CMPE 260
Digital System Design II
Laboratory Exercise 5: Binary Multiplier with Built in Self Test
Revised: 06-30-2016

This exercise develops a built in self test (BIST) to confirm the functionality of a ip-core generated multiplier. The objective of this exercise is to understand how to generate components using Xilinx ISE, as well as how to implement a hardware tester of digital systems. A multiplier is generated, then a BIST is developed that contains a linear feedback shift register (LFSR) and a multiple input signature register (MISR).

Background

Digital systems can become large and complicated quickly. A BIST provides an easy way for designers to ensure that a design is functioning correctly. By precomputing the expected result, it is easy to check whether or not the systems behavior is appropriate. It can also be used during optimizations to verify that the outcome of a computation has not changed as a result of any changes.

Prelab Work

- Design a maximal length LFSR (internal or external) for the multiplier in use. Be sure to include a circuit diagram. Use the LFSR table PDF on myCourses to assist with this. Indicate the sizes of all buses.
- Design a MISR that will be used to generate the signature. The circuit diagram should also include a black box multiplier. Indicate the sizes of all buses.

Program Specification

The design must meet the following specifications and use all components listed at least once. Note that not all required components are listed explicitly. It will be necessary to determine what they are and what ports they will need to have. The unlisted components can either be placed in separate processes of the wrapper, or implemented as separate files for greater re-usability.

LFSR

The LFSR has the following features:

- Inputs (One bit unless noted):
  - `clk`: Clock. Components should be rising edge triggered.
  - `rst`: Resets the LFSR to a hard coded, unchanging value. This ensures that the output is the same. Active low, synchronous.

  - `enable`: Controls whether or not the LFSR should be generating random values.

- Outputs:

  - `bit_pattern` (8 bits): The pseudo random sequence to be used as multiplier inputs.

Notes

- Be sure to used the LFSR table PDF from myCourses to guarantee that the LFSR is maximal length, meaning that it will cycle through all possible outputs. This ensures better pseudo random numbers and lessens the chances that the BIST misses a flaw.

MISR

The MISR takes the output of the multiplier and turns it into a signature over several clock cycles.

- Inputs (One bit unless noted):

  - `clk`: Clock. Components should be rising edge triggered.
  - `rst`: Resets the MISR to zero. Active low, synchronous.
  - `enable`: Controls whether or not the MISR should be ignoring input from the multiplier.
  - `mult_result` (8 bits): The product calculated by the multiplier.

- Outputs:

  - `signature` (8 bits): The signature generated by the MISR.

Notes

- When designing the MISR, keep in mind that it should, in addition to the input XORs, have XORs that match the tap locations of the LFSR. This helps ensures that the correct signature can only be obtained via the correct input by introducing yet another source of randomness to the design.
- Because the generated multiplier is a pipelined device, the first result it returns after the LFSR has started generating pseudo random numbers is likely not the product of those numbers. To avoid this being an issue, the MISR should not start calculating the signature for a few clock cycles after `enable` is asserted, or the wrapper can delay asserting `enable`.

Debouncer

How debouncing is performed is implementation defined, however some examples are provided.

Listing 1: Potential debouncing methods

```
-- noise filter the async_input signal
-- signal must be true for 2 cycles
-- first ff to remove metastability
-- implemented as a shift register
temp_output_proc : process (clk)
begin
    if (clk'event and clk = '1') then
        if reset_n= '0' then
            temp_signal <= (others => '0');
        else
            temp_signal(0) <= async_input;
            temp_signal(1) <= temp_signal(0);
            temp_signal(2) <= temp_signal(1);
            temp_signal(3) <= temp_signal(2);
        end if;
    end if;
end process temp_output_proc;

-- debounced, synchronized signal
-- Will remain high as long as async_input is '1'
Conditioned_output <= '1' when
            temp_signal(3 downto 1) = "111" else '0';

-- Will only be high for one clock cycle if
-- async_input stays high
single_pulse <= not temp_signal(3) and
            temp_signal(2) and temp_signal(1);

-- temp_output will stay high until reset
output_start_proc : process (clk) begin
if (clk'event and clk = '1') then
    if reset_n = '0' then
        temp_output <= '0';
    else
        if single_pulse = '1' then
            temp_output <= '1';
        else
            temp_output <= temp_output;
        end if;
    end if;
end if;
end process;
output <= temp_output;
```

In Listing 1, `Conditioned_output`, `single_pulse`, and `output` could all be used as the debounced signal. It is necessary to determine which one would be appropriate for what scenario. For example, using `output` would work well when a button should be pressed and the signal from the button should not be deasserted, even after it is released.

