$ cd final_proj/
[archit.jaiswal@ece-b312-recon2 final_proj]$ ls
design_1_wrapper.bit  sw
[archit.jaiswal@ece-b312-recon2 final_proj]$ make
arm-linux-g++ -O3 -Wall -ansi -g -c main.cpp
arm-linux-g++ -O3 -Wall -ansi -g -c Board.cpp
arm-linux-g++ -O3 -Wall -ansi -g -c Timer.cpp
arm-linux-g++ -O3 -Wall -ansi -g -c App.cpp
arm-linux-g++ -O3 -Wall -ansi -g -c DramTest.cpp
arm-linux-g++ -o zed_app main.o Board.o Timer.o App.o DramTest.o
[archit.jaiswal@ece-b312-recon2 final_proj]$ ls
App.cpp    Board.h              DramTest.cpp  main.o    Timer.h
App.h      Board.o              DramTest.h    Makefile  Timer.o
App.o      Convolve.h           DramTest.o    sw        zed_app
Board.cpp  design_1_wrapper.bit  main.cpp     Timer.cpp
[archit.jaiswal@ece-b312-recon2 final_proj]$ zed_schedule.py ./zed_app design_1_
wrapper.bit
Searching for available board....
Starting job "./zed_app design_1_wrapper.bit" on board 192.168.1.128:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[archit.jaiswal@ece-b312-recon2 final_proj]$
```

Figure 2: Running the dram_test on ZedBoard

2. <u>Successfully implemented Signal Buffer:</u>
   This implements a smart buffer, which loads a 16-bit element from the FIFO in every clock cycle and slides the input to an adjacent buffer in every clock cycle. The figure below shows the high-level structure of it. When all the elements are loaded, it will raise a "full" flag to notify the FIFO to stop sending data to it. It will also deassert the "empty" flag to notify the pipeline that it has a complete data window for further processing.



* Signal buffer : smart buffer fifo.

buffsize: 128 registers
each 16 bit width.

Implement as delay example

→ Memory gives 16 bits which is 1 element at a time.

→ Have ~~same~~ similar control signals as FIFO like
(1) write_data
(2) wr-en
(3) full
(4) rd_en
(5) empty
(6) rd data

wr data
wr data
wr-en
full
clk
rst
buffer
rd_data
rd-en
empty

```
if (wr-en)
    count ++
if (rd-en)
    count --

if (count < 128)
    empty = '1'
else
    empty = '0'
```

full logic will be opposite of it

