Simulation Report for Hand Gesture Recognition

Pradnesh Patil[1] **and Aniket Bhatia**[2]

[1]160010021 Electrical Engineering Department, IIT Bombay

[2]160020001 Electrical Engineering Department, IIT Bombay

Abstract

This report is the study of the technique used for recognizing basic hand gestures like swiping to control various activities and simulating the result to check the best parameters and conditions and algorithms for the swift functioning of the circuit.

1. Goal

The goal of the project is to identify various hand gesture like swiping left or right and using it to activate various functions. The gestures that we are trying to recognize are:-

1. Swiping Left and Right.
2. Making a Fist
3. Complete opening of the Palm.

The major use of such a system can be seen in the automobile industry which can reduce driver's visual and cognitive demands. In the new era of automated cars basic controls can be linked to various hand gesture to allow the driver to control the vehicles position. Hence such a system would play an indispensable role in the "automated world", ranging from door opening sensors to, as mentioned above, automated cars this system will find its use in umpteen places.

2. Importing Blocks of the Circuit and Idea Summary

2.1. The Optical Sensor Input

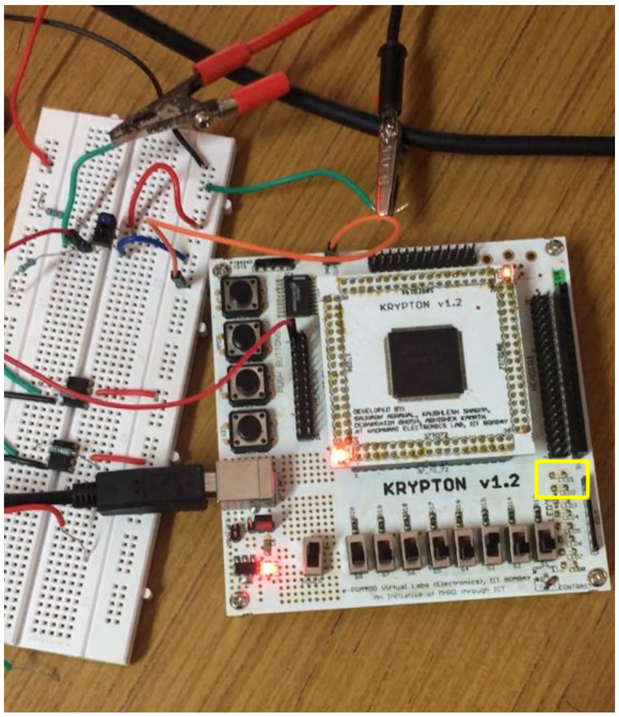
The input is generated using the optical sensor TCRT5000. The led and the photo diode are powered by a 5V supply and the output voltage is taken at the collector.

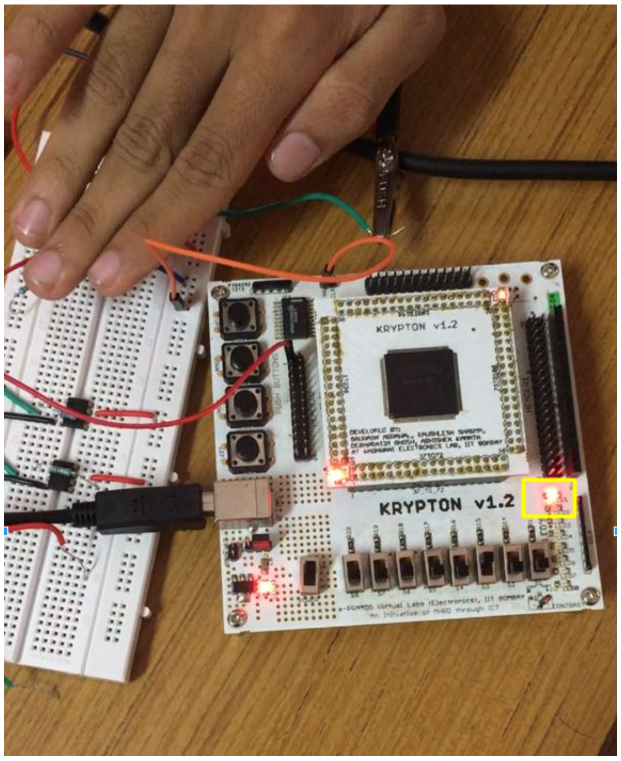
The data for the output depending on the distance of the hand from the sensor is as follows.

