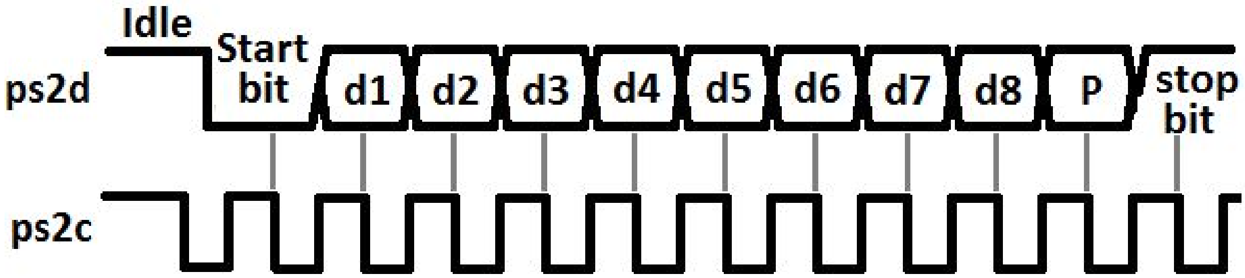
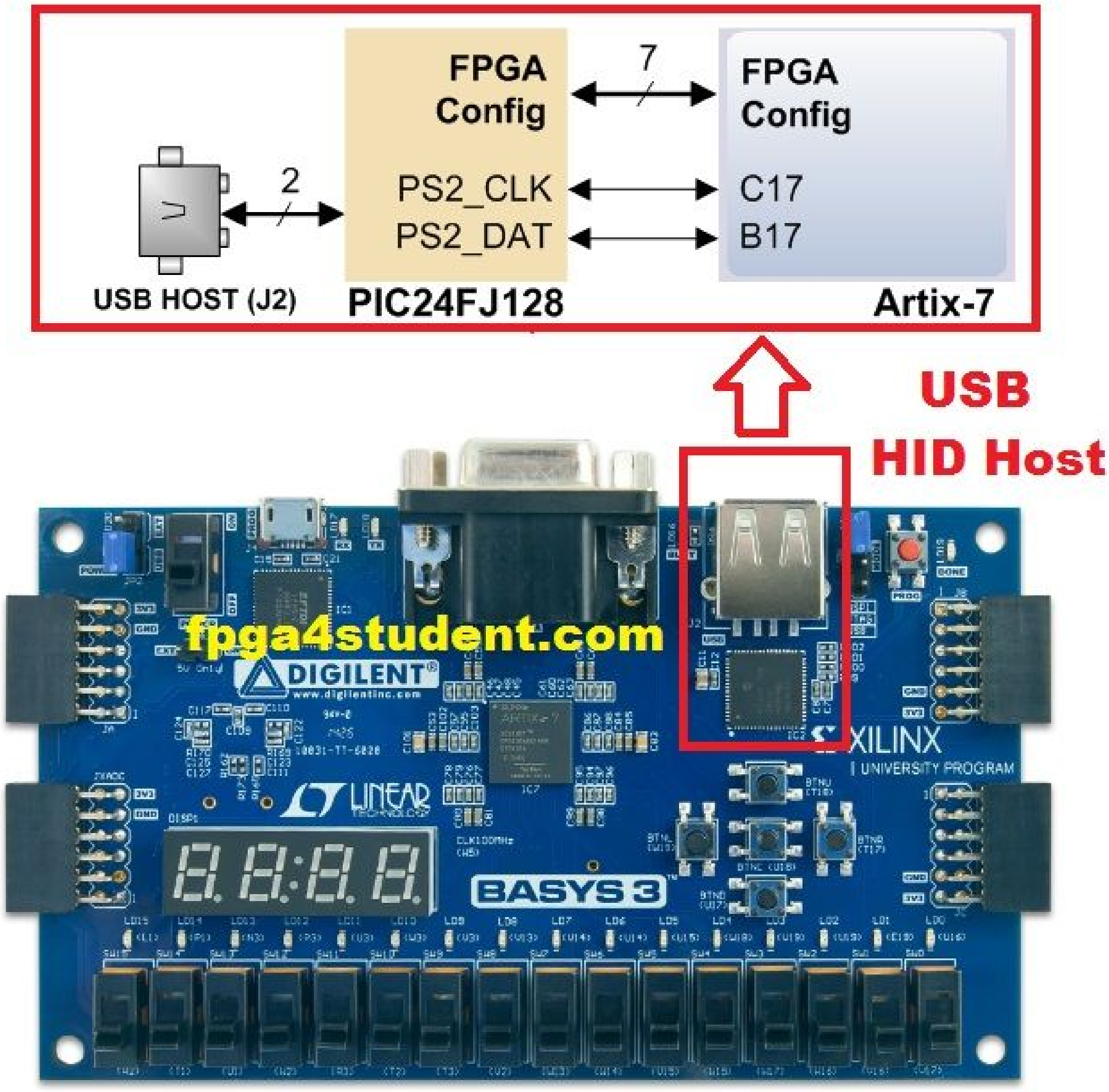
FPGA Keyboard Interface

