Baylor University

Department of

Electrical and Computer Engineering

ELC 5311 (graduates), 4396(undergrad)

Advanced Digital Logic Laboratories

Keith Schubert
Associate Professor
Department of Electrical and Computer Engineering
Baylor University

Contents

Lab 1

Basys 2 Programming

1.1 Passthrough

We are going to begin with a simple project to turn on LEDs when the switch under them is on. There are eight switches (called sw⟨7⟩ ...sw⟨0⟩), and eight LEDs (called Led⟨7⟩ ...Led⟨0⟩). Our first easy part will be to assign the LEDs to be identical to the switches, see Code 1.1.

Listing 1.1: Verilog code for pass-through

```
`timescale 1ns / 1ps
module pass_through_simple(
    input [7:0] sw,
    output [7:0] Led
    );

assign Led=sw;

endmodule
```

This code simply passes the switches through to the LEDs, but it demonstrates the **assign** statement, which is one of our basic ways of designing combinational circuits. The other important thing is to have a user constraints file (UCF). An example is below.

Listing 1.2: User Constraints File

```
# This file is a general .ucf for Basys2 rev C board
# To use it in a project:
# - remove or comment the lines corresponding to unused pins
# - rename the used signals according to the project

# clock pin for Basys2 Board
NET "mclk" LOC = "B8"; # Bank = 0, Signal name = MCLK
#NET "uclk" LOC = "M6"; # Bank = 2, Signal name = UCLK
NET "mclk" CLOCK_DEDICATED_ROUTE = FALSE;
#NET "uclk" CLOCK_DEDICATED_ROUTE = FALSE;

# Pin assignment for EppCtl
# Connected to Basys2 onBoard USB controller
#NET "EppAstb" LOC = "F2"; # Bank = 3
```

```
#NET "EppDstb" LOC = "F1"; # Bank = 3
#NET "EppWR"        LOC = "C2"; # Bank = 3

#NET "EppWait" LOC = "D2"; # Bank = 3


#NET "EppDB<0>" LOC = "N2"; # Bank = 2
#NET "EppDB<1>" LOC = "M2"; # Bank = 2
#NET "EppDB<2>" LOC = "M1"; # Bank = 3
#NET "EppDB<3>" LOC = "L1"; # Bank = 3
#NET "EppDB<4>" LOC = "L2"; # Bank = 3
#NET "EppDB<5>" LOC = "H2"; # Bank = 3
#NET "EppDB<6>" LOC = "H1"; # Bank = 3
#NET "EppDB<7>" LOC = "H3"; # Bank = 3


# Pin assignment for DispCtl
# Connected to Basys2 onBoard 7seg display
NET "seg<0>" LOC = "L14"; # Bank = 1, Signal name = CA
NET "seg<1>" LOC = "H12"; # Bank = 1, Signal name = CB
NET "seg<2>" LOC = "N14"; # Bank = 1, Signal name = CC
NET "seg<3>" LOC = "N11"; # Bank = 2, Signal name = CD
NET "seg<4>" LOC = "P12"; # Bank = 2, Signal name = CE
NET "seg<5>" LOC = "L13"; # Bank = 1, Signal name = CF
NET "seg<6>" LOC = "M12"; # Bank = 1, Signal name = CG
NET "dp" LOC = "N13"; # Bank = 1, Signal name = DP

NET "an<3>" LOC = "K14"; # Bank = 1, Signal name = AN3
NET "an<2>" LOC = "M13"; # Bank = 1, Signal name = AN2
NET "an<1>" LOC = "J12"; # Bank = 1, Signal name = AN1
NET "an<0>" LOC = "F12"; # Bank = 1, Signal name = AN0

# Pin assignment for LEDs
NET "Led<7>" LOC = "G1"  ; # Bank = 3, Signal name = LD7
NET "Led<6>" LOC = "P4"  ; # Bank = 2, Signal name = LD6
NET "Led<5>" LOC = "N4"  ;  # Bank = 2, Signal name = LD5
NET "Led<4>" LOC = "N5"  ;  # Bank = 2, Signal name = LD4
NET "Led<3>" LOC = "P6"  ; # Bank = 2, Signal name = LD3
NET "Led<2>" LOC = "P7"  ; # Bank = 3, Signal name = LD2
NET "Led<1>" LOC = "M11"  ; # Bank = 2, Signal name = LD1
NET "Led<0>" LOC = "M5"  ;  # Bank = 2, Signal name = LD0

# Pin assignment for SWs
NET "sw<7>" LOC = "N3";  # Bank = 2, Signal name = SW7
NET "sw<6>" LOC = "E2";  # Bank = 3, Signal name = SW6
NET "sw<5>" LOC = "F3";  # Bank = 3, Signal name = SW5
NET "sw<4>" LOC = "G3";  # Bank = 3, Signal name = SW4
NET "sw<3>" LOC = "B4";  # Bank = 3, Signal name = SW3