Distance (in mm)	**Output Voltage (in V)**
2	0.27
4	1.2
6	2.4
8	3.8
10	4.3
12	4.9

TABLE 1. Variation of Output Voltage with Distance

This is the basic building block of our circuit. When we pass our hand over the sensor, it's voltage drops. We use this fact to proceed. Since the output of the sensor ranges from 0-5V, it is anyway perfect for us to convert into the digital spectrum. In the digital world 0V is taken as 0 ad 5V as 1. For the digital part of our circuit we are using the CPLD Krypton Board. In this basic building block the output from the sensor goes into the krypton board which coded in a particular fashion as to respond correspondingly to the input. Here for the basic building block circuit when our hand is near the sensor, the particular LED doesn't glow, but when our hand is sufficiently close to the sensor it causes a particular LED to glow (as shown in the pictures below- the yellow box encloses the LED which is supposed to glow).





But the problem we had with this circuit is that, for the krypton board to recognize the input as 0 the voltage has to be very near to 0. So from the observation table our hand has to be very near to the sensor (¡4mm). This is practically undesirable. We thus need a solution for this. For the same reason we use the following- Comparator

2.2. The Comparator Circuit

We are using the IC LM 311 to use as an voltage comparator. The voltage switching point is set at around a voltage 4V. The supply voltage to the Op-Amp is +5V and ground. Referring to the table1 we can see that setting the switching point at 4 V gestures are recorded for any distance less than 9mm making it more user friendly to use the device. This is an important change because in real life we can't always keep out hands very close to the sensors.