```
If (count < 128)
    full = 0
else
    full = 1
```

AND it with the rd-en so it is not full when someone is reading

Figure 3: Smart buffer Structure

We performed rigorous testing to ensure its functionality by testing it on small input sizes and changing the read/write patterns. The waveform below shows that when alternate read and write signals are asserted, the buffer will not show a full flag or remove an empty flag until all the elements in the buffer are loaded. The waveform below shows the behavior of smart buffer for size = 5 for demonstration purposes. The following waveform shows that when a read occurs right after the buffer is full, the buffer will remove the full flag and assert an empty flag.

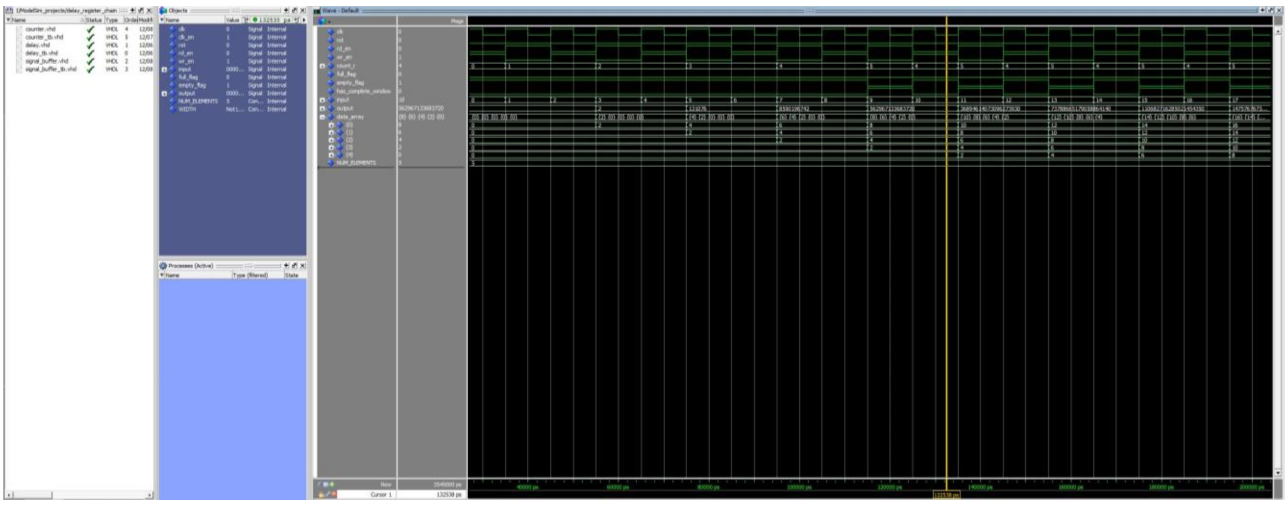


Figure 4: Alternate reads and writes for buffer size = 5 (writing only even numbers)

The screenshot below shows the behavior when the first write signal is asserted and remains asserted despite the full being buffer. In this case, the buffer successfully ignores the incoming data and keeps the output constant. Later, only the read is asserted, and the buffer will continue to show the same output and set the empty flag without decrementing the counter.

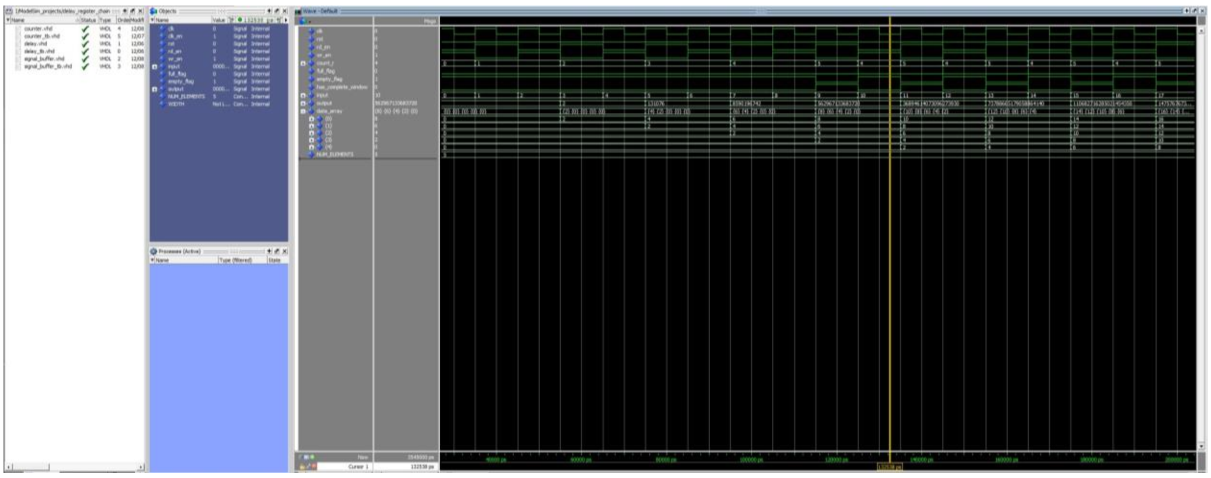


Figure 5: Discrete reads and writes

The following waveform demonstrates a case where the buffer handles simultaneous reads and writes. When the data is being read continuously while the new data is being written, the signal buffer will not raise the full flag and does not raise the empty flag. This gives the maximum throughput to the pipeline by allowing simultaneous reads and writes.

