A Design Report in
Design with CPLD's and FPGA
On

Key Board Controller

By:

Agnish Jain
M.E., Microelectronics-I[st] Year
I.I.Sc., Bangalore

Under the Guidance of:

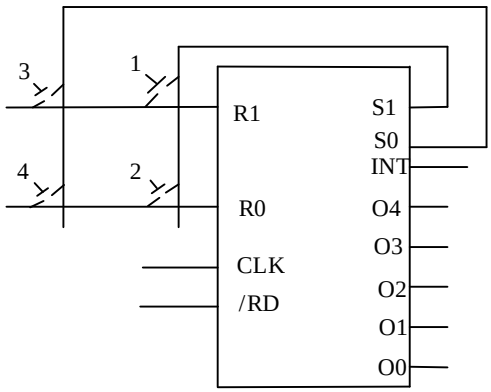
Mr. Kuruvilla Varghese
CEDT, I.I.Sc., Bangalore

2002-2003

INDIAN INSTITUTE OF SCIENCE
BANGALORE -560012.

Problem Specification:

Design a keyboard controller to scan 4 switches (2 x 2 matrix). Implement de-bounce and two key lockout.



Output (04 - 00) is enabled using /RD line. When /RD is not asserted output lines are tri-stated. Whenever a valid key is press is detected INTR line goes high. Subsequent reading operation de-asserts INTR line and clears output lines (All High). If a second key press occurs before reading an overrun flag is set. Output format is as shown below;

OR	S1	S0	R1	R0
O4	O3	O2	O1	O0

OR - Overrun Flag, S1 & S0 - Scan Line Status, R1 & R0 - Return Line Status

Different data formats for various conditions are given below:

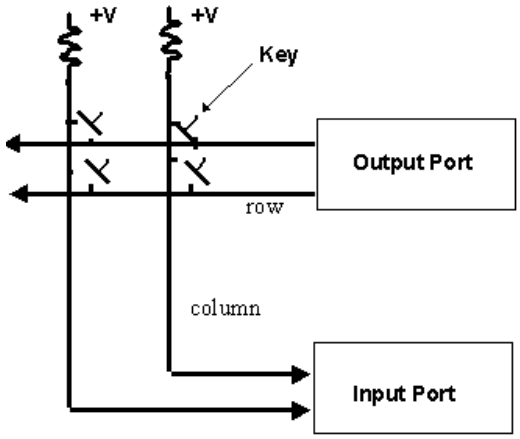
	O4	O3	O2	O1	O0
No Key	0	1	1	1	1
Key1	X	0	1	0	1
Key2	X	0	1	1	0
Key3	X	1	0	0	1
Key4	X	1	0	1	0
Overrun	1	0	1	0	1
Overrun	1	0	1	1	0
Overrun	1	1	0	0	1
Overrun	1	1	0	1	0

Introduction

The key Board controller is used to read a key press and is an interface between high speed synchronous devices like microprocessor & slow changing as well as asynchronous key switches.
The KeyBoard controller should be designed such that it is robust to noise and spike.
It also needs to give the correct output as desired by CPU. Proper handshaking signals are also required.
There are variations in its design depending on factors such as two key lockout

Theory

In keyboards, key switches are connected in a matrix of rows and columns.
When no keys are pressed, no current flows thru the pull-up resistors connected the +V, and the column lines (R0, R1) are held high.
If a low is (intentionally) output on a scan a row (S0, S1) and a key in that row is pressed, then the low will appear on the column that contains that key and can be detected on the input port.



The rows are continuously scanned, with one row being active (low) at any time. If any input goes low, the row and column information is captured(stored in ff_s and ff_r registers respectively).

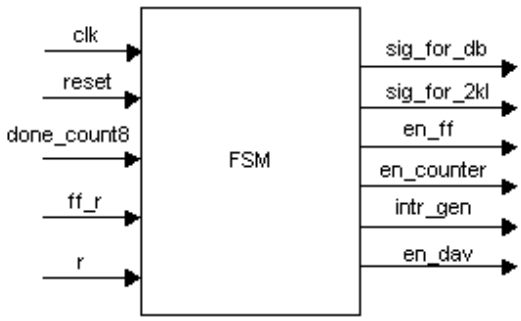
Getting meaningful from a keyboard requires:

- Detecting the key press.
- Debouncing the keypress.
- Implement the two Key Layout.

