DIGITAL COMMUNICATIONS AND VHDL

(EC-452) Lab Manual

Prepared by

Y.Sri Chakrapani, M.Tech
(Lecturer)
&
D.Swetha, M.Tech
(Lecturer)

DEPARTMENT OF ECE
BAPATLA ENGINEERING COLLEGE
BAPATLA

LIST OF EXPERIMENTS

Experiments Based on Hardware

1. Generation and Detection of PCM.
2. Generation and Detection of ASK.
3. Generation and Detection of FSK.
4. Generation and Detection of PSK&QPSK.
5. Generation and Detection of TDM
6. Generation and Detection of DPSK
7. Delta Modulation and Demodulation.

VHDL Modeling and Synthesis of the Following Experiments

8. Logic Gates.
9. Multiplexers/De-Multiplexers.
10. Design of ALU.
11. 4 bit Magnitude Comparator.
12. JK, D & T Flip-Flops
13. Synchronous Counters
14. Asynchronous Counters

1. Generation and Detection of PCM

Aim: To study the Pulse Code Modulation and Demodulation using PCM trainer kit

Apparatus: PCM kit, CRO and connecting probes

Theory:

Pulse code modulation is a process of converting a analog signal into digital. The voice or any data input is first sampled using a sampler (which is a simple switch) and then quantized. Quantization is the process of converting a given signal amplitude to an equivalent binary number with fixed number of bits. This quantization can be either mid-tread or mid-raise