Lab 2
USB/PS2 Keyboard Interface

Author : Anurag Daram
UID : 624002016

Instructor: Dr. Marcin Lukowiak
TA : Cody Tinker
TA : Dan Stafford

Abstract:

In this experiment consisting of 4 sections, the task was to implement a hardware interface for communication with PS2/USB keyboards. This lab required to create three different implementations of the interface which involved designing behaviorally with and without complex state machines. The third section of the lab involved implementation of the interface by using the internal clock of the FPGA and using a clock synchronizer logic to synchronize both the clock domains. The fourth section of the lab involved an introduction to the logic analyzer, wherein the FPGA could be monitored using the hardware manager tools. The task was to capture an entire break code and watch the waveform on the logic analyzer.

The data received from the keyboard or PS2 to the board involves transmission of a one byte make code when the key is pressed and a two byte break code is sent after it is released. The data received from the keyboard was displayed on the LEDs. The data pin consists of a start bit, 8 bits of data followed by the parity bit and a stop bit. The calculated and received were also displayed on the LEDs of the boards.

Design Methodology:

The first section of the lab involved implementing a design with a complex state machine and a simple datapath. A total of 11 states were used for this purpose which involved an IDLE state followed by 8 states for the data bits, one state for receiving the parity bit and one state for the stop bit. The 8 data bits are initially stored in a shift register and the parity bit is copied to a variable. In the stop bit or (RDS) state, the contents of the 8-bit data shift register are transferred to a 16-bit shift register and the parity bit is also copied to a 2-bit shift register which is then used to display the previous byte of data received and parity of the previously read data. The display is handled by using a 4 bit switch control input pin, and the combination of those values corresponded to displaying the contents of the data and parity registers. Then the design was simulated for testing and the results were analyzed for correctness. The calculated and received parity were compared in order to ensure that the result was correct. The parity was calculated by doing (XOR (0:3)) xor (XNOR (4:7)). It is performing an exclusive or of the first four bits and exclusive nor of the next four bits (odd parity).

The second section of the involved implementing a design with a simple state machine and a complex datapath. For this, only 3 states were used namely, (IDLE, Data acquisition, RDS). The data and the parity bits are captured in the Data acquisition state. It is implemented by using a counter which increments up-to 9 and then resets back to 0 when it reaches 9. Since the state ensures that every increment happens at the positive edge of the clock, therefore the counter is incremented at the ps2 clock event ensuring the data is received correctly. The parity is also calculated in the data acquisition state and the copy to registers is done in RDS state similar to the way performed for section 1 of the lab. This design was also synthesized and checked on the hardware for correctness.

The third section of the lab involved using the internal clock of the FPGA instead of the PS2 clock. The system clock was slowed down by using a clock divider and enable and then a clock synchronizer followed by the edge detector was used in order to reduce the possibility of metastability issues in flip flops and then for positive edge detection of PS2 clock respectively.

In section 4, the task was to use the logic analyzer module of the Vivado tools in order to monitor the signals on the FPGA in the hardware manager. It involved generating a clocking wizard IP core to divide the 100MHz clock signal and then add the new generated instance into the PS2Interface design source file and the code was synthesized. Then after the changing the constraint property and setting the sample data depth, the bit stream was generated and the logic analyzer was used to capture a break code and make code.

Results:

The simulation results in Figure 1, 5, 9, displays the various outputs in the waveform window and the selected one is the LED output. The system first stays idle and then during data_acq state, it stores all the data in the sreg 8 bit shift register. Once it enters the stop state, the contents of the sreg register are moved to the lower 8 bits of the 16 bit shex register. At every stop state the shex register first shifts the 8 lsb to the msb and then copies in the contents from the sreg register. Also, the parity is calculated in the temp_par signal and check_parity shift register stores the current parity at the lsb and previous parity in the msb. The spar and sparb register stores the parity received from the data. For the switches,

