Lesson 3: Busses and Address decoding Version 1.05

Busses:

In the previous lesson we discussed the *address bus*. You might wonder "what exactly is a bus?", and that's a good question. A bus is simply a group of one or more wires on which electrical signals are transmitted. Buses usually have multiple wires, in the case of the Atari Centipede PCB and the 6502 CPU, there are 16 wires that make up the *address bus*. Each wire carries it's own signal (0V or +5V) and the combination of all the wires represent some information that needs to be transmitted. One other feature of a bus is that multiple different devices are attached to a bus. At any given time one or more devices can *READ* data on the bus. However only ONE device can ever *write* data to the bus at any given moment in time. If multiple devices try to write data to the bus, the signals corrupt each other and the data is invalid. Those of you who are familiar with old Ethernet networks that used *hubs*, this is exactly what happened when a *collision* occurred. Old hub based networks are a bus topology, and all hosts can read the data on the network, but only one is allowed to talk at any given time without causing issues.

Busses and Tri-state devices:

If you were paying attention you may have noticed a few conflicting points.

1. TTL chips provide/send signals by pulling the ouput wires to GROUND, or setting the output wires to +5V.
2. Busses often have more than 1 device/chip attached.

You might be asking the question “if multiple devices are outputting either +5V or GROUND, and they both are signals, when aren’t all the devices talking and causing issues?” The answer is YES! If all a device can do is output a +5V or GROUND which are both valid signals, then each device is trying to alter the bus to whatever state it’s outputting, and that’s going to cause issues. However, there is a solution to this problem.

Tri-state devices:

To solve the problem described above, many TTL devices are actually *tri-state* devices. Rather than being able to only output 2 states, these devices have 3 states

- Chip *enabled* and output HIGH (+5V)
- Chip *enabled* and output LOW (GROUND)
- Chip *disabled* and output HIGH IMPEDANCE (often written **Z**)

When the chip is *enabled* it outputs one of the valid TTL logic states (+5V or GROUND). However the chip can be disabled. When it is in this state it goes into a *high impedance* mode which effectively removes it

from the bus. It is important to understand that *high impedance* is NOT 0V (GROUND) nor is it +5V, it is NEITHER state. In fact from the rest of the circuits point of view when a chip is in the high impedance state, it is like the chip is not even part of the circuit

> Tri-State device:
>
> A device that has 3 states, the normal logic states of HIGH and LOW, and a third state called *high impedance* that effectively disables the chip and makes it seem like it's not even in the circuit or on the bus at all.

The address decoding circuit:

Great! Since we have the tri-state devices this solves our problem of multiple devices attached to the same bus. All we have to do is ensure the devices are always in a *disabled* (*high impedance*) state, and only *enable* the chip that we wish to receive information from.

Now we just need a way (a circuit) to determine which chip whould be turned ON or *enabled*. This circuit needs to read some type of information (the address bus) and then *enable* the chip that is responsible for that address of memory. This circuit is called the *address decoding circuit* and is on the centipede-schematic-1.jpg image it is the circuit contained in the blue rectangle.

Exploring the address decoder circuit:

It is critical that the address decoding circuit works, other wise the system will not function properly. Some (most) systems have self test modes that will check RAM and ROM to determine whether they are working. However, if the address decoding circuit does not work, the right RAMS and ROMs will not be accessed at the right times and the self test can provide in-accurate results, they may lead to the system reporting RAMS or ROMS that do not work. Therefore before taking the results of a self test that states a bad RAM or ROM, you want to verify the operation of the address decoding circuit. Often if the RAM or ROM that is stated bad is the FIRST RAM or ROM the system checks, there is a decent chance that the address decoding circuit is bad.

Let's explore the address decoding circuit. Look at the centipede-schematic-1.jpg image. Find the chip in the address decoding circuit labeled with a green **D**. Notice this chip is labeled "J2 LS139". This means the chip referenced here is the chip on the board in position J2, and that chip is a 74LS139. Now it's time to use google to find a datasheet for the 74LS139 chip.