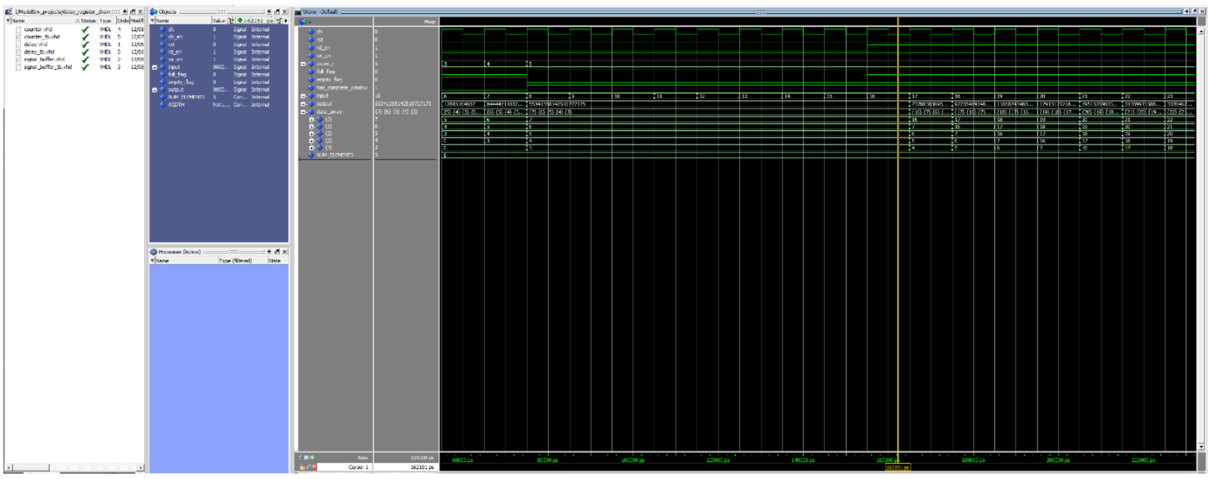


Figure 6: Simultaneous reads and writes

3. <u>Successfully created Kernel Buffer</u>

Next we implemented a kernel buffer, which instantiates the signal buffer and reverses its output before feeding it to the mult_add_pipeline. The waveform below shows the data_array of the kernel buffer where element “7” is present as the 5th element and “3” is present as the 1st element in a 5-element buffer.

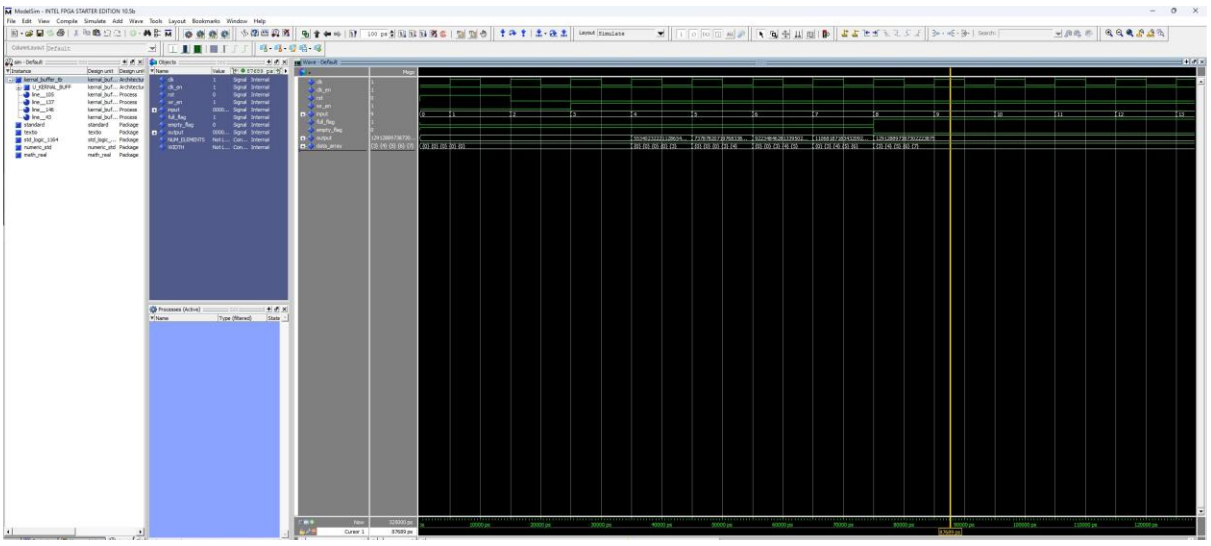


Figure 7: kernel buffer giving reversed output as intended

4. <u>Successfully integrated the buffer into the multiply-add pipeline</u>

We integrated the buffers into the mult_add_tree and implemented the valid_in/valid_out control mechanism by computing pipeline latency based on the number of input elements.

The following waveform shows that when the data is ready for output, valid_out flag is asserted to notify the RAM1 that valid data is available at the read port.

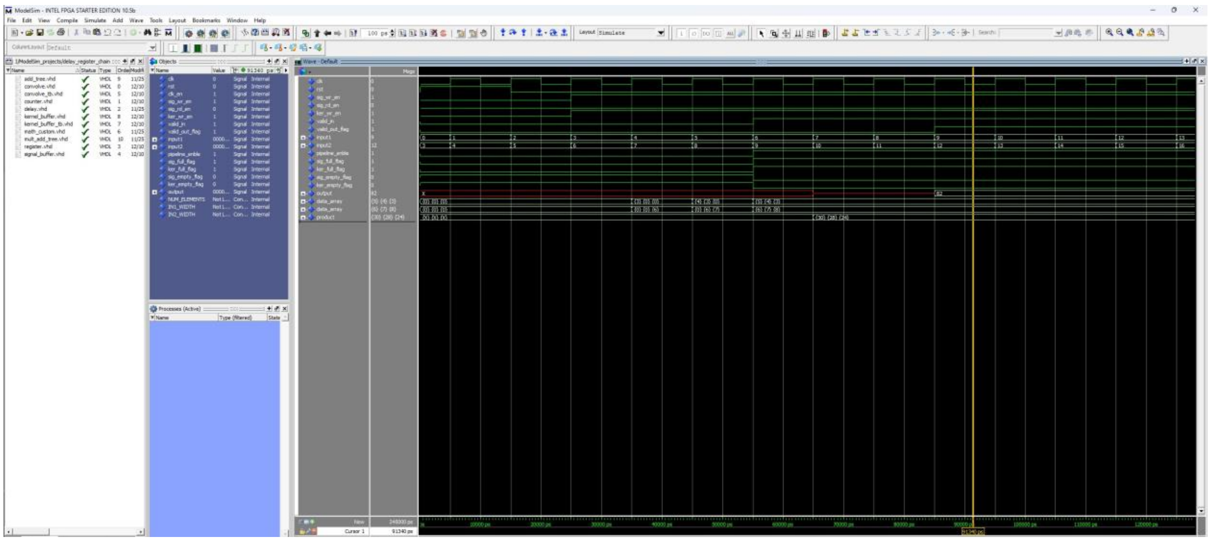
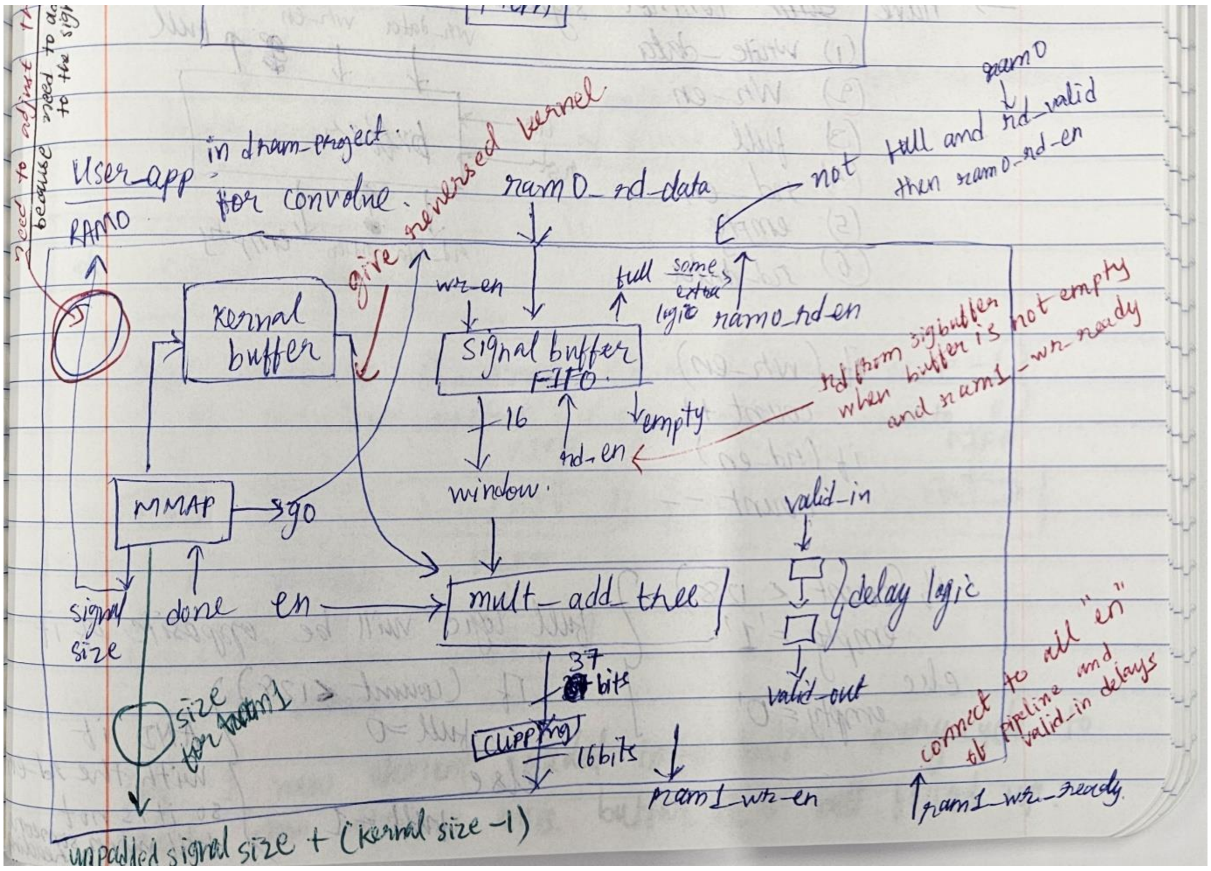


Figure 8: Mult_add_pipeline out for <u>3 element</u> signal and kernel buffer

5. <u>Successfully combined all the logic and pipeline in the user app of convolve IP</u>

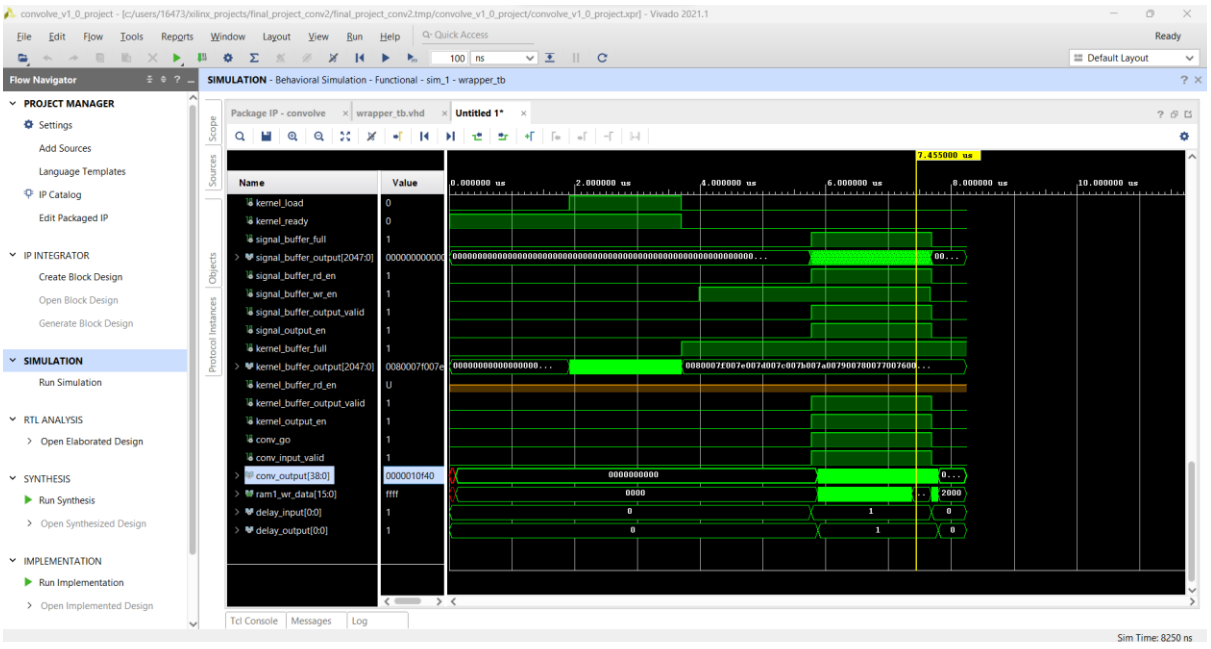
All the above-mentioned entities were integrated to develop the user app of the convolve IP. The high-level glue logic is shown in the following figure. By implementing this logic, we got a score of 99.997 after running the bit file on FPGA.



