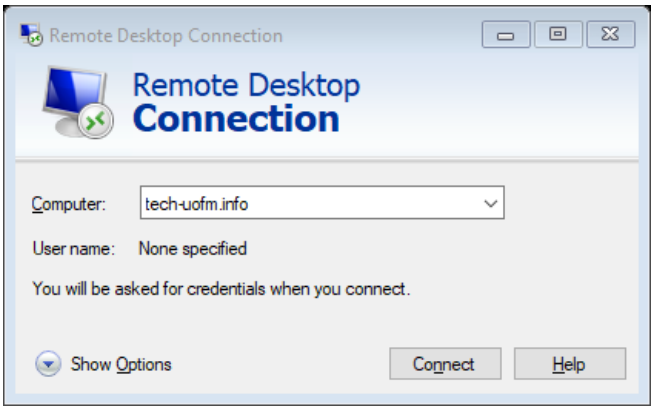
TECH 3233

Lecture #2 and #3
By Daniel Kohn (University of Memphis)
v2.0

The Simulations below are written for HADES which is a java based program. In the past, you could run these via a web browser, but due to security concerns almost all web browsers have that feature turned off.

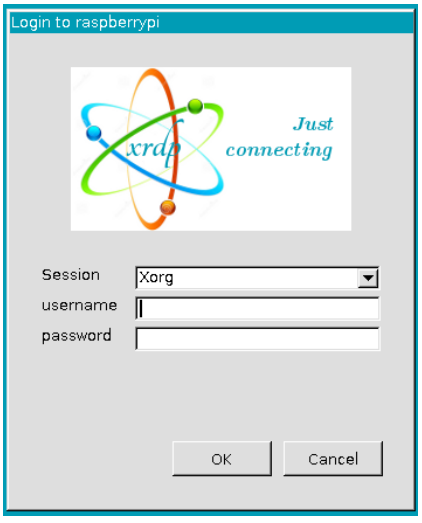
Due to this, the instructor has set up a server just for these simulations. To access, follow the following steps:

On a Win10 Machine, type into the search (on the task bar) “Remote Desktop Connection” (without the quotes) and open the program. The following window should appear:



In the textbox, type in tech-uofm.info as shown and then click on the Connect button. You might get a warning saying “the identity of the remote computer cannot be verified”, click on “Yes” to continue.

You should then get a log in screen that looks like:



For your login use your last name (all lower case), and for your password use the last 4 digits of your U#.

Once you are logged in you should see a Linux desktop like the following:

Now click on the terminal program by clicking on the  icon. This will bring up a command line interface.

You need to type in two commands (they are case sensitive and each ending each with an enter):

cd Hades

java -jar hades.jar

This will bring up the simulation software. To load the various simulations, you will use the menu at the top of the window FILE | Open. The file name of each simulation, if it exists, will be in [] after the picture caption. If there is a "/" then the file is in a subdirectory.

You can use the VIEW menu to zoom, and Layers | all the above layers to display labels on the simulation (most of the pictures below have this turned on)

When you are done with your session, click on the  icon and select Logout | Logout (note – other options will disrupt other users)

D-FF

In TECH 3232 you learned about the D Flip Flop. As you can see in dff.hds Data (D) can change, but until the Clock (C) is a rising edge, D does not get copied to Q. The data, is then stored in Q until the next rising edge of the clock occurs (overwriting the previously stored data.

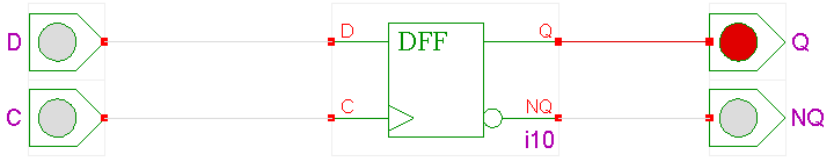


Figure 1- D-Flip Flop [file: dff.hds - note you must toggle the clock at least one time to reset the d-ff]

Register

But in computing, we need to store Bytes (and Words) not just bits. So we need to create a circuit that stores 8 bits on one control clock. Hence the Register (register.hds):

