

ECE 272 Lab #5

System Verilog

Marshall Saltz

11/17/2022

1. Introduction

This is an introductory lab to SystemVerilog. It involves taking code from the textbook as well as writing my own. The software used is Quartus Prime Lite 18, ModelSim, draw.io, and the digital ECE 271 textbook, Digital Design and Computer Architecture by Drs. David and Sarah Harris. The hardware used is my laptop, a DE10-Lite FPGA, and a USB-USBC cable. The logic block diagram will be drawn in draw.io, the top-level schematic in Quartus, and the modules that underly the diagram will be programmed in SystemVerilog through Quartus. The entire program will be simulated in ModelSim and pushed to the FPGA. The goal is to create a clock that monitors seconds, minutes, and hours in military time, resetting entirely at twenty-four hours.

2. Design

This design is modeled by using a clock divider, which divides 50 Mega Hertz into about one Hertz. The clock divider leads to a seconds counter, which counts up to fifty-nine seconds, after which the counter resets and the sixtieth second is carried over into the minutes counter by the synchronizer. The parser takes the seconds value and modulates it into tens and ones digits, which are then displayed on the seven segment display. The minutes and hours follow a similar process, with the twenty-fourth hour being carried back into the seconds.

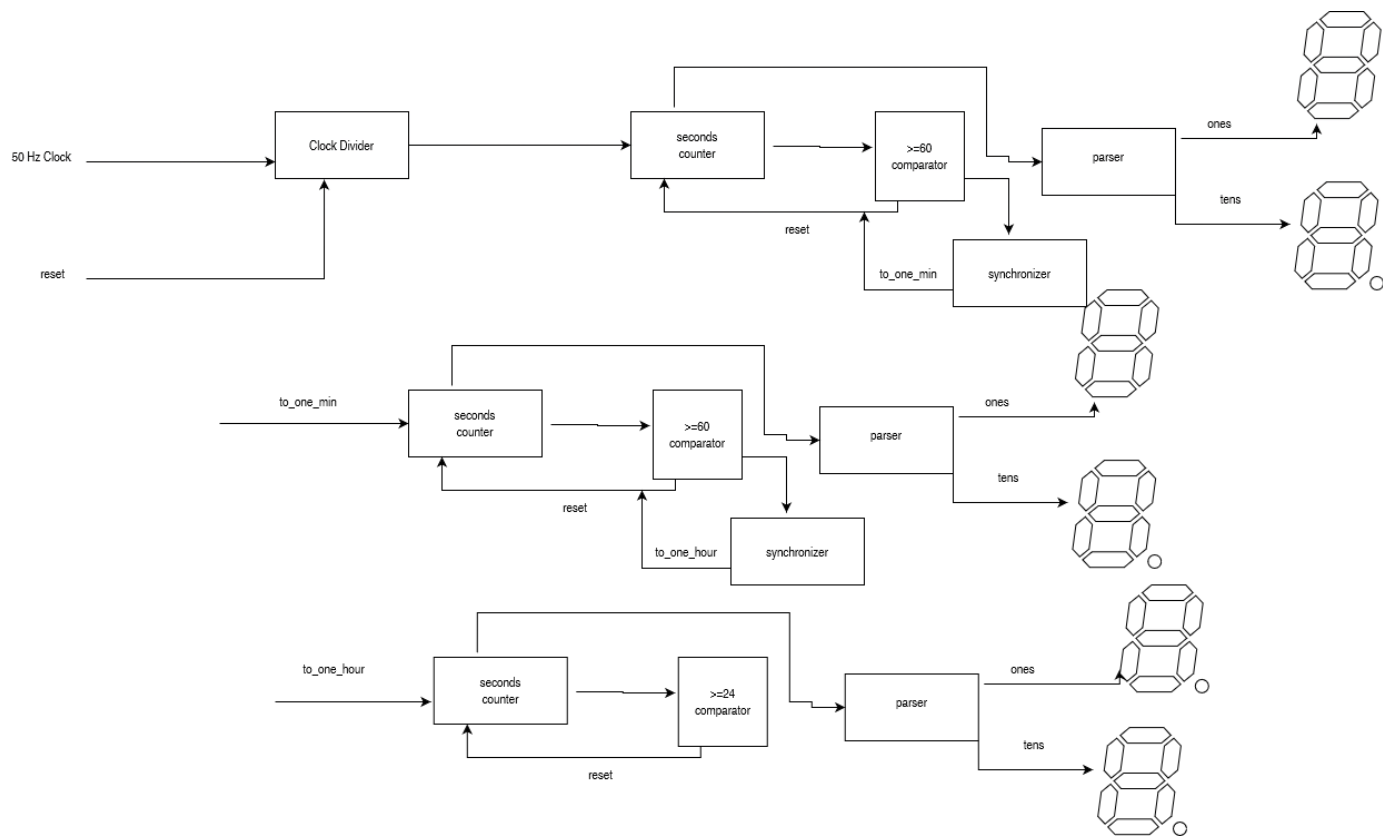


Fig. 1: Block Diagram

This figure describes the top-level logic of the program, which is created by blocks of SystemVerilog logic.

Interface Name	Interface Purpose	Input or Output	Number of Pins	Active High or Low	Board Label	FPGA Pin
clk	Gate Input	I	1	High	Clock	P11
