# Final Project Report:

## **1-D Time-Domain Convolution**

**Group-7**

Dinesh Reddy Ankireddy (Ufid:69325129)

Tharun Seemakurthi (Ufid: 76269441)

**Report: DRAM\_read (DMA), signal, kernel buffer and Convolution are fully functional and working.**

**Abstract:**

In this project, we have implemented 1-D Time domain convolution on FPGA using VHDL language. And also improve the performance.

**Introduction:**

Convolution is a common operation in DSP (Digital Signal Processing). we created a custom convolution circuit implemented on Zedboard which significantly improves in parallelism and performance compared to a microprocessor.

Convolution takes as input a signal (shown as the x array) and a kernel (shown as the h array). The output is another signal (y array), where each element of the output signal is the sum of the products formed by multiplying all the elements of the kernel with the appropriate elements of the input signal. A white background with black text

Description automatically generated

**FPGA design for Zed board**

**Design and Implementation:**

Part-1  
The DRAM\_read (DMA)Direct Memory Access entity.

This module is made connection between RAM and the User App. The input to the DMA interface is a ‘go’ signal after the go is asserted inside the DMA entity address generator will start. The data ‘dram\_rd\_data’ is the input to the DMA entity which is 32 bits of data and here the user app deals with 16 bits of data the data is divided and flipped into two parts. The components include FIFO, Handshake, Address Generator and a counter.

A paper with writing on it

Description automatically generated

1. **Address generator:**  
   This includes a state machine and a counter which counts the addresses from the start address given by the user app and stall operation which is coming from the program’s full signal from the FIFO entity.
2. **FIFO:**

It is used to handle two clock domains dram clock and user app clock here the input is from the RAM the data is 32 bits and the output data is 16 bits to the user app where the data input is flipped, and the read is enabled for the FIFO depend on the empty signal of the FIFO and read enable d signal from the user app. The FIFO incorporates a programmable full flag, activated when its capacity reaches 17 entries. When the number of available slots in the FIFO reduces to 17, the system restricts the acceptance of new data until there are a minimum of 2 additional free spaces. This programmable full flag addresses variations in delay between address generation and entry into the FIFO, ensuring efficient data handling.

1. **Handshake**

The purpose is handshake synchronizer to avoid metastability as the signals are crossing from the user domain to the dram domain. The ‘go’ signal is inserted delays in send acknowledge, ‘start address’ and ‘size’ are stored in registers and when go is asserted the start address is given to the address generator after getting acknowledgement for the ‘go’.

1. **Counter**

This entity is just a counter which counts from 0 to size number and after it reaches max size done is asserted, it depends on the read-enabled signal from the FIFO here the out put of the counter that is done is sent to reset the FIFO using the **dual flop** because it is crossing the clock domain to reset the FIFO.

Part-2

The Convolution

Here we will design the user app entity which has 2 main buffers that is a signal buffer or smart buffer and a kernel buffer along with this we will have a memory map, delay for the pipeline, and pipeline controller used to compute the convolution and produce the output consisting of 39 bits it is then clipped to 16 bits output in this way, If (38 down to 16 is not zero then output is all ones which are 16 bits ones, else the output is clipped 15 down to 0.

A paper with writing on it

Description automatically generated

1. **Signal buffer:**

Here this is implemented by using shift registers concepts. Here the data is input from the ram0\_rd\_data signal which is 16 bits from the dram\_read entity and the output of the data sent to the pipeline which is vectorized as 128 \* 16 format here the elements are 128 words and 16 bits. Here the counter is instantiated where the counter counts to 128 elements. Here there is a controller available which indicates if the counter is less than 128 than empty is asserted to ‘0’ otherwise ‘1’.If the count is 128 and reads enabled is ‘0’ than full is ‘1’ otherwise ‘0’.Here read enable depends on ram1\_wr\_ready and not empty logic

1. **Kernel buffer:**

This entity is similar to the signal buffer but here read enabled in the counter is hard coded to ‘0’ and the read enabled of the buffer depends on ram1\_wr\_ready and not full logic.

1. **Memory map:**

Here this module will give data to the kernel buffer and also gives the size value to ram0\_rd\_size which is signal size and 2\* kernel size-1 and deal with ram1\_wr\_size which is kernel size and signal size -1.

1. **Valid delay logic:**

Here input depends on ram1\_wr\_ready and (not signal buffer empty) logic and also includes a delay of 8 cycles because the pipeline took 8 cycles to get the output and the output is valid signal for the ram1\_wr\_valid signal.

**Results:**

**Dram (DMA)**

A screen shot of a computer

Description automatically generated

**Convolution**

A screenshot of a computer

Description automatically generated

**Challenging**

1.

A screenshot of a computer

Description automatically generated

I am confused about where to connect the reset signal in dram\_clock or user\_clock after using dual flop.

2. During the initial bitstream generation, we achieved a single success, followed by a halt. Subsequent debugging revealed that we overlooked the need to reverse the data intended for the pipeline.

3. We forgot to modify the memory map in the convolve part so there were number of issues later overcome it.