In Key Board Controller Design we continuously scan the status of R1-R0 lines by giving the output to the S0-S1 through a multiplexer.

Basically the output of scan lines S0-S1 is given through a Multiplexer as shown in the block schematic.

Finite State Machine Description



State-1 We first output lows to all the scan lines(S1-S0) and execute our first loop in which we wait till any valid key is pressed i.e. any one of the R1-R0 lines go low. As soon as this happens the state machine enters into a new state with asserting en_ff signal high which enables the ff_r and ff_s registers and they store the current value of S0-S1 and R0-R1.

State-2 Now since the key is detected we need to debounce it since we check to see whether the key is still pressed.(If the R1-R0 are now all high, then no key is pressed and the initial detection was caused by a noise pulse or a light brushing past a key, or etc.) If any of the R1-R0 are still low, then the assumption is made that it was a valid keypress. For this we enable a Debounce counter (by asserting en_counter high). We wait till 8 clock cycle (3 bit counter)for the key to get get debounce.When 8 clk cycles are over done_count8 is asserted high.
Sig_for_db is made high and Sig_for_2kl is asserted low to debounce the key.

State-3 As soon as done_count8 is asserted high we enter into this state and upon next clock cycle en_counter is asserted low which disables the debounce counter.And now we Check whether the earlier stored value of R1-R0 matches with the current value. If this happens we enable the data_valid ff and make intr_gen also high & enter into next state to check for two key lockout.
If both values don't match we go back to state-1.

State-4 Now the Sig_for_db is made low and Sig_for_2kl is asserted high to check for two key lockout. If any key is released(all high)then only state change will occur and the fsm will go back to state-1 to check for a new key.
Otherwise it will remain in the same state till the R1-R0 doesn't becomes all high(key realesed).
In FSM the rd_bar signal is not used and it the state transition is independent of its value.

Main sections of Key Board Controller

Debounce Counter

This is a 3-bit counter which is enabled by en_counter output from the FSM.
Whenever it becomes “111” done_count8 is asserted high & given to the FSM as an an input(which changes its state).
Here the done_count8 is also registered to avoid any glitches.

Free Running Counter

This counter gives output 10 and 01 on alternate clock cycle which is used to scan the R1-R0 lines and are on the output line depending on the bits of sel(Mux select line).
The Mux is also shown in Fig.1 Block Schematic and it assigns s a value depending on the bits sig_for_2kl and sig_for_db.(sel is concatenation of these bits).

Intr and Latch_intr Regr.

Both these become high as soon intr_gen becomes high.The difference between them is this only that former become low as soon as rd_bar becomes low wheras latter becomes low when rd_bar transits from 0 to 1.This is shown in the block schematic Fig.1.

Overrun Flag

This is asserted high when a key press is already detected and validated and it is not being read and a new key press is detected & is also valid.

Data_valid & final output stage

The Data_valid is asserted as soon as en_dav(from FSM) is made HIGH.It will store the latest value of valid key press
Independent of overrun flag as shown in the block schematic Fig.1.
Now its output is given to a register dmux_out through a Mux whose othr input is all HIGH(for the condition of problem to give output all HIGH when rd_bar is asserted and intr is not set) & its output itself.
Its select line is sel_out (concatenation of rd_bar and latch_intr)

Description of block Schematic

The figure-1 shows the block Schematic of Key Board Controller.The figure shows the output on scan line are given thru a Mux whose select line is sel.Its input are output of Free running counter,00(for two key lockout)
& output of ff_s Register.
ff_s and ff_r store the value of S1-S0 and R1-R0 depending on en_ff bit(from FSM).
The Debouncer counter is a 3-bit counter which is enabled when en_doboune is HIGH.
The intr, latch_intr registers stores the value of intr_gen(output of FSM).The former is cleared when rd_bar becomes LOW whereas latter becomes low when rd_bar goes from LOW to high. latch_intr is also the input to MUX with rd_bar used to select the output dmux_out(before TRI-state)from data_valid,all '1' and itself depending on the condition shown in VHDL code.
The output o4_o0 is enabled as soon as rd_bar becomes LOW otherwise it is tristated as desired in the problem spec.

