Lab 3 Tutorial

Exploring Verilog

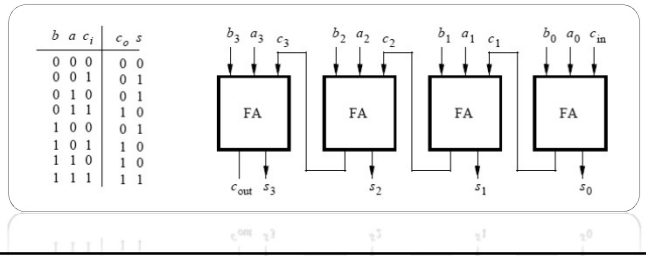
- Verilog can express logic beyond the basic gate-by-gate specification.
  - Typically in `always` blocks
- Example: `case` statements
  - Not the same as case statements in other languages!
  - **`case`** statements in Verilog provide output behaviour for all possible input values
    - Like specifying the output for all minterm cases
  - How would you do a multiplexer with this?

Part 1: Mux + case statement

- Use `case` statement and `always` block to implement a 7-to-1 multiplexer.
  - See the example of a `hex_decoder` from class.
- Key points:
  - Combination circuits w/ an `always` block.
    - Use * in the sensitivity list of your `always` block.
  - Don't forget the `default` case! Or else the tool will need to have memory (which we haven't done yet).
  - New storage term: `reg` (used similar to `wire`)
    - Use `wire` with assignment statement outside `always` blocks, and `reg` within them.

Part 2: Ripple Carry Adders

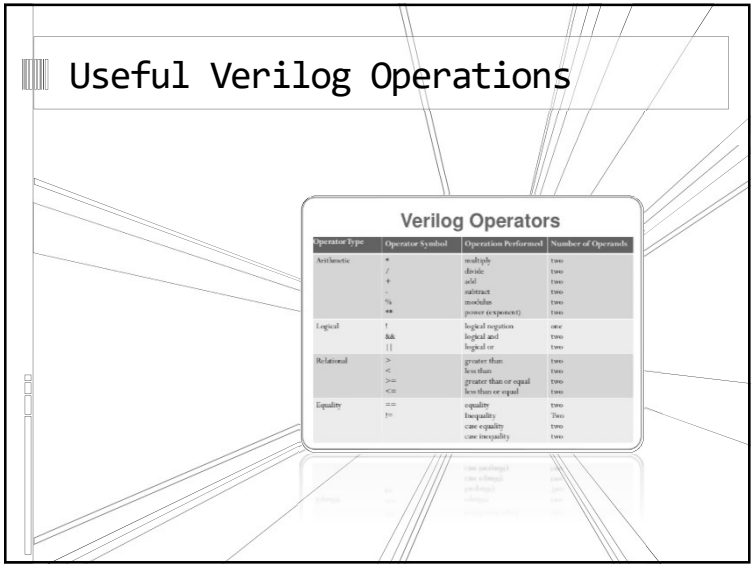
- Implement a Ripple Carry Adder by connecting (chaining) four full-adders together.
  - Must use hierarchical design!



Part 3

- Implement a simple ALU (Arithmetic Logic Unit) and display inputs/ouputs to LEDs and 7-segment display (in hex).
  - You should reuse work you did in Lab 2.
  - Uses mux to implement addition, subtraction, inversion, etc.
    - More Verilog operators! ☺

Useful Verilog Operations



Bitwise Operators

- Bitwise Operators (see Table 1 from Lab2)
  - If you use a bitwise operator with two n-bit operands, the result is also an n-bit vector.
- For example:
  - `3'b101 & 3'b011` → `3'b001`

Table 1: Verilog Operators

|	bitwise OR
&	bitwise AND
~	bitwise negation
^	bitwise XOR

- More general mathematical notation:
  - ($X_{n-1}X_{n-2}..X_1X_0$ `&` $Y_{n-1}Y_{n-2}..Y_1Y_0$) results in $W_{n-1}W_{n-2}..W_1W_0$ where W_i is (X_i `&` Y_i) for every `i` in `[0,n-1]`.
  - You can use any of these bitwise operators in place of &.