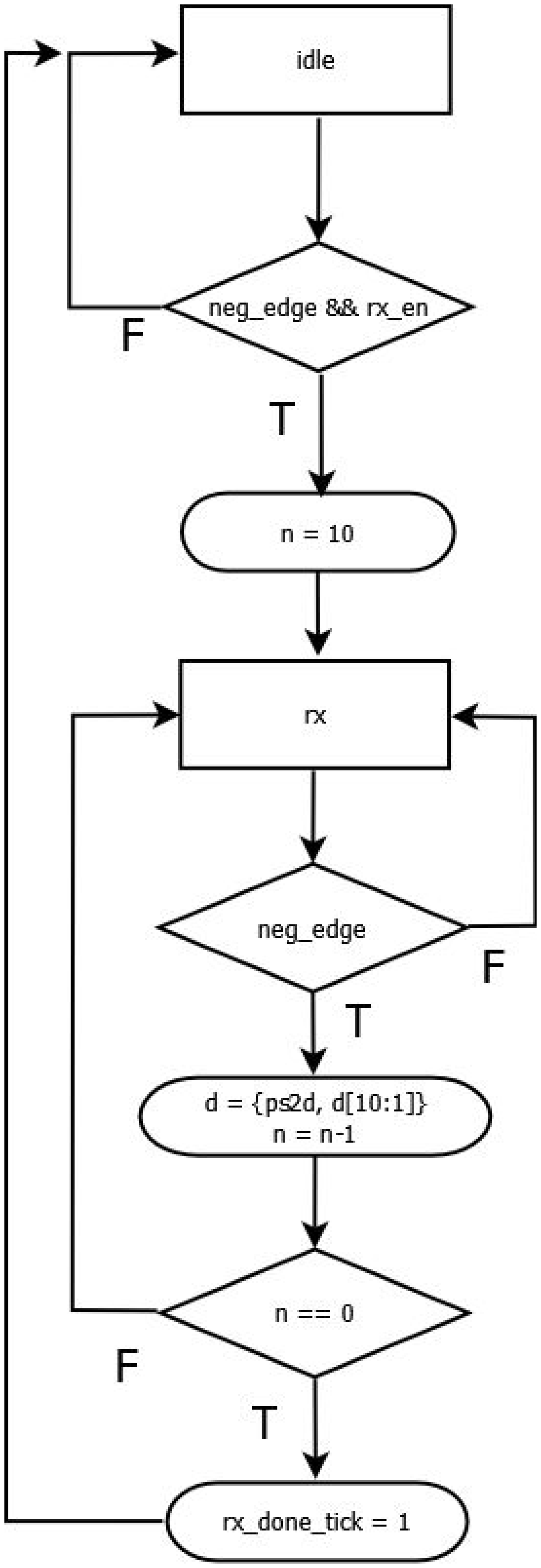
Implementation an FPGA keyboard interface that is simplified to process the alphanumeric & symbol keys, as well as the space, backspace, enter, tab, shift, and capslock keys. The keyboard used will have a PS2 connection, so we will need to implement a PS2 receiver circuit to receive scan codes from the keyboard when a key is pressed. We will then implement a keyboard interface circuit that processes the scan codes in a way that will make the keystroke responses natural and akin to how they would be in a simple text editor. Then using scancode we are obtaining the ascii value to pressed key and displaying it on 7-segment display.
We are also using ILA to view the scan code and ascii value of the pressed key.

The PS2 interface is used for keyboards and mouse that connect to a PC host.
The PS2 port has two wires used for communication: a clock line, ps2c, and a data line, ps2d. Data is communicated in an 11-bit packet, with the data intended to sampled on the falling edge of the clock signal provided by the keyboard.





The state machine diagram for the ps2 receiver. From the diagram, rectangles correspond to states, triangles to conditions, and ovals to variables that are adjusted.


idle
neg_edge && rx_en
F
T
n = 10
rx
neg_edge
F
T
d = {ps2d, d[10:1]}
n = n-1
n == 0
F
T
rx_done_tick = 1

The two states for the receiver are **idle** and **rx** (receive). In the **idle** state, if a negative edge is detected for ps2c, the start bit of a packet has been sent, if the receiver is also enabled, we go to the **rx** state. A counter variable n is set to 10 to count down for each remaining bit of ps2d we sample. In the **rx** state, if a negative edge for ps2c is detected, we sample ps2d by right shifting in the bit to the register d and then decrementing n. When n is equal to 0 we have sampled the 8 data, 1 parity, and 1 stop bits and are done, so we assert a one clock cycle done tick, and go back to the **idle** state.

