EECS 470 Project #1

- This is an individual assignment. You may discuss the specification and help one another with the (System) Verilog language. Your solution, particularly the designs you submit, must be your own.
- Due at 11:59pm ET on 29th January, 2021. *Late submissions are not accepted.*

---

1 Introduction

In this project, you will be designing a number of different priority selectors. A priority selector, or priority arbiter, has n pairs of request and grant lines. As the names imply, the selector chooses one of the asserted request lines and asserts its corresponding grant line. In the most general case, the selector can assert k grant lines, where $k \leq n$, though for this project, $k = 1$.

Priority selectors are heavily used in computer architecture, where we often have limited resources that need to be assigned optimally for best performance. The modules you write for this assignment will both remind you about the concepts of digital design you first learned in EECS 270 and begin to prepare you for the final project, where they can be reused.

2 Assignment

In this assignment you will be asked to design and test the following devices:

- A 4-bit fixed priority selector
  - Using `assign` statements for the combinational logic.
  - Using `always` blocks for the combinational logic.
- A 4-bit and an 8-bit fixed priority selector using a hierarchy of 2-bit selectors.
- A 4-bit rotating priority selector using a hierarchy of 2-bit selectors.

2.1 Before you start

Log into a CAEN computer, booted into Redhat Enterprise Linux. Open a terminal, now run the following commands after downloading project1.tar.gz to your current directory:

```
tar -xvf project1.tar.gz
cd project1
```

Figure 1: Project setup guide, from the command line

These commands create a directory for 470, create a directory for projects and then move into that directory. Then, the project 1 source is pulled from the course website and unpacked.

2.2 4-bit Fixed Priority Selector

For this section, you will design two different 4-bit fixed priority selectors. Both are to be declared as:

```
module ps4(
    input        [3:0] req,
    input              en,
    output logic [3:0] gnt
);
```

The signal `req[3]` is the highest priority request, and priority goes down to `req[0]`, which is the lowest priority request. In all cases no more than one of the grant lines should be asserted at any given time. If `en` is low, then no grant lines should be asserted. For example, if `en=1` and `req=4'b1111`, then `gnt=4'b1000`.

The design for this section should be put into a file named `P1a.v`. Using `assign` statements (and no `always` blocks) implement a 4-bit priority selector. You may want to use a K-map to find the logic equations needed for the grant lines, or you might be able to solve it ad hoc.

The `Makefile` provided with this project is already set to use this file along with a provided testbench, which can be found in the file `test.v`. Once you think you have the code correct, simply run the command `make` in the directory. Odds are very good that this won't work the first time you try it, and you'll have to figure out the errors. Compile errors will point you to a line number in your design, with some kind of hint as to what is wrong. If things compile it is still possible, perhaps even likely, that the testbench will find an error. If that happens, you are strongly encouraged to look at the testbench to try to understand what it is doing.

Once you have completed the above task, you need to do it again, but this time put the design in a file named `P1b.v`. This time you are to use `if/else` statements inside an `always` block. Make the necessary changes so that the `Makefile` builds the new file instead of the old one.

2.3 Hierarchical Priority Selectors

Consider the Verilog you wrote in `P1a.v` and `P1b.v`. If you were going to make a 128-bit priority selector, your design would be quite long and unreadable. One way to avoid this problem is to build your module in a way that it can be combined with itself to make a larger version. For example, it is fairly easy to use three 2-bit `and` gates to create a single 4-bit `and` gate. Think through how you would go about building this. Figure 2 shows how this is done in the case of the `and`. How would you go about doing this with priority selectors? It's not easy...

```
module and2(
    input        [1:0] a,
    output logic       x
);
    assign x=a[0] & a[1];
endmodule

module and4(
    input        [3:0] in,
    output logic       out
);
    logic [1:0] tmp;
    and2 left(.a(in[1:0]),.x(tmp[0]));
    and2 right(.a(in[3:2]),.x(tmp[1]));
    and2 top(.a(tmp),.x(out));
endmodule
```

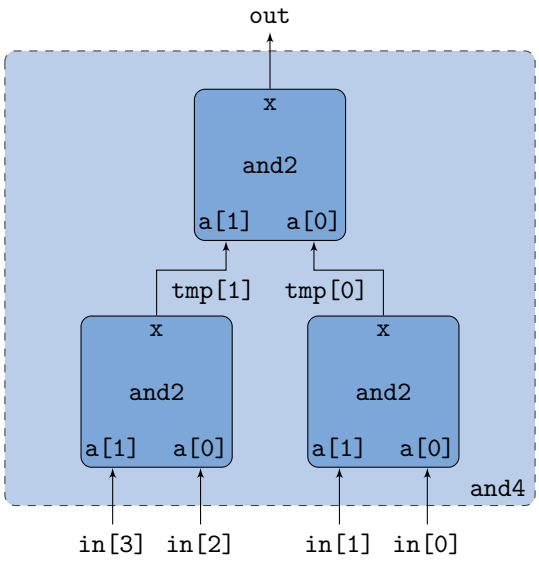

Figure 2: Hierarchical Module Example: a 4-bit `and` built with three 2-bit `and` modules.

Let's consider a 2-bit priority selector as a building block. In order to make a 4-bit device, you'd need three of 2-bit selectors in a tree structure, similar the `and`'s above. The right might get the two lowest priority request and grant lines, while the left got the two highest priority request and grant lines. Then the left and right would ask the top to choose which of them got the grant. In our previous 4-bit module we had a way to be told if we *could* grant to anyone, the enable line (`en`). But we didn't have a way of saying "Hey, I have a request that would like a grant." We will need to add that functionality, and you should call it `req_up` for "requesting something from the device above me." This signal should be asserted if either request is asserted no matter the value of the enable.

