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. **Structural Multiplier - Unsigned**
   - Verilog code for your design and testbench,
   - Simulation results. Both waveform and TCL console output.
2. **Structural Multiplier - Signed**
   - Verilog code for your design and testbench,
   - Simulation results. Both waveform and TCL console output,
   - Explain how many number of adder stages are necessary,
   - Explanation of Baugh-Wooley Method,
   - RTL and Technology Schematics,
   - Utilization Report,
   - Timing Report.
3. **Behavioral Multiplier**
   - Verilog code for your design and testbench,
   - Simulation results. Both waveform and TCL console output.
   - RTL and Technology Schematics,
   - Utilization Report,
   - Timing Report,
   - Comparison between structural and behavioral designs.
4. **Multiply and Accumulate (MAC)**
   - Verilog code for your design and testbench,
   - Simulation results. Both waveform and TCL console output.
   - RTL and Technology Schematics,
   - Utilization Report,
   - Timing Report,
   - Maximum achievable clock frequency.
5. **2D Convolution**
   - Explain 2D-Convolution briefly.
   - Write mathematical calculation of 2D-Convolution,
   - Explain how 2D-Convolution can be calculated by sliding the kernel window over the image,
   - Verilog code for your design and testbench.
   - Simulation Results. Both waveform and TCL console output.
   - MATLAB code and the result obtained from command window.

- **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.**

Structural Multiplier - Unsigned

1. Create a new module called **MULTS**. This module should have 8-bit inputs **A** as a multiplicand, **X** as a multiplier and 16-bit output **result**. You will use a simple binary multiplication algorithm as shown in **Figure 1**. First, you need to calculate the partial products and then perform summation of these partial products to obtain the final **result**.

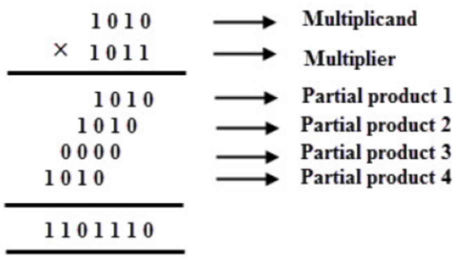

Figure 1: Binary Multiplication

2. Define 16-bit signals **PP[0], PP[1], PP[2], PP[3], PP[4], PP[5], PP[6], PP[7]** as partial products. Multiply the **multiplicand A** by each bit of the **multiplier X** to save partial products by using a **generate block** and **shift operators**. The calculation of the partial products is as follows:

$$\begin{aligned}
PP0 &= x_0 * A * 2^0 \\
PP1 &= x_1 * A * 2^1 \\
PP2 &= x_2 * A * 2^2 \\
PP3 &= x_3 * A * 2^3 \\
PP4 &= x_4 * A * 2^4 \\
PP5 &= x_5 * A * 2^5 \\
PP6 &= x_6 * A * 2^6 \\
PP7 &= x_7 * A * 2^7
\end{aligned}$$

3. Implement **three adder stages** to obtain the final result as shown in **Figure 2**. Use **16-bit Carry Lookahead Adders** for each adder stage. Define 16-bit wires sum1, sum2, sum3, sum4, sum5, sum6, sum6 as the outputs of the adders. The sum6 value will be written to the **result**.

4. Create a testbench to simulate your design.

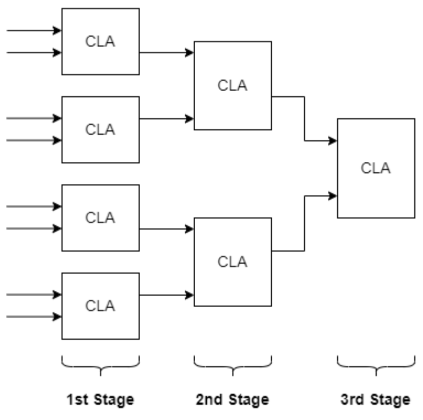

Figure 2: Three Adder Stages

Structural Multiplier - Signed

1. Create a new module called **MULTS_signed**. This module should have 8-bit inputs **A** as a multiplicand, **X** as a multiplier and 16-bit output **result**. You will use the same binary multiplication algorithm as shown in **Figure 1**.
2. Define 8-bit signals **PP[0], PP[1], PP[2], PP[3], PP[4], PP[5], PP[6], PP[7]** as partial products. Calculate these partial products by using a **generate block without shift operators**.
3. Apply **Baugh-Wooley Method** to the partial products as shown in **Figure 3** [2].

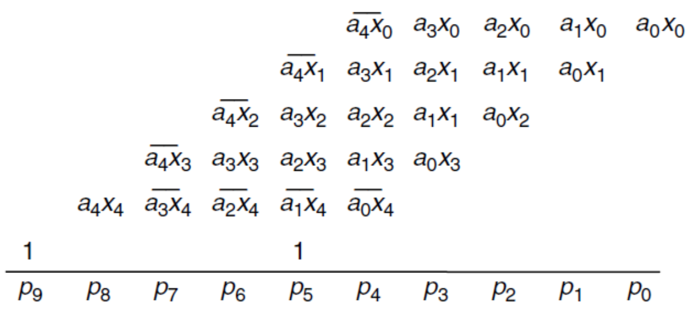
$\overline{a_4x_0}$ a_3x_0 a_2x_0 a_1x_0 a_0x_0
$\overline{a_4x_1}$ a_3x_1 a_2x_1 a_1x_1 a_0x_1
$\overline{a_4x_2}$ a_3x_2 a_2x_2 a_1x_1 a_0x_2
$\overline{a_4x_3}$ a_3x_3 a_2x_3 a_1x_3 a_0x_3
a_4x_4 $\overline{a_3x_4}$ $\overline{a_2x_4}$ $\overline{a_1x_4}$ $\overline{a_0x_4}$
1 1
p_9 p_8 p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0

