ISTANBUL TECHNICAL UNIVERSITY

# Digital System Design Applications

Experiment VII
CONVOLUTION CIRCUITS

In this experiment, a multiplier and a 2-D convolution [1] circuit will be designed by using structural and behavioral modelling. These designs will be described in Verilog and be verified using testbenches. A simple image processing example will be given.

## Preliminary

Students should do a research about 2D-convolution [1] and investigate the resources given in the lecture.

## Objectives

- To learn how to design multiplier circuits.
- To learn how to design 2D-Convolution circuit.
- To do simple examples about 2D convolution and its usage in the image processing.

## Requirements

Students are expected to be able to

- Design basic combinatorial and sequential circuits.
- Describe circuits by using structural and behavioral modelling with Verilog.
- Use Vivado: synthesize, simulate, implement designs, generate bitstreams and configure FPGA.

## Experiment Report Checklist

1. **Behavioral Multiplier**
    - Verilog code for your design and testbench.
    - Simulation results. Both waveform and TCL console output.
    - RTL and Technology Schematics.
    - Resource usage table of the implemented design.
    - Pad to pad path delays of the implemented design.
2. **Structural Multiplier**
    - Verilog code for your design and testbench.
    - Simulation results. Both waveform and TCL console output.
    - RTL and Technology Schematics.
    - Resource usage table of the implemented design.
    - Pad to pad path delays of the implemented design.
3. **2D Convolution**
    - Explain 2D-Convolution briefly.
    - Write mathematical calculation of final result for the center location of the image. (Write it by hand or with a computer program).
    - Verilog code for your design and testbench.
    - Simulation Results. Both waveform and TCL console output.
    - MATLAB code and the result obtained from command window.
    - RTL and Technology Schematics
    - Resource usage table of the implemented design.
    - Pad to pad path delays of the implemented design.

- **Projects and reports are to be done INDIVIDUALLY. High amounts of points will be deducted from similar works.**
- **Reports must be written in a proper manner. Divide your text to sections and subsections if needed, label your figures and connect your sections with proper explanations of your works. Reports filled with imprecisely placed tables and figures, with no verbal explanations in workflow, will not fare well.**
- **Check homeworks section in Ninova for submission dates.**

## Behavioral Multiplier

1. Add a new Verilog file named **convolution_circuits.v** to your project.
2. Create a new module called **MULTB**. This module should have 1-bit inputs **clk, reset, start**; 8-bit inputs $A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$, $B = (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$; 16-bit output **result**, $R = (r_{15}, r_{14}, r_{13}, r_{12}, r_{11}, r_{10}, r_9, r_8, r_7, r_6, r_5, r_4, r_3, r_2, r_1, r_0)$; 1-bit output **done**.
3. You should design the behavioral 8-bit multiplier by using only one **always** block.
4. If **reset** and **start** signals are both low, then all the outputs and the registers must be zero. When the multiplication is completed, assign one(1) to **done**.
5. Write a **testbench** to ensure that your circuit is working correctly.
6. Synthesize and implement your design.

## Structural Multiplier

1. Create a new module called **MULTB**. This module should have 1-bit inputs **clk, reset, start**; 8-bit inputs $A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$, $B = (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$; 16-bit output **result**, $R = (r_{15}, r_{14}, r_{13}, r_{12}, r_{11}, r_{10}, r_9, r_8, r_7, r_6, r_5, r_4, r_3, r_2, r_1, r_0)$; 1-bit output **done**.
2. You will use a simple binary multiplication algorithm as shown in **Figure 1**. You should multiply the **multiplicand A** by each bit of the **multiplier B**. You should define 16-bit registers PP0, PP1, PP2, PP3, PP4, PP5, PP6, PP7 to save partial products. After that, perform summation of these partial products and obtain the final **result R**. The calculation of the values of registers is as follows:

$$\begin{aligned} PP0 &= b_0 * A * 2^0 \\ PP1 &= b_1 * A * 2^1 \\ PP2 &= b_2 * A * 2^2 \\ PP3 &= b_3 * A * 2^3 \\ PP4 &= b_4 * A * 2^4 \\ PP5 &= b_5 * A * 2^5 \\ PP6 &= b_6 * A * 2^6 \\ PP7 &= b_7 * A * 2^7 \end{aligned}$$

3. If reset and start signals are both low, then all the outputs and the registers must be zero.
4. Use an **always** block to calculate partial products.
5. Use 16-bit Ripple Carry Adders that you designed in Experiment 4 in order to obtain the final **result R**. Define 16-bit wires sum1, sum2, sum3, sum4, sum5, sum6, sum6, sum7 as the outputs of the adders. The sum7 value will written to the result.
6. Your circuit should produce the final result, **at the rising edge of the second clock cycle** after the inputs applied. Hence, the **done** signal should be one at the second rising edge as well.