need to adjust the because need to add to the signal
User_app
in dram-project for convolve.
RAM0
give reversed kernel
ram0_rd_data
not full and ram0 rd_valid then ram0_rd_en
Kernal buffer
wr_en
Signal buffer FIFO
full
some extra logic
ram0_rd_en
rd from sigbuffer when buffer is not empty and ram1_wr_ready
16
rd_en
empty
window
MMAP
go
valid_in
signal size
done
en
mult_add_tree
delay logic
37 bits
valid_out
size for ram1
clipping
16bits
ram1_wr_en
connect to all "en" of pipeline and valid_in delays
ram1_wr_ready
unpadded signal size + (kernal size -1)

Figure 9: Convolve IP user_app structure

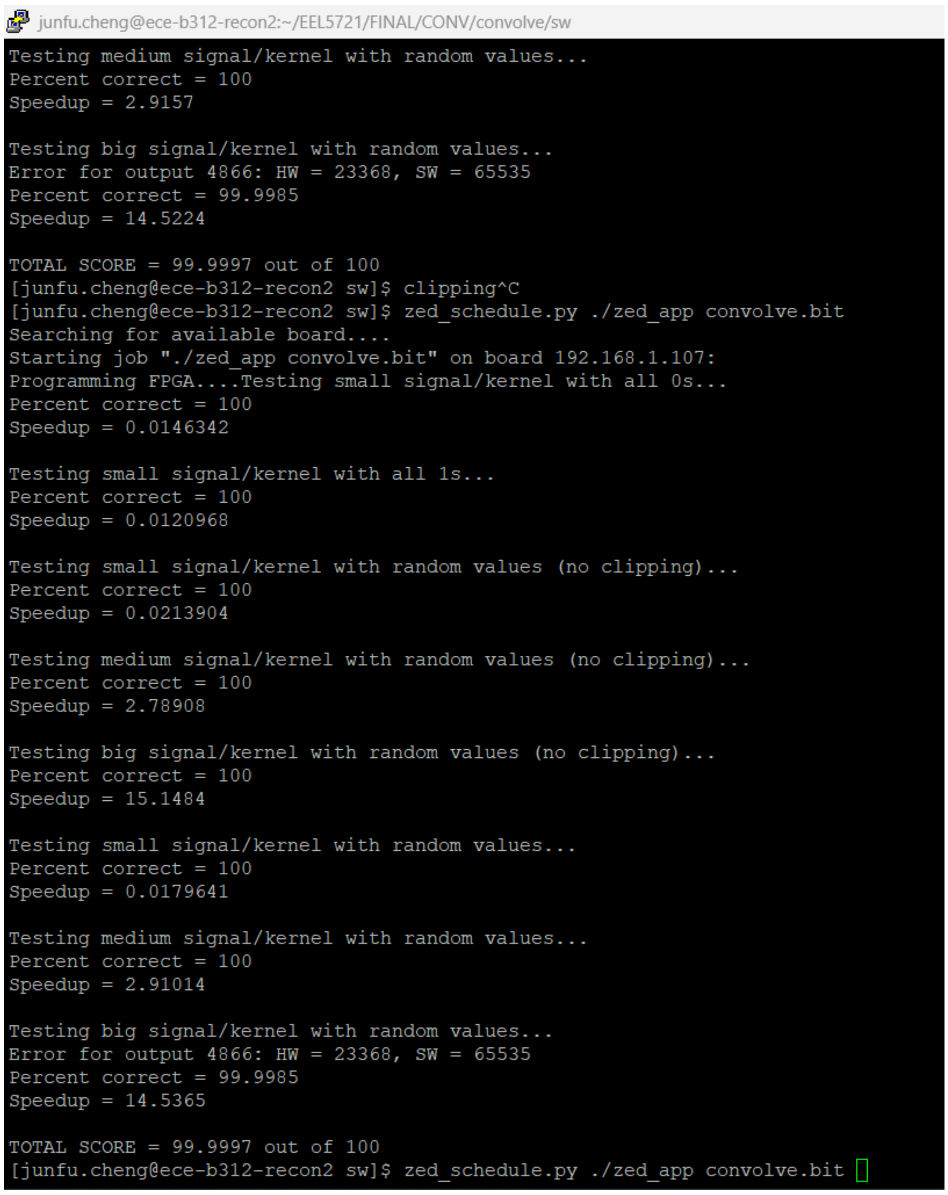
Figure below shows the simulation output of the convolve IP we created using the components and logic described above.



Name	Value
kernel_load	0
kernel_ready	0
signal_buffer_full	1
signal_buffer_output[2047:0]	000000000000
signal_buffer_rd_en	1
signal_buffer_wr_en	1
signal_buffer_output_valid	1
signal_output_en	1
kernel_buffer_full	1
kernel_buffer_output[2047:0]	0080007f007e
kernel_buffer_rd_en	U
kernel_buffer_output_valid	1
kernel_output_en	1
conv_go	1
conv_input_valid	1
conv_output[38:0]	0000010f40
ram1_wr_data[15:0]	ffff
delay_input[0:0]	1
delay_output[0:0]	1

Figure 10: Convolve IP simulation results

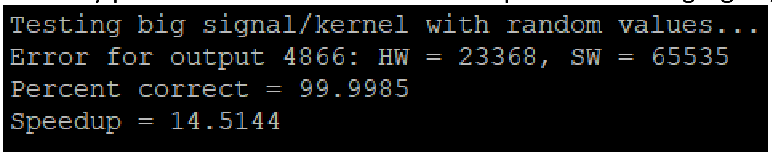
The following screenshot shows the results after running the generated bitstream of convolve IP.