reset	Gate Input	I	1	Low	Push Button	A7
seg1[6]	Gate Output	O	1	Low	7SEG	C14
seg1[5]	Gate Output	O	1	Low	7SEG	E15
seg1[4]	Gate Output	O	1	Low	7SEG	C15
seg1[3]	Gate Output	O	1	Low	7SEG	C16
seg1[2]	Gate Output	O	1	Low	7SEG	E16
seg1[1]	Gate Output	O	1	Low	7SEG	D17
seg1[0]	Gate Output	O	1	Low	7SEG	C17
seg2[6]	Gate Output	O	1	Low	7SEG	C18
seg2[5]	Gate Output	O	1	Low	7SEG	D18
seg2[4]	Gate Output	O	1	Low	7SEG	E18
seg2[3]	Gate Output	O	1	Low	7SEG	B16
seg2[2]	Gate Output	O	1	Low	7SEG	A17
seg2[1]	Gate Output	O	1	Low	7SEG	A18
seg2[0]	Gate Output	O	1	Low	7SEG	B17
seg3[6]	Gate Output	O	1	Low	7SEG	B20
seg3[5]	Gate Output	O	1	Low	7SEG	A20
seg3[4]	Gate Output	O	1	Low	7SEG	B19
seg3[3]	Gate Output	O	1	Low	7SEG	A21

seg3[2]	Gate Output	O	1	Low	7SEG	B21
seg3[1]	Gate Output	O	1	Low	7SEG	C22
seg3[0]	Gate Output	O	1	Low	7SEG	B22
seg4[6]	Gate Output	O	1	Low	7SEG	F21
seg4[5]	Gate Output	O	1	Low	7SEG	E22
seg4[4]	Gate Output	O	1	Low	7SEG	E21
seg4[3]	Gate Output	O	1	Low	7SEG	C19
seg4[2]	Gate Output	O	1	Low	7SEG	C20
seg4[1]	Gate Output	O	1	Low	7SEG	D19
seg4[0]	Gate Output	O	1	Low	7SEG	E17
seg5[6]	Gate Output	O	1	Low	7SEG	F18
seg5[5]	Gate Output	O	1	Low	7SEG	E20
seg5[4]	Gate Output	O	1	Low	7SEG	E19
seg5[3]	Gate Output	O	1	Low	7SEG	J18
seg5[2]	Gate Output	O	1	Low	7SEG	H19
seg5[1]	Gate Output	O	1	Low	7SEG	F19
seg5[0]	Gate Output	O	1	Low	7SEG	F20
seg6[6]	Gate Output	O	1	Low	7SEG	J20
seg6[5]	Gate Output	O	1	Low	7SEG	K20
seg6[4]	Gate Output	O	1	Low	7SEG	L18
seg6[3]	Gate Output	O	1	Low	7SEG	N18

seg6[2]	Gate Output	O	1	Low	7SEG	M20
seg6[1]	Gate Output	O	1	Low	7SEG	N19
seg6[0]	Gate Output	O	1	Low	7SEG	N20

Fig. 3: Interface

This is the interface which all the pins, inputs and outputs, are mapped to.

