EE 371 Autumn 2016 - Lab 1

William Li, Jun Park, Dawn Liang

October 16, 2016

Contents

1 ABSTRACT

This lab focuses on introducing us to the tools and methods of digital design. We were introduced to the various levels of abstraction in modeling and implementation, the tools involved in designing and testing hardware applications, as well as the overall design process for hardware and software and applications.

2 INTRODUCTION

First, we built four different types of counters using different modeling techniques: a 4-bit ripple up counter using gate modeling, a 4-bit synchronous up counter using both dataflow model and schematic entry, and a 4-bit synchronous Johnson up counter using the behavioural model. In the process of building and testing these counters, we were introduced to Icarus Verilog (iVerilog) and GTKWave software, for compiling our designs and simulating waveforms. We then loaded our designs onto an Altera Cyclone V FPGA, where we verified their functionality on hardware. Then we utilised Signal Tap II, a logic analyzer for probing designs in hardware. Finally, we were briefly introduced to the C programming language. We learned the basics of a C program in the CodeBlocks IDE by compiling a provided C project, and then we built a simple C car price calculator program that asks for relevant input data and outputs an approximate list price for a brand new vehicle.

3 DISCUSSION

3.1 Design

Design Specification We built four 4-bit counters: a ripple-up counter, two synchronous up counters, and a Johnson counter. Each counter had an active-low reset.
Additionally, we built a car list price calculator. Given the appropriate input data, the calculator would compute the estimated list price of a brand new vehicle.

Design Procedure In Verilog HDL, we designed and implemented four counters using the four different levels of modeling. The first three counters were implemented using the D-flipflop provided in the lab spec; the fourth counter was implemented via Quartus' schematic entry feature.

```
/*
   provided D-flipflop code
*/
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
```

The ripple-up counter was implemented at the structural level. The connections between the D-flipflops were explicitly assigned based on the gate level diagram.

The first synchronous up counter was implemented at the data flow level. We diagrammed the states and generated boolean expressions using truth tables and K-maps.

The johnson counter was implemented at the behavioural level.

The second synchronous up counter was built via schematic entry.
In this subsection you will first describe how you developed your the hardware and software for your design. You might want to include Verilog or C code fragments to illustrate specific points, the theorems you used to simplify logic expressions, the K-map(s) or the truth tables you used to simplify or verify your conclusion if that these are tools you used.

System Description The four counters take a clock and reset as the input, and output 4-bits counting up in binary. The fourth counter also takes an enable input, determining whether the counter is on/off.

The list price calculator takes the manufacturer's cost ($), the estimated markup (%), the pre-tax discount (%), and sales tax (%) as inputs. It outputs the estimated list price of a vehicle, calculated from those values.

Software Implementation What is your design????
Present your design starting from a top level functional view and potentially block diagram or high level architecture. Refine that view to present and explain each of the modules that comprise the major functional blocks. Discuss the flow of control through the design. Identify and discuss the specific processes/-tasks you have implemented in your design. Explain your design choices.

Hardware Implementation See software implementation. Now, draw the block diagrams of how you connected interconnected the major / minor modules in your system.
Include your logic equations and diagrams as well.

3.2 Test

Test plan

Test Specification

Test Cases

4 RESULTS

5 ANALYSIS OF ERRORS

6 SUMMARY AND CONCLUSION

7 APPENDIX

7.1 Designing and building VHDL applications - counters

7.1.1 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
```

```
/*
   Ripple-up counter tester

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

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

   // module declaration
   rippleUpCounter dut(.out, .clk, .rst);

   // test module
   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
```

7.1.2 Synchronous up counter

```
/*
   4-bit synchronous up counter, implemented using D-flipflops at
   dataflow level

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

module synUpCounter(out, clk, rst);
   // declare inputs/outputs
   output logic [3:0] out;
   input logic clk, rst;

   // connecting wires
   logic Q0, Q1, Q2, Q3;
   logic D0, D1, D2, D3;