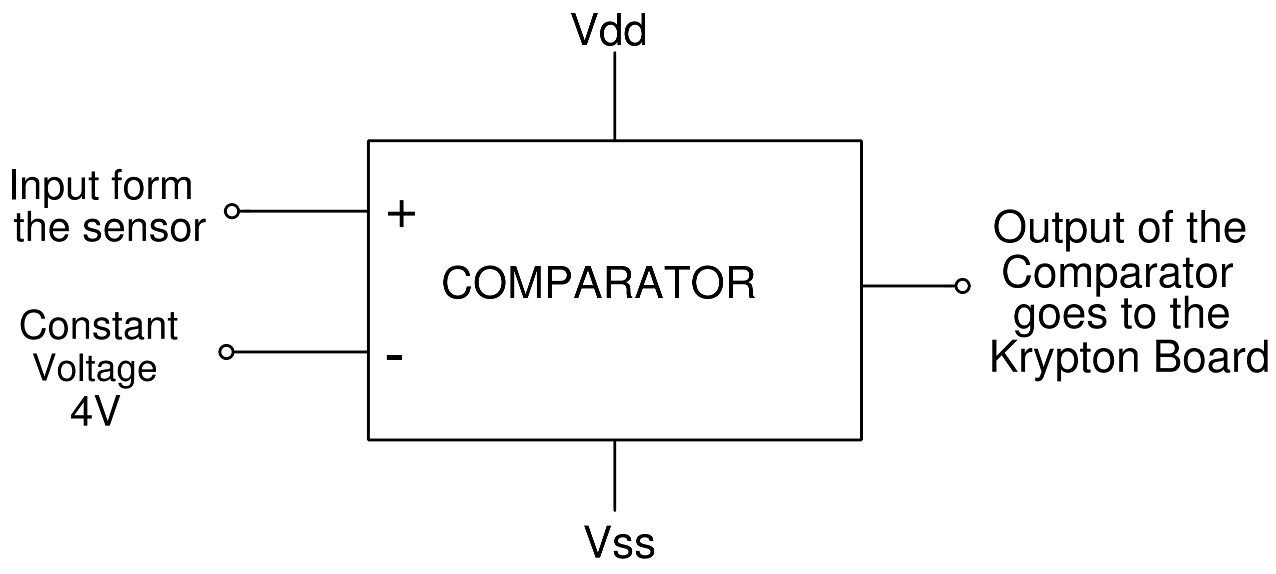


FIGURE 1. Block Diagram for a Comparator

2.3. The working algorithm on the Krypton Board

Now after taking inuput from the sensors we will give it to the Krypton Board. Coming to the application of left/right swipe the positioning of the sensors would be next to each other, separated by enough distance. When our hand passes over any of the sensors the output for that particular sensoe would be 1, since the voltage would drop below 4V. When there is no hand over the sensor the output of the sensor is 1.
For representation of the system (also for the simulation) we create a Finite State Machine (FSM), which is then represented in VHDL Language (& com-

piled in the Quartus Software, and Simulations done on the same). The algorithm goes as follows: We have three states : Rest, State1 and State2. The input 10 takes the system from rest to State1, a further input of 01 takes it from State1 to State2. This two inuts occur when you swipe left. After this when there is no hand over the sensors the input automatically becomes 11 which sends the system to the rest state. The output is therefore deemed 1 when the input changes from 10 to 01. For right swipe the occurrences of the events is exactly opposite. The FSM is shown in the below picture.

