Computer Aided Design (CAD) -Assignment Description

University of Tehran

Electrical and Computer Engineering Department

Fall 98

CA1: FPGA Hardware Resources in Multiplier Designs

In this assignment, you have to implement a pre-designed LUT-Based model of an approximate multiplier architecture, which has been specifically designed for FPGA-based systems. Also you should compare the resource utilization, performance and power of this approximate module with an accurate multiplier.

INTRODUCTION

DSP blocks vs LUT-based Multipliers. Multiplication is one of the basic arithmetic operations, used extensively in the domain of digital signal and image processing. Major FPGA vendors, such as Xilinx and Intel, provide hardwired DSP blocks to achieve fast multipliers. Despite the availability and high performance offered by the DSP blocks, there is a limited number of such blocks in an FPGA. Thus, the need of LUT-based multipliers is inevitable for some applications that have many multiplication operations. That is why we still need to design a multiplier in VHDL/Verilog and synthesize it on LUTs.

Accurate vs Approximate Multipliers. Although some applications require multipliers that are highly accurate, a wide range of applications do not require accurate computations and their operations can be approximated to get higher speed. By approximate multiplication, the computation of these applications comes with some error. However, these applications have inherent resilience to approximation induced errors and thereby demonstrate the ability to produce good and acceptable outputs despite some of the input data/intermediate computation being inaccurate or approximate. Examples of such applications can be found in the domains of image/signal processing, artificial intelligence and machine learning, and various other probabilistic algorithms.

- **PROJECT DESCRIPTION**

CLBs in Xilinx Artix7-series FPGAs. The configurable logic blocks (CLBs) are the main resources involved in implementing any kind of sequential or combinatorial circuit on Xilinx FPGAs. A CLB in modern FPGAs, such as Xilinx Artix7-series has eight 6-input lookup tables (LUTs) and an 8-bit long carry chain.

As shown in Figure 1(a), the 6-input LUTs can be used to implement either a single 6-input combinational function or two 5-input combinational functions. The LUTs are also used to control the associated carry chain.

In the 7-series FPGAs, each CLB has two slices and each slice has four LUTs: so each CLB contains eight LUTs and the associated carry chain, as shown in Figure 1(b).

As shown in Figure 1(a), the 6-input LUTs can be used to implement either a single 6-input combinational function or two 5-input combinational functions. Each 6-input LUT is filled by a 64-bit value (called the INIT value). The INIT value of an LUT is actually the 64 rows (each can be either 0 or 1) of the truth table of the function.

*In the below link, Page 273-276, you can see how the INIT value for an LUT can be defined. (*https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/7series_hdl.pdf*)

The appendix on the last part of this homework document also introduces sample VHDL and Verilog codes for filling an LUT. You just should change the assignment statement of the INIT variable to fill it with the proper value based on your function.

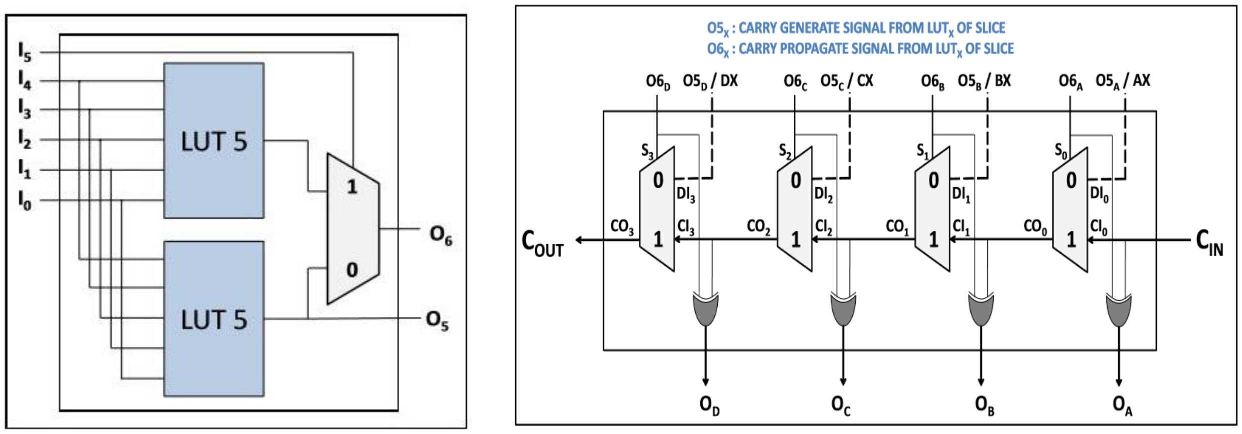


Figure 1. (a) 6-input LUT of Xilinx FPGAs (b) Carry chain of Xilinx FPGAs

FPGA Optimized Accurate Multiplier. **(Just For Study)** Utilizing the general structure of a $n \times n$ multiplier and exploiting the 6-input LUTs and fast carry chains of modern FPGAs, an accurate $n \times n$ array multiplier has been shown in Figure 2.