and it can be uniform or non-uniform based on the requirements. For example in speech signals, the higher amplitudes will be less frequent than the low amplitudes. So higher amplitudes are given less step size than the lower amplitudes and thus quantization is performed non-uniformly. After quantization the signal is digital and the bits are passed through a parallel to serial converter and then launched into the channel serially.

At the demodulator the received bits are first converted into parallel frames and each frame is de-quantized to an equivalent analog value. This analog value is thus equivalent to a sampler output. This is the demodulated signal.

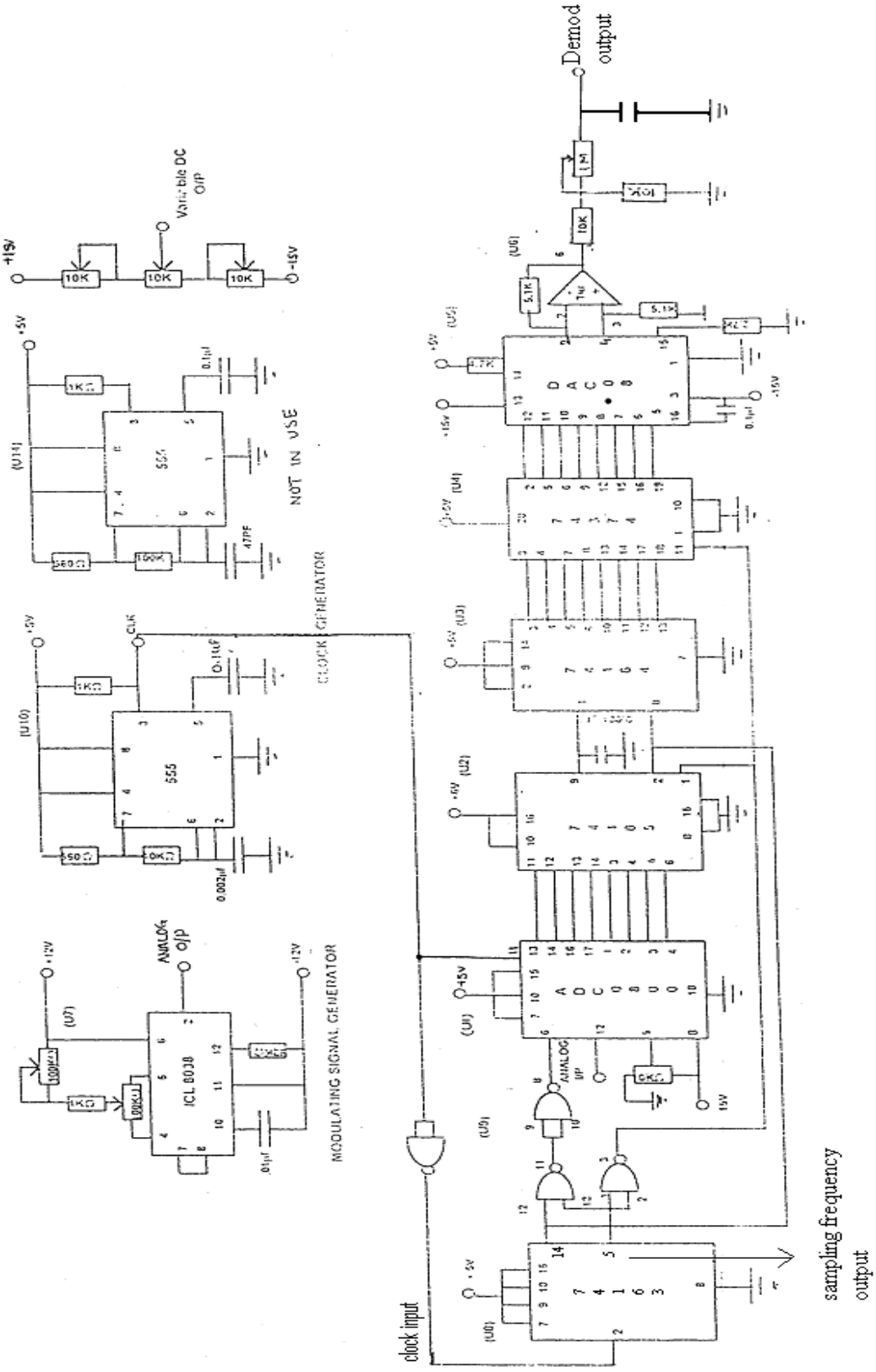
In the kit this is implemented differently. The analog signal is passed trough a ADC (Analog to Digital Converter) and then the digital codeword is passed through a parallel to serial converter block. This is modulated PCM. This is taken by the Serial to Parallel converter and then through a DAC to get the demodulated signal. The clock is given to all these blocks for synchronization. The input signal can be either DC or AC according to the kit. The waveforms can be observed on a CRO for DC without problem. AC also can be observed but with poor resolution.