You will need two modules, one is the 2-bit priority selector and the other is the 4-bit. These modules should be declared as follows:

```
module ps2(
        input        [1:0] req,
        input              en,
        output logic [1:0] gnt,
        output logic       req_up
);

module ps4(
        input        [3:0] req,
        input              en,
        output logic [3:0] gnt,
        output logic       req_up
);
```

To build the 2-bit selector, use either `P1a.v` or `P1b.v` as a template, and don't forget to include the `req_up` signal. Then you need to build the 4-bit device. Use fig. 2 as an example of the concept of the tree structure, and in particular, be sure you understand the Verilog before you proceed.

Your design for this section should be written into a file named `P1c.v`. In addition, you will need to modify the testbench to use this new module and its additional output. Make the needed changes to the Makefile to use this new design. Once you have this version working, build a `ps8` module using `ps4` and `ps2` modules, and save it in the same file. The module declarations should follow from `ps4`, above. For full credit, your design should not contain any additional "glue" logic. It should only contain the minimum number of ps modules and wires connecting them. To test this new module, write a new testbench and save it as `testC.v`. You will need to turn it in.

2.4 Hierarchical Rotating Priority Selectors

It is often the case that you would like to evenly service all devices connected to some common resource, instead of prioritizing one over the another. This avoids *livelock* where a lower priority device is never selected because something with higher priority is constantly requesting. One way to build such a selector is to change priority every clock tick. To do this, we clearly need to introduce some sequential logic.

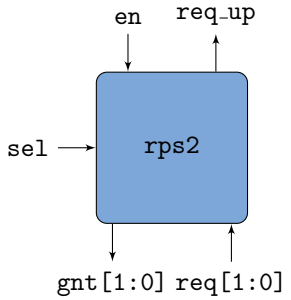

Figure 3: 2-bit rotating priority selector (`rps2`)

In this case, `sel` chooses which bit of the `req` bus will have the higher priority. For this assignment, if `sel=1`, then `req[1]` has the higher priority.

Figure 4 shows how the `sel` bus should be used in the 4-bit selector. Each level of the tree uses the same bit of the bus. The design for the `rps4` and `rps2` modules should go in a file called `P1d.v`. You should use `P1c.v` as a starting point by adding a select line to the `ps2` module and renaming `ps4` to `rps4`. Additionally, `rps4` will need 1-bit `clock` and `reset` lines, and a 2-bit `count` output. The module declaration is as follows:

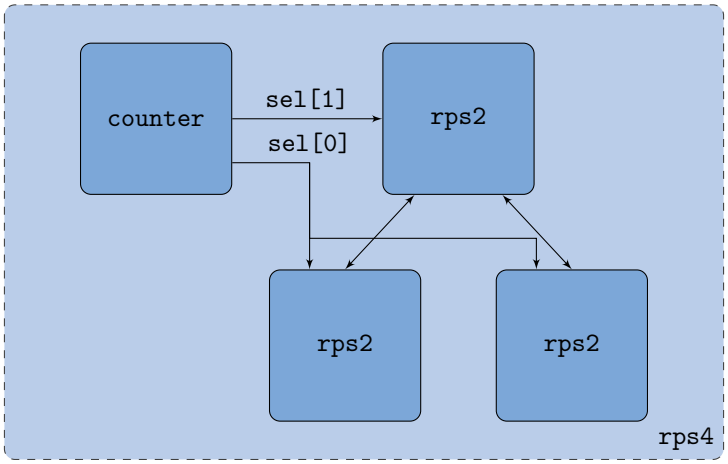

Figure 4: Submodules of the 4-bit rotating priority selector

```
module rps4(
        input               clock,
        input               reset,
        input        [3:0] req,
        input               en,
        output logic [3:0] gnt,
        output logic [1:0] count
);
```

A testbench, named `testD.v`, has been provided with this project. Be sure you've followed the directions above about which bit of the `sel` bus to route where and exactly how this bus works. In all cases, the testbench needs to be passed...

2.5 Additional Thoughts

If you get extra time, here are some things that you *should* look into. No points will be given for any of this.

- What is the `correct` signal doing in the testbench?
- In `testD.v`, there is a #6. What happens if you replace that with a #10? Why?
- Using a counter, you could do exhaustive testing (test every possible case) for the fixed priority devices. You could do it for the rotating ones too, but it would be harder. In general, when is exhaustive testing *not* a good idea?
- There is a `$random` function provided in Verilog for testing, which can be used to generate random test vectors. Read about it online and try to understand how it works. Why would you want to use this function?

3 Submission

After you are confident in your solution, make sure you have the following files in your directory

1. `P1a.v`
2. `P1b.v`

3. `P1c.v`
4. `P1d.v`
5. `testC.v`

Now, go up one directory (`cd ..`) and run:

```
/afs/umich.edu/user/j/b/jbbeau/Public/470submit -p1 <directory>
```

Note that the script takes a directory name as an argument, not an absolute path, so **you must run it from one level above your project 1 directory**.

At this point, you should see something similar to:

```
Submitting files in project1/
--- submitting project1/
--- submitting project1/Makefile
--- submitting project1/P1a.v
--- submitting project1/P1b.v
--- submitting project1/And.v
--- submitting project1/testAND.v
--- submitting project1/testD.v
--- submitting project1/test.v
--- submitting project1/P1c.v
--- submitting project1/P1d.v
--- submitting project1/testC.v
Submitted.
```

If there is a problem, it will be printed to the screen. Shortly after submission you should receive an email telling you if your submission passed the basic tests. This is not an autograder, so it's simply telling you that your design built successfully on the grading machine and that the module declarations looked approximately correct. If your submission didn't pass, copy the error message into an email to the course staff, and we will attempt to help.

Remember that for all projects in this class, you design must have proper simulation output after synthesis as well as before.