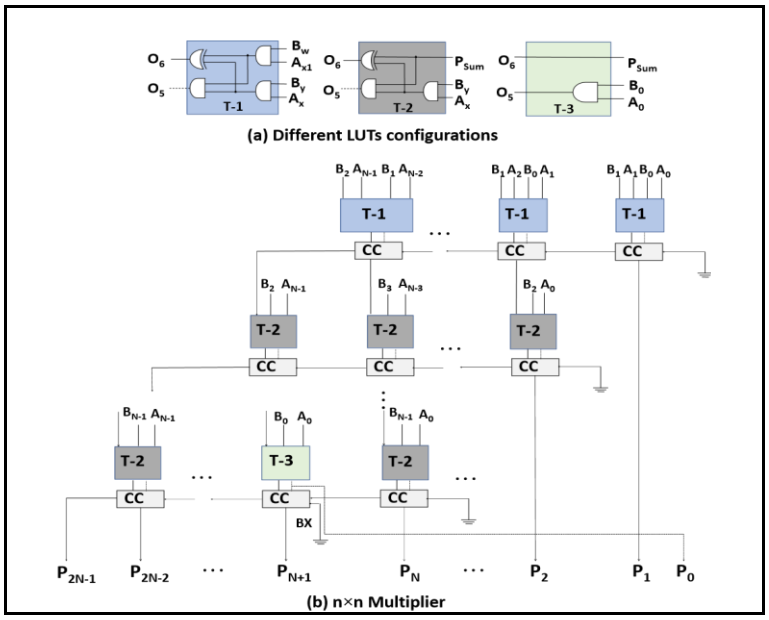
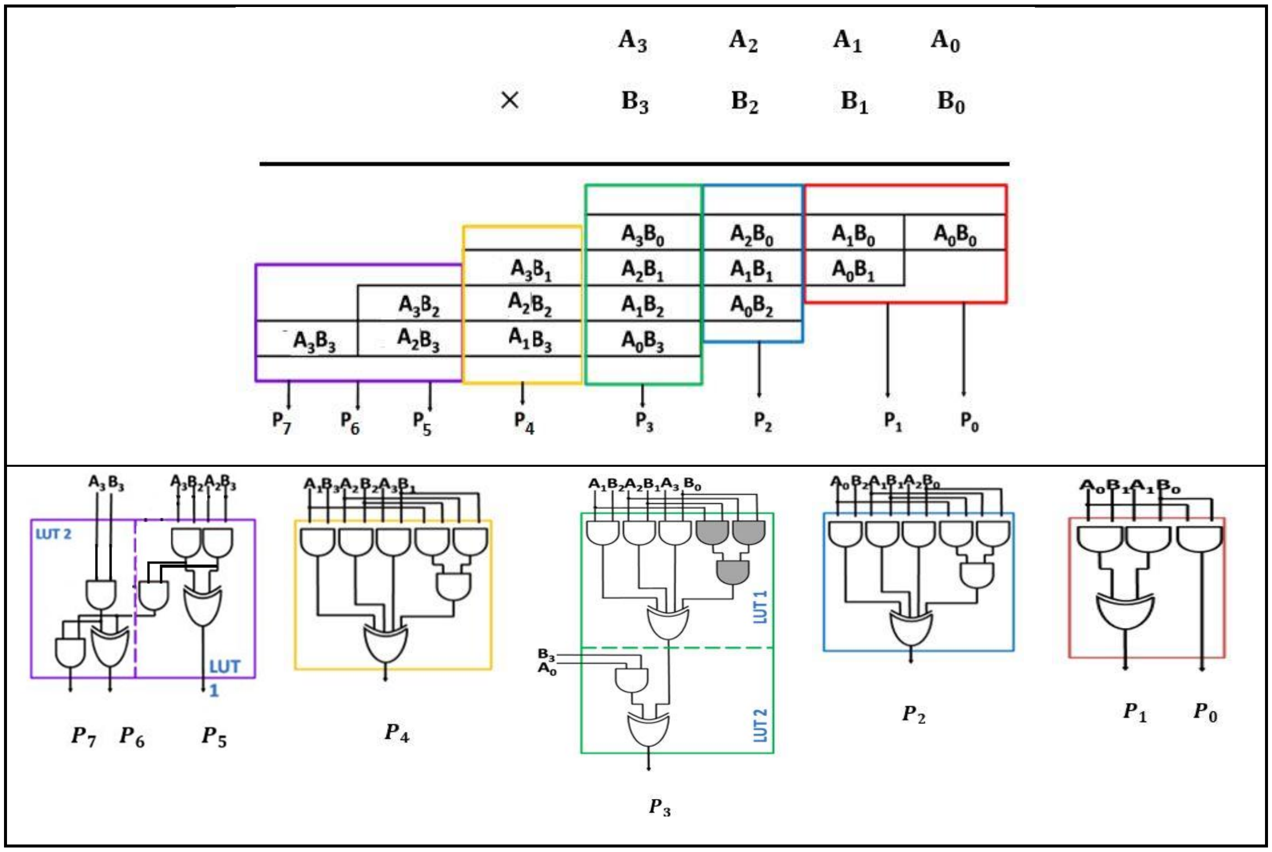


