GAL PROGRAMMING ASSIGNMENT WRITEUP
PROF. KIRTMAN
ECE 251 — COMP. ARCH.
JONATHAN LAM

PART 1

Part I (combinational logic) was done with basic discrete logic:

```
Y0 = !A;
Y1 = A & B;
Y2 = A $ B;
```

PART 2

I did Part 2 (the 3-bit up/down counter) three different ways. For all of the methods, the output enable was asynchronous (`CNT.oe = OE;`), and the other pins all made synchronous changes.

1. Using a finite state machine (`LAMHW1-FSM.*`)

Using the `SEQUENCE` command and some conditionals, the three counter bits could easily be made to advance, decrease, reset to zero, or stay the same.

```
FIELD CNT = [Z2..0];
CNT.sp = 'b'0;
CNT.ar = 'b'0;
SEQUENCE CNT {
    $REPEAT i = [0..7]
        PRESENT S{i}
            IF EN & CLR NEXT 'b'0;
            IF EN & !CLR & DIR NEXT S{(i + 1) % 8};
            IF EN & !CLR & !DIR NEXT S{(i + 7) % 8};
            IF !EN NEXT S{i};
            DEFAULT NEXT S{i};
    $REPEND
}
```

I realized that the variables for the counter could not be part of the same indexed variable as the variables for the combinational logic output (i.e., the indexed `Y0..Y5` variable), so the counter variables are labelled `Z0..Z2` for this program.

2. Using a truth table (`LAMHW1-TT.*`)

Rather than using conditionals, this approach mapped six inputs (3 counter variables and `EN`, `DIR`, `CLR`) to the D inputs of the three counter macrocells using a single truth table. Using a truth table throws a warning if indexed variables are used, so I renamed `Y3..Y5` to `Y3O`, `Y4O`, and `Y5O`.

The order of the variables here is important to make the truth table work with the `$REPEAT` macro and binary literals, which needed the `EN` input to be the MSB and the `DIR` and `CLR` to be the LSBs.

```
$REPEAT i = [3..5]
    PIN {i + 14} = Y{i}O;
    Y{i}O.sp = 'b'0;
    Y{i}O.ar = 'b'0;
    Y{i}O.oe = OE;
$REPEND

FIELD input = [EN, Y5O, Y4O, Y3O, DIR, CLR];
FIELD output = [Y5O.d, Y4O.d, Y3O.d];
TABLE input=>output {
    'b'1XXXX1=>'b'0;                    /* EN=1, CLR=1 */
    $REPEAT i = [0..7]
        'b'{i}XX=>'b'{i};               /* EN=0 */
        'b'{i + 8}10=>'b'{(i + 1) % 8}; /* EN=1, CLR=0, DIR=1 */
        'b'{i + 8}00=>'b'{(i + 7) % 8}; /* EN=1, CLR=0, DIR=0 */
    $REPEND
}
```