0000 → Current data value (shex[15:8]) for LED output

0001 → Previous data byte value (shex[0:7]) for LED output

0011 → Computed parity bits (check_par[0:1]) for LED output

0010 → Parity read from PS2 (sparb[0:1]) for LED output

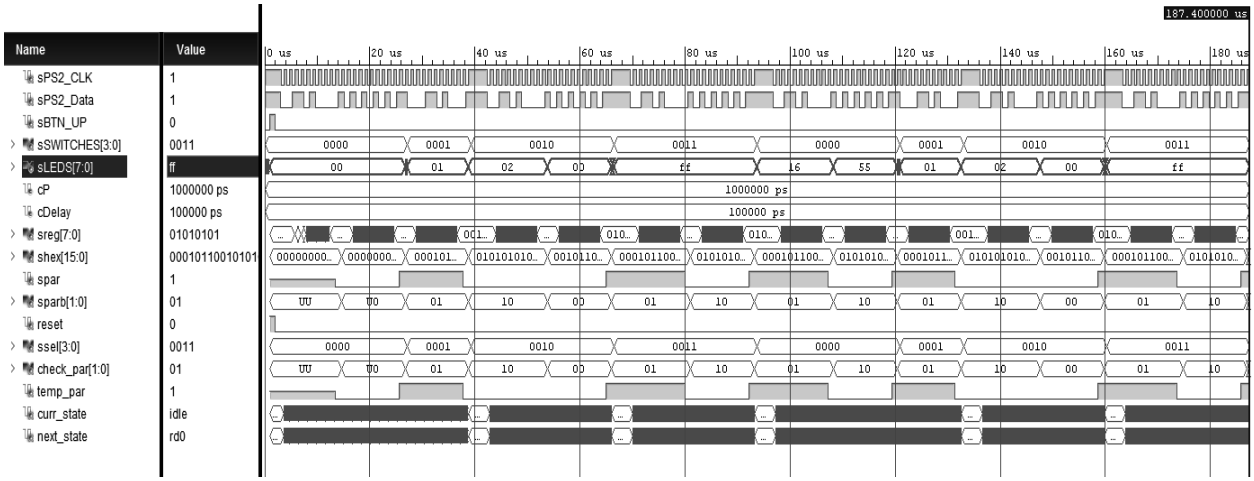

Figure 1: Simulation result of system with complex state machine

The waveform in figure 1 displays the output being received at every combination of the switch. Also, the system consists of 11 states namely (IDLE, RD0-RD7, RDP and RDS), the data acquisition starts in RD0 state and it shifts through the states till RD7 for receiving the data byte and RDP state for collecting the parity bit and in RDS state, all the transitions or shifting to larger registers occurs.

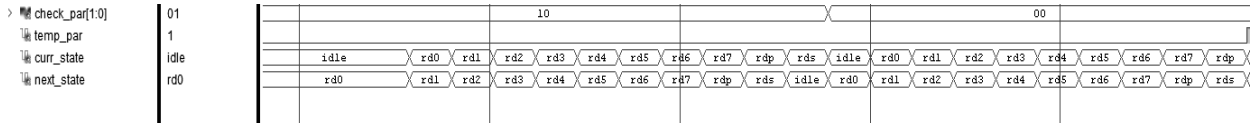

Figure 2: All the states of the complex state machine

```
---------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------
Detailed RTL Component Info :
+---XORs :
       5 Input      1 Bit         XORs := 1
       4 Input      1 Bit         XORs := 1
+---Registers :
                   16 Bit    Registers := 1
                    8 Bit    Registers := 1
                    2 Bit    Registers := 2
                    1 Bit    Registers := 2
+---Muxes :
       4 Input     16 Bit        Muxes := 1
       5 Input      8 Bit        Muxes := 1
       4 Input      2 Bit        Muxes := 2
       4 Input      1 Bit        Muxes := 3
---------------------------------------------------
```

Figure 3: Synthesis Report of complex state machine model

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	32	0	63400	0.05
LUT as Logic	32	0	63400	0.05
LUT as Memory	0	0	19000	0.00
Slice Registers	41	0	126800	0.03
Register as Flip Flop	41	0	126800	0.03
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Figure 4: Utilization report of system with complex state machine

Figure 3 and 4 represent the utilization of the system and the synthesis report. This design fairly large number of Look UP Tables and registers as compared to the models below. This is because of using many states and encoding them might take up a lot of resource space.