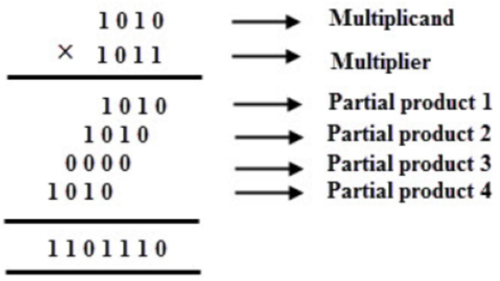

Figure 1: Binary Multiplication.

7. Write a test code to ensure that your circuit is working correctly.
8. Synthesize and implement your design.

## 2D-Convolution

1. An image is defined as a two-dimensional function, $f(x, y)$, where $x$ **and** $y$ are spatial (plane) coordinates and the **amplitude of** $f$ at any pair of coordinates $(x, y)$ is called the intensity of the image at that point. Each point refers to a pixel in an image.
2. In this experiment, we use 8-bit grayscale images in which value of a pixel is represented by 8-bits resulting in a 256 different shades of gray.
3. Filtering is used in a broad spectrum of image processing applications such as blurring, sharpening, embossing, edge detection and more [1]. Image filtering can be done in spatial and frequency domains. In this experiment, we will perform spatial filtering.
4. A linear spatial filter performs a sum-of-products operation (convolution) between an **image** $f$ and a **filter kernel** $w$. The **kernel** $w$ is a matrix whose size is $m \times n$, where $m = 2a + 1$, $n = 2b + 1$. The **image** $f$ is also a matrix whose size is $M \times N$.
5. Finally, the size of the result is $(M + m - 1) \times (N + n - 1)$.
6. The 2D-convolution formula is as follows [1]

   $$g(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i, j) f(x - i, y - j)$$

   where $x$ and $y$ are varied so that the center (origin) of the kernel visits every pixel in $f$ once. For a fixed location of $(x, y)$, the equation implements the sum of products and provides the new value at that location.
7. Consider a $3 \times 3$ **image** and $3 \times 3$ **kernel**. You need to calculate 2D-convolution **just for the center location**. Calculate the sum of products for the center pixel. You can use the MATLAB source shared with you in Ninova.
8. Create a new module which is called **CWODSP** into your **convolution_circuits.v** file.
9. This module is going to have 18 8-bit inputs **f11, f12, f13, f21, f22, f23, f31, f32, f33, w11, w12, w13, w21, w22, w23, w31, w32, w33**, 3 1-bit inputs **clk, reset, start**, a 24-bit output **result** and 1-bit output **done**.

10. Firstly, you need to make nine multiplications like: **m1=f11*w11, m2=f12*w12, ..., m9=x33*w33**. Use your own **Behavioral multiplier** circuit for this design.
11. After that, you need to add the results of these nine multiplication and obtain the output result as follows: **result = m1+m2+m3+m4+m5+m6+m7+m8+m9**. Use your own **24-bit RCA** adder circuits for this design.
12. When the multiplication is completed, assign one(1) to **done**.
13. Write a test code to ensure that your circuit is working correctly. Give arbitrary values to image f between [0,255] and use a Gaussian Blur kernel for w as follows:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

14. For verification, calculate the new value of the center location by using MATLAB, and compare the MATLAB and Verilog testbench results. The results must be same. MATLAB code will be provided in Ninova.
15. Synthesize and implement your design.

## References

[1] R. C. Gonzalez and R. E. Woods, Digital image processing. New York: Pearson, 2018.

[2] Nexys4 DDR Reference Manual

[3] Artix-7 Libraries Guide for HDL Designs

[4] Constraints Guide

[5] Stephen D. Brown and Zvonko G Vranesic, Fundamentals of Digital Logic with Verilog Design, McGraw-Hill, 2002.