

1-D Time Domain Convolution

Fall 2021

EEL5721 – Reconfigurable Computing

Graduate Team

Team Members

Vidhya Hari, Vijayasai Jalagam

Department of Electrical & Computer Engineering

University of Florida

vidhya.hari@ufl.edu, v.jalagam@ufl.edu

Under the Guidance of Dr. Greg Stitt

Abstract—Convolution is the process of combining two signals to produce a third signal and is one of the principal techniques used in Digital Signal Processing. In this project, this technique has been implemented using the Xilinx Zync Zedboard. This report gives a brief description of the components used, their working and how they are all combined to produce convolution.

Keywords—FPGA, metastability, pipeline, buffer, DMA, DRAM, kernel

Introduction

Convolution involves two signals, one of which is the unit impulse i.e. impulse function at the sample time $t = 0$. The output response of any system can be found out by simply convoluting the input signal $x(t)$ with the impulse signal $h(t)$. In digital systems, convolution is a bitwise operation on the discrete signal. The convolution of two signals in time domain is equivalent to multiplication in frequency domain.

The aim of the project was to implement 1-D time convolution of an input signal and a kernel signal. The input signal resides in the memory (DRAM) and is passed on to the FPGA using a FIFO. The kernel signal is directly obtained from software. For the purpose of this project, a fake DRAM that is implemented as a block RAM simulation is used.

The project is divided into the following parts:

1. DRAM-DMA interface
2. Convolution pipeline
3. Signal/Smart buffer
4. Kernel buffer

DRAM-DMA Interface

Steps implemented:

Address Generator: Created an address generator for the `dram_rd_ram0` entity and set the clock frequency to 133MHz. The address generator generates addresses once the start signal and transfer size are given.

Handshake: Created and used a handshake synchronizer to integrate the DMA interface and the address generators. This was done to eliminate the possibility of metastability.

FIFO: Once the data was read from `DRAM_RD_RAM0`, it had to be sent to the signal buffer, again crossing a clock domain. Hence a ‘first-word-fall-through’ FIFO synchronizer was created for this purpose. The FIFO has different clocks for read and write operations. Also, 32 bits are written to the FIFO from DRAM while only 16 bits are read out. Hence, the write and read widths were assigned as 32 and 16 respectively. A prog-full flag is used in the FIFO and it is typically asserted a few cycles before the FIFO is actually full. This is done to ensure that the address generator does not mistakenly generate addresses when the FIFO is full.

Convolution Pipeline

The pipeline datapath consists of a multiplier adder tree that takes in two array inputs, multiplies the respective elements and finally adds the outputs from the multipliers using an adder tree. The output of the adder tree defines the final result which is the output of the datapath. The datapath for this project needs a total of 128 multipliers and 127 adders. The input to the datapath is given by `C_KERNEL_SIZE` elements each of width `C_KERNEL_WIDTH`. The inputs are then devectorized into 16 bits each.

Two cases had to be ensured with respect to pipeline output – accept only 16 bits of output and indicate when output is valid. The following logic was implemented to handle these:

Output Clipping: The convoluted output may sometimes exceed 16 bits. Whenever this happens, the output has to be saturated to ensure the right value is obtained. For this purpose, we implement clipping where the output bits are all made 1's in case the result exceeds 16 bits. This is done to ensure that the result does not wrap around to 0's incorrectly. For this, we have used the `mult_add_tree_out` signal which checks if the pipeline result exceeds 16 bits. In case it does, the output is made all 1's.

Valid signals: Before the pipeline output is sent to memory, the DRAM should know if the output is valid. For this purpose, we use the `valid_in` and `valid_out` signals. The `valid_in` is asserted when the buffer is full and the DRAM is ready to read. The `valid_out` is asserted after a delay equal to the pipeline depth which tells the DRAM that the data is valid.

Signal Buffer

This is essentially a smart buffer and is used to minimize reading directly from memory. The buffer consists of 128 16-bit registers. It uses the sliding window mechanism to generate a new window of inputs to the pipeline every cycle. Since the window differs by only one element every iteration, the buffer is implemented as a shift register that shifts 16 bits each cycle. When the `wr_en` flag is high, the buffer takes in a 16-bit input and shifts it along the registers. The `wr_en` is asserted only when the `ram1_wr_ready` DMA interface is Full and empty flags are assigned in order to prevent overflow and underflow respectively. The full flag is asserted high when all 128 registers are occupied and no more values can be loaded. Similarly, the empty flag denotes when the buffer is devoid of inputs and cannot be read from. Once all the input elements are loaded, the buffer produces an output array of 128 16-bit elements. The output of the buffer is `C_KERNEL_SIZE * C_KERNEL_WIDTH` bits ie. $128 * 16 = 2048$ bits. This is then passed to the datapath which devectorizes this into 16-bit values.