Reduction Operators

- Reduction Operators have same symbol as bitwise, but
  - they take a single multi-bit operand, and
  - they result in a single-bit vector.
- For example:
  - `(& 3'b101)` results in `1'b0`
- General mathematical notation:
  - `(&` $X_{n-1}X_{n-2}..X_1X_0$`)` results in W_0 where W_0 is (X_{n-1} `&` X_{n-2} `& ... &` X_1 `&` X_0).
    - Think of this as feeding all `n` bits of `X` into a single n-input AND gate with output W_0
  - You can use any of the reduction operators in place of `&`.

Replication and Concatenation

- The binary value `011` (3 in decimal) is the same as `0011` or `000000011`.
  - Adding zeros in the most significant bits of a positive or an unsigned number does NOT change the number being represented!
- Example:
  - If the output of a module is 3-bits and you want to feed it to a 5-bit input of another module, you'd need to use both replication & concatenation!
    - See Background section on Lab3 handout.

More Complex Force Commands

```
force {<signal_name>} <initial value> <initial time>, <new value> <new time>
                      -repeat <repeat time> -cancel <cancel time>
```

```
force {a} 0 0, 1 20 -repeat 40
force {b} 0 0 ns, 1 40 ns -r 80
```

- What's happening here:
  - The waveform for `a` (see first `force` command) starts low (logic-`0`) @ `0`ns
  - At `20`ns, it goes high (changes to logic-`1`).
  - It stays high until `40`ns when this process repeats
    - (i.e., it goes back to 0 until `60`ns; at `60`ns it goes back to `1` until `80`ns and so on..)
    - `a` is represented by a square wave with a period of `40`ns
  - The square waveform for `b` has double the period of `a`.
- Don't forget to follow these force commands with a `run` command (e.g., run 160ns)!

Specifying a vector:

- What you might want to do:
  - `wire multibit_vector [3:0];`
- The you should do:
  - `wire [3:0] multibit_vector;`
- Additional notes:
  - You can specify a part of this vector's bits (a subset of wires) by writing: `multibit_vector[2:1]`

Frequently asked questions

- Can we use the `2to1mux` module from Lab 2 for prelab Lab3 Part1, or just logic gates?
- Answer:
  - In terms of your Verilog code, you need to use the template provided in page 3 of your Lab 3 handout, i.e., an `always` block with a `case` statement.
  - You should **not** use multiple 2-to-1 muxes like you did in Lab 2. The goal of Part I is to provide you with an alternate way of describing a multiplexer, what we call "behavioural Verilog".
  - The combinational circuit you will describe this way will of course be synthesized using logic gates by Quartus!

Frequently asked questions

- How would one draw a schematic for the mux in Part1?
- Answer:
  - Think of the 7-to-1 mux described in the template as a black box (a building block of your circuit). What will its `MuxSelect` input be connected to? How about the reg `Out`? You need to specify how these are connected to the various FPGA pins.

Frequently asked questions

- Should the default value of the 7 to 1 mux be the same as case 6?
- Answer:
  - It doesn't matter. You can think of the default case as a "don't care" since this is a 7-to-1 mux and not an 8-to-1 mux.
  - That being said, it usually helps with debugging if you hardwire the output of the circuit in that case, as in make it be a known value that does not rely on any of the inputs. For example, making it always be `0` would achieve that.

Frequently asked questions

- When it says that the output should be `1` (`8'b00000001`) if at least 1 of the 8 bits in the two inputs is `1` using a single OR operation. What do you mean by '*using a single OR operation*'?
- Answer:
  - A '*single OR operation*' is the same as an OR Reduction Operator. Information on reduction operators is available on page 2 of the handout.

Frequently asked questions

- When I try to run my `part3wave.do` on Modelsim I get an error message saying that it cannot find my instantiation of '`ripplecarryadder`' or '`HEXdisplay`'. I have imported these two modules in my project using quartus for Lab3Part3. Do I need to add some lines of code to my `part3wave.do` file to correct this?

Frequently asked questions

- Answer:
  - If the code for these modules is written in separate `*.v` files you need to specify them in your `vlog` command in ModelSim. What you do in your Quartus project has no effect on ModelSim; notice that we don't even ask you to set-up the Quartus project as part of you prelab.
  - You can specify more than one Verilog files in your `vlog` command; just separate them by spaces:

```
vlog -timescale 1ns/1ns mux.v
that_other_module_you_used.v a_third_module.v
```

  - ModelSim also allows the use of wildcards. For example, assuming you have all the Verilog files you need in the same directory (the one you cd-ed to before running your `*.do` script), you can simply do:

```
vlog -timescale 1ns/1ns *.v
```