NET "sw<2>" LOC = "K3";  # Bank = 3, Signal name = SW2
NET "sw<1>" LOC = "L3";  # Bank = 3, Signal name = SW1
NET "sw<0>" LOC = "P11";  # Bank = 2, Signal name = SW0
```

```
#NET "btn<3>" LOC = "A7";  # Bank = 1, Signal name = BTN3
#NET "btn<2>" LOC = "M4";  # Bank = 0, Signal name = BTN2
#NET "btn<1>" LOC = "C11"; # Bank = 2, Signal name = BTN1
NET "btn" LOC = "G12"; # Bank = 0, Signal name = BTN0

# Loop back/demo signals
# Pin assignment for PS2
#NET "PS2C"    LOC = "B1"   | DRIVE = 2  | PULLUP ; # Bank = 3, Signal name = PS2C
#NET "PS2D"    LOC = "C3"   | DRIVE = 2  | PULLUP ; # Bank = 3, Signal name = PS2D

# Pin assignment for VGA
#NET "HSYNC"   LOC = "J14"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = HSYNC
#NET "VSYNC"   LOC = "K13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = VSYNC

#NET "OutRed<2>"  LOC = "F13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED2
#NET "OutRed<1>"  LOC = "D13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED1
#NET "OutRed<0>"  LOC = "C14"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED0
#NET "OutGreen<2>"  LOC = "G14"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN2
#NET "OutGreen<1>"  LOC = "G13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN1
#NET "OutGreen<0>"  LOC = "F14"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN0
#NET "OutBlue<2>"  LOC = "J13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = BLU2
#NET "OutBlue<1>"  LOC = "H13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = BLU1

# Loop Back only tested signals
#NET "PIO<72>" LOC = "B2"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JA1
#NET "PIO<73>" LOC = "A3"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JA2
#NET "PIO<74>" LOC = "J3"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JA3
#NET "PIO<75>" LOC = "B5"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JA4

#NET "PIO<76>" LOC = "C6"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JB1
#NET "PIO<77>" LOC = "B6"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JB2
#NET "PIO<78>" LOC = "C5"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JB3
#NET "PIO<79>" LOC = "B7"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JB4

#NET "PIO<80>" LOC = "A9"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JC1
#NET "PIO<81>" LOC = "B9"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JC2
#NET "PIO<82>" LOC = "A10" | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JC3
#NET "PIO<83>" LOC = "C9"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JC4

#NET "PIO<84>" LOC = "C12"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JD1
#NET "PIO<85>" LOC = "A13"  | DRIVE = 2  | PULLUP ; # Bank = 2, Signal name = JD2
#NET "PIO<86>" LOC = "C13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = JD3
#NET "PIO<87>" LOC = "D12"  | DRIVE = 2  | PULLUP ; # Bank = 2, Signal name = JD4
```

1.2 Making a Counter for the Seven Segment Display

Now we want to drive the seven segment display. All the cathodes of the LEDs in the same position are hooked together. So for example, all four of the top LEDs' cathodes are hooked together, as are all the cathodes of the upper right, and so on. The top LED is called A or seg⟨0⟩, the letters and numbers proceed clockwise, and the middle segment is G or seg⟨6⟩. Each digit also has a decimal point, which is called dp, and their cathodes are also connected. The FPGA does sink the current in this case (*Why can it sink what it couldn't source?*), so a zero turns the LED on if the anode was also sourced.