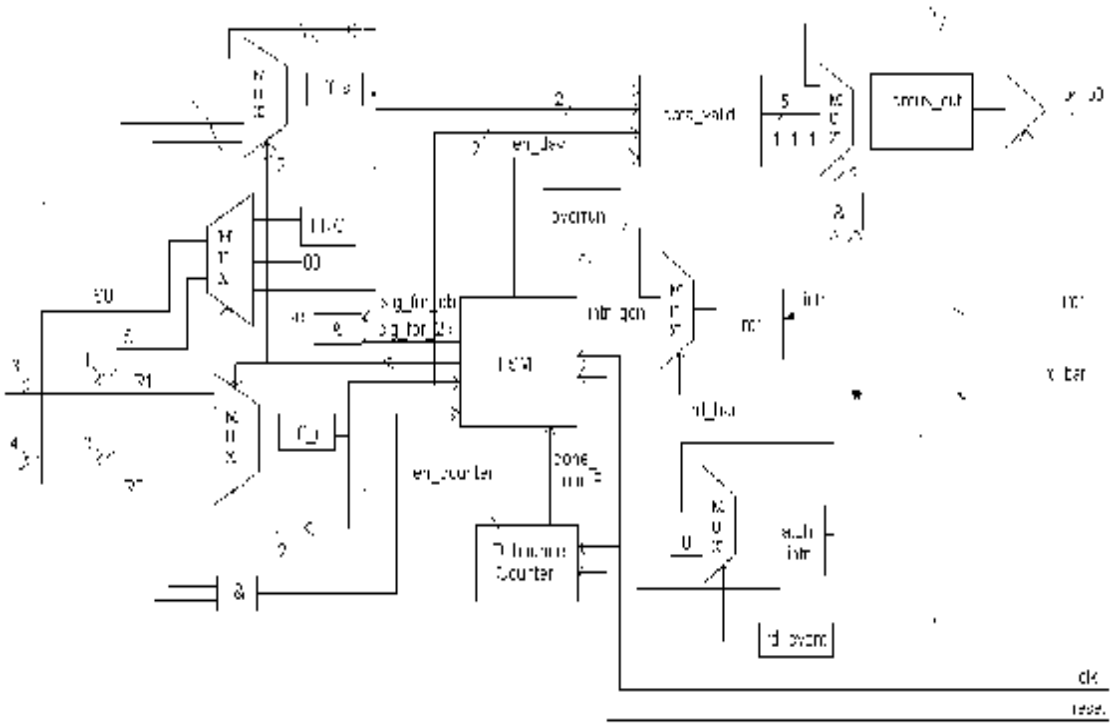


Figure-1
Block Diagram of Key Board Controller

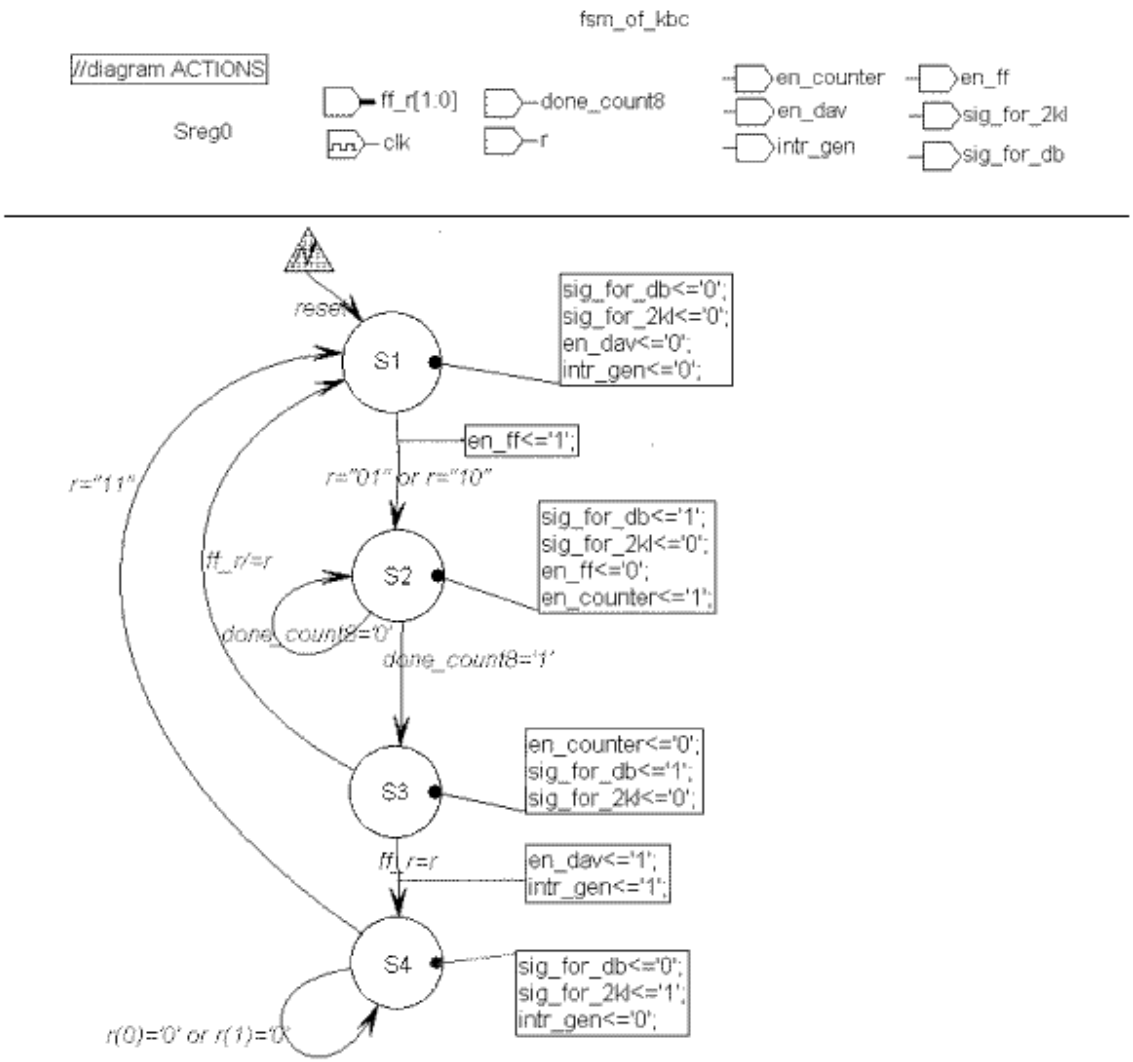


Figure-2
Finite State Machine of Key Board Controller.

