University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

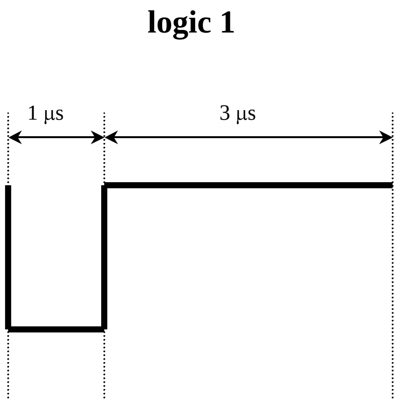
EECS 150 | R. H. Katz
Spring 2001 | P. Yan

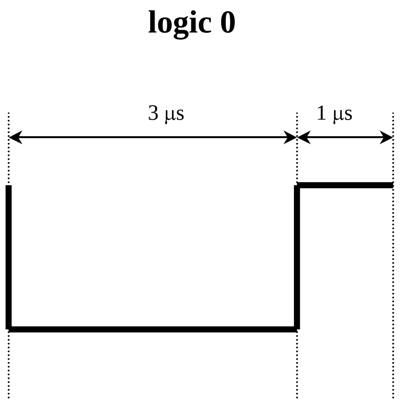
Project Checkpoint 1
Controller Interface

Controller Specifications

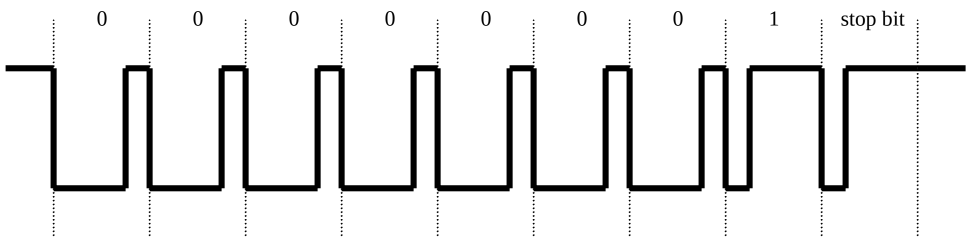
The Nintendo 64® controller has three terminals - VCC, GND, and a data line. The VCC and GND lines supply power, while the data line is a two-way transmission line that is used to communicate with the controller.

The data line is normally high (VCC), which indicates that the line is idle. Data bits are sent serially, with bits encoded as follows:





To receive the controller button information, the request byte 0x01 (hexadecimal) must first be sent to the controller. Also, a stop bit (logic 1) must be sent as a 9th bit of the transmission. Viewed on the oscilloscope, the request waveform should look like this:



Note that the most significant bit (MSB) of the request byte is sent first. Also, each bit is sent immediately after the previous bit, with no idle time in between bits.

After the request has been sent, the controller will respond with 32 bits of button information (and a stop bit at the end) within a few microseconds. (The response time is usually around 3 μs, but it varies depending on the controller. Some controllers may actually begin sending data while the stop bit of the request byte is still being sent! Be careful…)

Each button is sent as one bit, where a 0 indicates that the button is currently not being pressed and a 1 indicates that the button is being pressed. The button states are sent as a sequence of 32 bits in the following order:

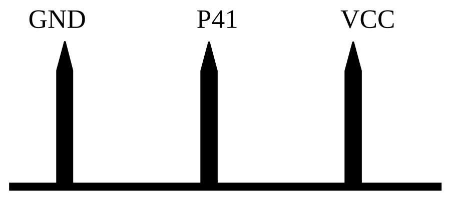
bit	button
0	A
1	B
2	Z
3	START
4	UP
5	DOWN
6	LEFT
7	RIGHT
8	[not used]
9	[not used]
10	L
11	R
12	C-UP
13	C-DOWN
14	C-LEFT
15	C-RIGHT
16 – 23	analog X-AXIS
24 – 31	analog Y-AXIS
32	[stop bit]

The state of the analog stick is sent as the last 16 bits, where each axis of the stick is sent as an 8 bit one's complement number. For example, on the x-axis, the value 0x00 would represent the middle, the value 0x7F would represent the analog stick being all the way to the right, and the value 0xFF would represent the stick being all the way to the left.

