EE 371 Autumn 2016 - Lab 1

William Li, Jun Park, Dawn Liang

October 16, 2016

1 Abstract

2 Introduction

This lab focuses on designing and building VHDL (Verilog Hardware Description Language) programs. We built four different types of counters (a four stage ripple up counter using gate modeling, a four stage synchronous up counter using both dataflow model and schematic entry, and a four stage synchronous Johnson up counter using the behavioural model). To build these counters, we were introduced to both Icarus Verilog (iVerilog) and GTKWave analysis. We designed the counters using iVerilog and tested the output waveforms using GTKWave analysis. We then loaded our designs onto an Altera FPGA where we then did further testing using Signal Tap II, a logic analyzer for detecting the output waveform on the FPGA. Finally, we began a brief introduction into the C programming language. We learned the basics of a C program through the CodeBlocks IDE by compiling the Project0.c file. Finally, we built a simple C car calculator program that asks for relevant input data and outputs an approximate list price for a brand new vehicle.

3 Designing and building VHDL applications - counters

```
module DFlipFlop(q, qBar, D, clk, rst);
   input D, clk, rst;
   output q, qBar;
   reg q;

   not n1 (qBar, q);
   always@ (negedge rst or posedge clk) begin
      if(!rst)
         q = 0;
      else
         q = D;
   end
endmodule
```

3.1 Ripple up counter

```
/*
   4-bit ripple-up counter, implemented using D-flipflops

   Authors: William Li, Dawn Liang, Jun Park
   Date: 16 Oct 2016
```

```
*/

module rippleUpCounter(out, clk, rst);
   // declare inputs/outputs
   output logic [3:0] out;
   input logic clk, rst;
   logic Q0, Q1, Q2, Q3, clkTemp1, clkTemp2, clkTemp3, clkTemp4;

   // connect flip flops; one ff per bit
   DFlipFlop d0(.q(Q0), .qBar(clkTemp1), .D(clkTemp1), .clk(clk),
       .rst(rst));
   DFlipFlop d1(.q(Q1), .qBar(clkTemp2), .D(clkTemp2), .clk(clkTemp1),
       .rst(rst));
   DFlipFlop d2(.q(Q2), .qBar(clkTemp3), .D(clkTemp3), .clk(clkTemp2),
       .rst(rst));
   DFlipFlop d3(.q(Q3), .qBar(clkTemp4), .D(clkTemp4), .clk(clkTemp3),
       .rst(rst));

   // output logic
   assign out = {Q3, Q2, Q1, Q0}; //not {Q0, Q1, Q2, Q3};
endmodule

// provided D-ff module
module DFlipFlop(q, qBar, D, clk, rst);
   input D, clk, rst;
   output q, qBar;
   reg q;

   not n1 (qBar, q);
   always@ (negedge rst or posedge clk) begin
      if(!rst)
         q = 0;
      else
         q = D;
   end
endmodule
```

```
/*
   Ripple-up counter tester

   Authors: William Li, Dawn Liang, Jun Park
   Date: 16 Oct 2016
*/

`include "rippleUpCounter.v"
module rippleUpCounter_testbench;
   logic [3:0] out;
   logic clk, rst;
```

```
   rippleUpCounter dut(.out, .clk, .rst);

   parameter PERIOD = 100; // period = length of clock
   initial begin
      clk <= 0;
      forever #(PERIOD/2) clk = ~clk;
   end

   initial begin
      rst=0; @(posedge clk);
      rst=1; @(posedge clk);
   end
endmodule
```