3. Using discrete logic (`LAMHW1-LOGIC.*`)

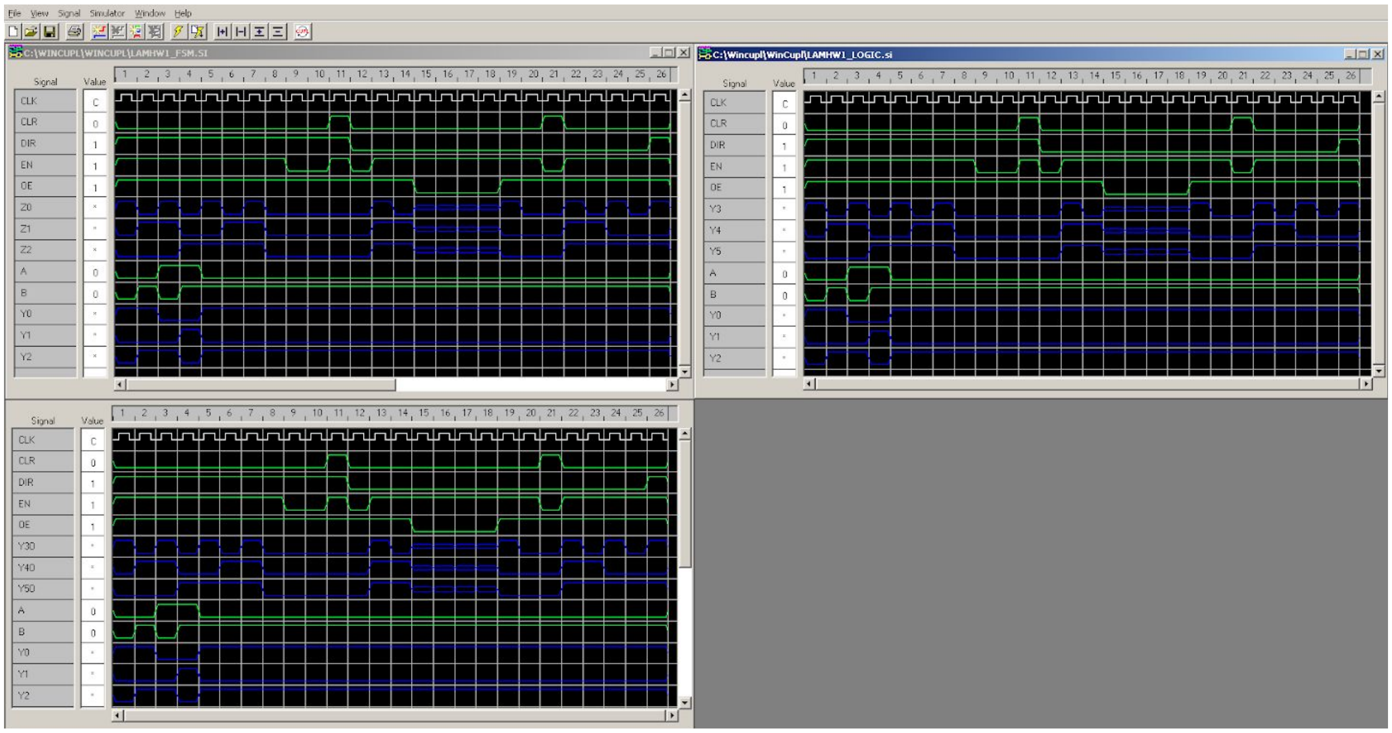
This actually turned out to be the quickest solution because of the simple logic and because I spent much more time learning to use more complex structures for the other two approaches before I tried this one. The logic follows a mostly intuitive path and is not minimized here in the code. (The only minimization I did was a K-map for `DIR`, `Y3`, `Y4`, `Y5` to get the innermost expression.)

```
FIELD CNT = [Y3..Y5];
CNT.sp = 'b'0;
CNT.ar = 'b'0;

Y5.d = (!EN & Y5)
     # (EN & !CLR & ((DIR & !(Y5 $ Y4 $ Y3) # !DIR & Y5) $ !(Y4 # Y3)));

Y4.d = (!EN & Y4)
     # (EN & !CLR & (!DIR $ Y4 $ Y3));

Y3.d = (!EN & Y3)
     # (EN & !CLR & !Y3);
```

TESTING AND RESULTS

Most of the testing was done using WinSim. The provided *`.si` files use the same order and vectors to show that all of these work the same, and for consistency to show that all of them provide the same (correct) output. A screenshot of the three simulations in WinSim is shown in Figure 1.