```
junfu.cheng@ece-b312-recon2:~/EEL5721/FINAL/CONV/convolve/sw
Testing medium signal/kernel with random values...
Percent correct = 100
Speedup = 2.9157

Testing big signal/kernel with random values...
Error for output 4866: HW = 23368, SW = 65535
Percent correct = 99.9985
Speedup = 14.5224

TOTAL SCORE = 99.9997 out of 100
[junfu.cheng@ece-b312-recon2 sw]$ clipping^C
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app convolve.bit
Searching for available board....
Starting job "./zed_app convolve.bit" on board 192.168.1.107:
Programming FPGA....Testing small signal/kernel with all 0s...
Percent correct = 100
Speedup = 0.0146342

Testing small signal/kernel with all 1s...
Percent correct = 100
Speedup = 0.0120968

Testing small signal/kernel with random values (no clipping)...
Percent correct = 100
Speedup = 0.0213904

Testing medium signal/kernel with random values (no clipping)...
Percent correct = 100
Speedup = 2.78908

Testing big signal/kernel with random values (no clipping)...
Percent correct = 100
Speedup = 15.1484

Testing small signal/kernel with random values...
Percent correct = 100
Speedup = 0.0179641

Testing medium signal/kernel with random values...
Percent correct = 100
Speedup = 2.91014

Testing big signal/kernel with random values...
Error for output 4866: HW = 23368, SW = 65535
Percent correct = 99.9985
Speedup = 14.5365

