ES 4 exam makeup

December 26, 2022

This makeup opportunity has two parts:

(a) First, you will work on the problems below, and submit your answers on Gradescope. These are design problems, more involved than a typical exam question and closer to what you'd expect in a lab project. You are welcome to use any tools and resources you like while developing your solution. You can use any of the course resources, look things up online, talk with classmates, simulate designs with GHDL or Modelsim, etc.

(b) Second, you'll sign up for a 15-minute zoom conversation where we'll talk about your work in part 1. You'll have some time to explain your solution, and we'll talk through anything that isn't quite right and try to improve it.

This isn't so much a "get points back" opportunity as it is a chance to show what you really do know in a different format with less time pressure.

Question 1: Edge counter

Draw a logic diagram for a circuit which counts the number of rising edges of an input signal. Let's call the signal `c`. Here's the catch: you're not allowed to use `c` as a clock. Instead, you'll need to use a separate clock (call it `clk`) as the clock input for every flip-flop in your design.[1]

You can assume that `clk` is much faster than the input signal `c` (at least 10× faster), so even if `c` changes rapidly, it will be high or low for multiple cycles of `clk`.

The circuit should be able to count to at least 11.

Question 2: PS/2 interface

Before USB, most keyboards used the PS/2 protocol (See https://en.wikipedia.org/wiki/PS/2_port for the history), which uses a simple serial interface with a clock and data signal.

When a key is pressed, the keyboard drives both the clock (`c` in the diagram below) and `data` to transmit a sequence of 11 bits. These bits are:

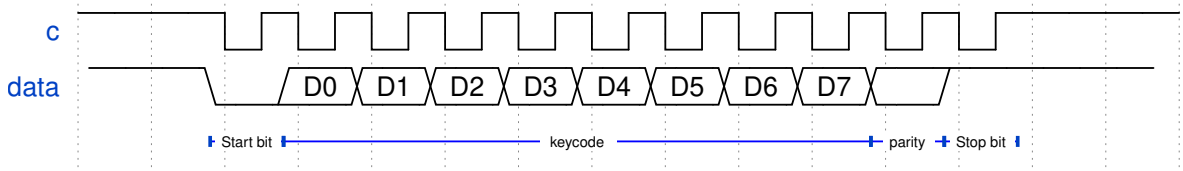
- A start bit, which is always 0. (The `data` line is high by default, so this can be used to detect the start of the transmission.)
- 8 data bits (least significant bit first)

[1] *Why?* This isn't just a contrived trick to make the problem hard (although it does make the problem harder). Remember that FPGAs are designed to have all of their flip-flops driven by a small number of shared clock signals (ideally one, and rarely more than four). Using other signals to drive flip-flops generally doesn't scale well.

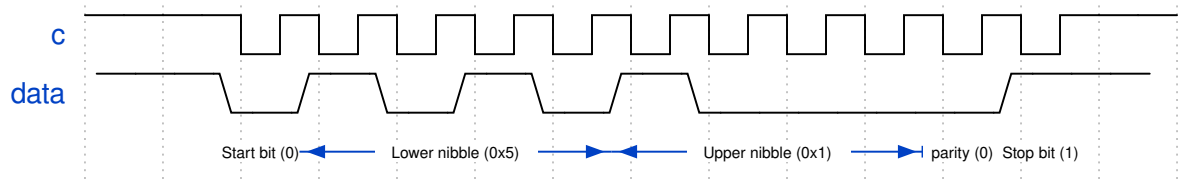
It also introduces problems with synchronization: if different sections of the design are driven by different clock signals, then there is the possibility of metastability issues anywhere a signal crosses from one clock domain to another.

- A parity bit, which is 0 when the number of 1s in the data is odd, and 1 otherwise. (This is known as "odd parity", because the data + parity bit together will have an odd number of 1s.)
- A stop bit, which is always 1.

These are annotated on the timing diagram below:



Here is an example of what is received for the 'Q' key, which has keycode `0x15`. Note that the values have to be read right-to-left because the least significant bit is transmitted first.



Your task is to design a digital circuit which can read the keycode when a key is pressed.

It should have the following inputs and outputs:

- Clock and data input from the PS/2 keyboard
- Fast clock signal input which drives all flip-flops in the design. Like the previous problem, you should not use the PS/2 clock as the clock signal for any flip-flops.
- 8-bit keycode output (you should not include the parity or start/stop bits in the output)
- Valid output, a single bit which indicates whether the keycode output is valid. Any downstream circuitry that uses the keycode needs to know when it is ok to read!

As you might suspect, the full protocol is a *little* more complex than this. There are a handful of keys with 2-byte keycodes, and there's a an additional byte that is transmitted when a key is released. To learn more, check out Ben Eater's excellent video about the PS/2 protocol: `https://youtu.be/7aXbh9VUB3U`. Your design does not need to handle these situations.

Evaluation

When we talk, I will be trying to figure out how well you understand the concepts that were covered on exam 2 – flip-flops, sequential circuits, and state machines. I don't have a precise rubric yet (this format is an experiment!) but I'll be looking for the following things:

- How well do you understand your solution? Can you explain why and how it works?
- Can you extend or modify the solution to handle other "what-if" cases?
- Do you understand digital design terminology? Can we have a conversation using precise technical language, or do you tend to misunderstand or misuse technical terms?

There are shortcuts to the correct answer (e.g., copying from a friend, or the internet, or writing VHDL code and looking at the netlist in Radiant), but there are no shortcuts to understanding. I do encourage you to talk with classmates, read whatever resources you find helpful, and to try modeling stuff in VHDL. But use these as aids to your learning, not in place of it! Take your time to think through the problems, build up your understanding of anything that you're fuzzy on, and come up with your own solution.