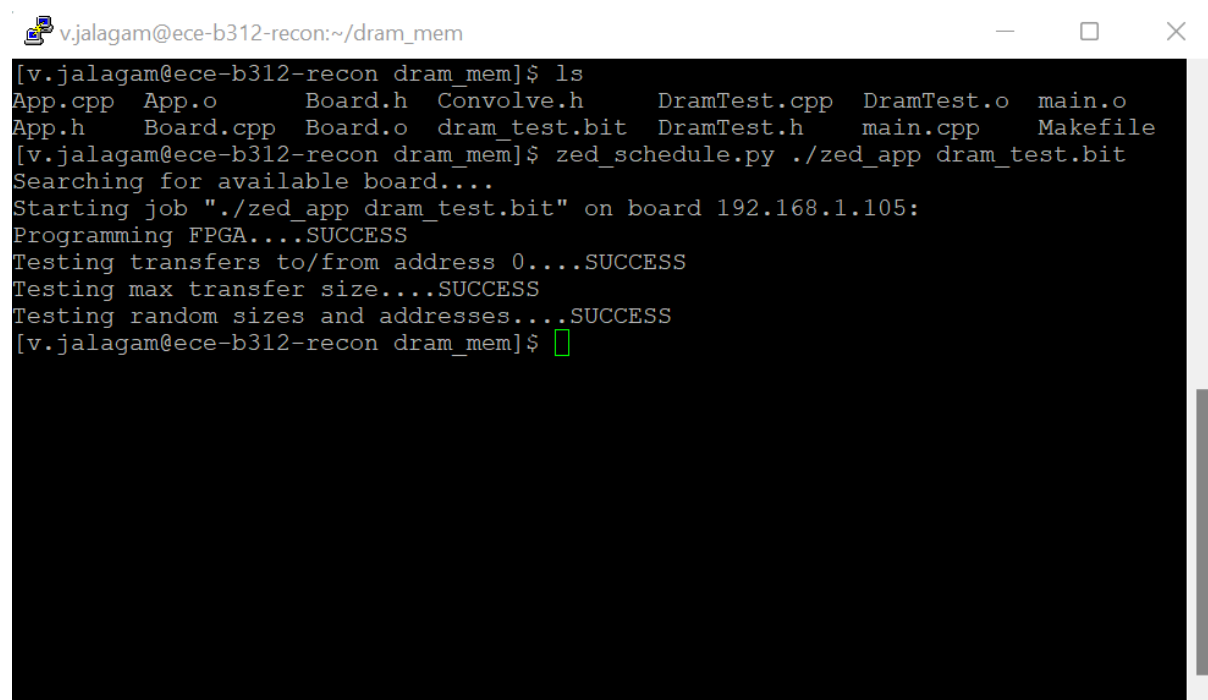
Kernel buffer

This is the second input to the convolution and only differs from the signal buffer in that it does not follow a sliding window mechanism ie. the inputs, once loaded, do not change each cycle. The kernel also consists of 128 16-bit registers. The `wr_en` flag is connected to the `kernel_load`

flag in memory map and both are asserted when the kernel is to be loaded with data. The kernel_full flag is connected to the kernel_loaded flag in the memory map to indicate that it has received all input elements. This typically happens after 128 memory map transfers.

RESULTS

The DRAM-DMA interface shows successful results when checked over the FPGA indicating that the memory transfers are happening as expected.

A terminal window with a black background and white text. The window title is 'v.jalagam@ece-b312-recon:~/dram_mem'. The terminal shows the following commands and output:

```
[v.jalagam@ece-b312-recon dram_mem]$ ls
App.cpp App.o Board.h Convolve.h DramTest.cpp DramTest.o main.o
App.h Board.cpp Board.o dram_test.bit DramTest.h main.cpp Makefile
[v.jalagam@ece-b312-recon dram_mem]$ zed_schedule.py ./zed_app dram_test.bit
Searching for available board....
Starting job "./zed_app dram_test.bit" on board 192.168.1.105:
Programming FPGA....SUCCESS
Testing transfers to/from address 0....SUCCESS
Testing max transfer size....SUCCESS
Testing random sizes and addresses....SUCCESS
[v.jalagam@ece-b312-recon dram_mem]$
```

Fig.1 Result of DRAM-DMA interface

The user_app module was successfully implemented and tested on the FPGA. This denotes successful working of the pipeline and the buffers.

```
vjalagam@ece-b312-recon:~/final_project
[v.jalagam@ece-b312-recon ~]$ cd final_project
[v.jalagam@ece-b312-recon final_project]$ ls
App.cpp  Board.cpp  convolve.bit  Convolve.o  Makefile  Timer.o
App.h    Board.h    Convolve.cpp  main.cpp    Timer.cpp  zed_app
App.o    Board.o    Convolve.h    main.o      Timer.h
[v.jalagam@ece-b312-recon final_project]$ 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 = 0.0197044

Testing small signal/kernel with all 1s...
Percent correct = 100
Speedup = 0.0534759

Testing small signal/kernel with random values (no clipping)...
Percent correct = 100
Speedup = 0.0141509

Testing medium signal/kernel with random values (no clipping)...
Percent correct = 100
Speedup = 2.68019

Testing big signal/kernel with random values (no clipping)...
Percent correct = 100
Speedup = 15.9938

Testing small signal/kernel with random values...
Percent correct = 100
Speedup = 0.0240964

Testing medium signal/kernel with random values...
Percent correct = 100
Speedup = 2.34033

Testing big signal/kernel with random values...
Percent correct = 100
Speedup = 14.5484

TOTAL SCORE = 100 out of 100
[v.jalagam@ece-b312-recon final_project]$
```

Fig.2 Result of user_app module

CONCLUSION

All individual blocks of the project have been tested for the expected functionalities. Both the DMA-DRAM module and the user_app module have been tested on the FPGA. The positive result shows time domain convolution has been successfully implemented on the zedboard. The project proved to be an absolute challenge to our understanding of the concept and subsequent reproduction through code. On the whole, it has been a wholesome experience and we thank Dr. Stitt for providing us this opportunity and guiding us throughout the semester. We also thank Robert Burwell for clearing our queries and helping us with the project work.