Figures 5 and 6 presents the simulation results of the PS2 implementation with a simple state machine and complex datapath.

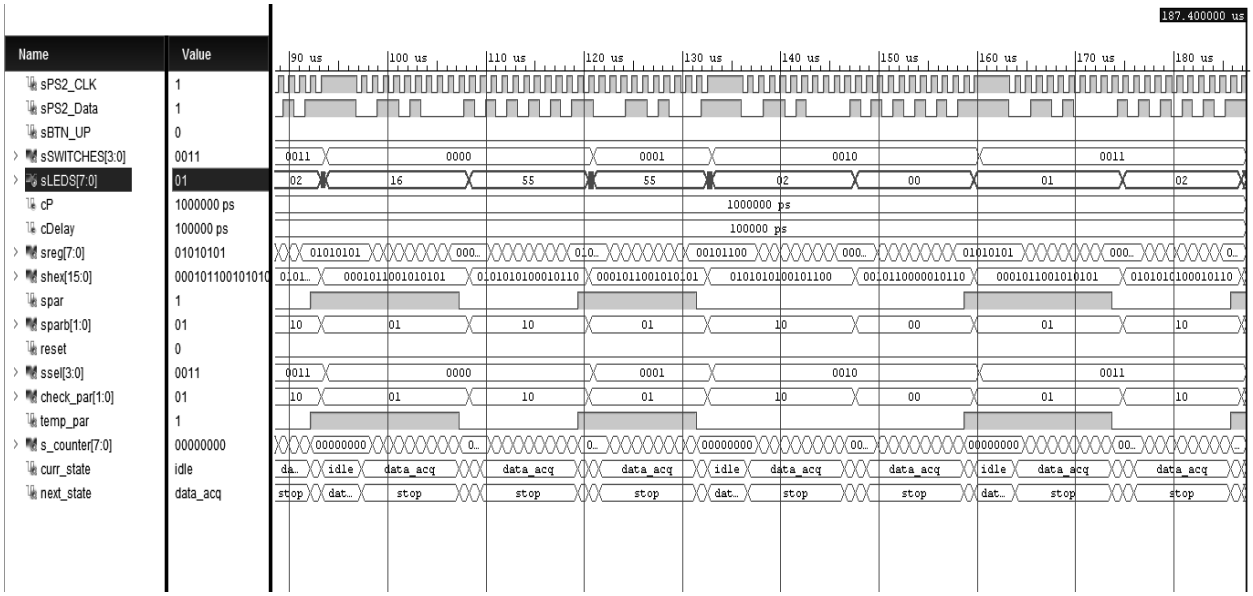

Figure 5. Overall simulation of system with simple state machine

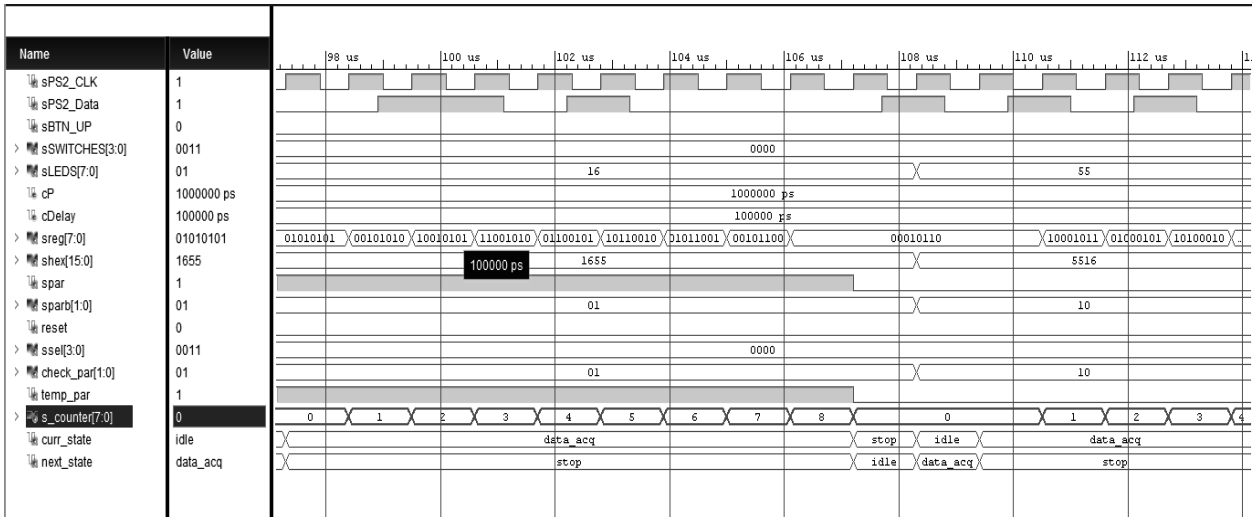

Figure 6. Simulation presenting the state transition and counter incrementing

As displayed in Figure 6, the counter is incremented up to 8 (0 -7 for the data acquisition bits and bit 8 for parity). After the parity and data byte have been copied to their individual registers, and the system enters stop state, the counter is then set back to 0 and again it starts incrementing once the system enters the data acquisition state.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	20	0	63400	0.03
LUT as Logic	20	0	63400	0.03
LUT as Memory	0	0	19000	0.00
Slice Registers	36	0	126800	0.03
Register as Flip Flop	36	0	126800	0.03
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Figure 7. Utilization Report for system with lesser states

```
---------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input      8 Bit       Adders := 1
+---XORs :
       5 Input      1 Bit         XORs := 1
       4 Input      1 Bit         XORs := 1
+---Registers :
                   16 Bit    Registers := 1
                    8 Bit    Registers := 2
                    2 Bit    Registers := 3
                    1 Bit    Registers := 2
+---Muxes :
       2 Input      8 Bit        Muxes := 1
       5 Input      8 Bit        Muxes := 1
       4 Input      2 Bit        Muxes := 1
       4 Input      1 Bit        Muxes := 4
       2 Input      1 Bit        Muxes := 2
---------------------------------------------
```

Figure 8. Synthesis Report

The synthesis and utilization report of the system with simple state machine displays lesser use of resources and Look-Up Tables. This method of implementing the interface for PS2 uses almost 40% lesser resources than the one with complex state machines. It might be because instead of using many states, the counter just uses a basic comparator and MUX based logic which is less expensive than creating multiple states and instantiating them.