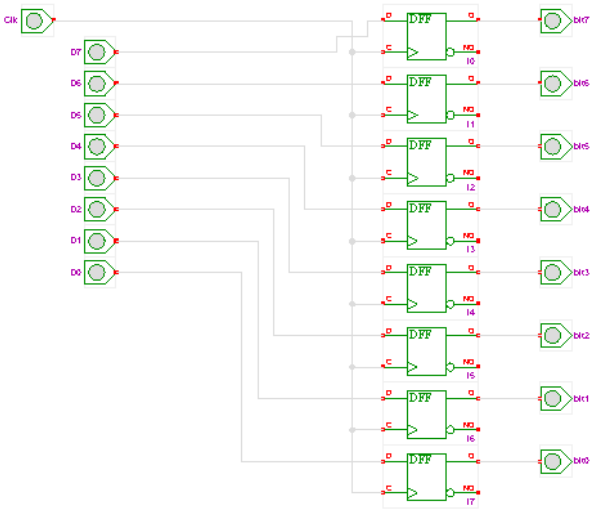


Figure 2- Register [tri-state_registers/register.hds- note you must toggle the clock at least one time to reset the register]

Here we have 8 bits of data coming in (D0-D7) and one Clock (clk). When the Clock does a rising edge, the data is copied to the output and stored until the next rising edge of the clock occurs, overwriting the data.

In the second simulation of a register, we replace binary inputs, with hex inputs and the output goes to hex displays (register-hex.hds).

Tri-State Buffer

But computers need more than one byte of data to operate, but there is an issues that must be addressed first. Let's say we want to store and display 2 bytes of data on a display (one after the other).

We would need to output the data from one register, then the next to the SAME DISPLAY. This would mean that we would need to connect D0 of both register one and D0 of register two to the same place (the input of the display). But what happens when you connect two outputs from digital logic to the same place.

Let's take the example below:

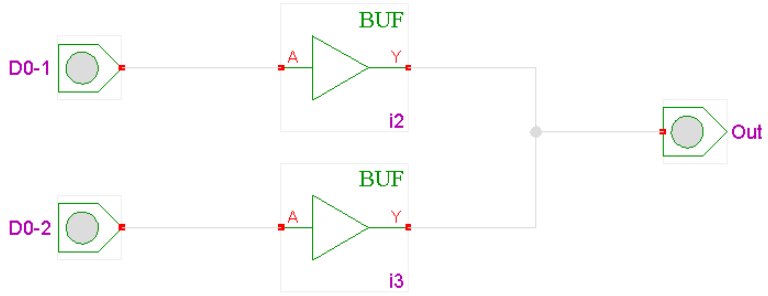


Figure 3- Buffer Example (both off) [2-Buffers.hds]

The circuit contains two buffers (note they are NOT inverters) connected to the same output. Currently both inputs are off, so the outputs of both buffers are LOW (Logic 0 or 0V). So we get a Low out

When both inputs are turned on, both outputs are HIGH (Logic 1 or 5V) (2-Buffers.hds)

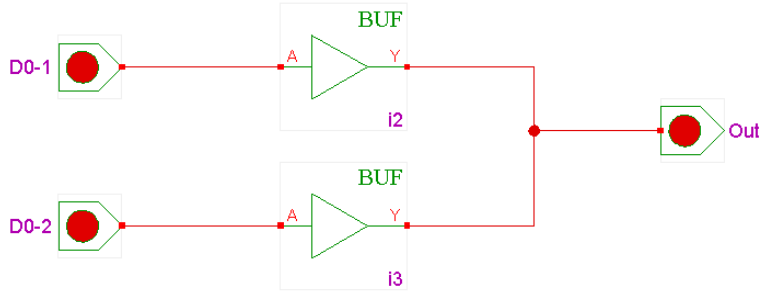


Figure 4 – Buffer Example (both on)

Currently both inputs are on, so the outputs of both buffers are nigh (Logic 1 or 5V). So we get a High out.