   // D-ff connections
   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));

   // output logic assignments
   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
```

```
/*
   4-bit synchronous up counter tester

   Authors: William Li, Dawn Liang, Jun Park
   Date: 16 Oct 2016
*/
module synUpCounter_testbench;
   // declare variables
   logic [3:0] out;
   logic clk, rst;

   // declare module
   synUpCounter dut(.out, .clk, .rst);

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

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

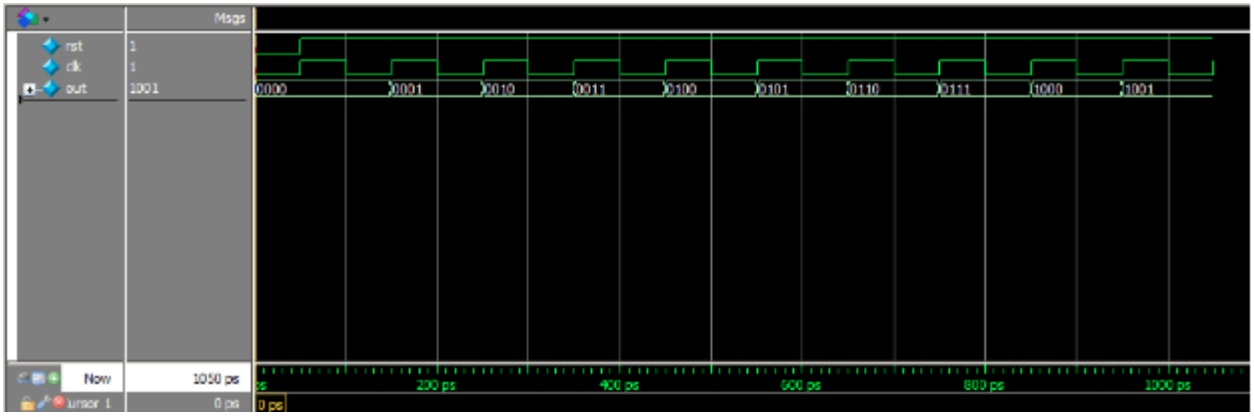

Figure 1: Ripple-up counter waveform in gtkwave

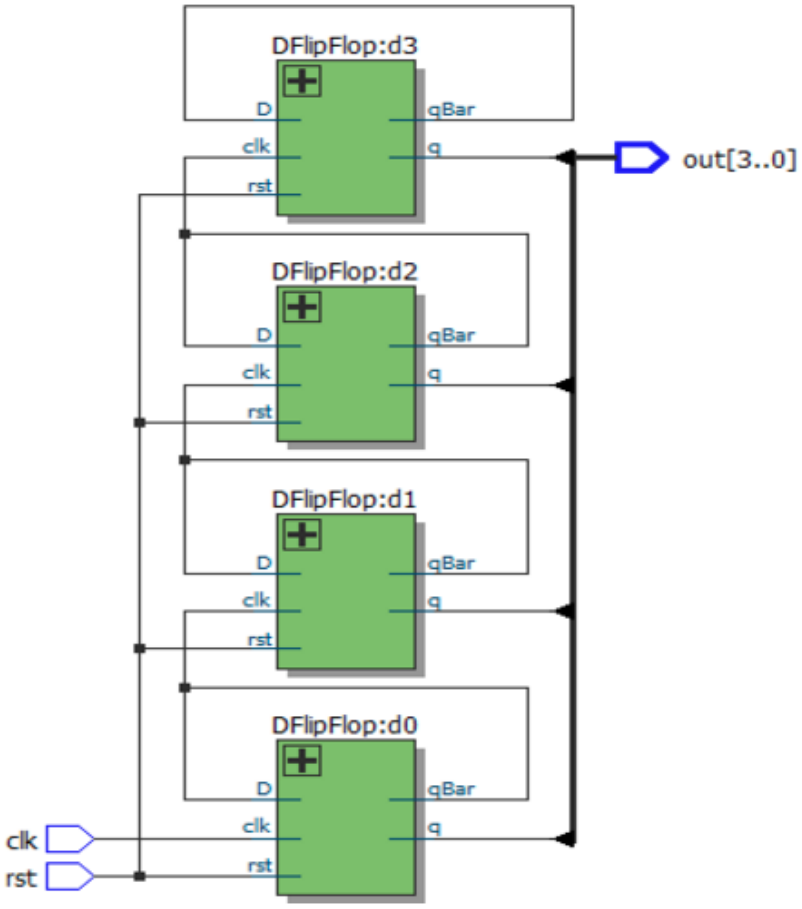

Figure 2: Ripple-up counter RTL view

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

7.1.3 Johnson up counter

```
/*
   4-bit Johnson up counter, implemeneted using D-flipflops at the
   behavioural level

   Authors: William Li, Dawn Liang, Jun Park
   Date: 16 Oct 2016
*/
module johnsonUpCounter(out, clk, rst);
   // declare inputs/outputs & variables
   output logic [3:0] out;
   input logic clk, rst;
   logic [3:0] temp;

   // counting every clock edge
   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

   // output assignment
   assign out = temp;
endmodule
```

```
/*
   Testing module for johnson up counter

   Author: Jun Park
   Date: 16 Oct 2016
*/
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);
```