Procedure:

The kit is self-explanatory. Identify the blocks according to the theory mentioned and generate a PCM modulated signal and demodulate to check if the same signal is obtained or not. The steps are also given below in a detailed fashion:

1. Power on the Future Tech PCM kit.
2. Measure the frequency of sampling clock.
3. Apply the DC voltage as modulating signal.
4. Connect the DC input to the ADC and measure the voltage.
5. Connect the clock to the timing and control circuit.
6. Note the binary work from LED display. The serial data through the channel can be observed in the CRO.
7. Also observe the binary word at the receiver end.
8. Now apply the AC modulating signal at the input.
9. Observe the waveform at the output of DAC.
10. Note the amplitude of the input voltage and the codeword. Also note the value of the output voltage. Show the codeword graphically for a DC input.

Kit diagram:


Demod output
Variable DC O/P
+15V
-15V
NOT IN USE
CLOCK GENERATOR
MODULATING SIGNAL GENERATOR
ANALOG O/P
clock input
sampling frequency output

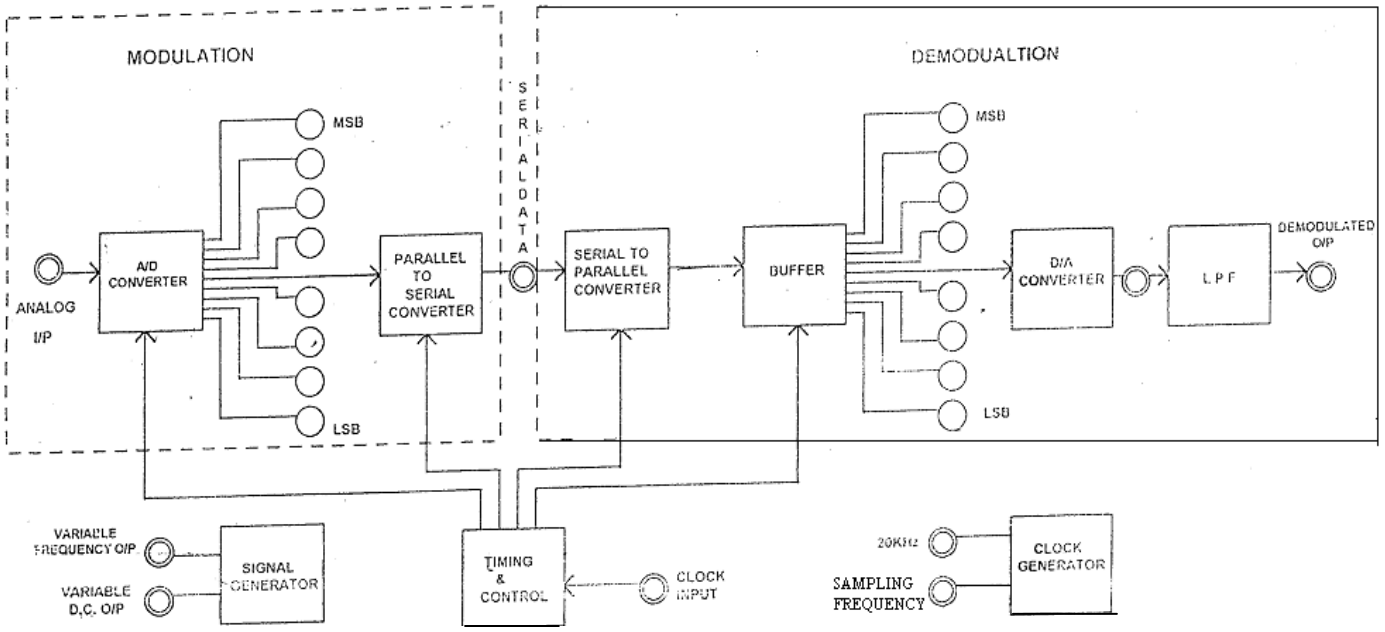
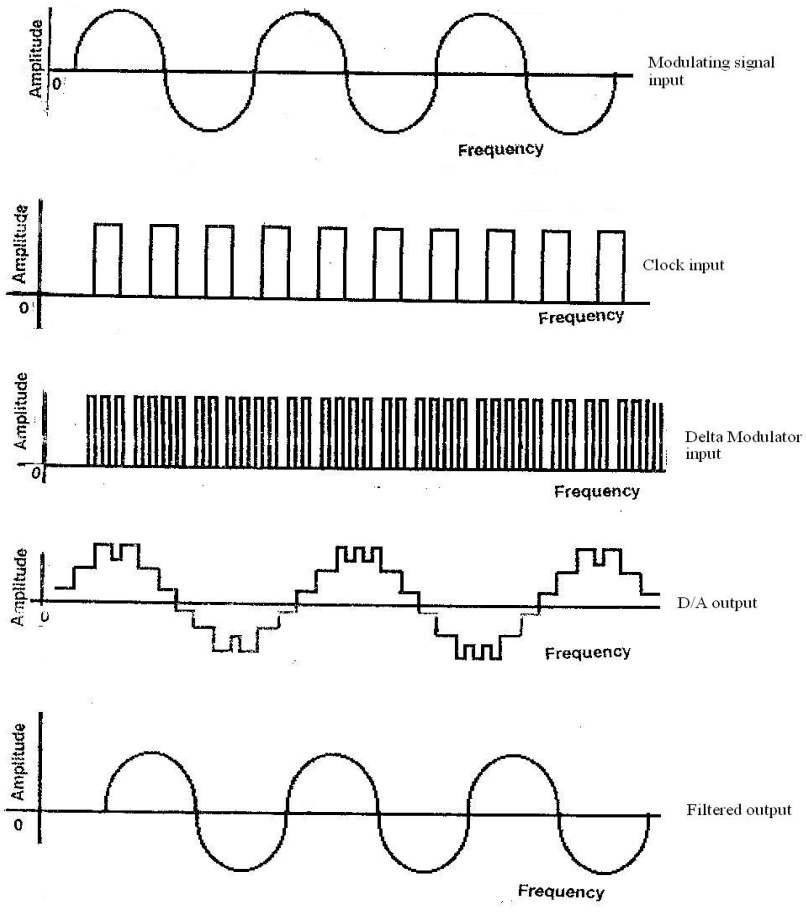
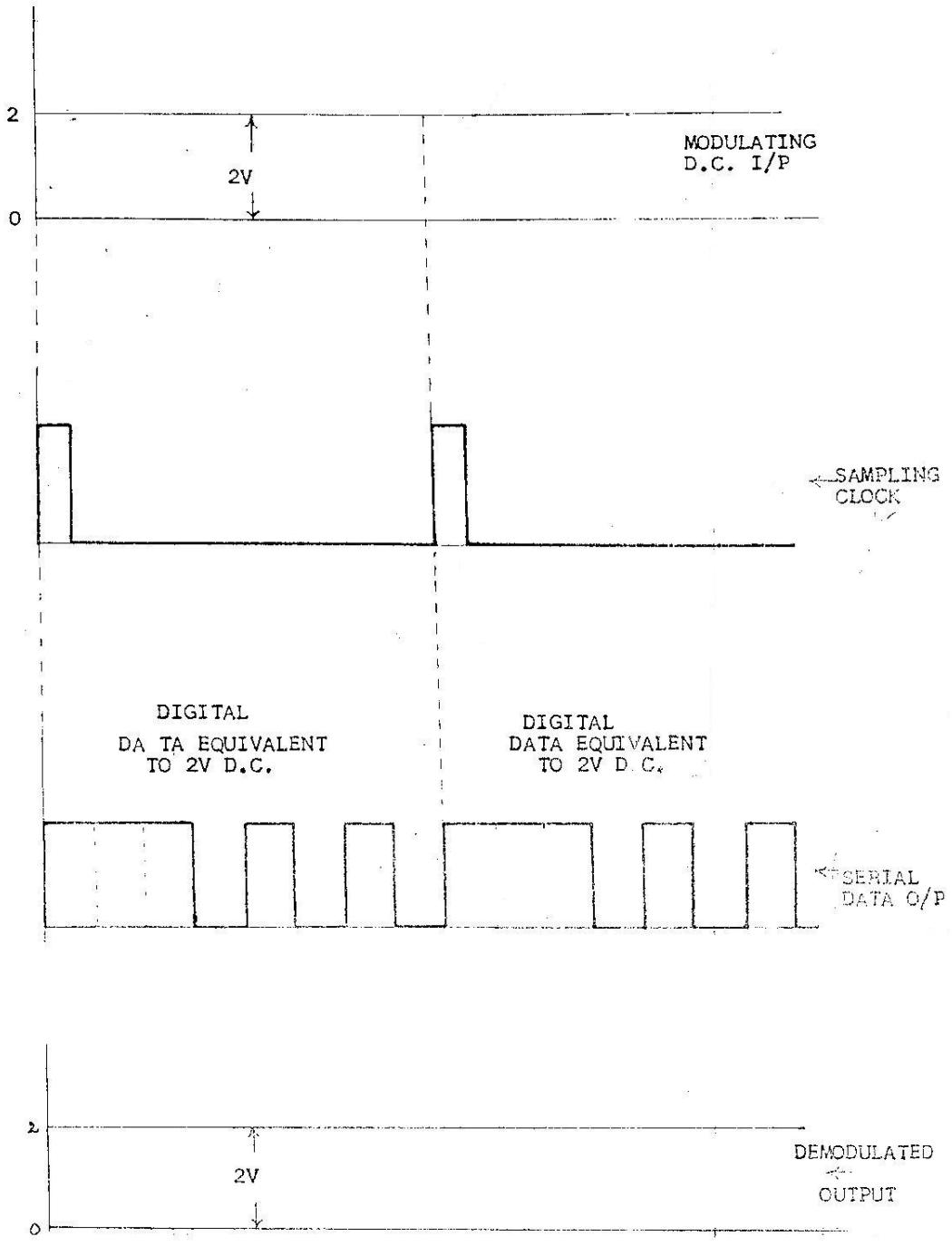