But what happens when one input is HIGH and the other is LOW….as follows:

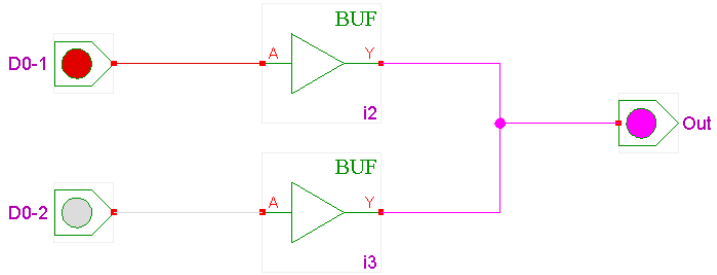


Figure 5- One on One off (SHORT Circuit!)

Since D0-1 is High (and the buffer output is HIGH or 5V) and D0-2 is Low (and its buffer produces a LOW output or a GND), this means that a 5V is connected to GND and thus we have a SHORT CIRCUIT (hence the pink/fuchsia color on the output).

So we need a device that allows us to select which bit we want to appear while DISCONNECTING the output of the other bit, so it will not interfere.

What if we added a switch to each buffer's output like so:

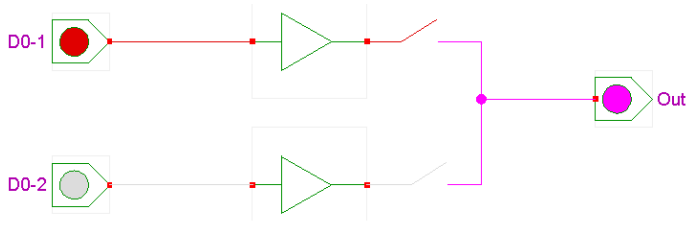


Figure 6- What if?

Now if we can control each switch we can then select which output we wish to see on the output. Memory does this, but uses ELECTRONIC Switches that are either closed, or HIGH IMPEDENCE (aka High-Z) but these can be controlled by a separate electronic signal. This is known as a Tri-State Buffer.

Below is an inverting Tri state buffer (so the input is inverted to the output, but goes high-z when the Output_Enable pin is inactive (in this case HIGH) (tri-state.hds).

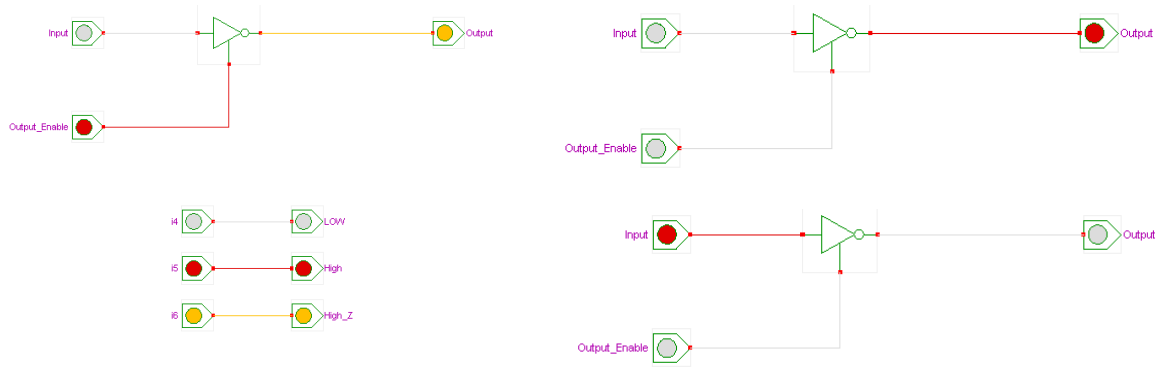


Figure 7- Tri-State Buffer [tri-state_registers/tri-state.hds]

Now that we can disconnect a register contents from it's output pin, we can now create two register that output to the same display. This can be seen in reg-tri.hds.

