ECE 440 - Project #4

Collin Heist

13th February 2020

Contents

Figures

Listings

1 Design

There was really no design consideration necessary for this project. The general circuit was provided to us, and the only interesting component was the instantiation and creation of the memory unit.

For the memory unit, I utilized the distributed memory generator with a 4-bit address, and a two-bit input and output bus. However, because the address inputs on the larger circuit diagram is only 2-bit. This tripped me up at first, as my simulation was showing indeterminate values (of course) and I didn't realize why. This was resolved by wiring the address input with two zeros appended. The isolated register was simple enough, and was within a synchronized block that updated the output signal with the results of the memory unit. The rest of the circuit implementation was as expected.

Initializing the memory was done using a **.coe** file, which is shown in **Listing 3.3**.

2 Simulations

2.1 Behavioral Simulation

The results of my behavioral and post-synthesis simulations are exactly identical. The behavioral simulation has the benefit of showing internal signals, which I've partitioned into inputs, those of the memory module, and the output signals.

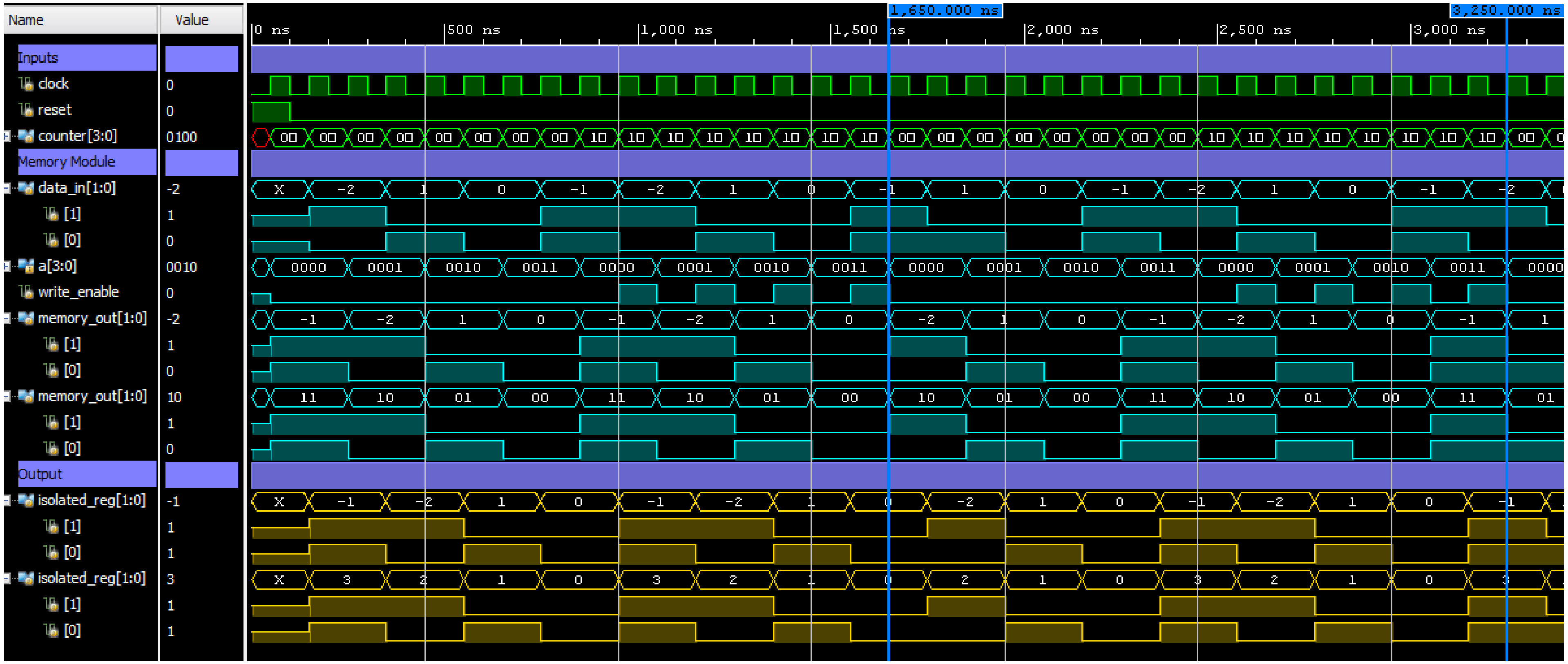

Figure 2.1: First Section of the Behavioral Simulation.