Fig 2: Block diagram of PCM

Model waveforms:




2
0
2V
MODULATING
D.C. I/P
SAMPLING
CLOCK
DIGITAL
DA TA EQUIVALENT
TO 2V D.C.
DIGITAL
DATA EQUIVALENT
TO 2V D.C.
SERIAL
DATA O/P
2
0
2V
DEMODULATED
OUTPUT

2. Generation and Detection of ASK

Aim: To study the operation of Amplitude Shift Keying modulation and demodulation with the help of circuit connections.

Apparatus:

1. Resistors	1.2KΩ	3
2. Transistor	BC 107	2
3. CRO	30MHz	1
4. Function generator	0-1MHz	1
5. Regulated Power Supply	0-30V, 1A	1
6. CRO Probes	---	1

Theory:

The binary ASK system was one of the earliest form of digital modulation used in wireless telegraphy. In an binary ASK system binary symbol 1 is represented by transmitting a sinusoidal carrier wave of fixed amplitude A_c and fixed frequency f_c for the bit duration T_b where as binary symbol 0 is represented by switching of the carrier for T_b seconds. This signal can be generated simply by turning the carrier of a sinusoidal oscillator ON and OFF for the prescribed periods indicated by the modulating pulse train. For this reason the scheme is also known as on-off shift testing. Let the sinusoidal carrier can be represented by Ec (t) $=A_c \cos(2\Pi f_c t)$ then the binary ASK signal can be represented by a wave s(t) given by $S(t) = A_c \cos(2\Pi f_c t)$, symbol 1 ASK signal can be generated by applying the incoming binary data and the sinusoidal carrier to the two inputs of a product modulator. The resulting output is the ASK wave. The ASK signal which is basically product of the binary sequence and carrier signal has a same as that of base band signal but shifted in the frequency domain by $\pm f_c$. The band width of ASK signal is infinite but practically it is $3/T_b$.

Procedure:

1. Connect the circuit as per the circuit diagram.
2. Switch on the supply.
3. Apply the sinusoidal carrier signal from the function generator to the collector terminal of the transistor with 10v (p-p) amplitude and10KHz frequency.
4. Apply the Binary signal from the pulse generator to the Base terminal of the transistor with 5v (p-p) amplitude and 2 KHz frequency.
5. Observe the output of ON/OFF keying from ASK modulator circuit using CRO.
6. Now vary the Amplitude and frequency of the binary signal and observe the output changes of ASK modulated Wave & compare it with the modulating data signal applied to the modulator input.
7. The output of the low pass filter is a demodulated binary signal.

Precautions:

1. Keep the connections tight.
2. Do not lift the IC pins. Check if the ICs of the kit are in tact.

Circuit diagram:

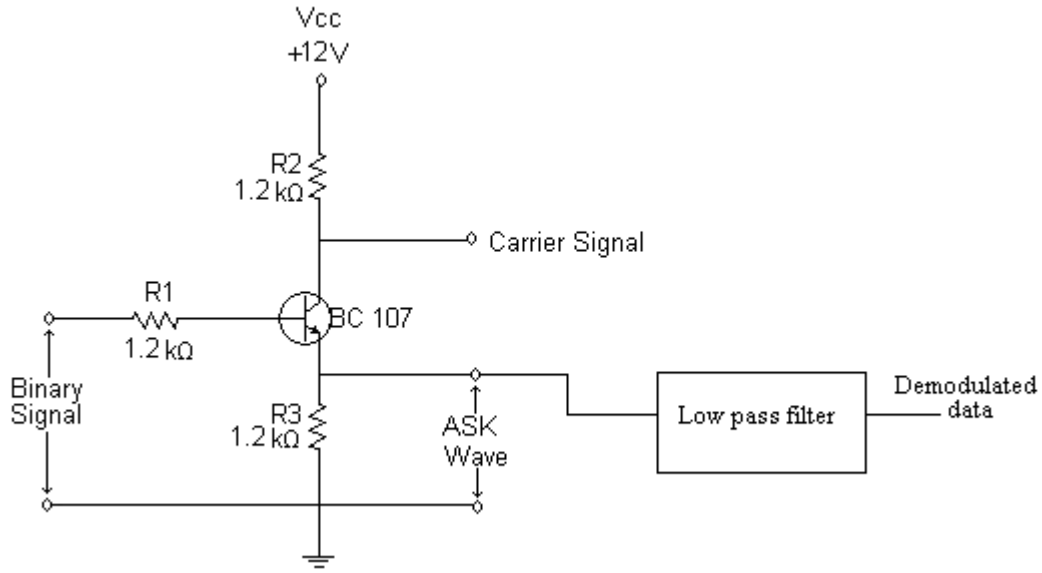
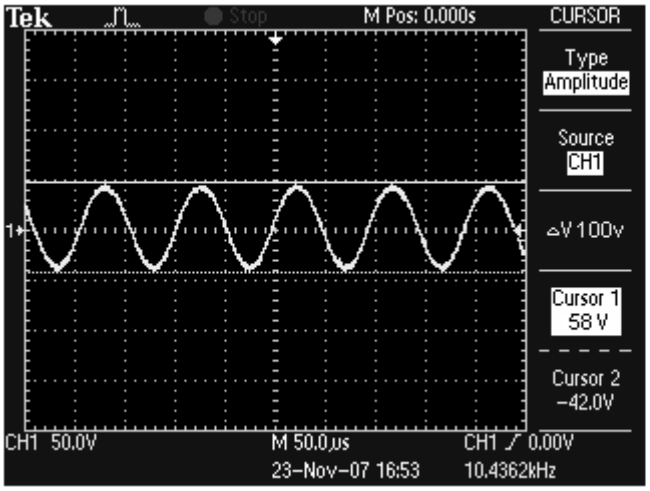


