Daily Design Problems

Design Problem #1 (due Fri. Jan 9)

- Design the lowest cost 4-bit comparator that you can
  - Costs:
    - 2 : Inverter
    - 3: 2-input NAND, NOR
    - 4: 2-1 Mux, 3-input NAND, NOR
    - 6: 4-1 Mux, 2-4 Decoder, 4-input NAND, NOR
    - 10: 8-1 Mux, 3-8 Decoder

Design Problem #2 (due Monday, Jan 12)

- Turn in a tentative solution for the 7-segment design problem for Lab 2

Design a circuit that translates a 4-bit binary code to a 7-segment display code. You will only have to decode the digits 0-9; you may display anything for the inputs 10-15. Shown below is how the 7-segments are labeled by convention, and how the digits are formed.

```
  a      faaab    b    aaab    aaab   f   b   faaa    faaa    aaab   faaab   faaab
f   b    f   b    b       b       b   f   b   f       f          b   f   b   f   b
  g                    ggg     ggg     ggg     ggg     ggg            ggg     ggg
e   c    e   c    c   e           c       c       c   e   c      c   e   c       c
  d      edddc    c   eddd    dddc        c   dddc    edddc      c   edddc   dddc
```

You must design your circuit using only the logic gates in your design kits. This includes the following:

'00 : 4x 2-input NANDs
'02 : 4x 2-input NORs
'04 : 6x inverters
'08: 4x 2-input AND
'10: 3x 3-input NAND
'20: 2x 4-input NAND
'138 : 3-8 decoder (with enable)
'139 : 2x 2-4 decoder
'151: 8-1 multiplexer
'153: 2x 4-1 multiplexer
'157: 4x 2-1 multiplexer
'163: 4-bit counter (for testing)

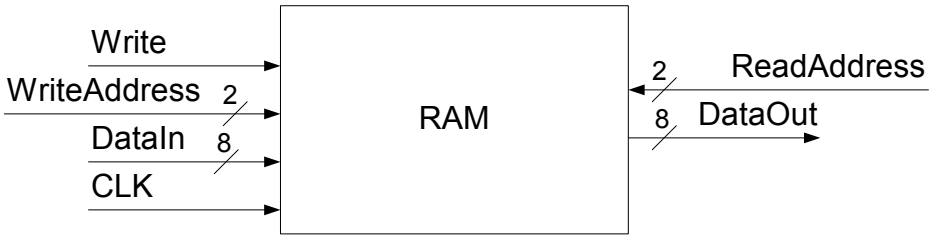
Design Problem #3 (due Wed, Jan 21)

An arbiter receives a request signal (REQi) from N modules, numbered 0 to N-1, each requesting exclusive access to a resource such as a bus or memory. The arbiter sends each of the N modules a corresponding grant (GNTi) signal indicating whether the request was granted. The arbiter must choose exactly one of the modules that asserts a request, and assert the corresponding grant signal. The decision is made by giving the lowest numbered modules the highest priority. Sketch the design of this arbiter circuit so that it has O(logN) delay.

[Extra credit] How would you implement this circuit if the arbiter had to use round-robin priority? That is, the ordering is the same (with wraparound) but the previous grant signal (oldGNTi) determines which module has the lowest priority.

Design Problem #4 (due Monday, Jan 26)

- Using registers, multiplexors and decoders, design a RAM with four entries
  - The ReadAddress indicates which entry is read to the DataOut
    - DataOut changes when the ReadAddress changes (asynchronous)
  - The WriteAddress indicates which entry is written from DataIn
  - The Write control signal indicates whether a write should be done
    - The write takes place on the CLK edge (synchronous)



