ECE532S Digital Systems Design

Assignment 2 - FPGA Design Practices

Due: Feb 00, 2019

FPGAs contain many powerful primitives for implementing digital logic. To design effectively for FPGAs, it is important to understand how these primitives work and when to use them. In this assignment, you will perform some exercises to better understand some of these elements.

This assignment consists of three parts. You will submit a report containing tabulated data and a discussion for each part, as well as separate Verilog code and test bench files. Look carefully at the *submission* listing of each section to see what is requested.

To help with setting up the evaluation environment, a zip archive with supporting Vivado scripts is provided. Read the README file in the archive to understand how to use the files.

1 Random Access Memories (RAMs)

Modern FPGAs have several different types of storage primitives. In the Xilinx Artix-7 FPGAs (like the one inside the Nexys 4 boards), you can implement storage elements in Flip Flops, Distributed RAMs and Block RAMs. To perform the following exercises, you will need to refer to the Xilinx user guides for Synthesis [1], Memories [2] and CLBs [3].

1.1 Problem

You must construct 3-port RAMs, with two read and one write ports, conforming to the following Verilog interface:

```
module ram532
#(
  parameter C_ADDR_WIDTH = 8,
  parameter C_DATA_WIDTH = 32
) (
  input  clk,                       // clock
  input  wen,                       // write enable
  input  [C_DATA_WIDTH-1:0] wdata,   // write data
  input  [C_ADDR_WIDTH-1:0] waddr,   // write address
  output [C_DATA_WIDTH-1:0] rdata_0, // read port 0 data
  input  [C_ADDR_WIDTH-1:0] raddr_0, // read port 0 address
  output [C_DATA_WIDTH-1:0] rdata_1, // read port 1 data
  input  [C_ADDR_WIDTH-1:0] raddr_1  // read port 1 address
);
```

The write data port consists of enable, data and address inputs. When write enable is high, the data contained in the `wdata` wires is to be stored in the RAM at address `waddr` in the next clock cycle.

The read ports are always active, thus the RAM value at `raddr` is produced at `rdata` at the next cycle. You may implement any write mode. Note that the depth of a memory for a given address width is 2^{width}. The default configuration as shown in the code sample implements a 256x32-bit RAM.

1.2 Tasks

Implement two RAM configurations: 256x32-bit and 1024x32-bit using Registers, Distributed RAM and Block RAM (Hint: Look up the `ram_style` directive). You should have six implementations in total. For each implementation, record the reported resource utilization in Slice LUTs, Slice Registers, and Block RAMs. In your code, you should direct the Vivado Synthesis tool to *infer* the kinds of memories you need rather than instantiate them directly.

Submission

In your report include:

1. A table listing the resource utilization obtained for each implementation
2. A discussion comparing the different storage primitives. When would you want to use a specific primitive to implement a certain memory?

Submit your Verilog code as a separate file. You may submit one file and add a comment describing how to target different primitives.

2 DSP Units

As for storage, FPGA vendors have included special primitives for performing certain types of computation. Of particular importance is the DSP Unit, known as the DSP48E1 in the Xilinx 7-series FPGAs. The DSP unit contains several different computation blocks connected with configurable paths. Your task is to design a simple computation module that makes the most use of these paths. Once again, look at the examples in the Vivado Synthesis Guide [1] for reference. You should review the Xilinx DSP reference [4] and the synthesis guide [1] to complete the problems in this section.

2.1 Problem

Create a Verilog module that implements the equation: $f(a, b, c) = (a + b) * c$ with the following interface:

```
module preadd532
#(
  parameter C_WIDTH = 16
) (
  input  clk,     // clock
  input  in_vld,  // input valid
  input  signed [C_WIDTH-1: 0] in_a, // a value
  input  signed [C_WIDTH-1: 0] in_b, // b value
  input  signed [C_WIDTH-1: 0] in_c, // c value
  output signed [2*C_WIDTH: 0] out, // output value
  output out_vld  //output valid
);
```

All three inputs a, b and c are valid when `in_vld` is 1. The corresponding output value should be valid when `out_vld` is high. Note this design allows for arbitrary delay between input and output values.

2.2 Tasks

Implement the following four versions of the function:

1. `C_WIDTH=16` that maps onto the DSP48E1 unit, it should use no LUTs for logic
2. `C_WIDTH=16` that is forced to use LUTs only
3. `C_WIDTH=32` that targets DSP units, but may also spill into LUTs
4. `C_WIDTH=32` that is forced to use LUTs only

Create a test bench, simulate the design and ensure it works correctly. Record the utilization and Fmax for the implementation being sure to include the number of DSP48E1 units. Note that Vivado does not directly produce the Fmax number [5]. If you only have one level of flip flops or just the DSP block, there is no path between clocked flip flop elements. To simplify reporting, if your circuit behaves that way, you may compute your Fmax using the worst unconstrained path delay as the critical path. Look for the worst delay in the "Unconstrained Paths" part of the timing summary.

Submission
In your report include:

1. A table listing the resource utilization and Fmax obtained for each of your four implementations
2. A description of how you made your implementation pack into a single DSP48E1 block and what changed when the width was set to 32-bits
3. A discussion comparing the utilization and Fmax of LUT vs DSP implementations. Are there cases when you would prefer using a particular resource?

Submit your Verilog code and test bench as separate files.

3 Resource Trade-offs

When designing circuits targeting FPGAs, you can control the resources and performance very finely by using different algorithms. In this final section, you will explore using memory for computation.

3.1 Problem

You are to implement the function $f(x) = x^3$ in a Verilog module using the following function interface:

```
module cube532
(
  input  clk,
  input  [7:0] in,  // 8-bit unsigned input
  output [23:0] out // 24-bit unsigned output (in^3)
);
```

This module should produce a new output one cycle after the input is registered.

3.2 Tasks

Create a version of `cube532` that uses a Block RAM to implement the function. Considering that a function is a mapping from a domain to another, we can compute a function by using the input as the address to a read-only memory and producing the output as simply the value at the address. In the support files, `cube.mem` contains this mapping for an 8-bit cube function. Use these values to implement the module and verify that the module works correctly through simulation. Create a second version of `cube532` that performs the computation directly using multiplication and verify its correctness. Obtain utilization results for the two implementations.

Submission
In your report include:

1. A table listing the resource utilization obtained for each implementation
2. A discussion comparing the different techniques. When would it be beneficial to use a RAM LUT versus performing the computation? Is it possible to combine the two techniques?

Submit your Verilog files and testbench as separate files.

References

[1] *Vivado Design Suite User Guide: Synthesis*, `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug901-vivado-synthesis.pdf`.

[2] *7 Series FPGAs Memory Resources: User Guide*, `https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf`.

[3] *7 Series FPGAs Configurable Logic Block: User Guide*, `https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf`.

[4] *7 Series DSP48E1 Slice: User Guide*, `https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf`.

[5] *AR# 57304 Vivado Timing - Where can I find the Fmax of a clock signal in the timing report?* `https://www.xilinx.com/support/answers/57304.html`.