Prelab Preparations

This checkpoint requires three wire wraps, one for each terminal of the controller. The VCC and GND terminals should be connected to the VCC and GND of the board. The data line should be connected to pin 41 of the Xilinx 4010XL chip.

Three consecutive pins in the pin array (the upper half of the board with lots of pins) will be used to connect to the controller, so those three pins should be wire wrapped, in the following order:

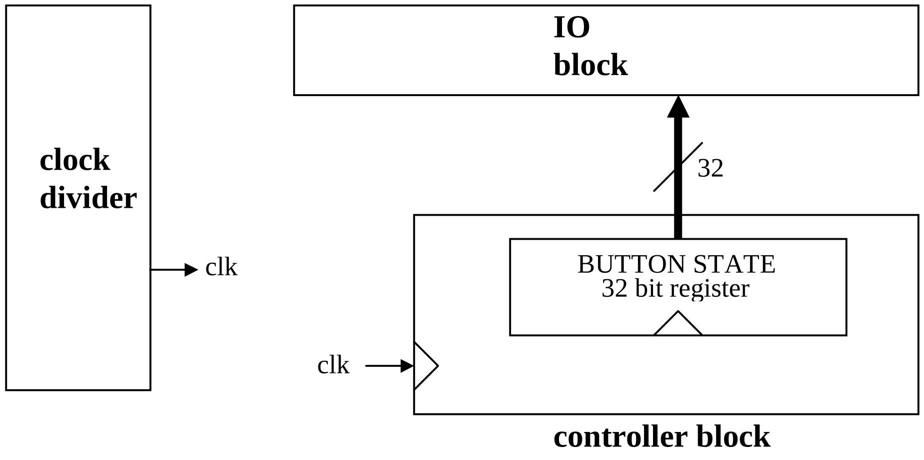


three consecutive pins

Design Structure

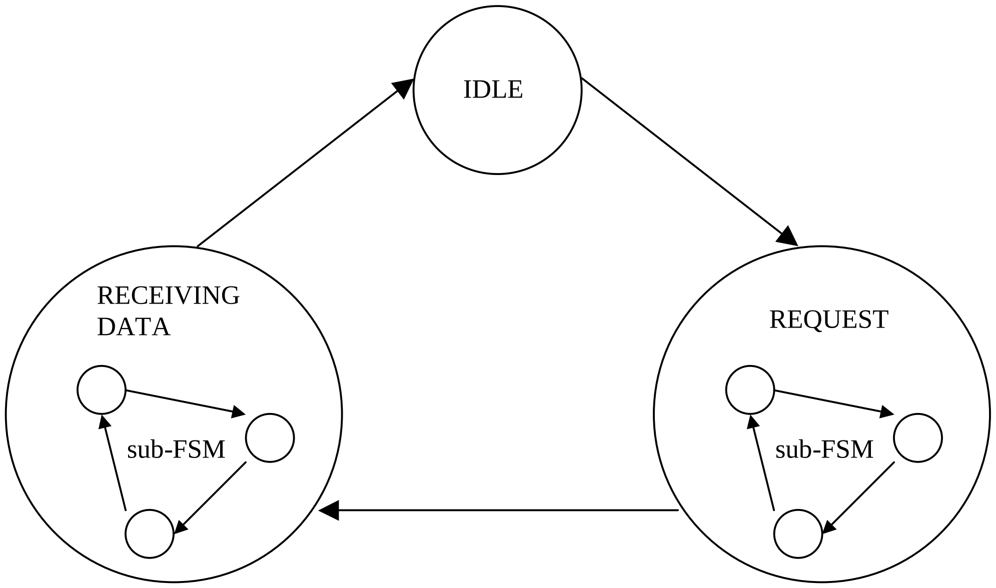
The goal of this checkpoint is to build an independent component of a videogame system. To accomplish this, the only interface to the rest of the system will be a 32 bit register which contains the state of the buttons.

For this checkpoint, the system outside of the controller block will only consist of an IO block (used to output to the LEDs) and a clock divider to provide the system with the necessary clocks (only a 4mhz clock is required for this checkpoint). The controller button signals will be output to the two seven-segment LED displays.



Controller State Machine

The top-level state machine to communicate with the controller is fairly simple, consisting of three general states – idle, request, and receiving-data.