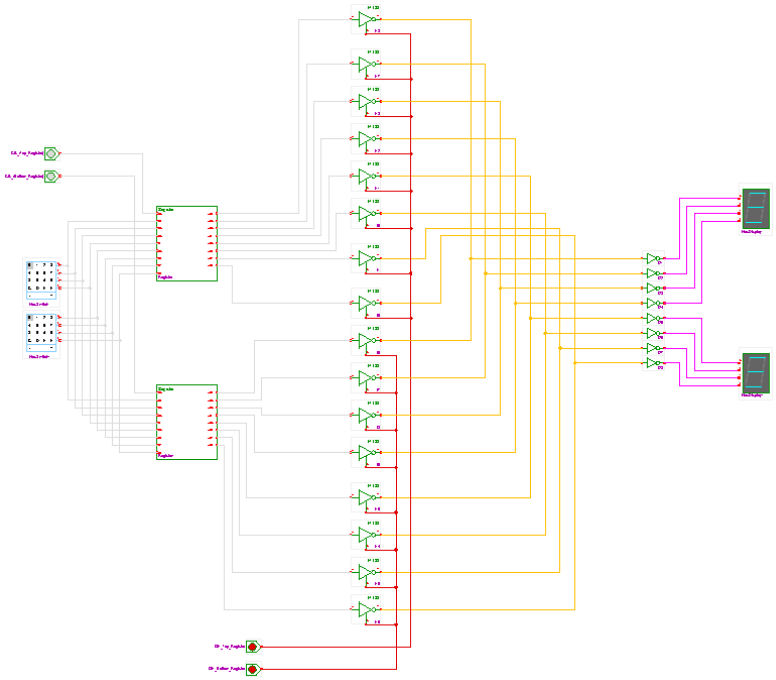


Figure 8 - Two Registers with one output display [tri-state_registers/reg-tri.hds – note you need to toggle both Clk's to clear errors before proceeding]

Now we can store different values in the two 8-bit registers (rectangles) and control when those values appear on the HEX Displays on the Right (using the two control lines at the bottom of the diagram). The two clocks independently tell the registers when to store the data (hex inputs on the left).

Demux

For a decent size memory IC we need to be able to select between may different memory locations (registers). So we need a circuit that selects one of many. For this we use the Demux circuit we mentioned last semester (demux1_4.hds):

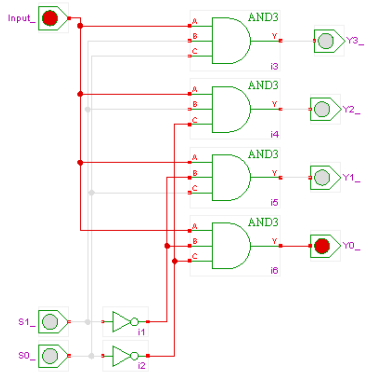


Figure 9 – Demux [mux_demux_memory/demux1_4.hds note – turn on and leave on input]

We leave the input on all the time for this application, and use S0 and S1 to select which output we want to activate. Since we have 2 bits as selects, we have the ability to choose between the 4 outputs (00 binary, to 11 binary or 0->3 or 4 different outputs).

This circuit can be expanded easily. Each time we add a new S line, we double the number of possible outputs. So adding an S2 (and the appropriate circuitry) we would double the Y's out (now going from binary 000 – zero, to binary 111 or seven).

Memory

Now we have all the components we need to make a much larger memory IC. We will start with a ROM (Read Only Memory) IC as an example (rom.hds)

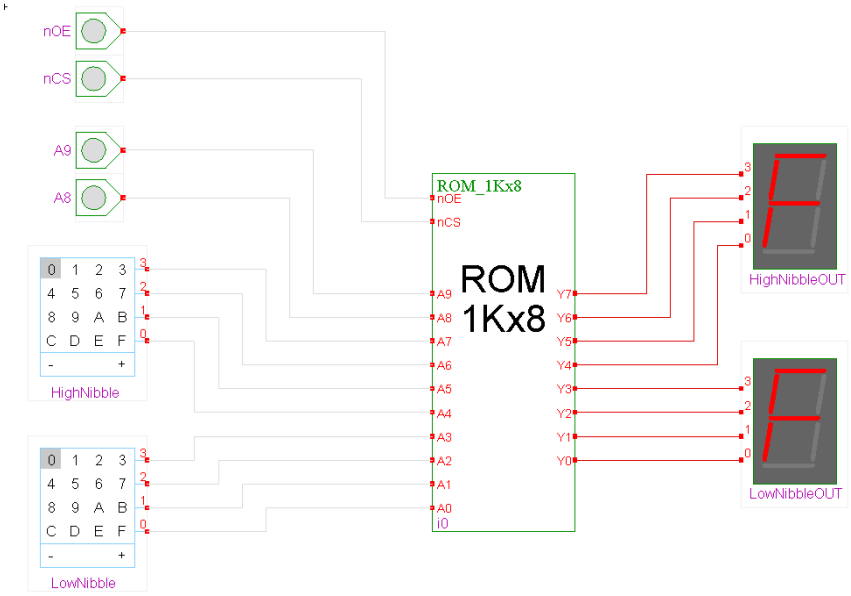


Figure 10 – Rom [rom/rom.hds]

