**3TB4 Tutorial/Prelab 1 Report**

**Lin Fu ful10 400234794**

**Keyin Liang liangk10 400236736**

Quartus Prime Introduction Using Verilog Designs:

Complication Report:

Graphical user interface, text, application

Description automatically generated

Simulation Report:

Graphical user interface, table

Description automatically generated

Introduction to Simulation of Verilog Designs:

Simulation Report:

A picture containing timeline

Description automatically generated

Simulation Report Modified:

A picture containing table

Description automatically generated

Complication Report:

Graphical user interface, text, application

Description automatically generated

Prelab Questions:

1. What is the reg data type and what is the wire data type in Verilog?

The reg data type is the hardware registers that stores data, the wire data type describes the connection between elements or hardware entities.

1. Can the wire data type be used on the left side of the assignment statement in a procedural block?

No, the wire data type can only be used on the left side when outside of the procedural block (always block).

1. What are the rules for module port connection?

First, declare input, inout or output as: input/output/inout [range\_val:range\_var] list\_of\_identifiers; Inputs are type net and can be connected to a variable of type reg or net. Outputs are type net or reg and must be connected to a variable of type net. Inouts are type net and can only be connected to a variable net type.

1. What are continuous assignment, blocking assignment and nonblocking assignment?

Continuous assignment: Assign values to the nets (LHS) whenever there is a change on the RHS.

Blocking assignment: Blocking the next execution until the current statement is executed. The statements are executed in order.

Nonblocking assignments: Simultaneously execute statements.

1. What is the difference in procedural coding when implementing combinational logic and sequential logic?

Combinational logic does not require a clock to operate, whereas sequential logic does.

1. How does one avoid inferred latches when using Verilog to describe circuits?

To avoid this, make sure that all of the if statements that are used in combinational always blocks have one last else statement to catch all missing conditions.

Always make sure every variable in the procedural block gets assigned.

Use default assignments at the beginning of the procedure, so that every signal is assigned.

1. What is the difference between the operators “<<” and “<<<”?

“<<<” is the arithmetic left shift and “<<” is a logical shift. They both shift left specific bits while “<<” fills the vacated bits with 0, “<<<” fills the vacated bit positions with the value of the sign bit if the expression is signed, otherwise fills with 0.

1. How to declare an array of 6 elements of a 7-bit wire?

Wire[6:0] array[5:0]

One bit data width D flipflop:

A picture containing text

Description automatically generated

One bit data width D flip-flop with active low synchronous reset:

Text

Description automatically generated with low confidence

One bit data width D flip-flop with active low synchronous reset and active low enable:

Graphical user interface, text

Description automatically generated

D latch with synchronous enable control:

Text

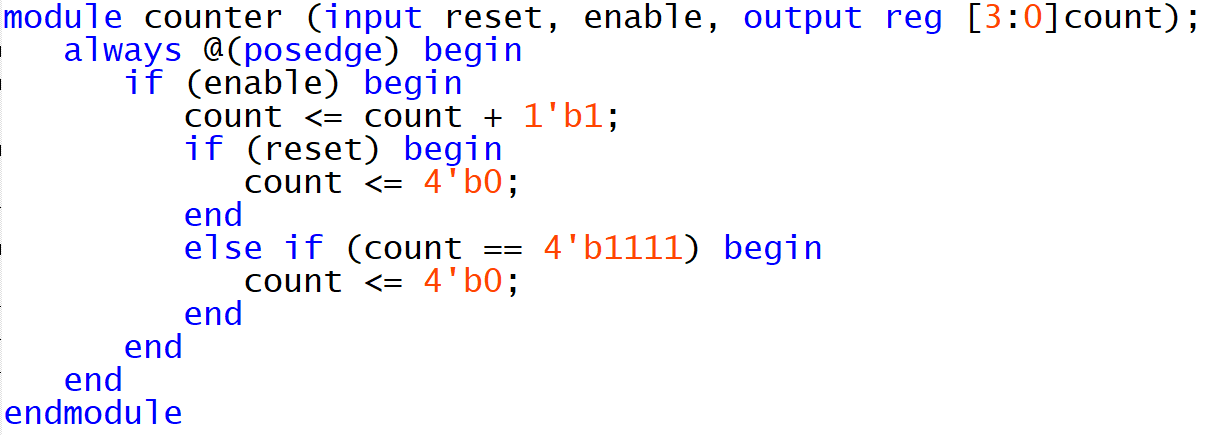
Description automatically generated with medium confidence

4-to-1 multiplexer:

A picture containing text

Description automatically generated

4-bit counter with reset and enable controls:



Truth Table for 7-Seg

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Input | | | | Output | | | | | | |
|  | A | B | C | D | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| L | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| I | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| N | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| F | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| U | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| OFF | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

K-map for segment 0 and 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Segment 0 | |  |  |  |
| AB\CD | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 0 | 0 |
| 01 | 1 | 0 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 |
| 10 | 0 | 0 | 1 | 1 |
| Segment 1 | |  |  |  |
| AB\CD | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | 0 | 1 | 1 | 0 |
| 10 | 0 | 0 | 0 | 1 |

Segment 0 logic =

Segment 1 logic =

Graphical user interface, table

Description automatically generated