VHDL Source code

```
--Vhdl code for Key Board Controller
--Submitted By Agnish Jain M.E. Microelectronics I-st Yr
--Guided By Kurvilla Varghese
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity kbc is
  port (clk,reset,rd_bar: in std_logic;
        r:in std_logic_vector(1 downto 0);
     --r line
          s:buffer std_logic_vector(1 downto 0);
     --s line
          intr:buffer std_logic;

          o4_o0:out std_logic_vector(4 downto 0));
     --five bit output.
end kbc;

architecture arch_kbc of kbc is

--state assignment
  type state_type is (S1, S2, S3, S4);
  signal sreg0:state_type;

  signal ff_s,ff_r :std_logic_vector(1 downto 0);

--For 3 bit counter
  signal c_bit: std_logic_vector(2 downto 0);     --3 bit
counter
  signal en_counter: std_logic;                   --enable
counter to be used in the Debounce
  signal done_count8: std_logic;                  --used to
indicate that c_bits are "111"

--For debonce 2 key lock out checking
  signal sig_for_db,sig_for_2kl:std_logic;
  signal frc_for_s: std_logic_vector (1 downto 0);
     --Free Running Counter bits

  signal intr_gen,en_ff,en_dav:std_logic;    ----signals --
--for fsm
```

```
  signal latch_intr :  std_logic;
  signal sel,sel_out:std_logic_vector(1 downto 0);
     --select lines for Mux

--Output Stage Registers
  signal data_valid:std_logic_vector(4 downto 0):="00000";
  signal out_dmux:std_logic_vector(4 downto 0):="00000";

--Overrun flag
signal overrun:  std_logic;
attribute synthesis_off of overrun: signal is true;

--architecture begins

begin


--storing the starting s and r
process(reset,clk,r,s,en_ff)
 begin
  if reset='1' then
      ff_r<="00";
      ff_s<="00";
   elsif en_ff='1'then
      ff_r<=r;
      ff_s<=s;
   end if;
end process;

--3 bit Counter used to debounce the circuit
process(reset,clk,c_bit,en_counter)
 begin
  if(reset='1') then
   c_bit<=(others=>'0');
   elsif(clk'event and clk='1') then
    if(en_counter='1') then
       c_bit<=c_bit+1;
     else
       c_bit<=(others=>'0');
     end if;
   end if;

--Register for done_count8
done_count8<='0';
  if(c_bit="111")then
          done_count8<='1';
  end if;
```

```
end process;

------------------Finite State Machine--------------------
--------
Sreg0_machine: process (clk, reset,s,r,done_count8,ff_r)

begin

if (reset='1') then
     Sreg0 <= S1;
elsif clk'event and clk = '1' then
     case Sreg0 is
          when S1 =>
               if r="01" or r="10" then
                    Sreg0 <= S2;
               end if;
          when S2 =>
               if done_count8='1' then
                    Sreg0 <= S3;
                else
                    Sreg0 <= S2;
               end if;
          when S3 =>
               if ff_r/=r then
                    Sreg0 <= S1;
               elsif ff_r=r then
                    Sreg0 <= S4;
               end if;
          when S4 =>
               if r="11" then
                    Sreg0 <= S1;
               elsif r(0)='0' or r(1)='0' then
                    Sreg0 <= S4;
               end if;