Fig: 1 Amplitude Shift Keying and demodulation Circuit

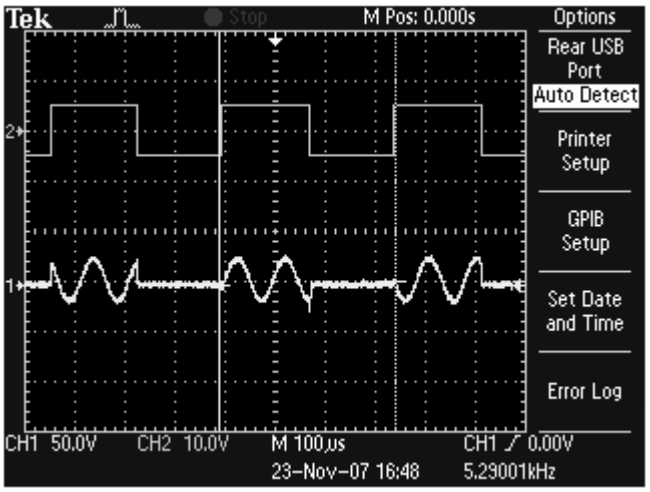
Model waveforms:



Amplitude: 10v (p-p)

Frequency: 10 KHz

(a)



Amplitude: 4v (p-p)

Frequency: 2KHz

Amplitude: 4V (p-p)

Frequency: 2KHz

(b)

Fig: 2 Waveforms of (a) Carrier signal (b) Data signal & ASK wave

3. Generation and Detection of FSK

Aim: To study the operation of Frequency Shift Keying modulation and demodulation with the help of kit.

Apparatus: FSK kit, CRO and connecting probes

Theory:

Frequency Shift Keying is the process generating a modulated signal from a digital data input. If the incoming bit is 1, a signal with frequency f1 is sent for the duration of the bit. If the bit is 0, a signal with frequency f2 is sent for the duration of this bit. This is the basic principle behind FSK modulation.

Basically a 555 timer is used as an Astable multivibrator, which generates a clock pulse of frequency determined by the values of R and C in this circuit. This is divided by 2, 4, 8 and 16 using 74163 IC, and two of these outputs are used in a NAND logic gates circuit, to generate a FSK modulated wave. To this NAND gates' circuit a binary data sequence is also supplied. The circuit operation causes a frequency f1 for bit 1, and f2 for bit 0.

In the demodulator circuit, the FSK modulated signal is applied to a high Q tuned filter. This filter is tuned to the frequency of either 0 or 1. This filter passes the selected frequency and rejects the other. The output is then passed through a FWR (Full Wave Rectifier) circuit and the output is now above zero volts only. It is then passed through a comparator; if the input to the comparator is greater than threshold value, the output is 1, else it is 0. This digital output of the comparator is the demodulated FSK output.