3.2 Synchronous up counter

```
module synUpCounter(out, clk, rst);
   output logic [3:0] out;
   input logic clk, rst;

   logic Q0, Q1, Q2, Q3;
   logic D0, D1, D2, D3;

   DFlipFlop dff0(.q(Q0), .qBar(), .D(D0), .clk(clk), .rst(rst));
   DFlipFlop dff1(.q(Q1), .qBar(), .D(D1), .clk(clk), .rst(rst));
   DFlipFlop dff2(.q(Q2), .qBar(), .D(D2), .clk(clk), .rst(rst));
   DFlipFlop dff3(.q(Q3), .qBar(), .D(D3), .clk(clk), .rst(rst));

   assign D0 = (~Q0)&rst;
   assign D1 = ((Q0&~Q1) | (Q1&~Q0))&rst;
   assign D2 = ((Q2&~Q1) | (Q2&~Q0) | (Q1&Q0&~Q2))&rst;
   assign D3 = ((Q3&~Q2) | (Q3&~Q1) | (Q3&~Q0) | (~Q3&Q2&Q1&Q0))&rst;

   assign out = {Q3, Q2, Q1, Q0};

endmodule

module DFlipFlop(q, qBar, D, clk, rst);
   input D, clk, rst;
   output q, qBar;
   reg q;

   not n1 (qBar, q);
   always@ (negedge rst or posedge clk) begin
      if(!rst)
         q = 0;
      else
         q = D;
```

```
  end
endmodule
```

```
module synUpCounter_testbench;
  logic [3:0] out;
  logic clk, rst;

  synUpCounter dut(.out, .clk, .rst);

  parameter PERIOD = 100; // period = length of clock
  initial begin
    clk <= 0;
    forever #(PERIOD/2) clk = ~clk;
  end

  initial begin
    rst=0; @(posedge clk);
    rst=1; @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
           @(posedge clk);
```

```
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
      rst=0; @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
      $stop();
   end
endmodule
```

3.3 Johnson up counter

```
module johnsonUpCounter(out, clk, rst);
   output logic [3:0] out;
   input logic clk, rst;
   logic [3:0] temp;

   always_ff@(negedge rst or posedge clk) begin
      if (rst == 0)
```

```
      begin
         temp <= 4'b0000;
      end
      else if (clk == 1'b1)
      begin
         temp <= {~temp[0], temp[3:1]}; // right shift and negate most
             signf bit
      end
   end

   assign out = temp;
endmodule

module DFlipFlop(q, qBar, D, clk, rst);
   input D, clk, rst;
   output q, qBar;
   reg q;

   not n1 (qBar, q);
   always@ (negedge rst or posedge clk) begin
      if(!rst)
         q = 0;
      else
         q = D;
   end
endmodule
```

```
module johnsonUpCounter_testbench;
   logic [3:0] out;
   logic clk, rst;

   johnsonUpCounter dut(.out, .clk, .rst);

   parameter PERIOD = 100; // period = length of clock
   initial begin
      clk <= 0;
      forever #(PERIOD/2) clk = ~clk;
   end

   initial begin
      rst=0; @(posedge clk);
      rst=1; @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
             @(posedge clk);
```

```
            @(posedge clk);
            @(posedge clk);
            @(posedge clk);
            @(posedge clk);
            @(posedge clk);
            @(posedge clk);
            @(posedge clk);
      rst=0; @(posedge clk);
            @(posedge clk);
            @(posedge clk);
      $stop();
    end
endmodule
```

3.4 Synchronous up counter (schematic entry)

4 Learning the C language

```
#include <stdio.h>
int main(void)
{
    // declare and initialise variables
    // inputs
    float man_cost, mark_up, sales_tax, discount;

    // outputs
    float list_price = 0.0;

    printf("Car list price calculator\n\n");

    printf("Enter the manufacturer's cost ($): ");
    scanf("%f", &man_cost);

    printf("Enter the estimated markup (%%): ");
    scanf("%f", &mark_up);

    printf("Enter the sales tax (%%): ");
    scanf("%f", &sales_tax);

    printf("Enter the dealer's pre-tax discount (%%): ");
    scanf("%f", &discount);

    printf("Manufacturer's cost = $%.2lf\n", man_cost);
    printf("Est. dealer's markup = %.2lf%%\n", mark_up);
    printf("Pre-tax discount  = %.2lf%%\n", discount);
    printf("Sales tax         = %.2lf%%\n", sales_tax);
    printf("----------------------\n");
```

```
    list_price = man_cost * (1 + mark_up / 100) * (1 - discount/100) *
        (1 + sales_tax/100);

    printf("Your car will cost %3.2f\n", list_price);

    return 0;
}
```

---

5 Failure Modes Analysis