Figure 2. Area and energy efficient implementation of *n*×n multiplier

This accurate multiplier combines the processes of partial products generation and their accumulation into a single step, to efficiently utilize FPGA resources. For this purpose, it classifies the LUTs into three configurations: T-1, T-2 and T-3 as shown in Figure 2(a). As shown in Figure 2(b), the first row of this accurate multiplier has LUTs with T-1 configuration and it generates the first two rows of partial products of a $n \times n$ multiplier and accumulates them using the associated carry chain. The remaining $n - 2$ rows have LUTs with configuration T-2 for generation of the remaining partial products and their accumulation through the carry chains. The last row has an LUT of Type T-3 for generation of product bits P_N+1 and P_0. Using this accurate multiplier, some approximation techniques can be proposed to enable gains in area, latency and energy.

FPGA Optimized Approximate Multiplier. **(You Should Implement)** Figure 3 shows an approximate design that is optimized to improve latency and energy gains of the accurate multiplier shown in Figure 2 by removing the associated carry chains for partial products accumulation. The first step in this approximate design is grouping the partial products into multiple layers, as shown by the different color boxes in Figure 3.



A_3 A_2 A_1 A_0
$\times$ B_3 B_2 B_1 B_0
A_3B_0 A_2B_0 A_1B_0 A_0B_0
A_3B_1 A_2B_1 A_1B_1 A_0B_1
A_3B_2 A_2B_2 A_1B_2 A_0B_2
A_3B_3 A_2B_3 A_1B_3 A_0B_3
P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0
LUT 2
LUT 1
LUT 1
LUT 2
B_3
A_0
P_7 P_6 P_5
P_4
P_3
P_2
P_1 P_0

Figure 3.(a) Grouping partial products in each layer,(b)Function of partial product group's LUT

As shown by the red box in Figure 3(a), the first two columns in each layer can be accurately produced by a single LUT. The corresponding red box in Figure 3(b) shows the implementation of the function performed by the LUT. Similarly, the blue box in Figure 3(b), utilizes a single LUT to accurately generate and add the partial products in the third column. The yellow box in third last column of each layer, is utilized for approximately computing the sum of partial products. The approximation in the function performed by the yellow box is the result of predicting the carry-out from the preceding column. The purple boxes are utilized to compute the final two product bits in each layer. As shown in Figure 3(b), the purple boxes utilize two LUTs for improving the accuracy of the most significant bit. As all the partial product bits grouped by green boxes contain four partial product terms, the proposed approximation utilizes two 6-input LUTs per group for generating and computing the sum of these bits. As shown by green box in Figure 3(b) a 6-input LUT is used for computing the first three partial product terms in each group. This LUT also implements three AND gates, identified by shading, to predict the carry from preceding bit locations. The computed three partial product terms and the predicted carry are added together to generate an approximate sum. Also by using a second LUT, this sum is again added with the fourth partial product term to compute the final approximate sum of the group.