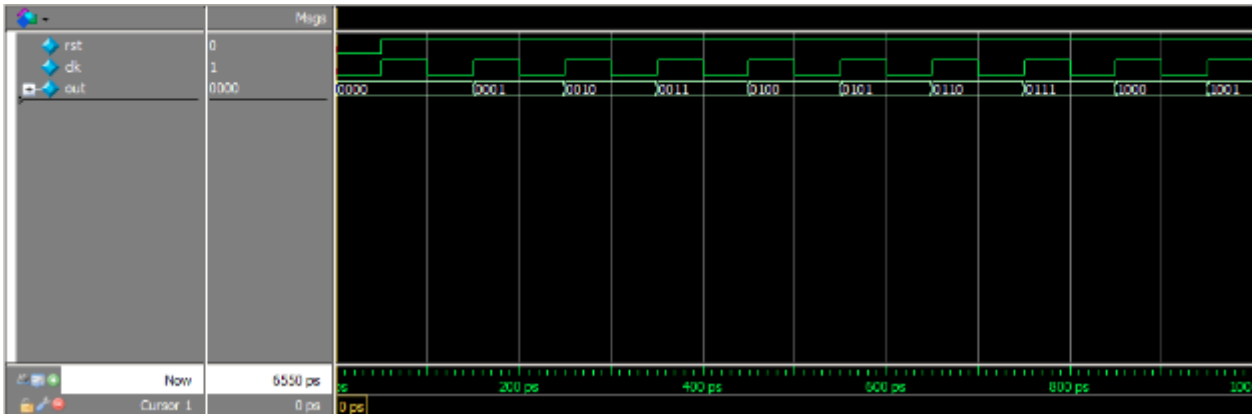

Figure 3: Ripple-up counter waveform in gtkwave

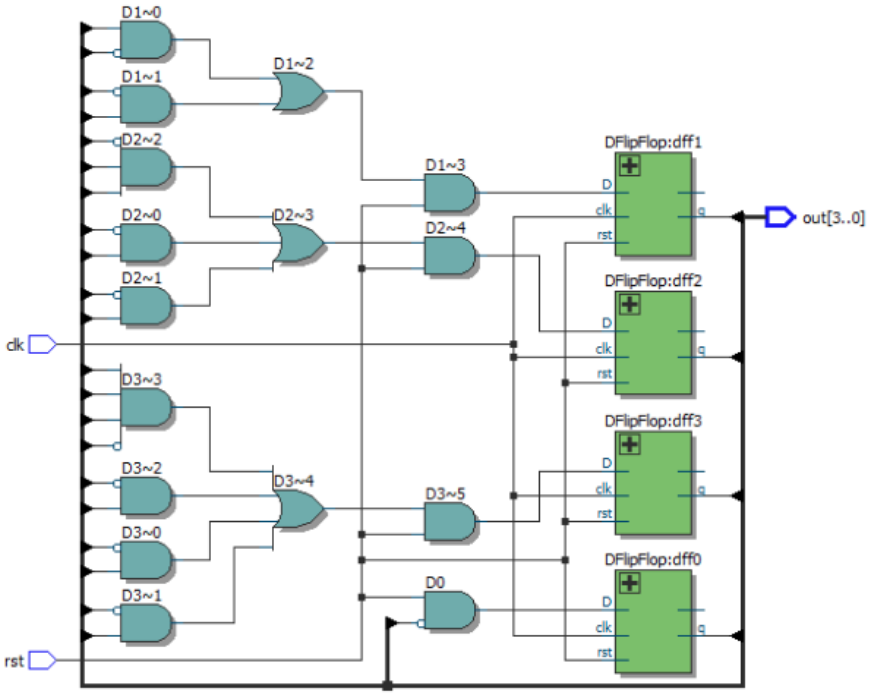

Figure 4: Ripple-up counter RTL view

```
            @(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
```