So now we have two nibbles (high and low) with two extra bits (A8, A9) representing the address. nCS and nOE enable the output of the rom (and must be low…hence the "n"). If either pin is high, the output will be disconnected from the output pins (going into the High Z mode) and disconnecting the IC from the output bus.

You should be able to get various memory locations by adjusting the Address lines (A0..A9) as you wish in the simulations (and is the same as Lab #1).

More Memory

But this is only 1K of memory. Most computers need more memory than that. So how do we add additional memory? Well first we need additional Address lines. Each time we add one address line we can double the memory we can address. So if we add one address line the to the circuit above, we can use that line to control one of the nCS lines (sending it though an inverter for the 1 IC and directly to the address line for the 0 IC). So when A10 is off, it would active the IC 0 (since A10 connected to nCS would activate the output of the 0 IC) and when A10 is on, it is inverted so the 0 IC gets a 1 to nCS turning it's output off, and 1 IC gets a 0 to nCS (though the Inverter) to turn out its output.

Figure 11- Hex Dump

Taking it a bit further, we can add two address lines, going though a demux to select 1 of 4 memory ICs to output their values. Note that A0..A9 are connected to each IC as the ADDRESS BUS, and A10 and A11 go though the demux to select the IC to use. This is shown in the following example (romx4.hds)

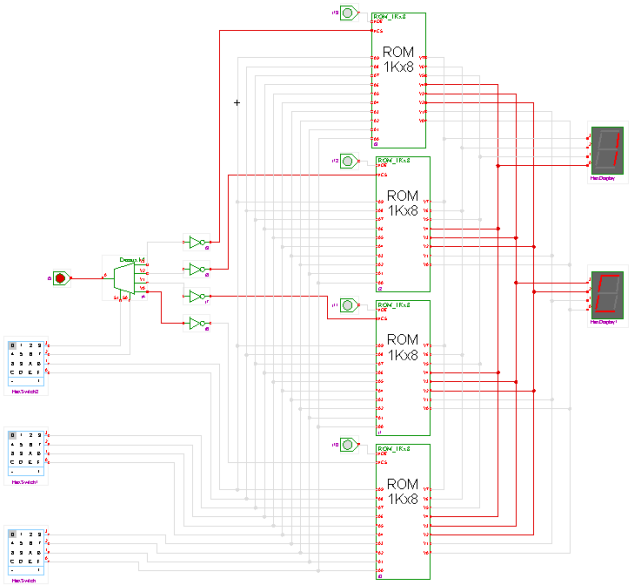


Figure 12- 4 Roms [romx4/memory.hds]

RAM Memory

Lastly we get to RAM (Random Access Memory) which can be read from or written to. Lets look at a 4 bit by 4 location RAM (ram.hds):