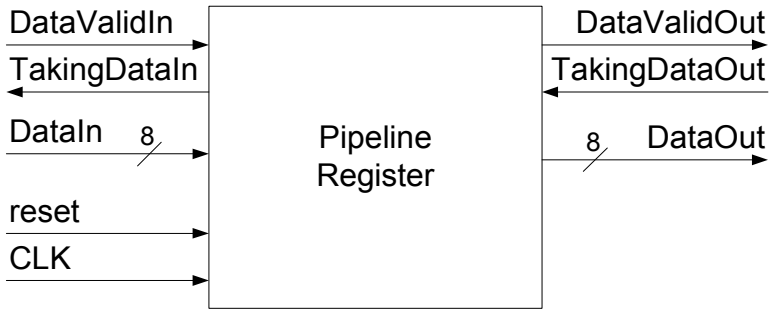
Design Problem #5 (due Wednesday, Jan 28)

- Using the memory you designed in Problem #4, design a FIFO
  - DataIn is added to the tail of the FIFO when Write is asserted (synchronous - happens on the CLK edge)
  - The head of the FIFO is always available on DataOut
    - the current head is removed when Read is asserted (synchronous - happens on the CLK edge)
  - reset causes the FIFO to become empty (synchronous)
  - Empty is 1 when the FIFO is empty, Full is 1 when the FIFO is full
    - the N-1 hack for Full is fine
  - You may use multiplexors, decoders, adders, counters, comparators, ...

Design Problem #6 (due Wednesday, Feb. 4)

- Design a pipeline register that uses the simple DataValid/TakingData handshake on both the input and output. This pipeline register acts like a FIFO with 2 entries. When it has one value, it can both send data out while receiving data in and thus acts just like a register. But if it is empty, then it does not send data, and if it is full, it does not take data.

  Try to come up with as simple a design as you can for this pipeline register. That is, don't use memory and counters like you did for the FIFO.



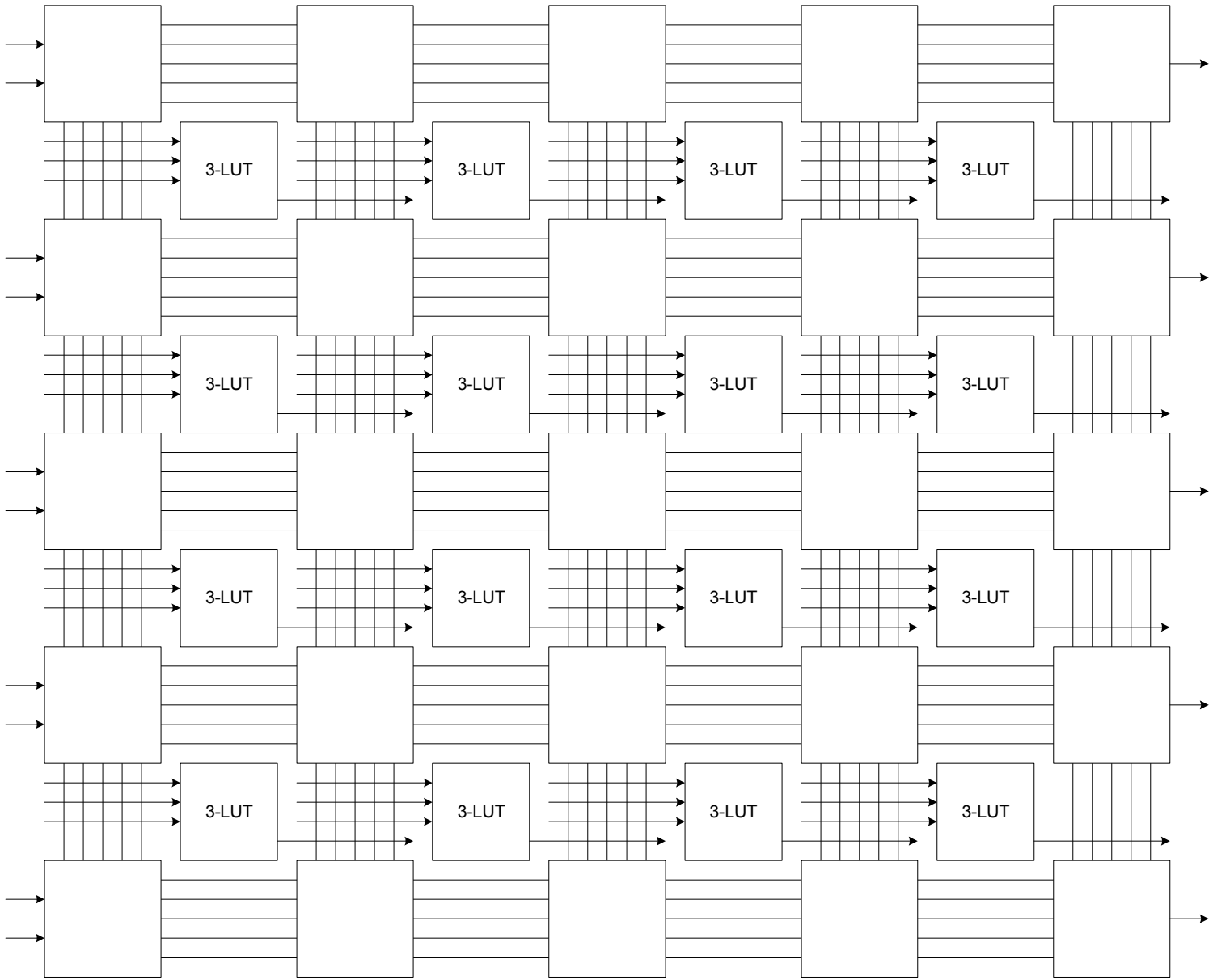
Design Problem #7 (due Friday, Feb. 13)