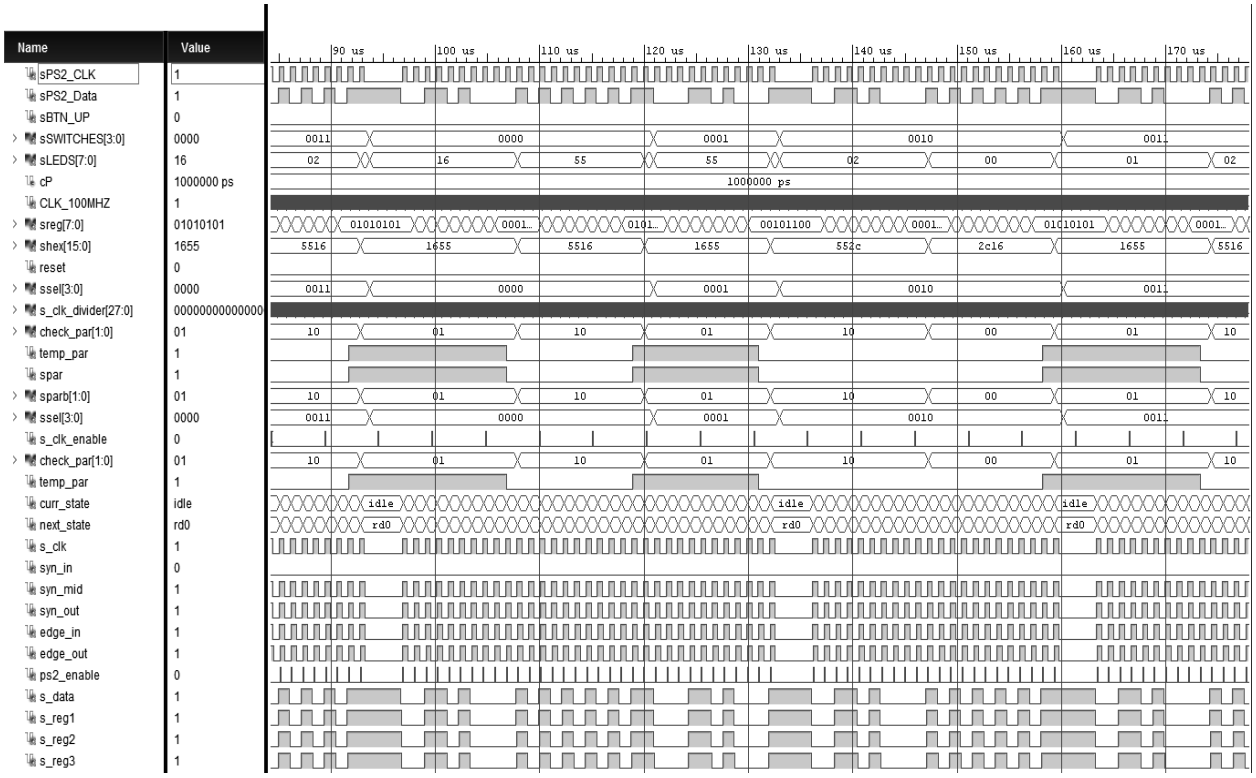

Figure 9: Overall simulation of synchronized clock domains

As shown in figure 9, since the system clock runs at 100MHZ whereas the PS2 clock is slow, in order to synchronize between them, an edge detector circuit is used in order to manage between both the clock edges and a synchronizer circuit is used as a solution to resolve metastability issues. Also, the data is also passed through flip flops in order to maintain correct synchronization between the clock and data line. The ps2_enable is the output of the edge detector and the data is fed into the sreg register from reg3 instead of directly using the PS2_DATA line.

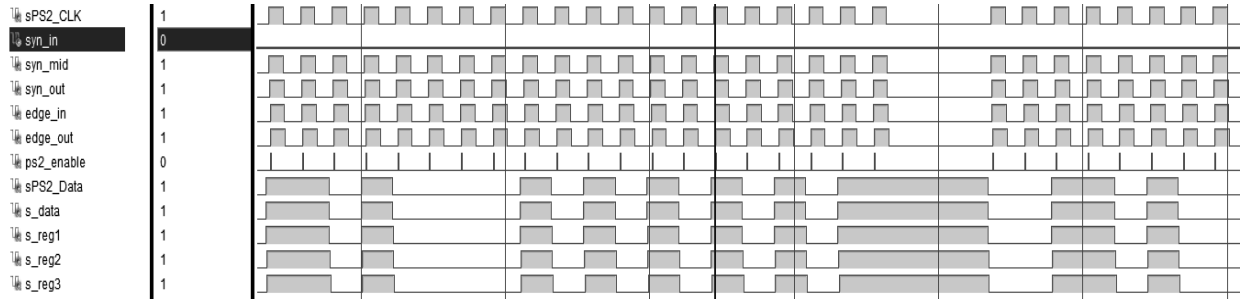

Figure 10: The synchronizer and edge detector circuit simulations

The PS2_CLK was fed in as an input to the synchronizer and edge detector circuit. The syn_in, syn_mid,edge_in and edge_out are the interconnecting wires between the flip flops of the synchronizer and edge detector and ps2_enable gets the output of the edge detector. In order to avoid metastability issues, the ps2_data is also sent through the multiple registers so that they are both synchronized. Figure 10 displays the waveform output of synchronizer with edge detector circuit.

```
Report RTL Partitions:
+-+--------------+------------+----------+
| |RTL Partition |Replication |Instances |
+-+--------------+------------+----------+
+-+--------------+------------+----------+
---------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------
Detailed RTL Component Info :
+---Registers :
                   16 Bit    Registers := 1
                    8 Bit    Registers := 1
                    1 Bit    Registers := 8
+---Muxes :
       4 Input     16 Bit        Muxes := 1
       3 Input      8 Bit        Muxes := 1
       4 Input      1 Bit        Muxes := 1
---------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------
```

