CheckPt2 is easy!!!

You get to listen to cool Super Mario Kart sound!

BUT………………

This lab can be very tricky.

BUT………………

Mark is here to help!

Introduction:

So what is lab about?

The end result:
You press a button on the controller and will hear one of Mario Kart music!

Only one sound will be played at a time.

We will have 6 MARIO KART sounds.

16 seconds of sound total.

4 2sec sound and 2 4sec sound.

Controller Interpreter:

2 objectives:
1. Figure out which button has been pressed.
2. Figure out which sound to play and enable PLAY signal.

PLAY = enable signal. (enables the Audio Module)

PLAY IS A PULSE SIGNAL!!! DEBOUNCE IT!!!!
What if you hold down your buttons?
You update your controller block 1000 times per second. (checkpt1. Each request/receive cycle = 1ms)
This means no two bits will be high at the same time.
What if you hold down your buttons?
That is why you debounce your signals.
Your play signal is LOW after 1 cycle.
Audio Controller ignores it.

Don’t worry about…… 😊

EPROM:
What is EPROM?
In checkPt2 we use EPROM to store sound data for sound effect of our video game!
Think of EPROM as a DATABASE you can retrieve information from.
We READ from EPROM in checkpt2.
There is no synchronization problems. Pretend EPROM works infinitely fast. You give it an address and the data will be available during the same clock cycle.

EPROM has 19 bits of address. $2^{19} = 512K$ bytes of data.
$<3$ bits to divide EPROM$<15$ bits of address$<1$ bit toggle$= 2^{3}$ bits $= 8$.
We are using the top 3 bits to divide the EPROM into 8 sections. This is because they are the 3 MS bits.

DAC Digital to Analog Converter:
We are using AD1866 convert digital signals to analog signal.
Very important:
You have 2 signals you have to worry about.
You control these two signals to control the DAC.
LL (Latch Left)
DL (Data Left)
The falling edge of LL cause the LAST 16 bits of data which clocked into the serial register to be shifted into the DACs, thereby updating DAC outputs.
DL is the data line. It contains the data inputs.
LL controls data input into the DAC.

Why is LL important?
Your data could be constantly changing. LL allows a user to latch only values that are important.

So the big idea is….
1. Shift 16 bits IN into SR inside the DAC
2. Once 16 bits are ready, let DAC play these values by lowering LL.

Lowering LL and drive data onto DL is in tricky synchronization is very important.
You want to lower the LL line immediately after the last bit has been shifted in, but BEFORE the next clock edge.
Otherwise you will shift in ONE junk bit for every 16 bits shifted. Shifting in bunch of 0s!!
This means 4096 junk bits are inserted for every 2 second sound.
BAD! BAD!

2 Solutions!
1. Invert the clock for the DAC. Phase shift DAC.
Or
2. Use a negative edge triggered FF on the signal sent to LL. This phase shift LL.
Make sure either the DAC or the rest of your project use a negative edge triggered clock.

Audio Module:
Communicates with the EPROM and Controller Interpreter.
* Use a state machine
You have 2 time cycles going on.
1. Every 1ms, you receive ONE button signal, then you play one of 6 sound available.
2. You output ONE sample every 16KHz. This 2nd time cycle is what your Audio Module works on. Each sample is 16 bits long. You need to read through 2 addresses for each sample.

3. This mini-time cycle runs until you have outputted to the DAC all of the bytes stored in the EPROM.

WIRE WRAPPING

* Lots of work!
* Check your discrete packs if nothing works.
* Don’t worry about interference among wires
* Keep your chips together. Save room for later chips.

3. Use some kind of counter to count through the bytes. Be careful about the 4 sec sounds.

Important:
You need some kind of RUNNING signal inside of Audio Module.

You can only play one sound at a time, you don’t want another play signal to interrupt while playing.

If (running) ignore another play signal.
Also only play if not running.

COOL! We are finished

GOOD LUCK GUYS!

Mark, Jeff, and Randy H.