To drive the seven segment display we will then need to convert from a 4 bit binary number to the seven segment display cathode pattern. We will leave the decimal point off. This can be seen in Code 1.3.

Listing 1.3: Verilog code for converting 4 bit binary to sseg.

```
`timescale 1ns / 1ps
module sseg_driver(
    input [3:0] num,
    output reg [0:6] sseg,
    output dp
    );
        assign dp = 1;

        always@* begin
                case(num)
                0:        sseg <= 7'b0000001;
                1:        sseg <= 7'b1001111;
                2:        sseg <= 7'b0010010;
                3:        sseg <= 7'b0000110;
                4:        sseg <= 7'b1001100;
                5:        sseg <= 7'b0100100;
                6:        sseg <= 7'b0000010;
                7:        sseg <= 7'b0001111;
                8:        sseg <= 7'b0000000;
                9:        sseg <= 7'b0001100;
                10:       sseg <= 7'b0001000;
                11:       sseg <= 7'b1100000;
                12:       sseg <= 7'b0110001;
                13:       sseg <= 7'b1000010;
                14:       sseg <= 7'b0110000;
                default:  sseg <= 7'b0111000;
                endcase

        end
endmodule
```

All the anodes for a digit are hooked together, so since there are four digits there are four anode lines (called an⟨3⟩ . . . an⟨0⟩). The output of the fpga is not strong enough to supply all the current for the LEDs, so a pnp transistor is used to switch power (*Why does a pnp switch power better, and an npn switch ground better?*). This means a zero, is needed to turn the pnp on and a 1 will turn it off.

This slicing allows control of 32 LEDs with 12 wires and one fourth can be on simultaneously. Since each LED will be off three fourths of the time, we need to switch them faster than the eye can see, and since the diode's pn junction holds charge for a bit (discharges like a capacitor, why?) and thus continues to glow, and the eye continues to see light for a bit. The Basys 2 board has a 50MHz clock, so we need to count to about a million, which is 20 bits. We thus need a counter that can count to whatever number fo bits we want.

Listing 1.4: Verilog code for counting a parameterized number of bits.

```
`timescale 1ns / 1ps
```

```
module counter#(
        parameter BITS=20
        )(
    input clk,
        input reset,
    output reg [BITS-1:0] count
    );
                initial
                        count <=0;

                always @(posedge(clk), posedge(reset))
                        if(reset)
                                count <= 0;
                        else
                                count <= count+1;

endmodule
```

To show this off, we will have a slow count displaying, with say a count around a second or so. This is about 50, which is close enough to 6 bits for our use. We will thus first make a 26 bit counter, triggering the rotation of the anodes off bits 18 and 19, and the next count off bit 25. We will thus make one more counter, this time with 16 bits so we can drive all four segments. We just need to change our top level module and ucf to reflect this.

Lab 2

Barrel Shifter

Do Experiment 3.11.1 Multifunction Barrel Shifter from the book.

Lab 3

Stack

Do Experiment 4.7.7 Stack from the Book.

Lab 4

PWM Dimmer

4.1 Pulse Width Modulation

Pulse width modulation is a direct way to drive analog devices with a digital output. Since it does not require any additional equipment, it is very popular in a number of applications. The basic idea is similar to a bang-bang control[1], but unlike typical bang-bang controls the cycle time is set very short so the output is effectively a weighted averaged, since most systems function like a low pass filter. The weighting is due to the duty cycle, which is the fraction of the cycle time that the output is on/high. If the output power of the digital line is P and the duty cycle is d, then the effective power is $P_e = dP$.

Pulse width modulation is very effective in simple servos, LEDs, and other simple devices that need to be driven by a digital controller. They are particularly common in hobby applications, but also find use in telecommunications, monitors, fans, and pumps. You can time them off the leading edge, trailing edge, or center of the edges, which results in three slightly different versions, but all are identified as PWM versions.

4.2 Assignment for all

Do Experiment 4.7.2 PWM and LED Dimmer from the Book.