          when others =>
               null;
     end case;
end if;
end process;

-- signal assignment statements for combinatorial outputs
--sig_for_db_assignment:
sig_for_db <= '1' when (Sreg0 = S2) else
              '1' when (Sreg0 = S3) else
              '0' when (Sreg0 = S4) else
              '0';
```

```
--sig_for_2kl_assignment:
sig_for_2kl <= '0' when (Sreg0 = S2) else
               '0' when (Sreg0 = S3) else
               '1' when (Sreg0 = S4) else
               '0' ;

--en_dav_assignment:
en_dav <= '1' when (Sreg0 = S3 and ff_r=r) else
          '0';

--intr_gen_assignment:
intr_gen <= '1' when (Sreg0 = S3 and ff_r=r) else
            '0' when (Sreg0 = S4) else
            '0';

--en_ff_assignment:
en_ff <= '1' when (Sreg0 = S1 and (r="01" or r="10")) else
               '0' when (Sreg0 = S2) else
               '0' ;


--en_counter_assignment:
en_counter <= '1' when (Sreg0 = S2) else
              '0' when (Sreg0 = S3) else
              '0';
------------------end  of fsm section----------------------
------


--Free Running Counter
process(clk,frc_for_s)
 begin
  if clk'event and clk='1' then
      frc_for_s(1)<=frc_for_s(0);
      frc_for_s(0)<=not(frc_for_s(0));
  end if;
end process;

--output s Mux implementation
sel<=sig_for_2kl & sig_for_db;

with sel select
        s<="00"  when "10",
            ff_s  when "01",
            frc_for_s when others;

--Asserting intr signal
```

```
process(reset,clk,intr_gen)
 begin
  if(reset='1') then
   intr<='0';
  elsif(clk'event and clk='1') then
        if(rd_bar='0') then
          intr<='0';     -- Clear interrupt after read.
        elsif(intr_gen='1') then
        intr<='1';
        end if;
  end if;
end process;

--Latch intr signal to be latched for final output Mux
process(rd_bar,intr_gen)
begin
     if intr_gen='1'then
          latch_intr<='1';
     elsif(rd_bar'event and rd_bar='1') then
          latch_intr<='0';
     end if;
end process;

--Overrun Flag
process(reset,clk,intr,intr_gen,overrun)
 begin
  if(reset='1') then
      overrun<='0';
  elsif(clk'event and clk='1') then
   if(intr='0'and intr_gen='1')then               --When --
--the first data is read overrun is cleared.
          overrun<='0';
     elsif(intr='1'and intr_gen='1')then
               overrun<='1';
          end if;
               data_valid(4)<=overrun;
    end if;
end process;

--registering the valid data after it is
debounced(en_dav='1')
process(en_dav,reset,s,r)
begin
if(reset='1')then
     data_valid(3 downto 0)<=(others=>'0');
elsif(en_dav='1')then
     data_valid(3 downto 0)<=s & r;
```

```
end if;
end process;

--final output stage
sel_out<=rd_bar & latch_intr;
process(reset,data_valid,sel_out)
begin
 if (reset='1')then
     out_dmux<=(others=>'0');
 elsif sel_out="00" then
     out_dmux<=(others=>'1');
 elsif sel_out="11" then
     out_dmux<=data_valid;
 end if;
end process;

--Implementing tristate output
with rd_bar select
     o4_o0<=out_dmux      when '0',
               (others=>'Z')when others;

end arch_kbc;
```