Let's explain the scan codes that we will receive from the keyboard and how to process them.

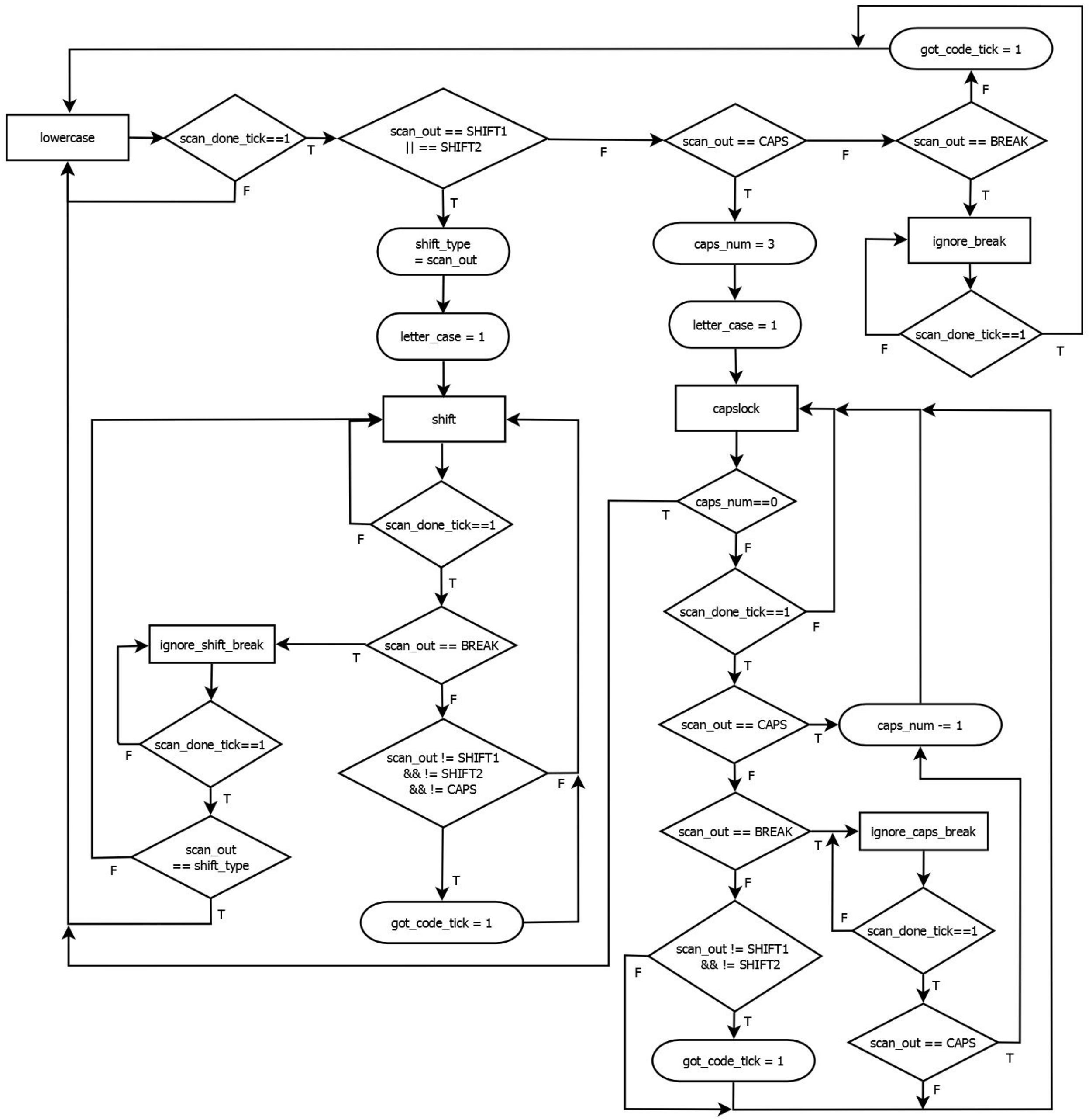
Each keyboard key has a unique hexadecimal code called a scan code.

For example, if the A key is pressed and immediately let go, we will receive the codes: **21 F0 21** . The code F0 is known as the break code and is sent when a key is let go, followed by a repeated scan code for the key.