Wrapper

This module will tie all of the components together into one, including the seven segment display.

- Inputs:
  - `A,B` (4 bits each): Factors to be multiplied.
  - `clk`
  - `rst`: Active low, synchronous reset. Should reset all components.
  - `test_mode`: Active high signal to enter test mode. Otherwise multiplication of `A` and `B` should be performed.
- Outputs:
  - Those necessary for the seven segment display.

Previous Code

The code developed for the Seven Segment Decoder will be needed in this lab to show either the signature or the product of the multiplier. The given code that the decoder relies on will also be needed. A dedicated multiplexer will also be needed, or can be implemented using VHDL statements like if then else, or cases.

Test Bench

Once again one test bench is required, however a second may be a good idea. See the previous exercise for details on the issues involved with testing the seven segment display.

Required Test Bench

This test bench should instantiate the wrapper component for complete testing of the BIST. Some scenarios to test(in various orders):

- Resetting the BIST.
- Multiplying values.
- Acquiring a signature with different values being multiplied. The different values should not cause the signature to change.

No assert statements are required.

Lab Procedure

1. Create a new directory and Xilinx ISE project for this exercise.
2. Copy the previously developed seven segment display code to the new folder.
3. Generate the multiplier.
   (a) Select "New Source", under the Project Menu.
   (b) Select IP (CORE Generator & Architecture Wizard)
   (c) Enter a name for your multiplier, then choose Next.
   (d) Generate a Multiplier, found under Math Functions/Multiplier, that has the following characteristics:
      i. Parallel multiply
      ii. Data type: Unsigned (for both inputs)
      iii. Width: 4-bits (for both inputs)
      iv. Use LUTs
   (e) Look in the ipcore_dir folder of your project to view the multiplier's vhd file. The entity block shows what inputs and outputs are named.
   (f) Verify the multiplier's functionality with the test bench from the ALU exercise. In case ModelSim throws a "Library XilinxCoreLib not found" error:
      i. In ModelSim, select New → Library. . . from the File Menu.
      ii. Select a map to an existing library,
      iii. Enter XilinxCoreLib as the library name, browse to `C:/Modeltech_6.3c/Xilinx_libs`, and select "XilinxCoreLib".
      iv. Click OK twice and re-run the simulation.
4. Write a properly commented and properly formatted VHDL program according to the preceding specifications.
5. Test the design using a behavioral simulation. Testings the display signals is not necessary.
6. Synthesize the design and retest it using a post route simulation. Once again, testing the display signals is not necessary.
7. Prepare the design for hardware, following the procedure laid out in Exercise One. Use Nexys_3_IO.pdf from myCourses to help plan the layout.
8. Program the board and verify that it is working.
9. Demonstrate steps 5, 6, and 8 to the instructor, who will sign the grading sheet.
10. Screen shot the appropriate waveforms for the report, making sure that they will be readable. It is not necessary to include the clock cycles while waiting for the signature generation, though the first and last few cycles must be shown. Put the outputs of the MISR and multiplier on the waveforms as well, since the seven segment signals do not have to be present.
11. Submit the source code and report online to the appropriate myCourses dropbox.

Lab Report

Write a report that meets the rules of professional technical writing and follows the lab report format on myCourses. Additional items that must be included:

- Prelab designs, in computer drawn form.

- Area used, which is reported via number of occupied slices, FFs used, and LUTs used.
- Timing results (best case achievable). No code modification should be required since a clock was used.
- Marked simulation results. This does not include any of the multiplier only test benches. They are only used to ensure that issues are caused by the BIST, not the multiplier. While this may seem counterintuitive, the purpose of this exercise is to write a BIST, so it is not likely to be correct the first time.