Simulation Result

The following result shows the simulation results.

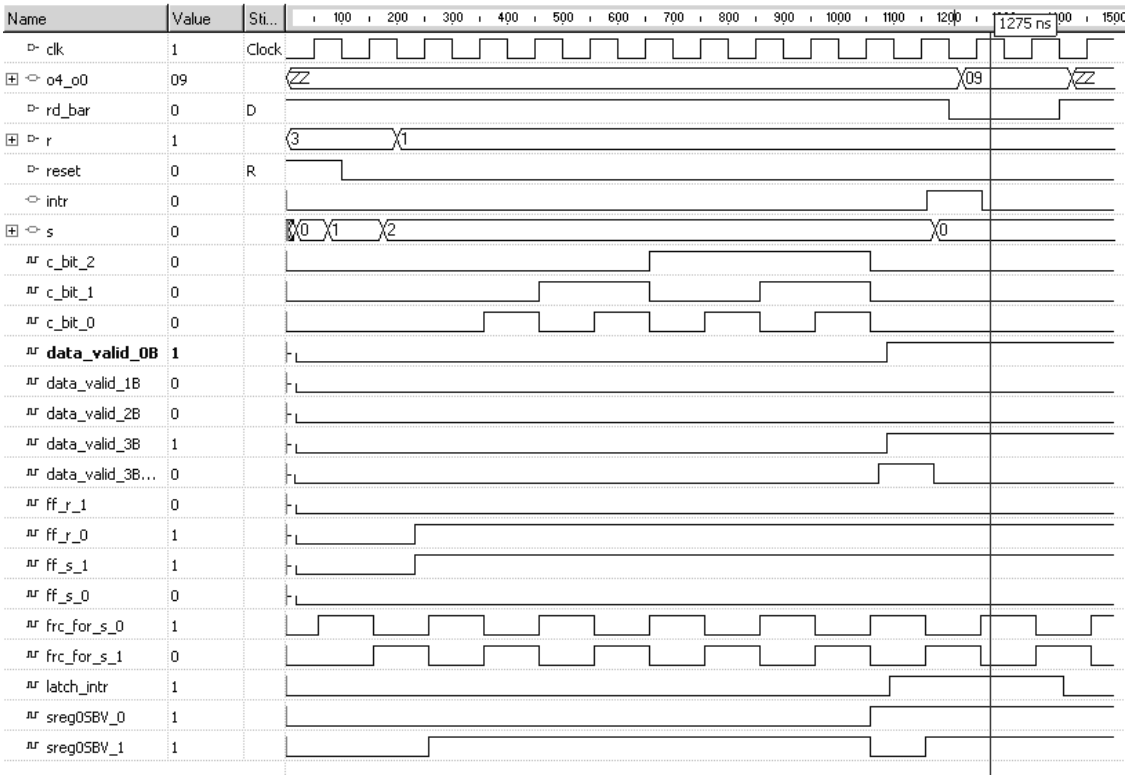


Figure-3 correct Output

Shows the o4_o0 3rd signal(after 1200 nsec.) when a valid key(key 3) is pressed and is debounced.