Figure 3: Baugh-Wooley Method for 4x4 Multiplier [2]

4. Define 16-bit signals **PP_shifted[0], PP_shifted[1],...,PP_shifted[7]** and make **shift** operations to the partial products by using a **generate block**.
5. Implement a similar adder structure as shown in **Figure 2**. Use **16-bit Carry Lookahead Adders** for each adder stage to obtain the final result.
6. Explain that whether three adder stages are enough or number of stages should be increased.
7. Create a testbench to simulate your design. Change the radix to **signed decimal** in the waveform.
8. Synthesize and implement your design.
9. Make a brief research about **Baugh-Wooley Method** and explain it.
10. Add the followings to your report:
    - Explain how many number of adder stages are necessary,
    - Explanation of Baugh-Wooley Method,
    - Utilization Report,
    - Timing Report.

Behavioral Multiplier

1. Create a new module called **MULTB**. This module should have 8-bit inputs **A**, **B** and 16-bit output **result**.
2. Design a behavioral **8-bit signed multiplier** using an **always block**. Use **signed** keyword for both input operands and output result.
3. Write a **testbench** to show that your circuit works as expected.
4. Synthesize and implement your design. Make a comparison between the structural and behavioral signed multiplier designs.
5. Add the followings to your report:
   - Utilization Report,
   - Timing Report,
   - Comparison between structural and behavioral designs.

Multiply and Accumulate (MAC)

1. Create a new module called **MAC**. This module should have 1-bit inputs **clk**, **reset**; 24-bit inputs **data**, **weight** and 20-bit output **result**. **MAC** unit will perform convolution operations by computing products of data and weights and adds these products to an accumulator. The structure of the MAC unit is shown in **Figure 4**.

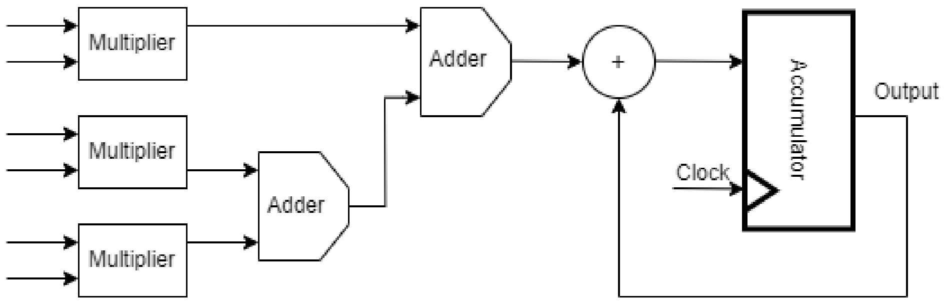

Figure 4: MAC Unit with 3 Parallel Multiplier

2. Implement three parallel **behavioral signed multipliers** that you designed in the previous step. Create a **parametric behavioral signed adder** to add the results of these multipliers. The calculations are as follows:

$$\begin{aligned}
product[0] &= data[7:0] * weight[7:0] \\
product[1] &= data[15:8] * weight[15:8] \\
product[2] &= data[23:16] * weight[23:16] \\
sum[0] &= product[0] + product[1] \\
sum[1] &= product[2] + sum[0] \\
result &:= result + sum[1]
\end{aligned}$$

3. Implement the **accumulator** by using an **always block** and **non-blocking assignments** (circuit is sequential). For each clock cycle results will be accumulated. Define a 2-bit register **count** to count the number of accumulations. When **reset** signal is high, the **count** value and output **result** value will be assigned to zero.

4. Create a **testbench** to simulate your design. The **data input** should be your 9-digits Student ID, and the weights should be {-1,-1,-1,-1,8,-1,-1,-1,-1}. Accumulate the products of data and weights in 3 clock cycles.

5. Synthesize and implement your design. Find the maximum achievable **clock frequency**.

6. Add the followings to your report:
   - Utilization Report,
   - Timing Report,
   - Maximum achievable clock frequency.

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 5×5 **image** and 3×3 **kernel**. You need to calculate 2D-convolution of the image. The **result** will be **3 x 3 matrix**. Calculate the sum of products for each **3 x 3 area** in the image. You need to slide the kernel window both horizontally and vertically. You can use the MATLAB sources shared with you in Ninova.

8. Use the **MAC unit** that you designed in the previous step to calculate 2D convolution.

9. Write a test code to ensure that your circuit is working correctly. Give arbitrary values to 5x5 image **f** between [0,255] and use a 3x3 **Laplacian Kernel** for **w** as follows:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

10. For verification, calculate the new value of the 9 different locations by using MATLAB, and compare the MATLAB and Verilog testbench results. The results must be same. MATLAB code will be provided in Ninova.

References

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

[2] B. Parhami, Computer Arithmetic Algorithms and Hardware Designs, Oxford University Press, 2010.

[3] Nexys4 DDR Reference Manual

[4] Artix-7 Libraries Guide for HDL Designs

[5] Constraints Guide

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