ECE251 - PLD Assignment 2

Henry J Son

May 18th, 2020

1 Introduction

For this assignment, the goal was two fold: to program (1) combinatorial logic using two inputs and having three outputs, and (2) a three-bit, up/down counter with synchronous clear, count enable, and output enable using five inputs and having three ouputs, onto a 22V10 PLD chip. The program was written using WinCupl: Atmel Version.

1. **Combinatorial logic:**
   With two inputs (A and B), there would be three outputs (Y0..Y2), with the operations 'NOT', 'AND', and 'XOR', respectively. In other words:
   - ***Y0*** = NOT A
   - ***Y1*** = A AND B
   - ***Y2*** = A XOR B
2. **Three-bit, Up/Down Counter:**
   With five inputs (clock, direction, enable, clear, output enable), there would be three outputs (Y3..Y5), where Y5 is the most significant bit and Y3 is the least significant bit. The functionality for each input is as follows:
   - ***clock*** = counter output changes on the rising edge
   - ***direction*** = count increments if direction=1 and decrements if direction=0
   - ***enable*** = count if enable=1; hold current count if enable=0
   - ***clear*** = reset to zero on the next clock if clear=1 and counter is enabled
   - ***output enable*** = Y3-Y5 are active digital outputs if output enable=1 and Hi-Z if output enable=0

2 Implementation

2.1 Combinatorial logic

The implementation for the combinatorial logic was straightforward since the instructions can be found in the manual linked in the introduction. Two pins were defined as the inputs (2 and 3) and three pins were defined as the outputs (14-16). Then, the output pins were set to their respective operations, where:

- ***PIN 14*** = Y0
- ***PIN 15*** = Y1
- ***PIN 16*** = Y2

and

- ***Y0*** = NOT A
- ***Y1*** = A AND B
- ***Y2*** = A XOR B

.

2.2 Three-bit, Up/Down Counter

The code for this gets a lot more interesting than that for the combinatorial logic. A 555 timer is used to clock the chip and this clock has been assigned to Pin 1 because of device specifications. When it came to programming the rest of the counter, there were essentially four tasks at hand: define binary states, mode states, input states, and state machine.

- A 'FIELD' was created to hold the values of the binary count, which are also the outputs in this case. Then, the states of the binary count were defined sequentially and assigned to variables.

```
FIELD count = [Y5..3]
$define s0 'b'000
$define s1 'b'001
$define s2 'b'010
$define s3 'b'011
$define s4 'b'100
$define s5 'b'101
$define s6 'b'110
$define s7 'b'111
```

- Another 'FIELD' was created to hold the values that would control the mode of the counter; in this case, those values would be the clear, enable, and direction. 'Clear' would essentially reset the counter, while 'enable' would determine whether the counter will continue or halt, and 'direction' determines whether the counter will go up or go down. Afterwards, each of the three modes are declared with their respective binary states for each value in the FIELD. For

instance, the counter should count 'up' when clear is 0, enable is 1, and direction is 1 (therefore 011), while the counter should count 'down' when clear is 0, enable is 1, and direction is 0 (hence 010). For 'clear', since decimal values 4 through 7 have a form of 1XX, where X is either 0 or 1, these values will force a clear.

```
FIELD mode = [clr, en, dir]
up = mode:'b'011;
down = mode:'b'010;
clear = mode:'[4..7]
```

- To prevent any loose inputs in the 'count' field, extensions were used and defined. The .sp, .ar, and .oe extensions can be found in the user manual. These declarations will prevent any undefined behaviour from occuring.

```
count.sp = preset;
count.ar = reset;
count.oe = oe;
```

- Lastly, the state machine has to be defined in order for the counter to function properly. For each 'PRESENT', the three modes and a default state are declared, where the modes are the ones that was discussed previously ('up', 'down', and 'clear'). The default is when none of the 'mode' states are HIGH or 1; if this is the case, the 'PRESENT' state will not change.

```
SEQUENCE count {
    PRESENT s0
        if up next s1;
        if down next s7;
        if clear next s0;
        default next s0;
    PRESENT s1
        if up next s2;
        if down next s0;
        if clear next s0;
        default next s1;
    ...
}
```

2.3 Simulation

Due to technical difficulties in getting WinCupl to start, the simulation part of the assignment could not be completed.

2.4 Physical Build

Due to the COVID-19 situation, this assignment could not be physically built without access to Cooper EE's lab. Nonetheless, a schematic was designed as if it was to be used to build the physical device.