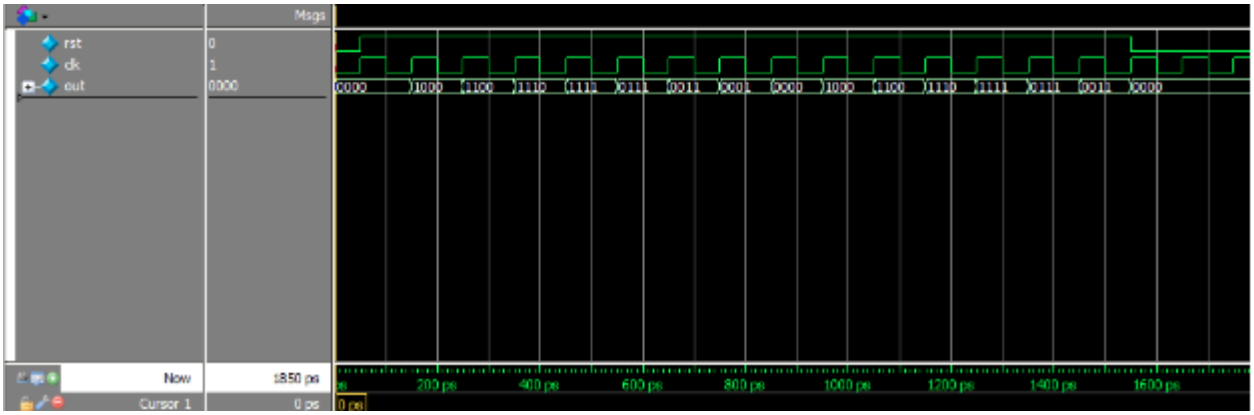

Figure 5: Ripple-up counter waveform in gtkwave

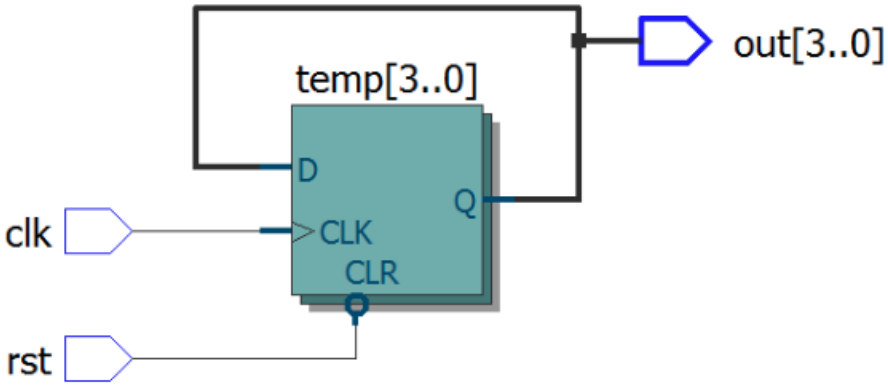

Figure 6: Ripple-up counter RTL view

7.1.4 Synchronous up counter (schematic entry)

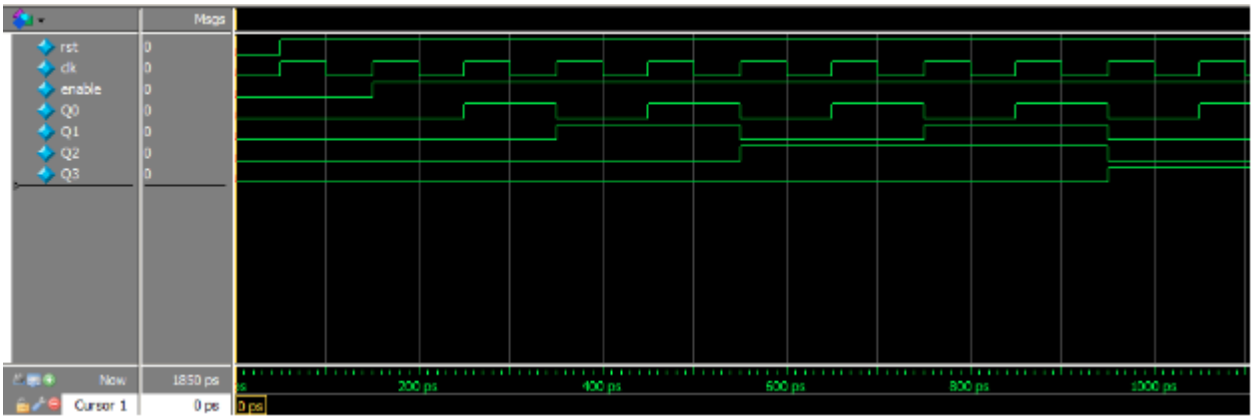