3. Results

I took screenshots of the ModelSim outputs, which are included below. The board successfully counts seconds, minutes, and hours. In ModelSim, the outputs show the transitions between ones seconds to tens seconds to ones minutes to tens minutes to ones hours to tens hours, after which it is reset back to ones seconds.

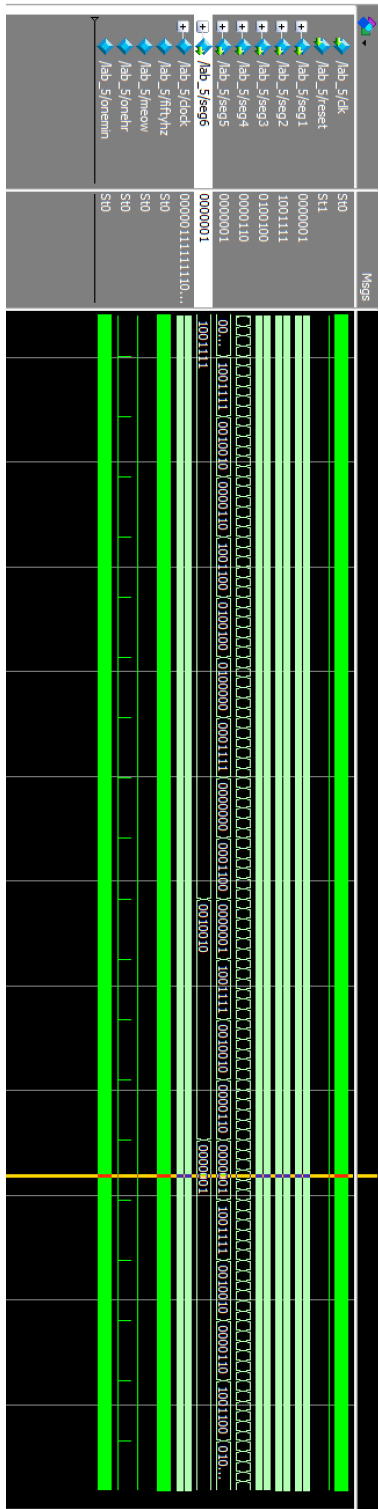


Fig. 4: Ones Hours Into Tens

The above figure shows the change from single digit hours to double digit hours.