This chip is a 2-> 4 line demultiplexer the 74LS139 chip actually contains 2, 2->4 line demultiplexers, the portion of the address decoder shown by **D** only uses one of those two demultiplexers. Now what is a demultiplexer? Well the key to understanding it is the look at the state table (or function table) which defines what the chip will output based on the different possible inputs. The state tables is shown below.

DM74LS139

Inputs			Outputs			
Enable	Select					
G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

If you look at the "INPUT" side there are 3 different inputs pins. Pin 15 on J2 is the *enable* line and it corresponds to input G on the state table. Pin 14 corresponds to A, and pin 13 to B. The output side of the table shows you all the different possible outputs (Y0-Y3) depending on the different inputs of (A,B and G). By looking at the table you see that if the enable pin (G or pin 15 @ J2) is logic state HIGH (H) (disabled) then ALL of the outputs are logic state HIGH (H) REGARDLESS of the inputs of A and B. So in effect when this chip is *disabled* all of its outputs are HIGH.

However when the chip is *enabled* (G is LOW) the chip will output a LOW signal on EXACTLY one output line, all others lines will be output HIGH. The output line that will be LOW will be determined by the input of A and B. This chip effectively reads the inputs (which are ultimately run back to certain lines on the address bus, and based on which lines are active it will activate (send low) EXACTLY one of the output lines or NONE of the output lines. The function of this chip is actually to turn on 1 of the 4 ROMS on the Centipede PCB board (or ensure all ROMS are actually turned OFF, if the chip is disabled).

Let's look at the schematic again, you'll see the outputs (the right side of the chip) are labled ROM0, ROM1, ROM2, ROM3 (each with a line over them). These ultimately are the ROM select chips that are run to the individual ROMS "chip selects/enable" input. The line over the ROM0, ROM1, ROM2, ROM3 specifies an inverted condition, that is the ROM is *enabled* when the line is at logic level LOW rather than enabled on logic level HIGH.

So putting this all together we see that if the 74LS139 at J2 is disabled NO ROMS will be enabled. Whether J2 is enabled depends on the input to G (pin 15), this is actually the output of a few more chips, but for the purposes of this exercise all we need to know is that J2 will be enabled ONLY if the CPU is trying to READ data AND the address bus line 13 is logic state HIGH (binary 1).

Now when J2 is enabled then EXACTLY one of the ROMs will be enabled based on the inputs A (pin 14) and B (pin 13). Pin A runs directly to Address bus line 11, and pin B runs to address bus line 12. Therefore which ROM chip is enabled depends on the combination of address bus lines 11 and 12.

We can take this knowledge to try to understand when exactly each ROM chip will be activated. We know for ANY ROM to be active adresss bus 13 MUST be 1, then the ROM chip activated will be based on address bus lines 12 and 11.

Let's create a chart to see the different combinations of A, and B which the G being HIGH, and map them to the respective address lines. We will ignore AB15 and AB14 as on Centipede those lines actually are not even used.

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	0	0													0
1	0	1													1
1	1	0													2
1	1	1													3

We now start to get a picture of what addresses will activate each ROM. You see the only 3 address lines that are needed to select a ROM are AB13, AB12, AB11. These alone determine the ROM selected, the other address lines can be either 0 or 1 for any position. From that however we can figure out what ranges of memory each ROM responsed to. For example for ROM 0 to be active, AB13 MUST =1, AB12 MUST =0, and AB11 MUST=0 all other address lines are irrelevant. So the smallest address that ROM 0 would respond to are the addresses where all the lines AB0 – AB10 are 0

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	0	0	0	0	0	0	0	0	0	0	0	0	0		0

If we convert this into HEX we see that is address 0x2000. Now the highest address the ROM will respond to is where all the other address bits are 1's

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	0	0	1	1	1	1	1	1	1	1	1	1	1		0

This number in hex is 0x27FF. So the range of addresses that ROM0 responds to are 0x2000-0x27FF.

Repeat the process for each ROM

Start Address for ROM1

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	0	1													1

ROM1 Hex Start Address = _____

End Address for ROM1

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	0	1													1

ROM1 Hex End Address = _____

Start Address for ROM2

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	1	0													2

ROM2 Hex Start Address = _____

End Address for ROM2

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	1	0													2

ROM2 Hex End Address = _____

 Brian Brzezicki

Start Address for ROM3

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	1	1													3

ROM3 Hex Start Address = _____

End Address for ROM3

G (15)	B (13)	A (14)													
AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0		ROM
1	1	1													3

ROM3 Hex End Address = _____

You should find that that the following ROMS respond (or are *mapped*) into the following addresses.