Figure 1. WinSim Simulations
(From top right, CCW: LOGIC, FSM, TT)


File View Signal Simulator Window Help
C:\WINCUPL\WINCUPL\LAMHW1_FSM.SI
C:\Wincupl\WinCupl\LAMHW1_LOGIC.si
Signal
Value
CLK
CLR
DIR
EN
OE
Z0
Z1
Z2
Y3
Y4
Y5
Y30
Y40
Y50
A
B
Y0
Y1
Y2

Once the simulation produced the correct results, writing the program to the Chipmaster and implementing it was straightforward and without bugs. (I only tested TT and FSM versions on the physical circuit, but having the same results in the simulation makes it likely that the LOGIC version will also produce a working circuit.) A schematic of the implementation is shown in Figure 2, and the specific implementation is shown in Figure 3. The project can be seen working in the video.

NOTES
A 555-timer was used to generate a low-frequency astable clock signal. A 7-segment display (M74 1318) and the 4511 7-segment driver were added to display the output of the counter for easier viewing of the counter's value.

When setting OE high, the 7-segment display shows "0," but the counter bits should be high-impedance as per the simulation. (This is probably the default behavior of the 4511 when its inputs are not driven.)

In the video and in the annotated diagram, the GND wire is actually connected to pin 11 (an input), but still worked normally. When connecting GND to the correct input, the circuit behaved the same.

Figure 2. Schematic Diagram of Implementation

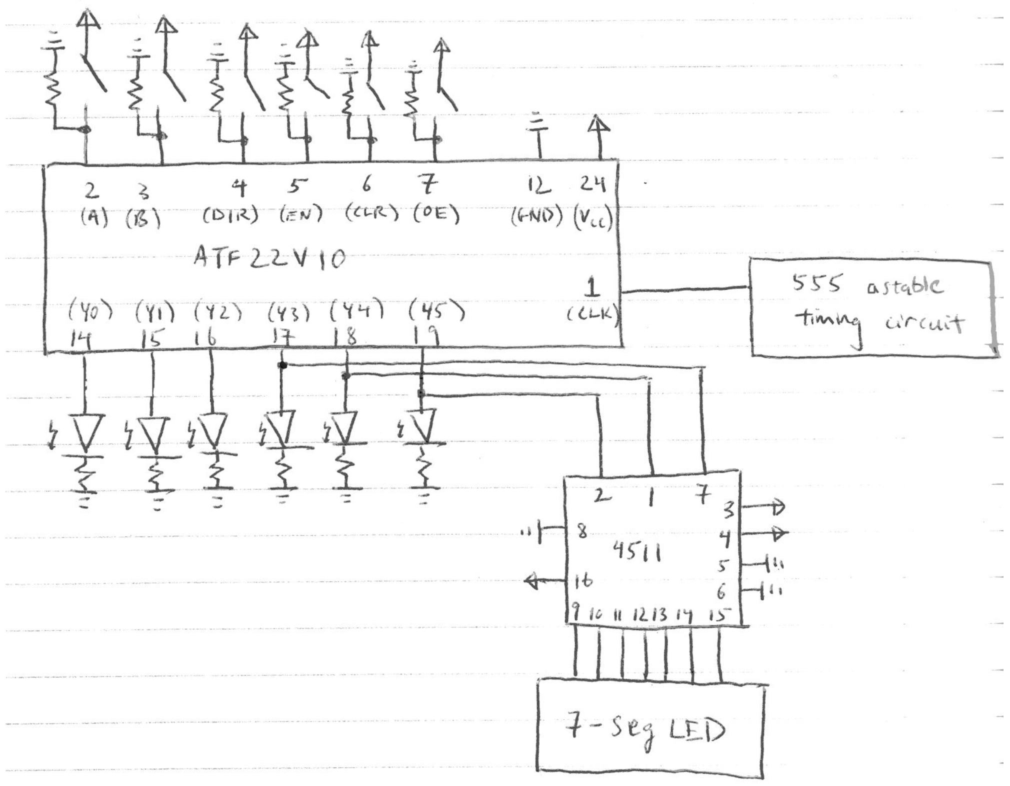


Figure 3. Annotated Image of Implementation