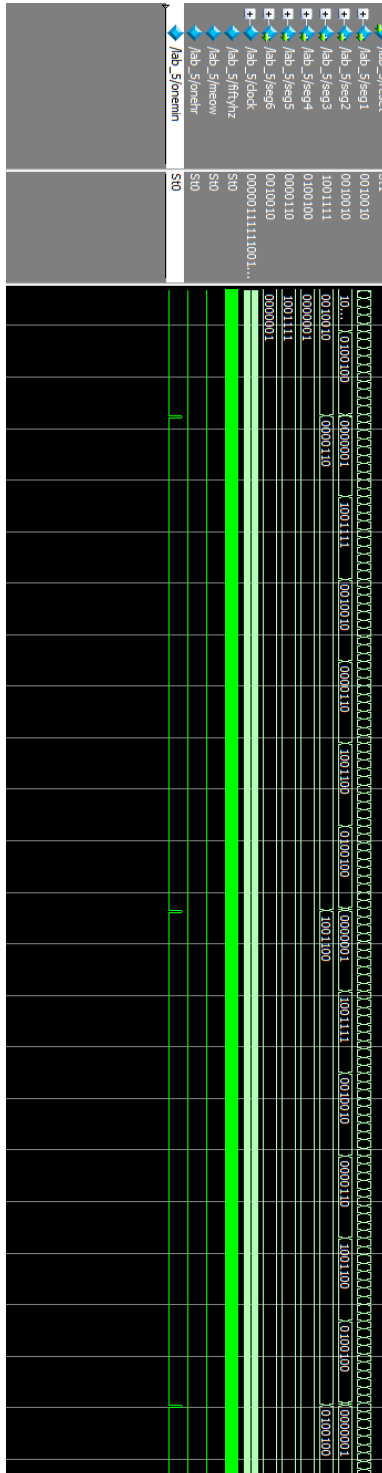


Fig. 5: Tens Seconds Into Minutes

The above figure shows the transition from double digit seconds into single digit minutes.

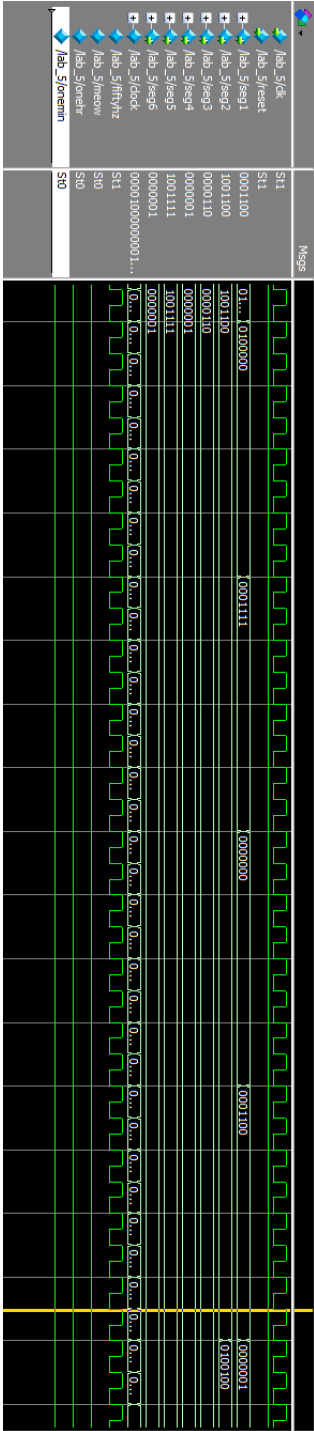


Fig. 6: Ones Seconds Into Tens Seconds

The above figure shows the transition from single digit seconds into double digit seconds.

4. Experiment Notes

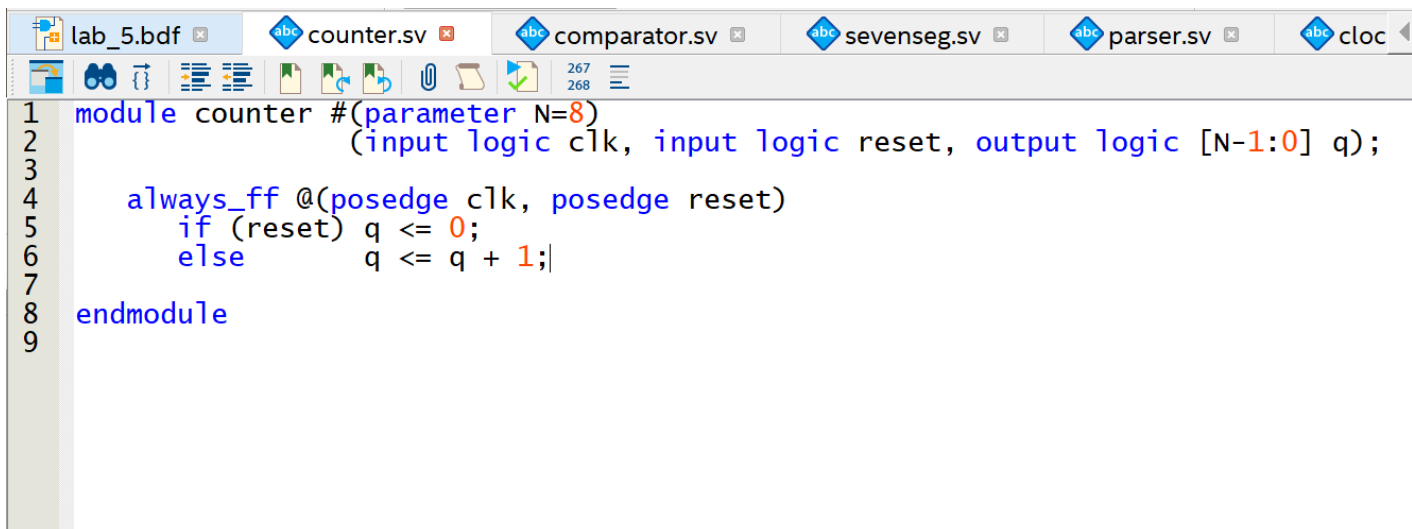
Coding. That's what I'm good at, so this lab will be easy-peasy, right? Wrong! While it was not the coding I struggled with, but rather ModelSim and adjusting the runtime for the clock to display each transition from seconds to minutes to hours. ModelSim was by far the biggest struggle with this lab, and the worst part was I had an angry FPGA staring at me counting menacingly ever upwards, reminding me that I was running out of time to complete the lab. The coding went well, and there were many resources available to assist me with that portion, but I struggled with ModelSim.