Figure also shows the states of FSM which becomes “11”(S4 as par the diagram of FSM) for implementing the two key lockout.

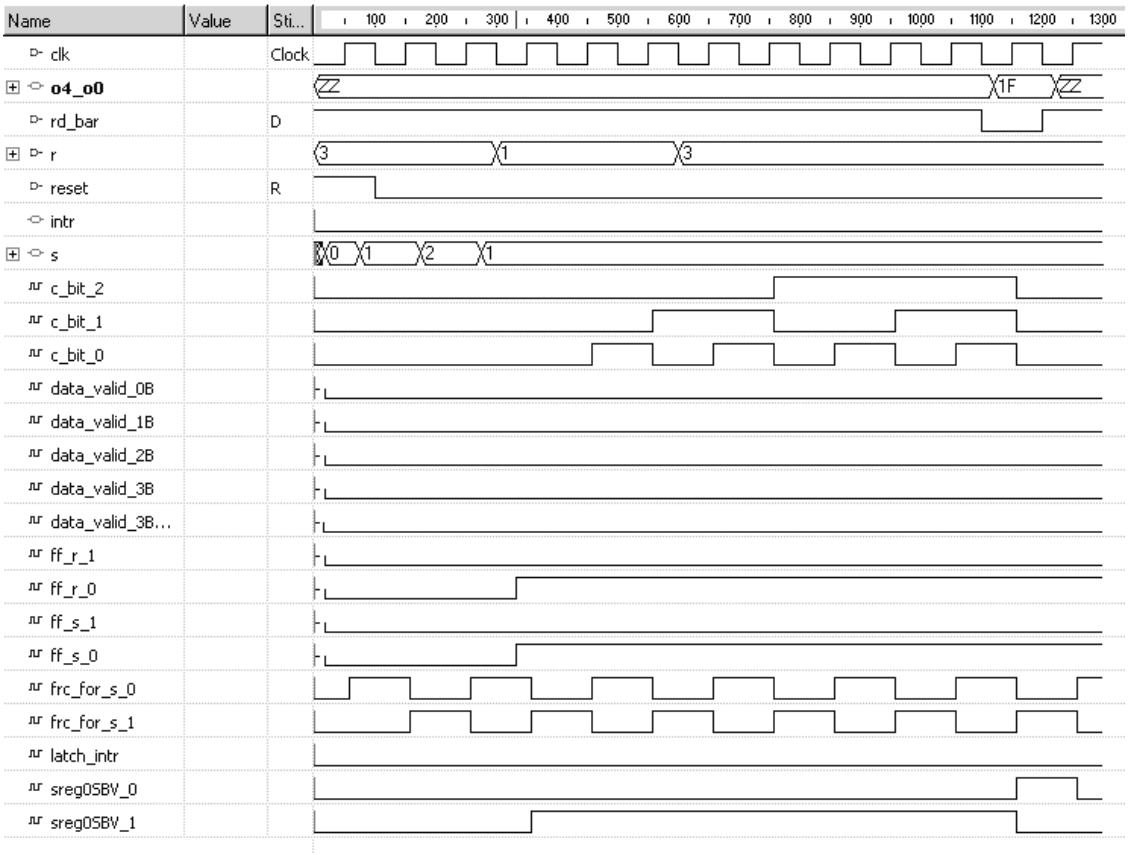


Figure-4 Timing Diagram

showing the condition of all output lines(04_o0)all High(1F) when intr(6th signal)is low(no key pressed)& rd_bar signal becomes low after 1100 nsec. The figure also shows that if a key press occur is detected(at 300 nsec.) & it changes (at 600 nsec.) during debounce operation then the key press is not stored & intr line is not asserted.