Normally when you press and hold a key, the character begins to repeat. This is known as the typematic condition and usually begins after a key is held for a half second, upon which the scan code repeats at a frequency of 10 Hz. For example, the codes received for the T key being held for some time and then released are: **2C 2C … 2C F0 2C** . In the typematic condition, the scan code of the held key repeats every 0.1 seconds, and when the key is released the break code is sent followed by the repeated scan code.

What if we hold a shift key and then press some characters? The first scan code will be 12 or 59 for the left or right shift key, followed by the normal scan codes for characters or symbols. For example, holding shift and typing “qwe”, then letting go of shift transmits the codes: **59 15 F0 15 1D F0 1D 24 F0 24 F0 59**. If we were to use caps lock instead we would press caps lock once, then “qwe”, then caps lock again, which would send the codes: **58 F0 58 15 F0 15 1D F0 1D 24 F0 24 58 F0 58**.

To implement the keyboard interface, we will design a FSM with 6 states: **lowercase**, **ignore_break**, **shift**, **ignore_shift_break**, **capslock**, **ignore_caps_break**. We made a state machine diagram that we will consider in portions, starting with the lowercase state.

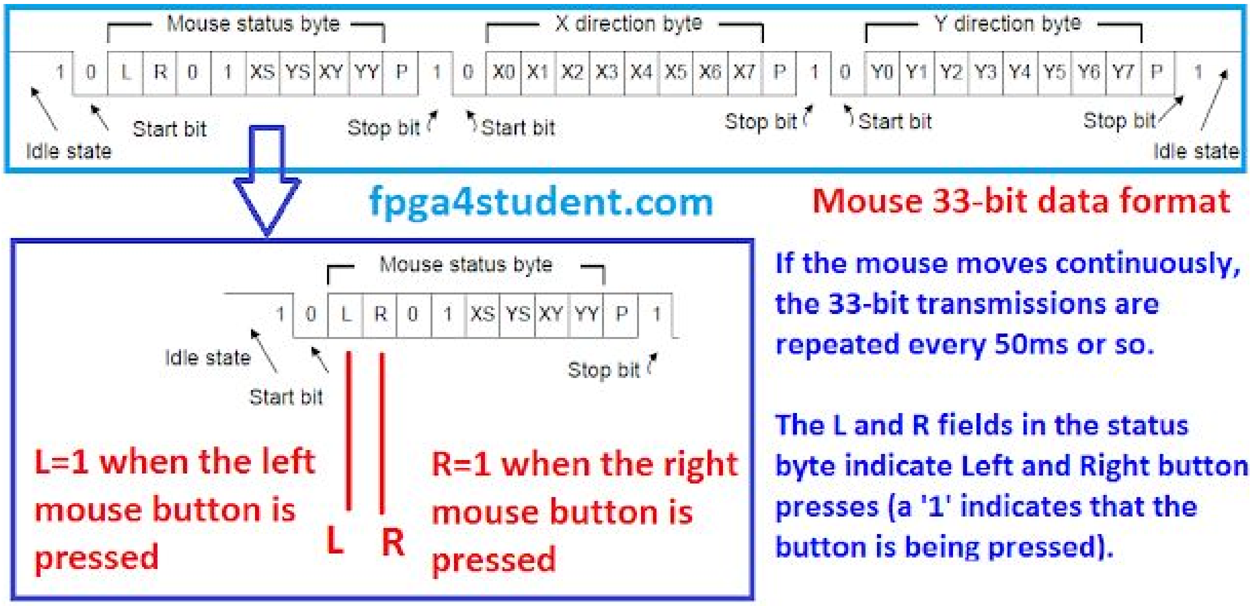


we will need to derive a circuit that takes in a scan code and spits out an ASCII code. The circuit should also take a special 1-bit input denoting if the output should be uppercase, which we will route in from the output of the previous keyboard interface circuit.

Finally to wrap things up we will design a circuit to interface the keyboard, key2ascii, and 7_segment display circuit in order to send ASCII codes to a 7-segment display.

MOUSE INTERFACE WITH FPGA

The 11-bit word data includes a ‘0’ start bit, 8 bits of data (LSB first), an odd parity bit, and followed by a ‘1’ stop bit. Three 11-bit words are transferred from the mouse to the FPGA, as shown in the following figure.



L=1 when the left mouse button is pressed

R=1 when the right mouse button is pressed

If the mouse moves continuously, the 33-bit transmissions are repeated every 50ms or so.

The L and R fields in the status byte indicate Left and Right button presses (a '1' indicates that the button is being pressed).

Here also we are interfacing mouse via ps2 interface and displaying output as seven segment display and on ILA.

The output for keyboard can be seen in this video

https://drive.google.com/open?id=18QEOP-upFJI90fIZe0OU6fyTu6WzD4Gh