5. Study Questions

1. How off is your clock?

Answer: My clock is off by 1.4901 seconds. I got this answer by dividing 50 MHz by 2^{25} seconds.

6. Appendix



```
1 module counter #(parameter N=8)
2     (input logic clk, input logic reset, output logic [N-1:0] q);
3
4     always_ff @(posedge clk, posedge reset)
5         if (reset) q <= 0;
6         else      q <= q + 1;
7
8 endmodule
9
```

Fig. 7: counter.sv

This is the counter module, which counts seconds, minutes, and hours.

```
1 module comparator #(parameter N = 8, b = 60)
2   (input logic [N-1:0] a,
3     output logic eq, neq, lt, lte, gt, gte);
4
5   assign eq = (a == b);
6   assign neq = (a != b);
7   assign lt = (a < b);
8   assign lte = (a <= b);
9   assign gt = (a > b);
10  assign gte = (a >= b);
11
12 endmodule
13
```

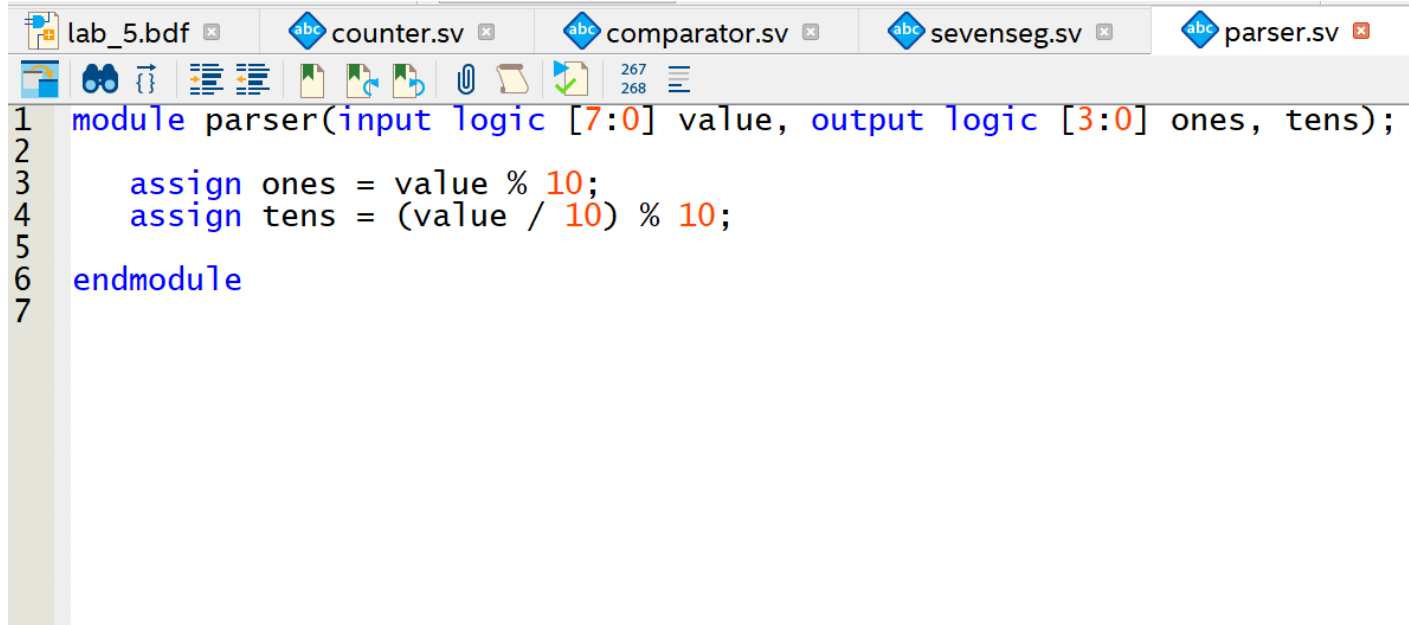
Fig 8: comparator.sv

This is the comparator module, which compares seconds, minutes, and hours to a fixed parameter.

```
1 module sevensseg(input logic [3:0] data, output logic [6:0] seg);
2
3     always_comb
4     case(data)
5         0: seg=7'b0000001;
6         1: seg=7'b1001111;
7         2: seg=7'b0010010;
8         3: seg=7'b0000110;
9         4: seg=7'b1001100;
10        5: seg=7'b0100100;
11        6: seg=7'b0100000;
12        7: seg=7'b0001111;
13        8: seg=7'b0000000;
14        9: seg=7'b0001100;
15        default: seg=7'b0000001;
16    endcase
17 endmodule
18
```

Fig 8: sevensseg.sv

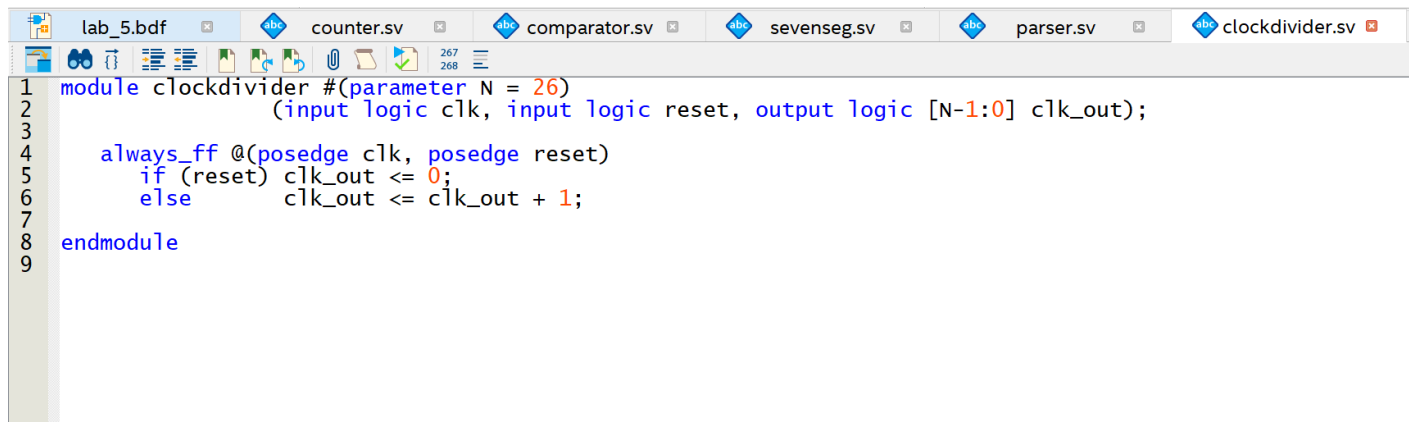
This module is the seven segment display output for each digit.