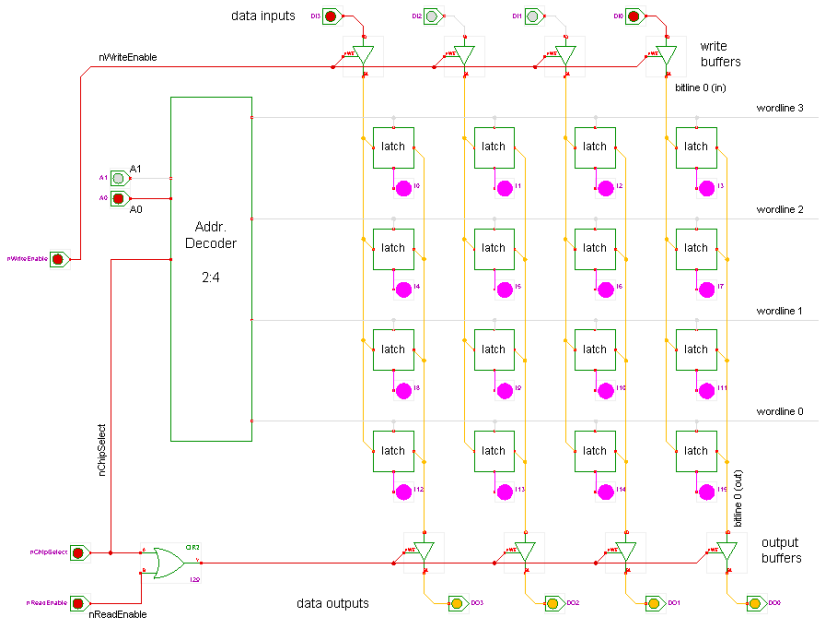


Figure 13- Ram Example [ram/ram.hds]

Note that when the simulation starts, the memory (latches at wordline 0 .. 3) are in an unknown state, since we have yet to write to those addresses.

To **write** to the address:

1. Set the address lines (where to store)
2. Set the Data lines at the top of the screen (what to store)
3. Active nChipSelect
4. Activate the nWriteEnable

Once this is done the value will be stored in the address (wordline) you set in step one. Undo steps 4 and 3 and you can then start again.

To **read** from an address:

1. Set the address (where to read from)
2. Active nChipSelect
3. Activate nReadEnable

And the data will appear on the output.

NOTE D0..D4 at the top of the page and D0..D4 at the bottom of the page on a REAL Ram IC are the same PIN (just cant be done in this simulator).

RAM IC

Our last simulation shows a 256x8 ram IC (ram256x8.hds)

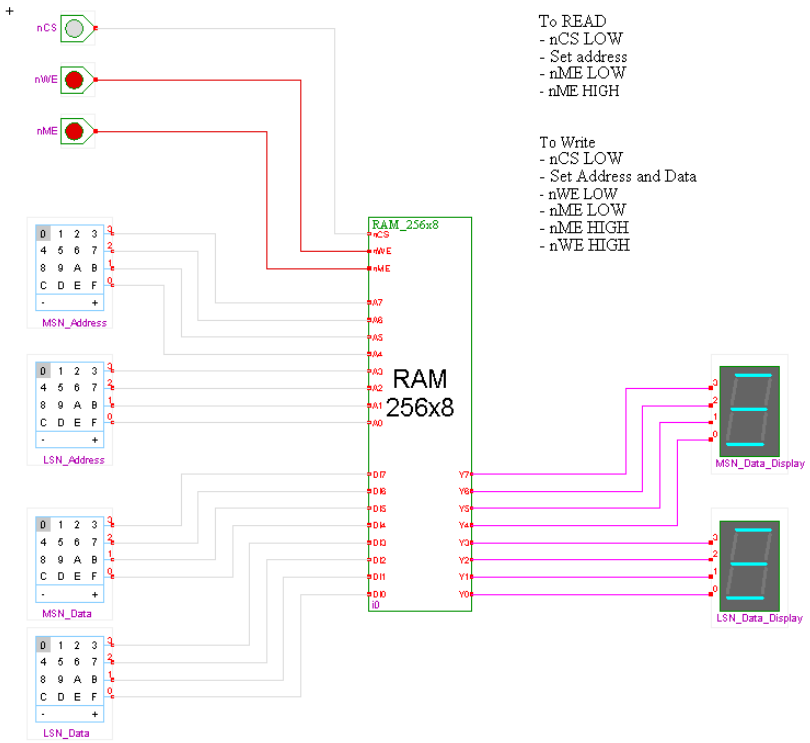


Figure 14 - Ram IC [ram256x8/ram256x8.hds]

You can write and read as shown in the simulation (upper right hand side). You can get a hex dump by a right-click | edit as you did before.

Closing Notes

For Computers to work you need both RAM and ROM, since ram is wiped when the power is turned off, computers need to know what to do when they power on, so every processor needs a program stored in ROM to say what to do on power up and the CPU must execute that code on power up or reset.

Also note that the above really describes STATIC MEMORY. This old style memory is slower than modern dynamic memory, but shows the basic concepts of the Data Bus (D0..D7), Address Bus (A0..Ax) and control line.