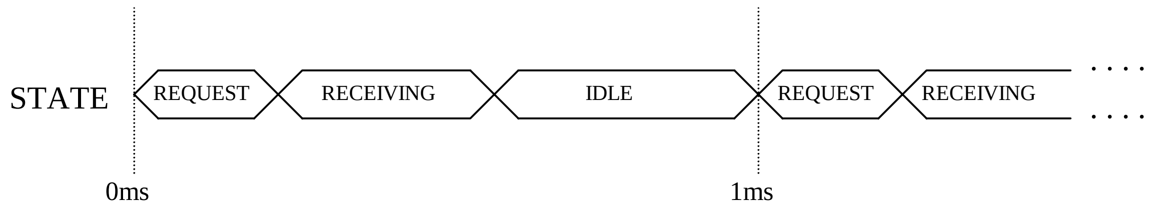


As seen in the diagram, the request and receive states actually consist of a sub-FSM, which looks similar to the top-level FSM (it's just a loop). You will need to design those two sub-FSMs.

Of course this is not the only way to design the controller interface. There may be other methods that yield more elegant designs; this design is merely a suggestion. But regardless, you will need a sub-FSM for the request and receive processes.

FSM Timing

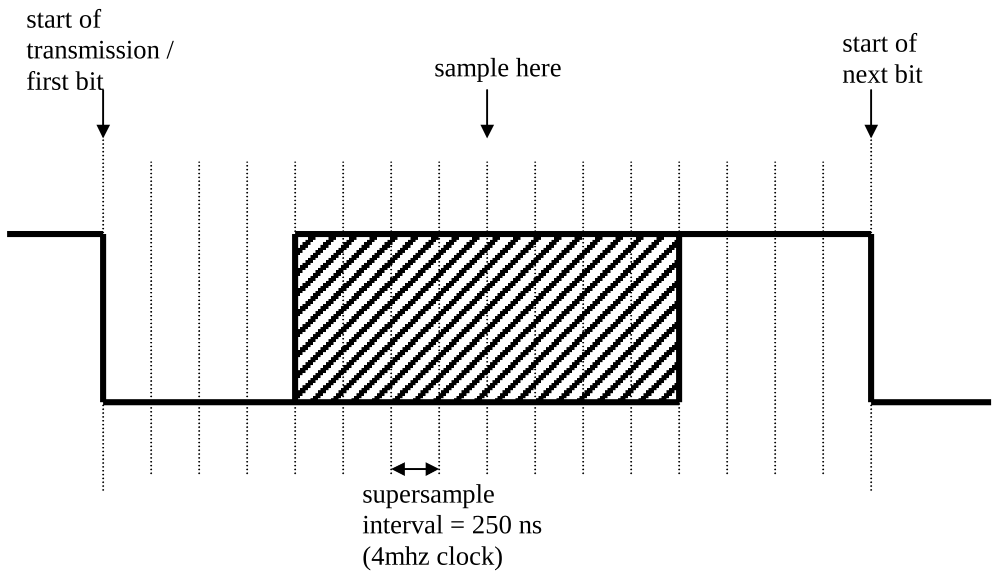
The top-level FSM should cycle every 1ms. It should be idle most of the time, but every 1ms it should send a request, then immediately receive the data that the controller sends, and then return to the idle state where it will wait until the next millisecond.



For sending the request signal, it is necessary to count 1µs intervals, since each bit consists of four 1µs sections. This can be done by counting four 4mhz clock cycles.

Receiving signals is much trickier, since it is asynchronous, i.e. the transmission might start at any point in time. Thus, an edge detector is necessary to detect the start of the transmission (a negative edge). Also, it is necessary to supersample the signal to detect the edge and to keep synchronized.

To determine whether each bit is a 0 or 1, we can sample in the middle of the 4µs interval of each bit, since 0 and 1 differ only in the middle two 1µs sections.


start of transmission / first bit
sample here
start of next bit
supersample interval = 250 ns (4mhz clock)

There is a tricky timing issue in the receiving stage that arises because the board clock is not exactly the same frequency as the controller clock. This may cause the FSM to become unsynchronized with the controller signal.

To get around this, your FSM should resynchronize at the start of each bit. Once a bit has been sampled, the state machine should wait for the next negative edge (the start of the next bit) and resynchronize with the signal.