TOTAL SCORE = 99.9997 out of 100
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app convolve.bit
```

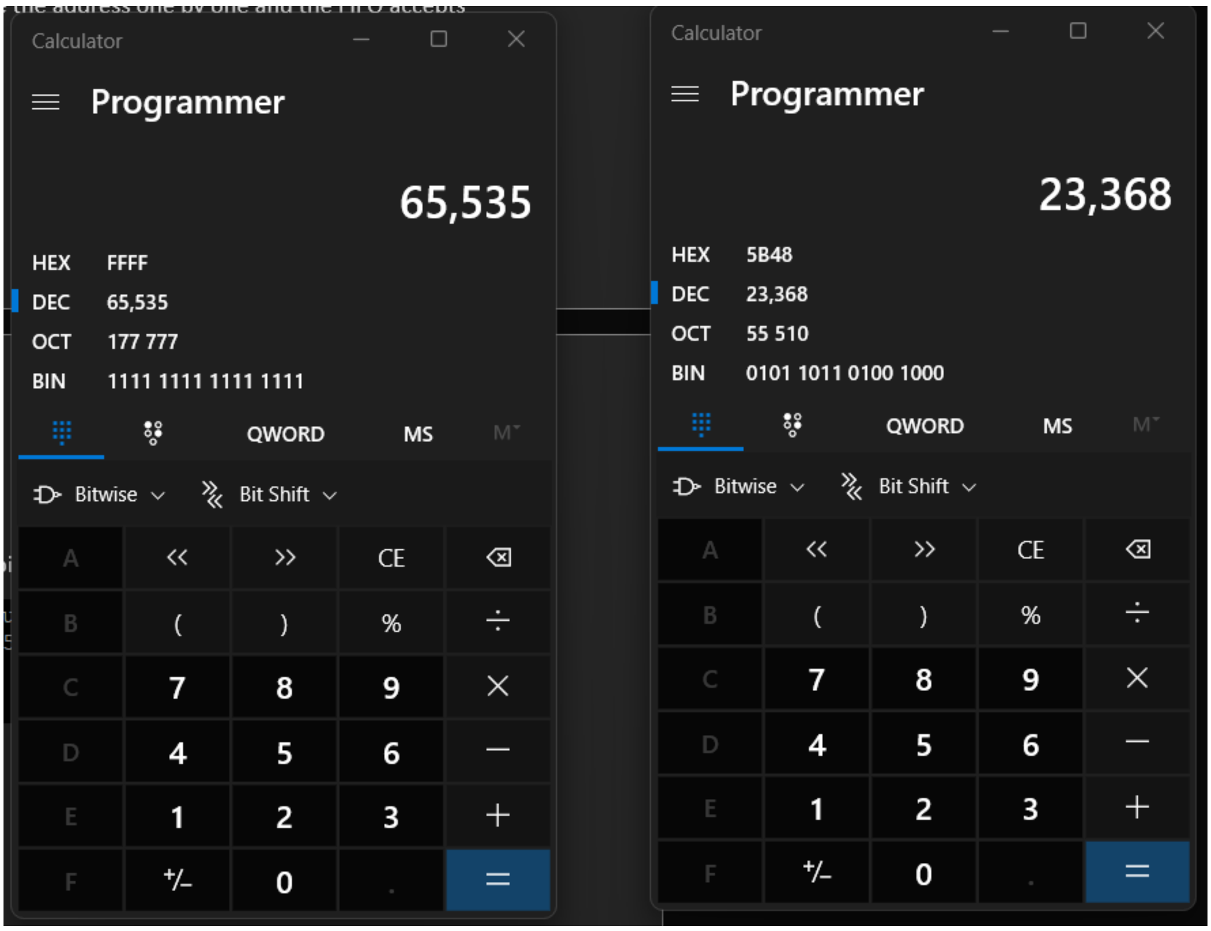
Figure 11: Score after running the convolve.bit on ZedBoard

The only problem we have in convolution part is in the big signal/kernel test.



```
Testing big signal/kernel with random values...
Error for output 4866: HW = 23368, SW = 65535
Percent correct = 99.9985
Speedup = 14.5144
```

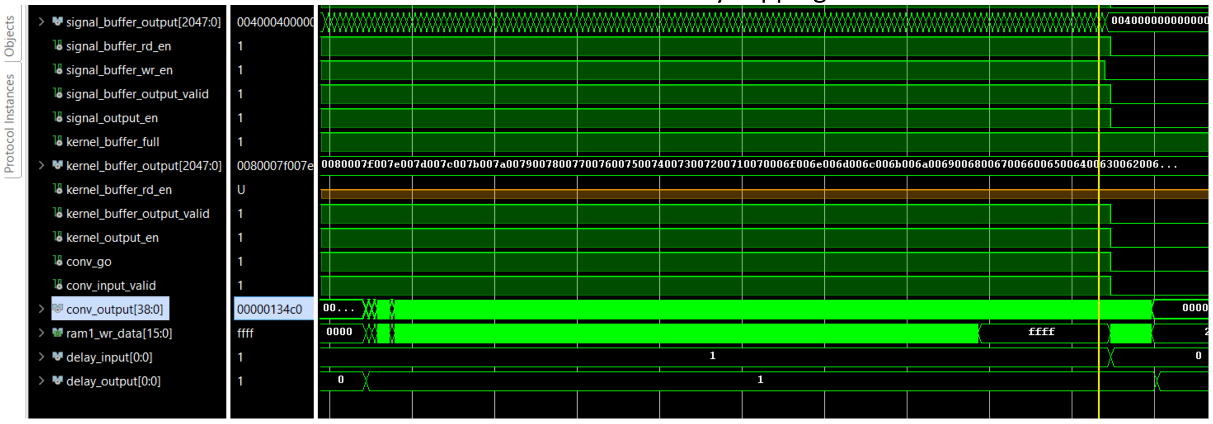
It looks like the clipping issue.



```
Calculator
Programmer
65,535
HEX FFFF
DEC 65,535
OCT 177 777
BIN 1111 1111 1111 1111
```

```
Calculator
Programmer
23,368
HEX 5B48
DEC 23,368
OCT 55 510
BIN 0101 1011 0100 1000
```

But our simulation result show that our IP does not have any clipping issue at all:



```
signal_buffer_output[2047:0]   004000400000   004000000000000
signal_buffer_rd_en            1
signal_buffer_wr_en            1
signal_buffer_output_valid     1
signal_output_en               1
kernel_buffer_full             1
kernel_buffer_output[2047:0]   0080007f007e   0080007f007e007d007c007b007a007900780077007600750074007300720071007000 6f006e006d006c006b006a00690068006700660065006400630062006...
kernel_buffer_rd_en            U
kernel_buffer_output_valid     1
kernel_output_en               1
conv_go                        1
conv_input_valid               1
conv_output[38:0]              00000134c0     00...   0000
ram1_wr_data[15:0]             ffff           0000   ffff   2
delay_input[0:0]               1              1   0
delay_output[0:0]              1              0   1
```

6. Successfully completed dram test IP for DMA interface

We have also successfully completed the DRAM IP and performed clock domain crossing of multiple bits by implementing a handshake synchronizer.

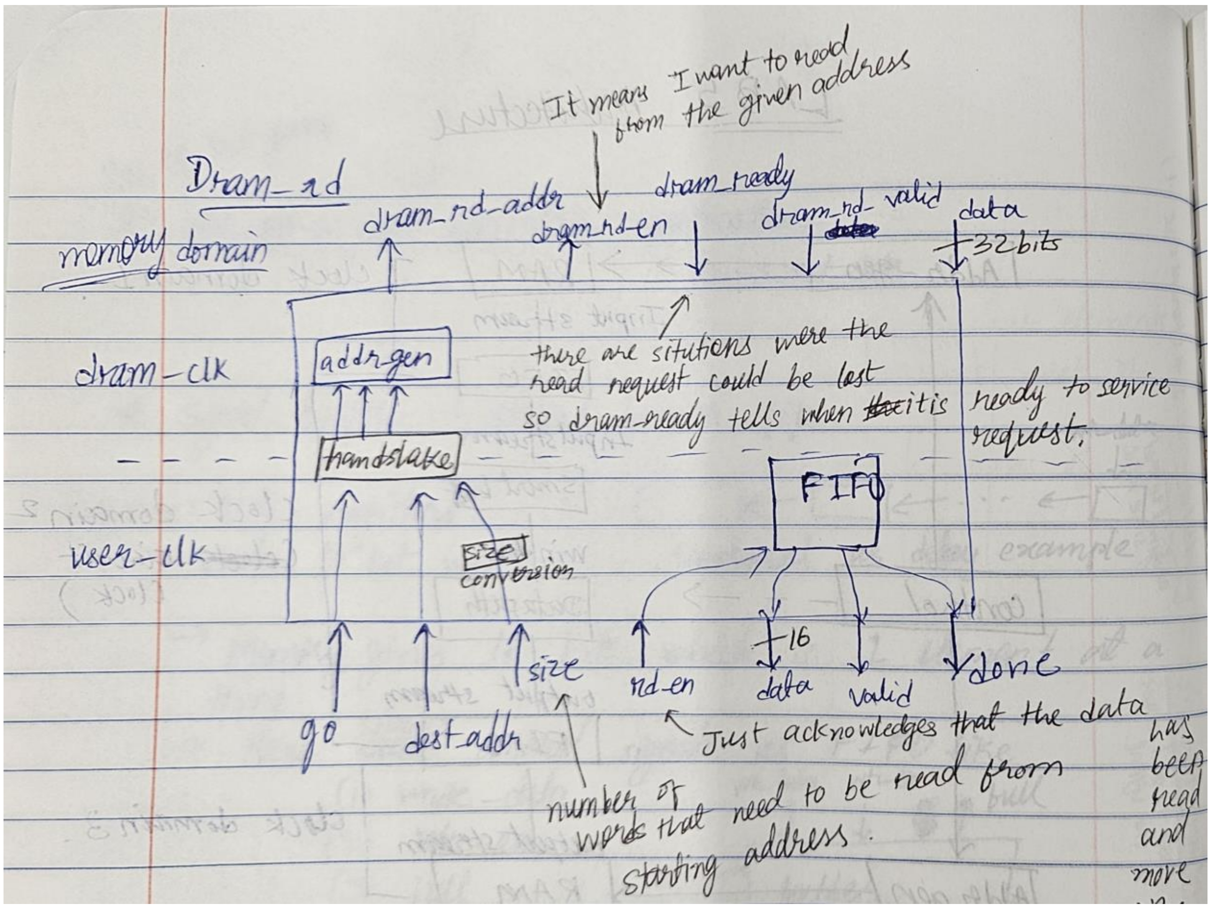
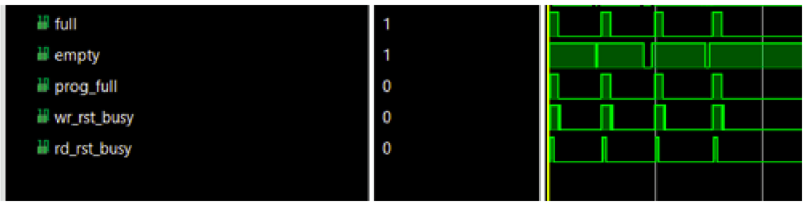

It means I want to read from the given address
Dram_rd
memory domain
dram_rd_addr
dram_rd_en
dram_ready
dram_rd_valid
data
32 bits
dram_clk
addr gen
handshake
there are situtions were the read request could be lost so dram_ready tells when ~~the~~ it is ready to service request.
FIFO
user_clk
size conversion
go
dest_addr
size
rd_en
16
data
valid
done
Just acknowledges that the data has been read and move
number of words that need to be read from starting address.

Figure 12: Architecture of DRAM_rd

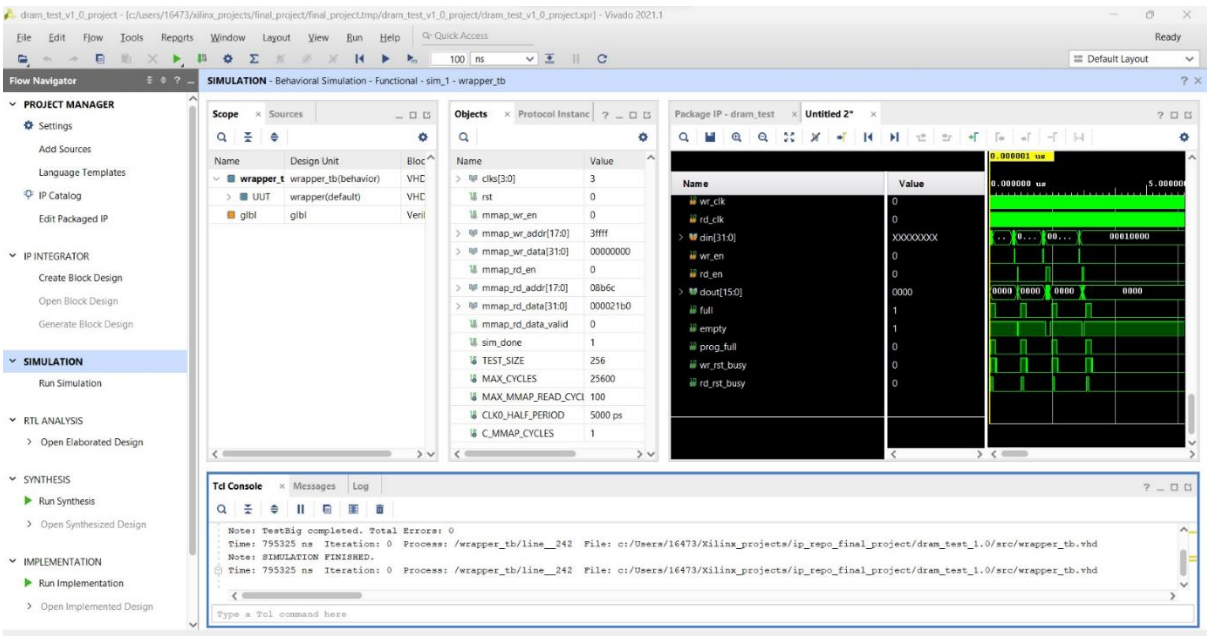
Our D_RAM_RD works completely without any error. We write the handshake component to relay the go signal, start address signal and size signal to activate the address generator, then the address generator generates the address one by one, and the FIFO consequently accepts the data from RAM0.

We created a FIFO called dram_fifo, which has a native type interface First Word Fall Through (no reading latency) and is programmable fully. The most important thing for the FIFO is the guarantee that the remaining data saved FIFO in the last tasks does not mess up the data in the current tasks FIFO accepted. We need to assert a reset port for FIFO to clean the data. The reset signal should be asserted to ‘1’ and maintained at ‘1’ for over one cycle. If you set the reset for only one cycle, the FIFO will stall at the RESETTING BUSY status and will not accept data anymore. We re-design the address generator, and we design a counter used for monitoring output and for debug convenience.