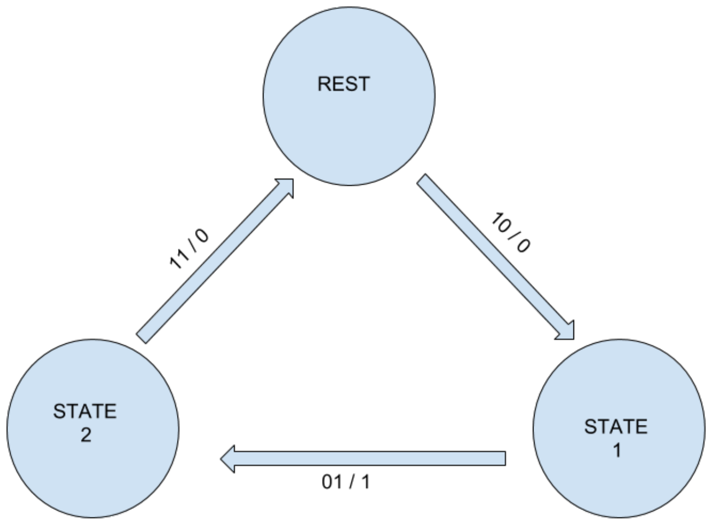
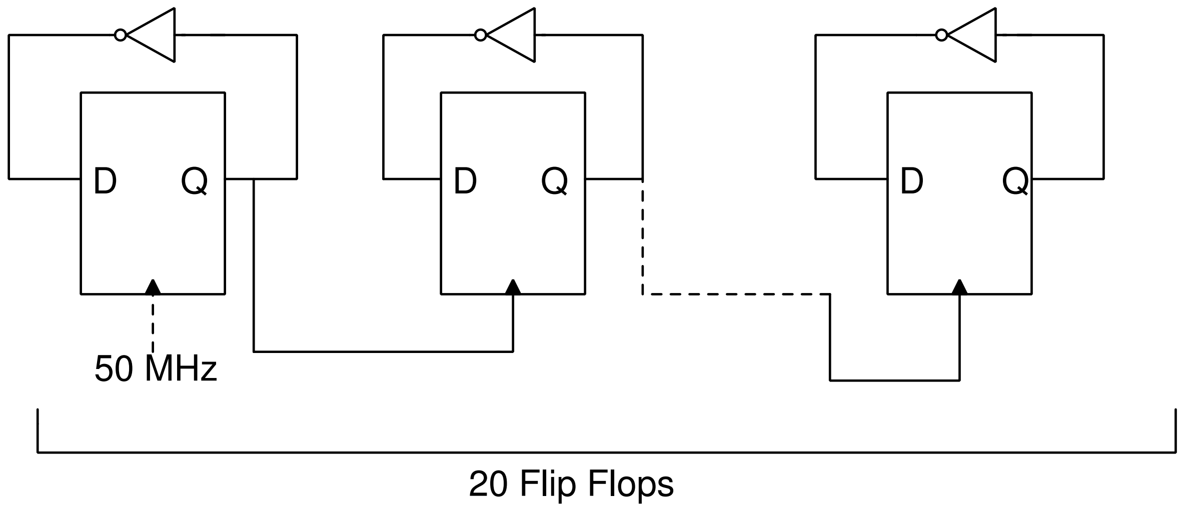
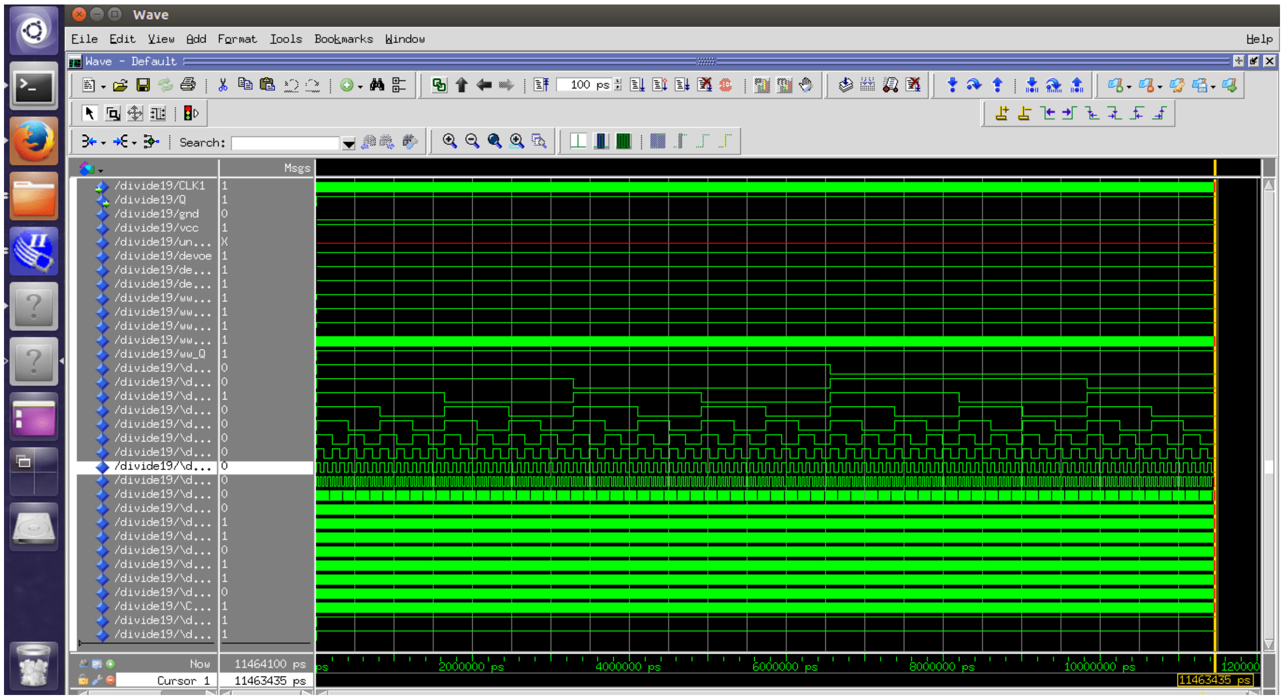