- On the next page is a simple FPGA architecture that uses 3-input LUTs. Implement a 4-bit comparator, which compares two 4-bit unsigned numbers and produces a greater-than output, using this array. Solve this problem using the following steps, showing all your work. (Hint: you don't need close to all the LUTs.)

  a) Factor the comparator function into a multi-level logic circuit, each node of which is a 3 (max) input function. Name each of these functions using a letter that represents the output and give the function each computes.

  b) Place each of the functions from part a in the FPGA array by writing the name in the appropriate 3-LUT.

  c) Route the circuit together using the interconnection network. Use an X to show the connection of each input and output of the 3-LUTs used. Each of the switch boxes at the intersection of the row and column channels can connect each wire on one side to a wire on each of the other sides. These connections can be made in either direction. For each connection that you use in a switch box, draw an arrow to show the connection. Circuit inputs arrive at the left and outputs leave at the right.


3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT
3-LUT

Design Problem #8

- Consider the problem of sending data from one system to another when the two systems use different clocks. There are many problems with this kind of communication where you don't know how fast the clocks are running. Here are some ideas about communicating to get you started thinking about this. For each idea, list as many problems as you can that may make it fail.
- *Do not write a book. Keep your answers brief and don't spend too much time on this.*
  - 1) Send a new data value on each sender clock, using as many wires as there are bits. (This works fine for synchronous systems – what problems can happen here?)
  - 2) Same as 1, but make sure the sender clock is slower than the receiver clock.
  - 3) Same as 1, but send the sender's clock along with the data.
  - 4) Same as 1, but use two control signals 'DataValid' and 'DataTaken' to establish a handshake between the two systems. That is, don't send new data until the receiver has asserted DataTaken.

Design Problem #9

- Consider the problem of implementing a 2D Waveguide Mesh
  - (Perry p.134)
- This computation can be done spatially, ie. in parallel with a lot of identical units
  - But we run out of hardware
  - And we compute much faster than we need to (1000x)
- How can we reuse hardware so that we can simulate large meshes with just a little hardware and more time?
- How can we extend this to simulation HUGE meshes, using both lots of hardware and lots of time?