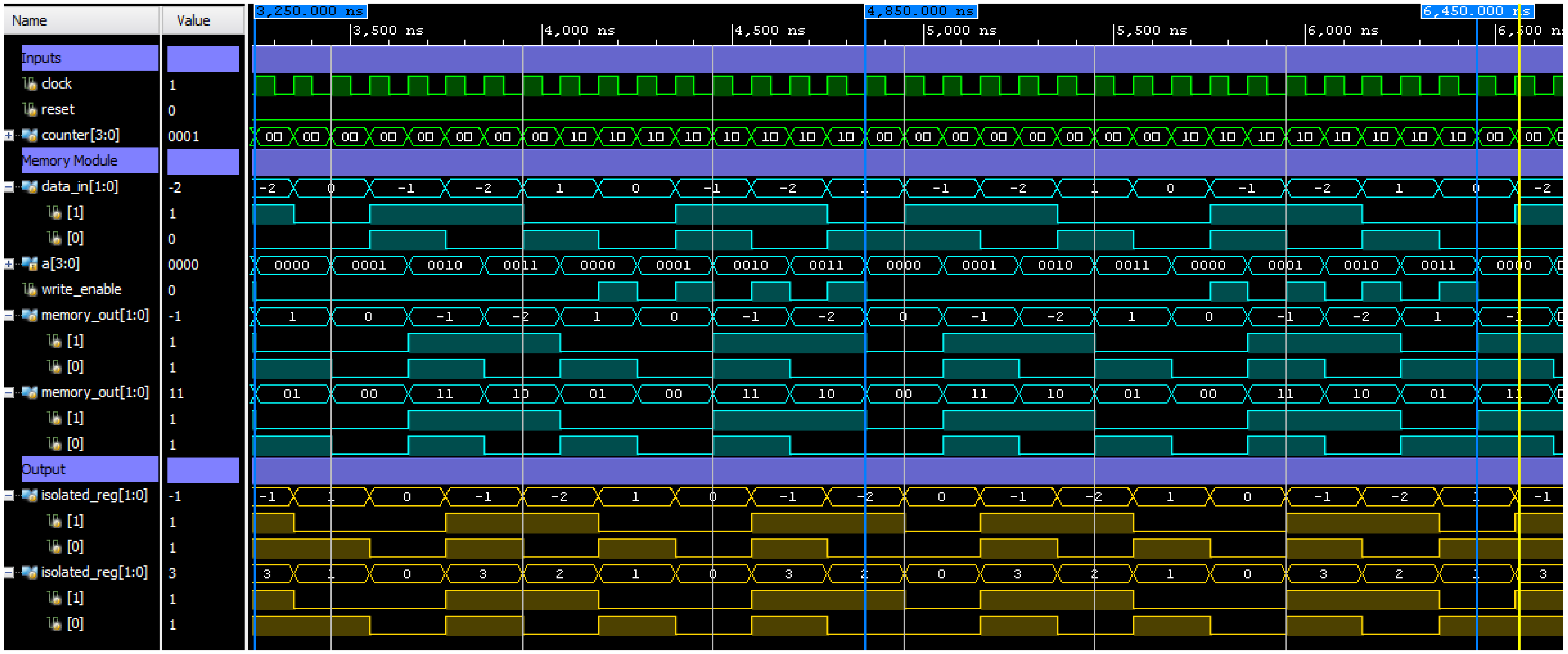

Figure 2.2: Second Section of the Behavioral Simulation.

2.2 Post-Synthesis Timing Simulation

Showing only the primary inputs and outputs, the identical output data patterns can be seen. Memory accesses occur in an identical pattern, as the counter is initialized to 0, incremented by the clock, and thus is not variable in any way.

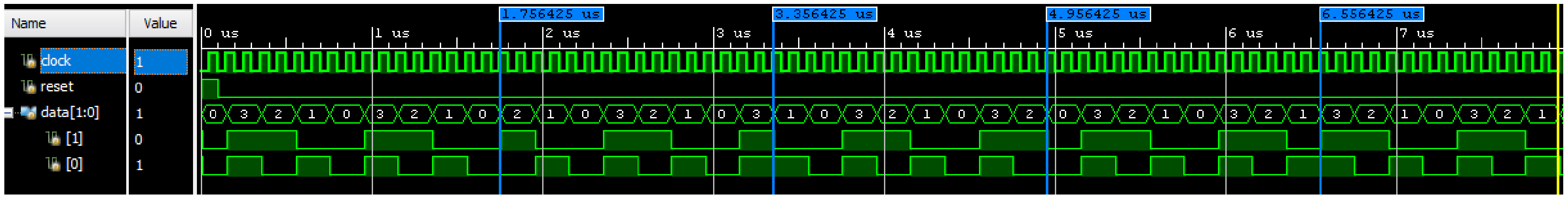

Figure 2.3: Post-Synthesis Timing Simulation.

3 Source Code

Listing 3.1: *Weird Circuit* Module

```
`timescale 1ns / 1ps

module weird_circuit(
  input logic clock,
  input logic reset,
  output logic [1:0] data
);

// Internal Signals
logic write_enable;
logic [3:0] counter;
logic [1:0] memory_out, isolated_reg, data_in;

// Implement subtractor, and write-enable
assign data_in = (isolated_reg - 2'b01);
assign write_enable = (counter[3] & counter[0]);
assign data = isolated_reg;

// Instantiate Memory Unit
dist_mem_gen_0 memory(
  .a({2'b00, counter[2:1]}),
  .d(data_in),
  .clk(clock),
  .we(write_enable),
  .spo(memory_out)
);

always_ff @(posedge clock) begin
  if (reset) begin
    counter <= 4'b0000;
    end
  else begin
    counter <= counter + 4'b0001;
    isolated_reg <= memory_out;
    end
end

endmodule
```

Listing 3.2: Testbench

```
`timescale 1ns / 1ps

```

```
module testbench();

// Global Parameters
parameter CLK_PRD = 100;
parameter HOLD_TIME = (CLK_PRD * 0.3);
parameter MAX_SIM_TIME = (16 * 5 * CLK_PRD);

// Internal logic signals
logic clock, reset;
logic [1:0] data;

// Instantiate the GCD core as a UUT
weird_circuit dut(.*);

// Prevent simulating longer than MAX_SIM_TIME
initial #(MAX_SIM_TIME) $finish;

// Generate Clock Signal
initial begin
  clock <= 0;
  forever #(CLK_PRD / 2) clock = ~clock;
end

// Main Simulation
initial begin
  reset = 1;

  // Global Reset
  #100; reset = 0;

  @(posedge clock); #HOLD_TIME;  // Align with clock

  forever begin
    @(posedge clock);
    $display("%b", data);
  end
end

endmodule
```

Listing 3.3: Memory Initialization File

```
memory_initialization_radix = 2;
memory_initialization_vector = 11 10 01 00;
```