4.3 Graduate Student Additional Work

Incorporate your dimmer in the LED time-multiplexing circuit of 4.5.1. In order to do this, you need more than 8 switches. Since this is only for testing, use switches 0-7 for the original circuit and also use 0-3 for the dimmer. This has duplication but does not produce any issues for demonstration purposes. Note: basic code for the system is available in the starter code from the book available from the class GitHub.

[1]Bang-bang controls are best known in thermostats, where the system is on or off, but not in any state in between.

Lab 5

Detection

Do either Experiment 5.5.1 Dual-Edge Detector or 5.5.2 Alternative Debouncing Circuit from the book.

Lab 6

Babbage Difference Engine Emulation Circuit

6.1 Newton's Differences

Newton's method of differences is foundational to interpolation, where it comes in as divided differences, and as a simplification of calculus. Essential each difference is approximating a derivative. You calculate the differences between successive steps, say $f(n)$ and $f(n-1)$ for the values of n that are in a region of interest. If the differences are constant you are done, if not take the differences of these. Repeat until they become constant. Since differences approximate derivatives, repeatedly performing differences on finite polynomials will eventually result in a constant. For example consider performing Newton's differences on $f(n) = n^2$,

n	0	1	2	3	4
$f(n) = n^2$	0	1	4	9	16
$f_1 = f(n) - f(n-1) = n^2 - (n-1)^2 = n^2 - (n^2 - 2n + 1) = 2n - 1$	-	1	3	5	7
$f_2 = f_1(n) - f_1(n-1) = 2n - 1 - (2(n-1) - 1) = 2n - 1 - (2n - 3) = 2$	-	-	2	2	2

Notice after two differences you end up with a constant difference of 2, which is the same as the second derivative of the function. Note this is not an accident, since it is following the leading coefficient. It works for both the points and algebraically due to the leading coefficent being the important part, but note the intermediate terms are not exactly the derivative[1], but are needed to keep the scales right given the jump between each integer.

You can also try this on $f(n) = n^3$ (or any other finite polynomial),

n	0	1	2	3	4
$f(n) = n^3$	0	1	8	27	64
$f_1 = f(n) - f(n-1) = n^3 - (n-1)^3 = 3n^2 - 3n + 1$	-	1	7	19	37
$f_2 = f_1(n) - f_1(n-1) = 3n^2 - 3n + 1 - (3(n-1)^2 - 3(n-1) + 1) = 6n - 6$	-	-	6	12	18
$f_3 = f_2(n) - f_2(n-1) = (6n - 6) - (6(n-1) - 6) = 6$	-	-	-	6	6

For non-polynomials or infinite polynomials it will keep going but the intermediates are meaningful due to Taylor Series approximation. If you are interested in going further with this take a course on numerics (a much beloved field of mine...), though this is as much as we need.

6.2 Babbage's Difference Engine

This is what Babbage implemented in his famous difference engine (and expanded on in the analytic engine), and which Lady Ada Lovelace became the first programmer due to her love of and skill with mathematics.

[1] This leads to the need for Newton to develop infinitesimals to fix it, that in turn led to Calculus

Together this idea brings together three of the great events of computer history, which is why we will build it (something Babbage was not able to due to the cost and time. The process is nicely described in Experiment 6.5.7 Babbage Difference Engine Emulation Circuit from the book. Steps 1-4 are the methodology, step 5 just asks you to redo the design for a second system. Design and implement for both polynomials ($f(n) = 2n^2 + 3n + 5$ and $h(n) = n^3 + 2n^2 + 2n + 1$). Calculate the algebraic form

6.3 Graduate Students

Babbage's difference engine can be generalized to where you can input the parameters for each of the sub-functions. Design a system that would work for any second or third order polynomial by inputting coefficients. Include the design with explanation (you don't have to build it) in your report in addition to the work the undergrads must do.

Appendix A

Github

Git is a version control system originally developed by Linus Torvalds to do the Linux OS development. Github is a git web service that I will be using to develop the labs. You can get the manual and code from:

`https://github.com/KeithEvanSchubert/Advanced_Digital_Logic.git`