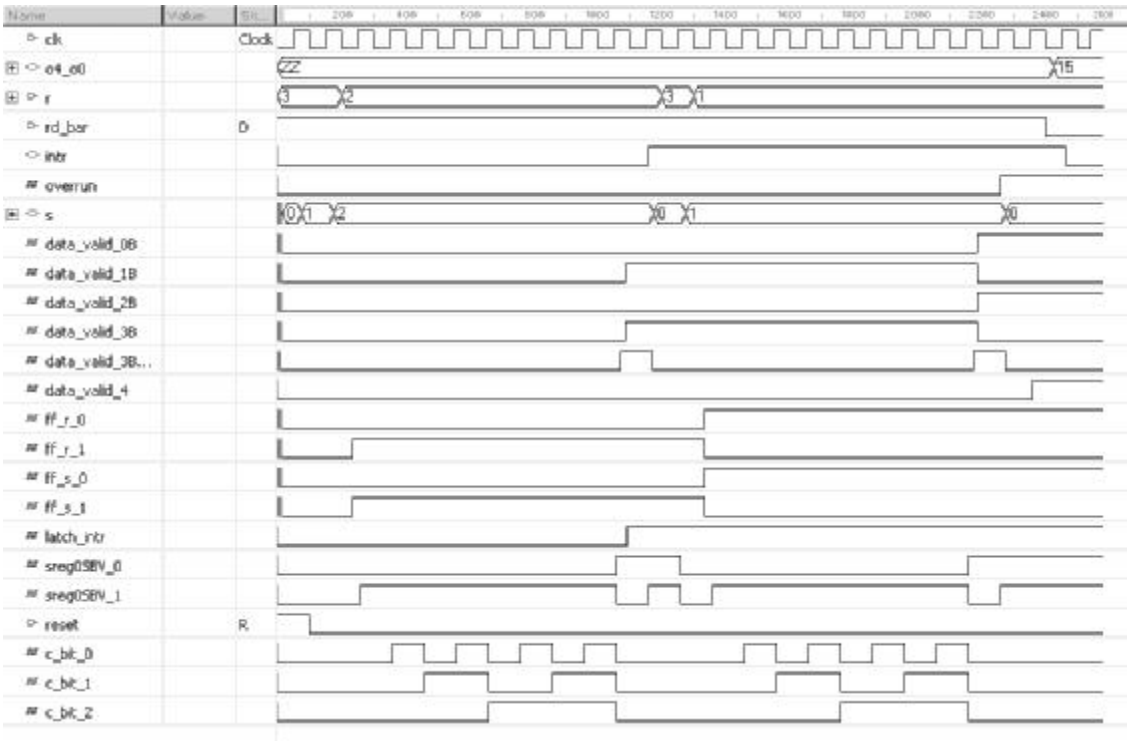


Figure-5 Timing Diagram

shows theoverrun flag(6th signal) being set(at 2200 ns) when a second key press occurs before reading.

Worst Case Path Summary

tPD = 83.0 ns for o4_o0(0)
tS = 41.0 ns for intr.D
tSCS = 46.0 ns for intr.D using clock signal clk and fMAX = 21 MHz
tCO = 88.0 ns for o4_o0(0)
tPO = 77.0 ns for latch_intr.AP
tRO = 26.0 ns for intr.AR
tER = 24.0 ns for o4_o0(4).OE

Macrocell Utilization.

Description	Used	Max
Dedicated Inputs	3	3
Clock/Inputs	2	2
I/O Macrocells	15	32
Buried Macrocells	13	32
PIM Input Connects	44	156

77 / 225 = 34%

Conclusion:

The timing parameters generated by Active HDL sim simulator shows that the maximum frequency at which we can operate is 21 MHz. Which is sufficient for low speed devices such as hand driven key board, control board used in applications such as instrumentation etc.

This project is designed for only 4 keys but it is clearly visible that it can be easily that it can be expanded for more no. of switches by increasing the no. of scan lines and R lines thereby increasing the no. of bit count of FRC and storage register size.

The design is made with providing sufficient time for debounce to occur(3 bit debonce counter 8 clk cycles). If this time is further reduced we can increase the output rate thus providing faster output.

The design can also be added to include a FIFO which stores a number of keys so that when FIFO is full it can give interrupt to Microprocessor to read a block of keys simultaneously rather then interrupting the MPU for each key press.this increases the overall speed of operation also.

Bibliography:

Digital Design By: JOHN F. WAKERLY.

INTEL 8279 Programmable Key Board controller Data sheet.

VHDL for Synthesis By: KEVIN SKEHILL.