3 Schematic Diagram

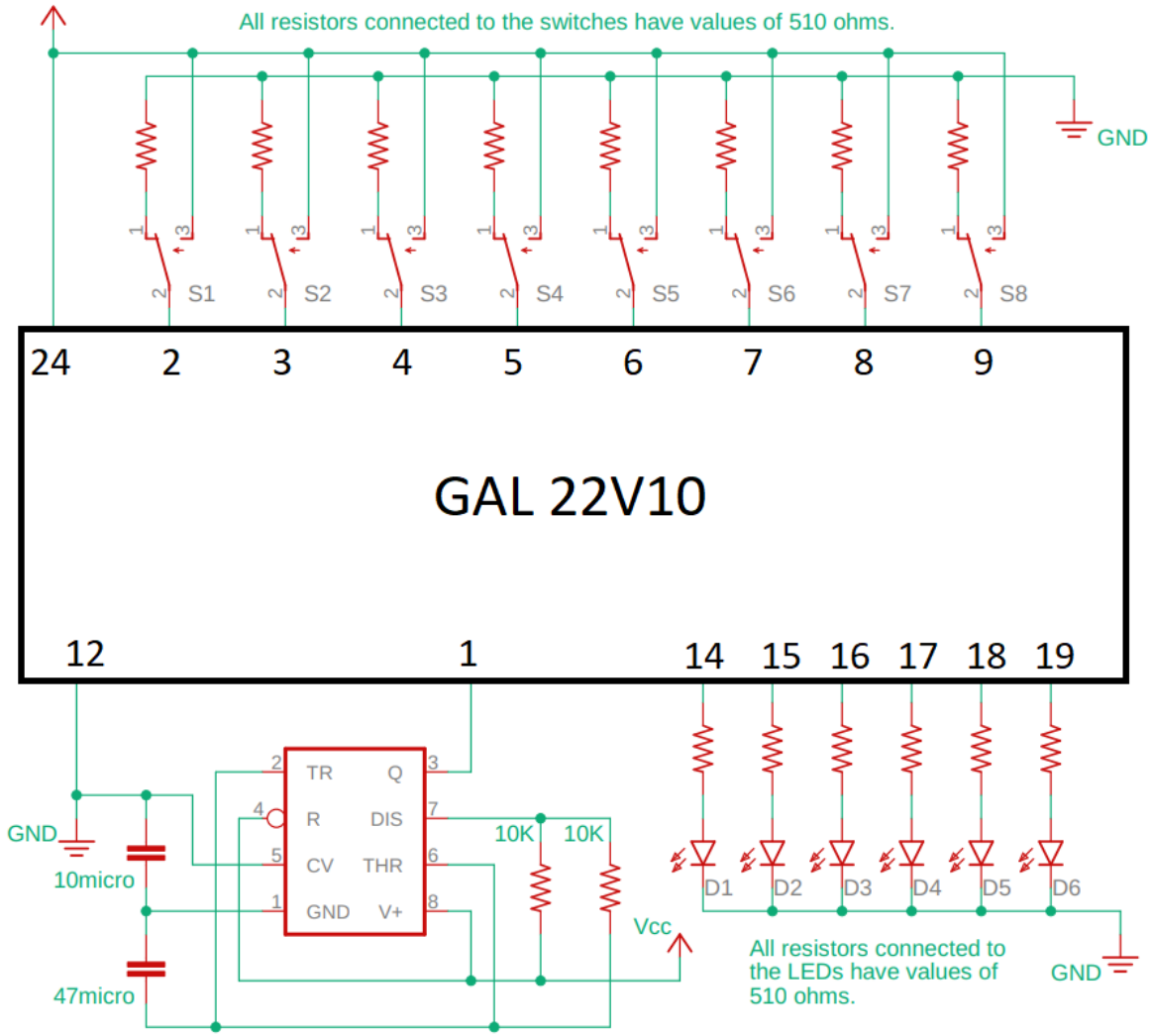

Figure 1: Schematic drawing using EagleCAD and Microsoft Paint.

4 Final Code

Final Code

```
Name     Assignment2 ;
PartNo   ATF22V10C ;
Date     05/18/2020 ;
Revision 01 ;
Designer sonbyj01 ;
Company  Cooper Union ;
Assembly None ;
Location ;
Device   g22v10 ;

/* *************** INPUT PINS ********************* */
PIN 1 = clk;          /* clock */
PIN 2 = a;
PIN 3 = b;
PIN 4 = dir;          /* direction */
PIN 5 = preset;
PIN 6 = reset;
PIN 7 = oe;           /* output enable */
PIN 8 = clr;          /* clear */
PIN 9 = en;           /* enable */

/* *************** OUTPUT PINS ********************* */
PIN 14 = Y0;
PIN 15 = Y1;
PIN 16 = Y2;
PIN 17 = Y3;
PIN 18 = Y4;
PIN 19 = Y5;

/* *************** LOGIC EQUATIONS ********************* */
Y0 = !a;          /* 'NOT' operation */
Y1 = a & b;       /* 'AND' operation */
Y2 = a $ b;       /* 'XOR' operation */

/* *************** 3 BIT COUNTER ********************* */
/* define binary states */
FIELD count = [Y5..3]
$define s0 'b'000
$define s1 'b'001
$define s2 'b'010
$define s3 'b'011
$define s4 'b'100
$define s5 'b'101
$define s6 'b'110
```

```
$define s7 'b'111

/* define mode states */
FIELD mode = [clr, en, dir]
up = mode:'b'011;
down = mode:'b'010;
clear = mode:'[4..7]

/* set input states */
count.sp = preset;
count.ar = reset;
count.oe = oe;

/* state machine */
SEQUENCE count {
    PRESENT s0
        if up next s1;
        if down next s7;
        if clear next s0;
        default next s0;
    PRESENT s1
        if up next s2;
        if down next s0;
        if clear next s0;
        default next s1;
    PRESENT s2
        if up next s3;
        if down next s1;
        if clear next s0;
        default next s2;
    PRESENT s3
        if up next s4;
        if down next s2;
        if clear next s0;
        default next s3;
    PRESENT s4
        if up next s5;
        if down next s3;
        if clear next s0;
        default next s4;
    PRESENT s5
        if up next s6;
        if down next s4;
        if clear next s0;
        default next s5;
    PRESENT s6
        if up next s7;
```

```
        if down next s5;
        if clear next s0;
        default next s6;
    PRESENT s7
        if up next s0;
        if down next s6;
        if clear next s0;
        default next s7;
}
```