The image shows a screenshot of a Verilog code editor. The top toolbar includes icons for file operations (new, open, save, print), editing (undo, redo, copy, paste), and simulation (run, step through). The tab bar at the top shows several files: lab_5.bdf, counter.v, comparator.v, sevenseg.v, and parser.v. The parser.v file is the active one, displaying the following code:

```
1 module parser(input logic [7:0] value, output logic [3:0] ones, tens);
2
3     assign ones = value % 10;
4     assign tens = (value / 10) % 10;
5
6 endmodule
7
```

Fig 10: parser.v

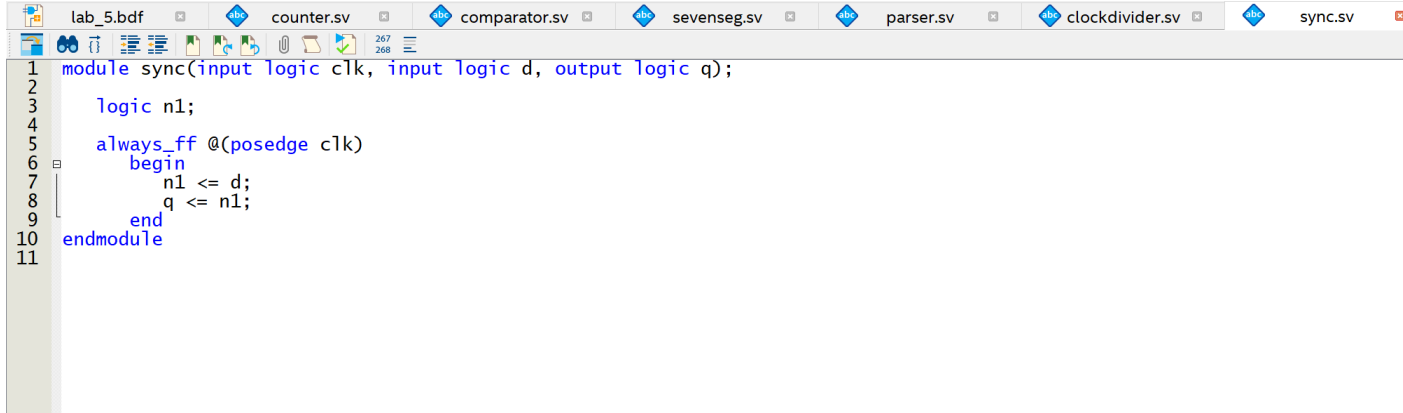
This parser separates two digit numbers into the tens and ones values.

The image shows a screenshot of a Verilog code editor. The top toolbar and tab bar are similar to the previous figure. The tab bar now includes an additional file, clockdivider.v, which is the active file. The clockdivider.v file displays the following code:

```
1 module clockdivider #(parameter N = 26)
2     (input logic clk, input logic reset, output logic [N-1:0] clk_out);
3
4     always_ff @(posedge clk, posedge reset)
5         if (reset) clk_out <= 0;
6         else      clk_out <= clk_out + 1;
7
8 endmodule
9
```

Fig 11: clockdivider.v

This is the clock divider, which reduces 50 MHz.

The image shows a screenshot of a Verilog code editor. The top of the window displays a series of open files: 'lab_5.bdf', 'counter.v', 'comparator.v', 'sevenseg.v', 'parser.v', 'clockdivider.v', and 'sync.v'. The 'sync.v' file is the active one, showing the following code:

```
1 module sync(input logic clk, input logic d, output logic q);  
2     logic n1;  
3  
4     always_ff @(posedge clk)  
5     begin  
6         n1 <= d;  
7         q <= n1;  
8     end  
9 endmodule
```

Line numbers 1 through 11 are visible on the left margin.

Fig 12: sync.v

This is the synchronizer, which carries the 60th second, 60th minute, or 24th hour into the next module.