```
full          1
empty         1
prog_full     0
wr_rst_busy   0
rd_rst_busy   0
```

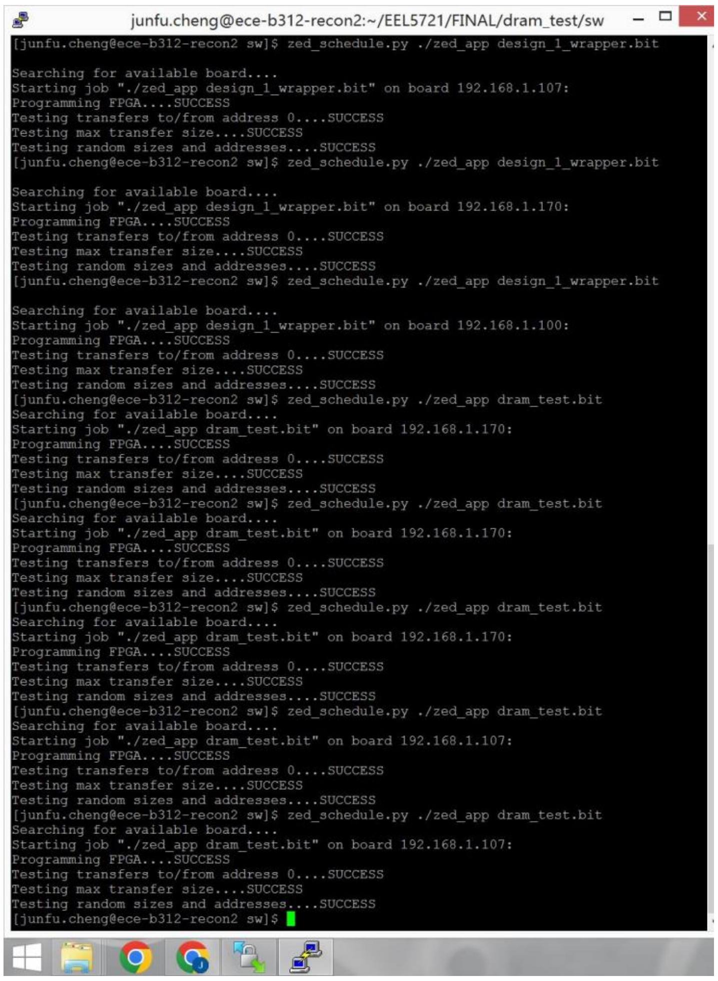
The following simulation output proves that our design works perfectly fine.



Name	Value
clks[3:0]	3
rst	0
mmap_wr_en	0
mmap_wr_addr[17:0]	3ffff
mmap_wr_data[31:0]	00000000
mmap_rd_en	0
mmap_rd_addr[17:0]	08b6c
mmap_rd_data[31:0]	000021b0
mmap_rd_data_valid	0
sim_done	1
TEST_SIZE	256
MAX_CYCLES	25600
MAX_MMAP_READ_CYCL	100
CLK0_HALF_PERIOD	5000 ps
C_MMAP_CYCLES	1

```
Note: TestBig completed. Total Errors: 0
Time: 795325 ns  Iteration: 0  Process: /wrapper_tb/line__242  File: c:/Users/16473/Xilinx_projects/ip_repo_final_project/dram_test_1.0/src/wrapper_tb.vhd
Note: SIMULATION FINISHED.
Time: 795325 ns  Iteration: 0  Process: /wrapper_tb/line__242  File: c:/Users/16473/Xilinx_projects/ip_repo_final_project/dram_test_1.0/src/wrapper_tb.vhd
```

The screenshot below shows that the dram_test IP works correctly on ZedBoard as well.



```
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app design_1_wrapper.bit

Searching for available board....
Starting job "./zed_app design_1_wrapper.bit" on board 192.168.1.107:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app design_1_wrapper.bit

Searching for available board....
Starting job "./zed_app design_1_wrapper.bit" on board 192.168.1.170:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app design_1_wrapper.bit

Searching for available board....
Starting job "./zed_app design_1_wrapper.bit" on board 192.168.1.100:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app dram_test.bit
Searching for available board....
Starting job "./zed_app dram_test.bit" on board 192.168.1.170:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app dram_test.bit
Searching for available board....
Starting job "./zed_app dram_test.bit" on board 192.168.1.170:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app dram_test.bit
Searching for available board....
Starting job "./zed_app dram_test.bit" on board 192.168.1.170:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app dram_test.bit
Searching for available board....
Starting job "./zed_app dram_test.bit" on board 192.168.1.107:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[junfu.cheng@ece-b312-recon2 sw]$ zed_schedule.py ./zed_app dram_test.bit
Searching for available board....
Starting job "./zed_app dram_test.bit" on board 192.168.1.107:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[junfu.cheng@ece-b312-recon2 sw]$
```

Figure 13: dram_test bitfile output on ZedBoard

Note: new_dram_rd.vhd requires the dram_FIFO