Figure 7: Ripple-up counter waveform in gtkwave

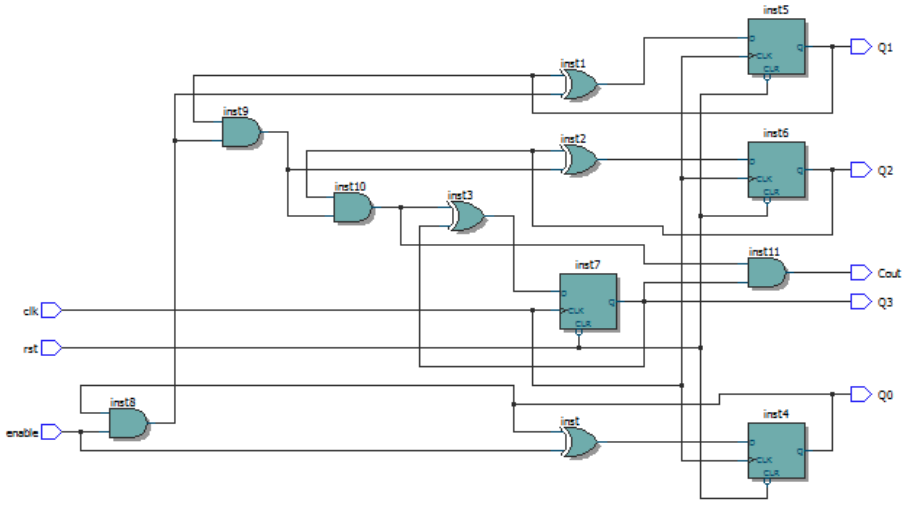

Figure 8: Ripple-up counter RTL view

7.2 Learning the C language

```c
#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;
}
```

8 FAILURE MODES ANALYSIS