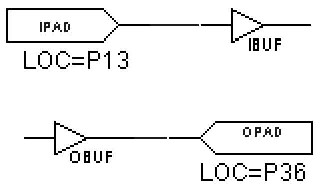
Design Implementation

IO Block

In order to input and output signals, there are several fundamental components necessary: IBUF and IPAD for input, OBUF and OPAD for output, IBUF and OBUFT and IOPAD for two-way signals (which is not used in the IO block, but in the controller block).

All of the pads (IPAD, OPAD, IOPAD) are used to make connections to pins. They must be configured by right clicking a pad and editing the symbol properties. A "LOC" parameter must be added, and its value should be "P41," for a connection to pin 41, or "P36" for a connection to pin 36, etc.

In addition to pads, each pad should have a corresponding buffer (IBUF, OBUF, or OBUFT). Connections should be made like this:



The IO block will take certain buttons' status and output them to the seven-segment LED displays. Here is a pin connection summary of which buttons' status will be sent to which pin (LED):

pin (LED)	button
39 (CR3-A)	UP
44 (CR3-G)	DOWN
40 (CR3-F)	LEFT
38 (CR3-B)	RIGHT
36 (CR3-C)	A
29 (CR3-E)	B
35 (CR3-D)	START
49 (CR4-A)	C-UP
51 (CR4-G)	C-DOWN
50 (CR4-F)	C-LEFT
48 (CR4-B)	C-RIGHT
45 (CR4-E)	L
47 (CR4-C)	R
46 (CR4-D)	Z

Remember that the LEDs are active low, so there must be an inverter before a signal gets output through the OBUF and OPAD.

Clock Divider

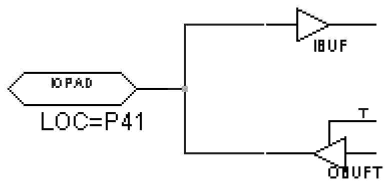
There is an onboard clock that runs at 16mhz, which is hardwired to pin 13 (an IPAD should be used to connect to this clock signal). A counter can be used to divide the clock in divisions of multiples of two. By using the 16mhz clock as the clock input for a counter and always enabling the counter, the LSB of the counter will be a 8mhz clock, the next bit will be a 4mhz clock, etc.

When generating clock signals, it is VERY IMPORTANT to put a BUFG before sending the clocks to components. This will prevent clock skew, which happens when different clocks become unsynchronized.

For this checkpoint we will only need a 4mhz clock, but in the future you may wish to add another clock frequency. Just remember to use a BUFG for each clock.

Controller Block

The controller block requires the use of a bi-directional data line which should be connected to pin 41. This type of IO connection is a little more complicated than normal input/output since it requires the use of IOPAD and OBUFT. Such a connection should be made like this:



Notice that the OBUFT is a tri-state buffer. Be aware that the control signal "T" is active low, so the output buffer will be enabled (i.e. signals will be output to the pad) when T = 0. When the IOPAD is not being used to output signals, the OBUFT should be turned off (to Z state).

To store the button state, a 32 bit parallel-loadable register should be used so that each button's state can be updated at the same time. This will also require that you have a temporary register to store bits as they are read from the data line.

Things To Do

- Create a clock divider - right now only a 4mhz clock is sufficient
- Create an IO block to output button state to LEDs
- Create the controller block

Extra Credit

There is a slight problem with this controller implementation, and it is that if you unplug the controller (or start your FSM without the controller plugged in), the FSM will become stuck or read incorrect data (if you have a different FSM design).

For extra credit, you must design your FSM so that it can detect whether the controller is plugged in so that you can unplug and plug in the controller at any time without affecting your FSM. You should do something reasonable if you unplug the controller while a button was held down. In this case, your button status register should probably be reset to all 0s.

Name ______________________________   Name ______________________________

Project Checkpoint 1
Checkoff Sheet

Design

Request sub-FSM state diagram	______________
Receive sub-FSM state diagram	______________

Implementation / schematics

Clock divider	______________
IO block	______________
Controller block	______________

Testing

Request waveform properly sent (oscilloscope) [controller response should appear after request]	______________
Everything works [LEDs light up when buttons pressed]	______________

finished 1st week (extra credit… yay!)	______________
finished 2nd week (normal credit)	______________

Extra credit – controller detection	______________