Figure 11: Synthesis report of PS2 implementation with synchronizer

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	28	0	63400	0.04
LUT as Logic	28	0	63400	0.04
LUT as Memory	0	0	19000	0.00
Slice Registers	43	0	126800	0.03
Register as Flip Flop	43	0	126800	0.03
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Figure 12: Utilization report of PS2 implementation with synchronizer

After looking at the three implementations of the PS2, the design with the simple state machine uses the least resources and is good for using as a component in large systems but then the design with complex state machine actually gives a better control over handling the data and is a good approach conceptually. All the three implementations worked fine on the hardware wherein the LEDs were able to display the correct ASCII values of the keys pressed and also were able to display both previous and current calculated and received parities. The fourth section of the lab involved using the logic analyzer in order to monitor the FPGA from Vivado itself. It was a good exercise in the sense that for few situations displaying on LEDs is not always a good option and being able to debug from the hardware manager is an efficient way.

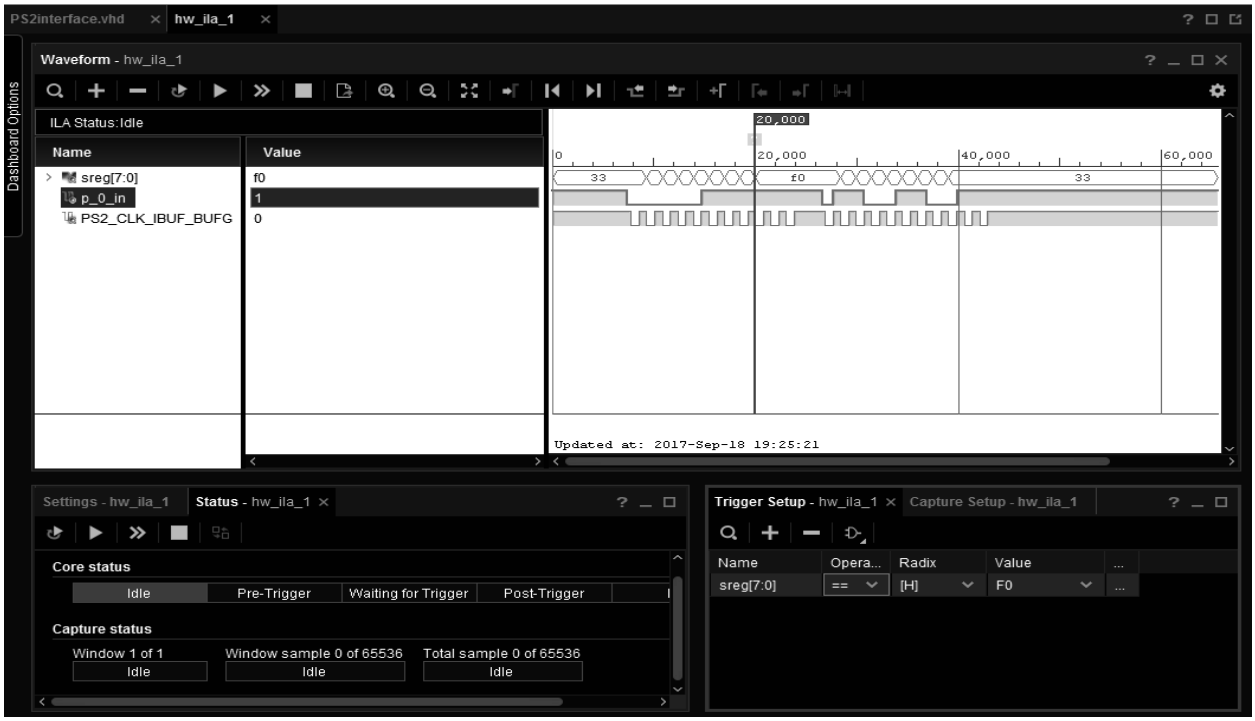

Figure 13: Analyzer window of 'H' key being pressed

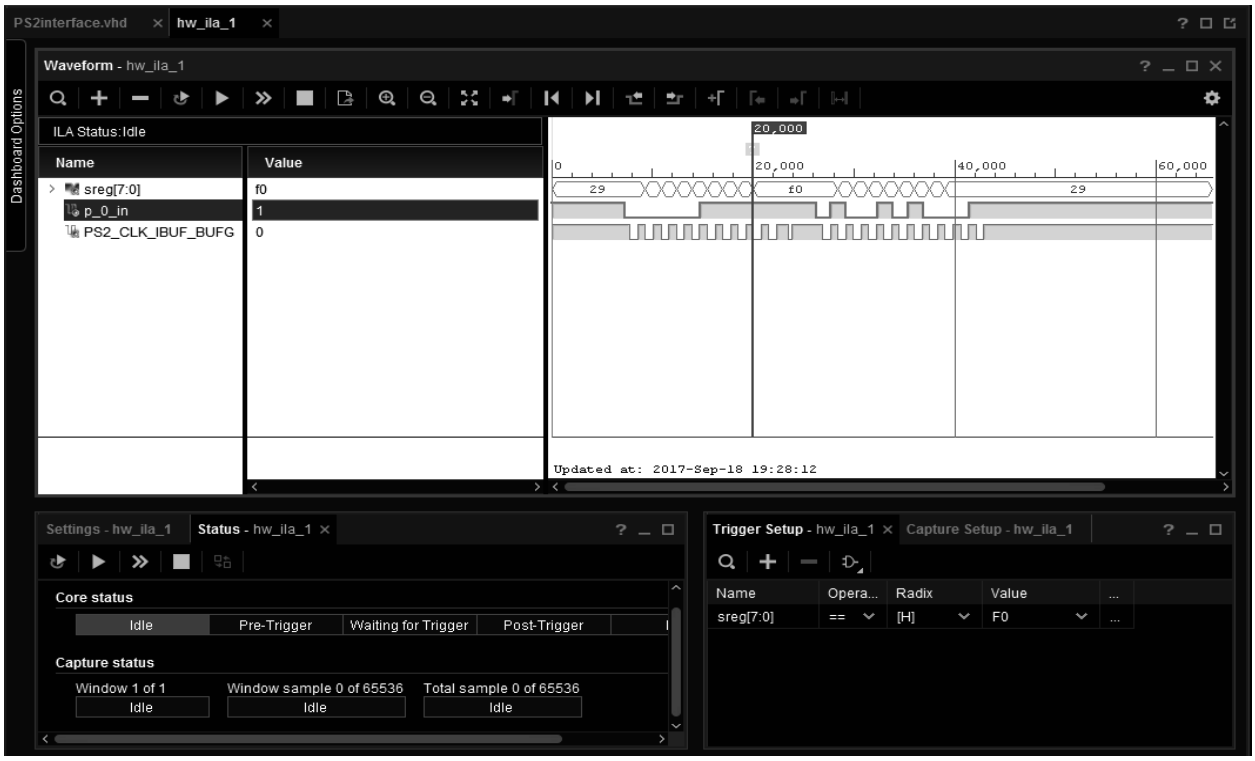

Figure 14: Analyzer window of 'SpaceBar' key being pressed

As shown in figures 13 and 14 the ASCII value of 'H' key is 33 and 'SpaceBar' key is '29'. The first value shows the make code being received and then the two byte break code with F0 as the first byte and then the make code of the key released. This was tested by connecting the keyboard to the board followed by the keys being pressed.