FIGURE 2. Finite State Diagram for recognizing a swipe

3. Simulation Setup

For the simulation to be done we need a clock which samples the input atleast every 20ms. So we will require a clock of 50 Hz. But only have a 50 MHz clock available to us. So we will use a clock divider, which takes the input a 50 MHz clock and the output as 50 Hz clock. The simulation result for the code for this s attached in the lower sections. The circuit for this purpose consisited of D-Flip Flops. Their arrangement is shown below.





```
/divide19/CLK1    1
/divide19/Q       1
/divide19/gnd     0
/divide19/vcc     1
/divide19/un...   X
/divide19/devoe   1
/divide19/de...   1
/divide19/de...   1
/divide19/ww...   1
/divide19/ww...   1
/divide19/ww...   1
/divide19/ww...   1
/divide19/ww_Q    1
/divide19/\d...   0
/divide19/\d...   0
/divide19/\d...   1
/divide19/\d...   0
/divide19/\d...   0
/divide19/\d...   0
/divide19/\d...   0
/divide19/\d...   0
/divide19/\d...   0
/divide19/\d...   0
/divide19/\d...   0
/divide19/\d...   1
/divide19/\d...   1
/divide19/\d...   0
/divide19/\d...   1
/divide19/\d...   1
/divide19/\d...   0
/divide19/\C...   1
/divide19/\d...   1
/divide19/\d...   1
Now               11464100 ps
Cursor 1          11463435 ps
```