Deliverables:

In this assignment you should implement and synthesize a **4*4 Multiplier** structure based on these methods:

1) **Accurate multiplier with DSP blocks**. Xilinx FPGAs have dedicated hardware resources. If you use these hardware resources, the synthesizer uses them for synthesizing your design (instead of LUTs). There are several ways for using these hardware resources: Using IP Cores, instantiating the resources in the code, and using Coding Style. In this case, your design must use Block Multiplier IP Core.
2) **Accurate multiplier with LUTs**, in which all the logics are implemented on CLBs. In this case you simply use the * operator with 4-bit inputs and 8-bit output and leave the synthesis process to the tool.
3) **Approximate multiplier with LUTs** based on Figure 3 that has been specifically designed for FPGA-based systems. You must use the proper **Coding Style** method.

For more information see the: "Xilinx 7 Series FPGA Libraries Guide for HDL Designs" *https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/7series_hdl.pdf*

- You must design this module with **Verilog** or **VHDL.**
- Your design must be **Synthesizable**.
- Choose your target FPGA in the synthesis tool as **Artix7-series** family.
- Write a proper **testbench** to test your design and a **Comprehensive Report** of the steps involved in the project. The testbench should apply random 64 pair of inputs (two 4-bit data) and to see if the design works correctly. For the approximate multiplier, it should report the average error (the average of |(approximate result)-(accurate result)| for all 64 inputs).
- Report and analyze the **Device utilization, Performance,** and **Power consumption** results generated by **Vivado** software and **Accuracy** calculated by the testbench for different multiplier architectures (1), (2), and (3).
    - For Performance report the ….
    - For Device utilization, report the number of LUTs, DSP blocks, flip flops,… that the design has.

Refrence papers:

[1]."Area-Optimized Low-Latency Approximate Multipliers for FPGA-based Hardware Accelerators"
[2]."SMApproxLib:Library of FPGA-based Approximate Multipliers"

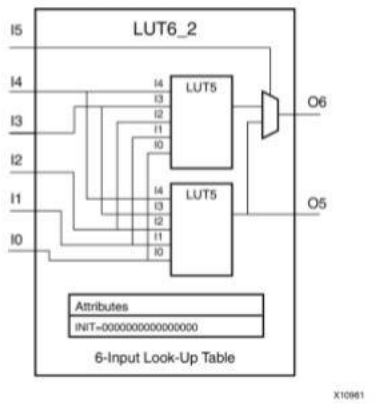
Good luck :)

HELP

Xilinx 7 Series FPGA Libraries Guide for HDL Designs

LUT6_2

Primitive: Six-input, 2-output, Look-Up Table



Verilog Instantiation Template

```
// LUT6_2: 6-input, 2 output Look-Up Table
//         7 Series
// Xilinx HDL Libraries Guide, version 14.1

LUT6_2 #(
   .INIT(64'h0000000000000000) // Specify LUT Contents
) LUT6_2_inst (
   .O6(O6), // 1-bit LUT6 output
   .O5(O5), // 1-bit lower LUT5 output
   .I0(I0), // 1-bit LUT input
   .I1(I1), // 1-bit LUT input
   .I2(I2), // 1-bit LUT input
   .I3(I3), // 1-bit LUT input
   .I4(I4), // 1-bit LUT input
   .I5(I5)  // 1-bit LUT input (fast MUX select only available to O6 output)
);

// End of LUT6_2_inst instantiation
```

VHDL Instantiation Template

Unless they already exist, copy the following two statements and paste them before the entity declaration.

```
Library UNISIM;
use UNISIM.vcomponents.all;


-- LUT6_2: 6-input  2 output Look-Up Table
--         7 Series
-- Xilinx HDL Libraries Guide, version 14.1

LUT6_2_inst : LUT6_2
generic map (
   INIT => X"0000000000000000") -- Specify LUT Contents
port map (
   O6 => O6,  -- 6/5-LUT output (1-bit)
   O5 => O5,  -- 5-LUT output (1-bit)
   I0 => I0,   -- LUT input (1-bit)
   I1 => I1,   -- LUT input (1-bit)
   I2 => I2,   -- LUT input (1-bit)
   I3 => I3,   -- LUT input (1-bit)
   I4 => I4,   -- LUT input (1-bit)
   I5 => I5    -- LUT input (1-bit)
);

-- End of LUT6_2_inst instantiation
```