ROM	Start Address	End Address
ROM0 (D1)	0x2000	0x27FF
ROM1 (E1)	0x2800	0x2FFF
ROM2 (F/H1)	0x3000	0x37FF
ROM3 (J1)	0x3800	0x3FFF

 Brian Brzezicki

Exercise: Using the Fluke 9010A and the logic probe to see the address decoding circuit work

In this exercise you will use the Fluke 9010A to read different addresses from the various ROMs on the Centipede PCB board. You will use the logic probe to see the changes to the output of J2 as different memory ranges are selected.

1. Setup the fluke and the logic probe properly (you should know how to do this correctly by now, but remember power the Fluke on FIRST before the Centipede PCB)
2. In the Fluke hit the "Setup" Button, choose "more" until "Active Line Force" is selected, choose "No"
3. Hit "Bus Test" to make sure all is working OK.
4. On the Fluke, hit the "read" button, choose the address 0x0000, and it "Loop". From what you should have learned NO ROMs are in this range so NONE of the ROM selects should be active (LOW) they all should be high. (You'll probalby notice that ROM3 is transistioning, ignore that for now)
5. Put the tip of the logic probe on each ROM selection output of J2 (pins 9,10,11,12) note they all are HIGH which is disable. You may also note that on the Atari Centipede PCB each ROM chip has it's own ROM select test point near the ROM, you may want to measure it here instead of the action pins on J2. (If you get confusing results from ROM3 select, ignore it for now)
6. Now on the Fluke request a memory address related to ROM0, let's "read" address 0x2000. Don't forget to hit the "Loop" button to make the read continuous. Put the tip of the logic probe to the ROM0 select (pin 12 on J2, or use the ROM0 test point near D1).
7. Notice thate ROM0 select pin is "flipping" signifying that the ROM0 is being enabled. (it is enabled on LOW)
8. Test the ROM select pins for ROM1, ROM2. Notice they are still high.
9. Now on the Fluke request a memory address related to ROM1, let's "read" address 0x2800. Don't forget to hit the "Loop" button to make the read continuous. Put the tip of the logic probe to the ROM1 select (pin 11 on J2, or use the ROM1 test point near E1).
10. Notice thate ROM1 select pin is "flipping" signifying that the ROM1 is being enabled. (it is enabled on LOW)
11. Test the ROM select pins for ROM0, ROM2. Notice they are still high. Do not test the select for ROM3 yet.
12. Test the ROM select pin for ROM3, this one is a little more interesting. On Centipede it is actually electrically hard coded to always be ON (low) except when it should be specifically off. (reading from another RAM or ROM etc, then it will go off (high). If you use a logic probe you will actually see a transition. However this transition is it actually going off, when the RAM is being read at 0x0000. From the perspective of the logic probe it's hard to tell the difference, you just see the transistion. If you use an oscilliscope you can see the ROM3 line transitions from high (off) as the RAM transistions low (on)
13. Try other addresses to verify that the ROMs are being enabled or disabled as you'd expect.

Challenge:

Now that you've learned this, look at chip H3 (74LS42) in the address decoding circuit. This chip is responsible for activating other components such as playfield RAM, the WATCHDOG and the POKEY. Based on what you learned above, try to determine what hex addresses activate the POKEY chip via H3. Use the fluke and the logic probe to verify your results. (answer at end of page)

Answer to challenge

The POKEY is enabled via H3 when the following address bits are set as follows ("- " is either 0 or 1)

AB13	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0
0	1	0	0	-	-	-	-	-	-	-	-	-	-

or hex addresses

0x1000 (when address AB0 - AB9 = 0) through 0x10FF (when address bits AB0-AB9 = 1)

Special thanks to KLOV users TROXEL and BARITONOMARCHETTO for reading through this guide, doing the exercises, catching many many typos, and generally making this a better document.