4. The VHDL Code to control the swiping gesture

```
    library std;
use std.standard.all;

library ieee;
```

```
use ieee.std_logic_1164.all;

package EE224_Components is
   component INVERTER is
port (a: in std_logic; b : out std_logic);
   end component;
   component AND_2 is
port (a, b: in std_logic; c : out std_logic);
   end component;
   component OR_2 is
port (a, b: in std_logic; c : out std_logic);
   end component;
   component NAND_2 is
port (a, b: in std_logic; c : out std_logic);
   end component;
   component XOR_2 is
port (a, b: in std_logic; c : out std_logic);
end component;
component OneBitAdder is
port( x0,y0,cin : in std_logic;
      cout, s0 : out std_logic);
   end component;
component eightbitadder is
port(  x : in std_logic_vector(7 downto 0);
y: in std_logic_vector(7 downto 0);
z : out std_logic_vector( 7 downto 0));
end component ;
end EE224_Components;



library ieee;
-- std_logic type and associated fulibrary std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

library work;
-- package of component declarations..
use work.EE224_Components.all;
```

```
library ieee;
use ieee.std_logic_1164.all;

library std;
use std.standard.all;

entity INVERTER is
  port (a: in std_logic;
         b: out std_logic);
end entity INVERTER;
architecture Behave of INVERTER is
begin
  b <= not a;
end Behave;



library ieee;
-- std_logic type and associated fulibrary std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

library work;
-- package of component declarations..
use work.EE224_Components.all;

library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;


library ieee;
use ieee.std_logic_1164.all;
entity AND_2 is
```

```
  port (a, b: in std_logic;
         c: out std_logic);
end entity AND_2;
architecture Behave of AND_2 is
begin
  c <= a and b;
end Behave;


library ieee;
-- std_logic type and associated fulibrary std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

library work;
-- package of component declarations..
use work.EE224_Components.all;

library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;


library ieee;
use ieee.std_logic_1164.all;

entity OR_2 is
  port (a, b: in std_logic;
         c: out std_logic);
end entity OR_2;
architecture Behave of OR_2 is
begin
  c <= a or b;
end Behave;
```

```
library ieee;
-- std_logic type and associated fulibrary std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

library work;
-- package of component declarations..
use work.EE224_Components.all;

library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

library ieee;
use ieee.std_logic_1164.all;
entity NAND_2 is
  port (a, b: in std_logic;
         c: out std_logic);
end entity NAND_2;
architecture Behave of NAND_2 is
begin
  c <= not (a and b);
end Behave;


library ieee;
-- std_logic type and associated fulibrary std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

library work;
-- package of component declarations..
use work.EE224_Components.all;
```

```
library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

library ieee;
use ieee.std_logic_1164.all;library ieee;
use ieee.std_logic_1164.all;


entity XOR_2 is
  port (a, b: in std_logic;
         c: out std_logic);
end entity XOR_2;
architecture Behave of XOR_2 is
begin
  c <= (a xor b);
end Behave;

library ieee;
-- std_logic type and associated fulibrary std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

library work;


library ieee;
use ieee.std_logic_1164.all;

entity D_FF is
  port (D, CLK: in std_logic; Q: out std_logic);
end entity;
architecture whatDoYouCare of D_FF is
begin

   process (CLK)
   begin
```

```
if CLK'event and (CLK = '1') then
     Q <= D;
end if;
   end process;

end whatDoYouCare;
-- package of component declarations..
use work.EE224_Components.all;

library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

entity divide2 is
  port ( CLK: in std_logic; Q: out std_logic);
end entity;
architecture whatDoYouCare1 of divide2 is

component D_FF is
  port (D, CLK: in std_logic; Q: out std_logic);
end component;

signal Qnot ,Q1,d: std_logic;

begin

dff : D_FF port map(d, CLK, Q1);
--here, we were supposed to initialize it with something, but in logi
-- always have some value already, so we just define a signal d, whos
-- will work as the initial input

Qnot <= not Q1;
D<= Qnot;
Q <= Q1;

end whatDoYouCare1;
```

```
-- package of component declarations..
use work.EE224_Components.all;

library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

entity divide19 is
  port ( CLK1: in std_logic; Q: out std_logic);
end entity;
architecture struct of divide19 is

component divide2 is
  port ( CLK: in std_logic; Q: out std_logic);
end component;

signal ck: std_logic_vector(19 downto 0);
signal d : std_logic;

begin

d0 : divide2 port map (CLK1, ck(0));
d1 : divide2 port map (ck(0), ck(1));
d2 : divide2 port map (ck(1), ck(2));
d3 : divide2 port map (ck(2), ck(3));
d4 : divide2 port map (ck(3), ck(4));
d5 : divide2 port map (ck(4), ck(5));
d6 : divide2 port map (ck(5), ck(6));
d7 : divide2 port map (ck(6), ck(7));
d8 : divide2 port map (ck(7), ck(8));
d9 : divide2 port map (ck(8), ck(9));
d10 : divide2 port map (ck(9), ck(10));
d11 : divide2 port map (ck(10), ck(11));
d12 : divide2 port map (ck(11), ck(12));
d13 : divide2 port map (ck(12), ck(13));
d14 : divide2 port map (ck(13), ck(14));
d15 : divide2 port map (ck(14), ck(15));
d16 : divide2 port map (ck(15), ck(16));
d17 : divide2 port map (ck(16), ck(17));
```

```
d18 : divide2 port map (ck(17), ck(18));
d19 : divide2 port map (ck(18), Q);

end struct;

------------------------THE FSM MACHINE--------------------------
-- package of component declarations..
use work.EE224_Components.all;

library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.al l;

entity fsm is
  port ( clk, x: in std_logic_vector(1 downto 0); y: out std_logic);
end entity;

architecture Struct of fsm is

component D_FF is
  port (D, CLK: in std_logic; Q: out std_logic);
end component;

component divide19 is
  port ( CLK1: in std_logic; Q: out std_logic);
end component;

signal s0, s1, ns0, ns1, clknew : std_logic;

begin


ns1 <= (not(s0) and not(s1) and not(x(0)) and x(1)) or (not(s0) and s
ns0 <= (not(s0) and s1 and x(0) and not(x(1)));
y <= (not(s0) and s1 and x(0) and not(x(1))) or (q1 and q0 and x0 and

divider : divide19 port map (clk, clknew);
dff1 : D_FF port map (ns0, clknew, s0);
dff2 : D_FF port map (ns1, clknew, s1);
```

```
end struct;
-----------------------------------------------------------------
```