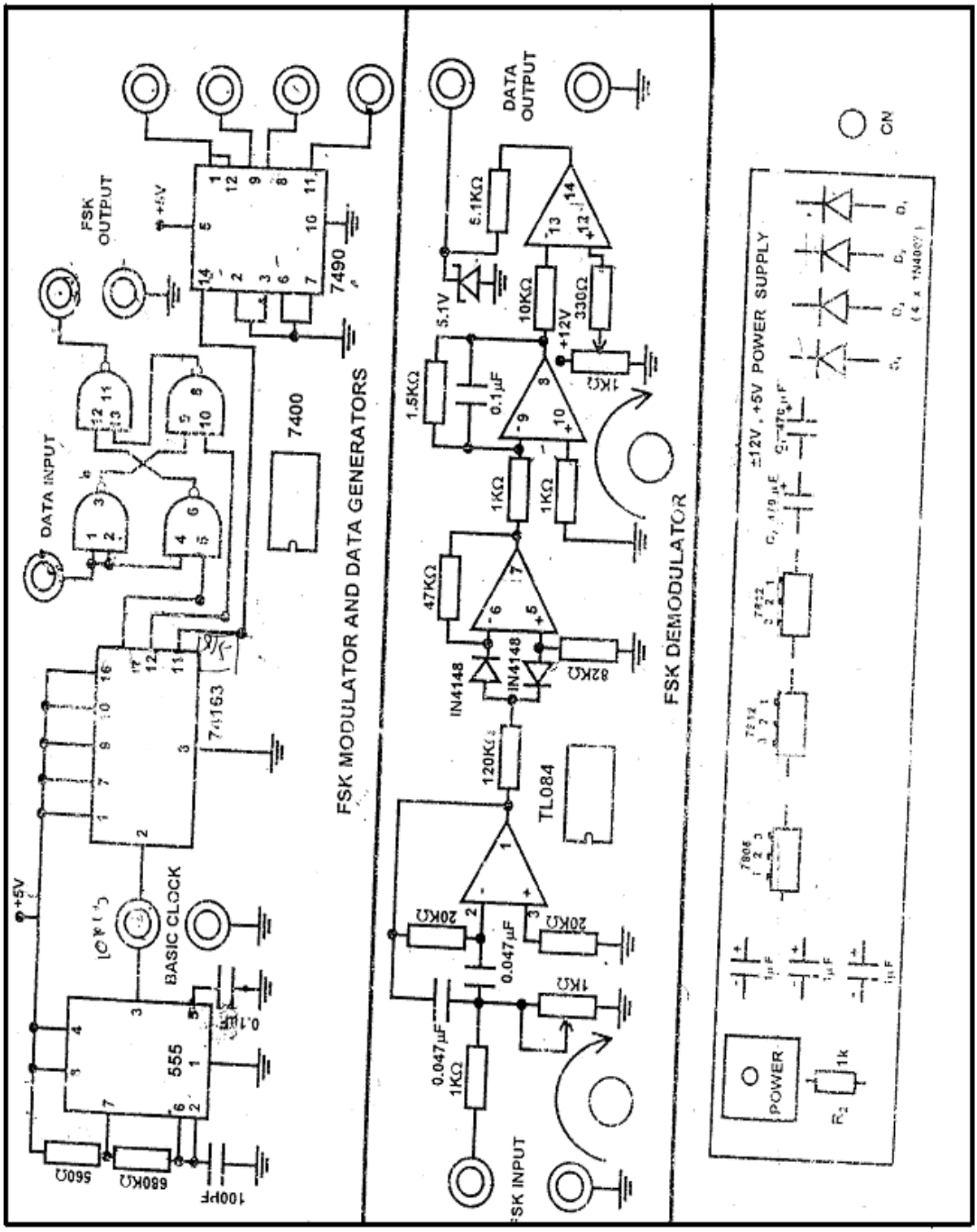
Procedure:

1. Power on the kit. Apply one binary sequence as input message to the FSK modulator. This sequence is taken from decade counter outputs.
2. Give the modulated FSK signal to the demodulator input. Observe the output of the demodulator on the CRO along with the original data sequence. Adjust the tuning controls (of filter and comparator) so that input and demodulated outputs are the same. Do not touch the tuning controls after the adjustment is done.
3. Apply another data sequence input from the decade counter as input message and observe the demodulated output message. Both will be identical. Note the FSK modulated and demodulated waveforms. Sketch them with proper indications.

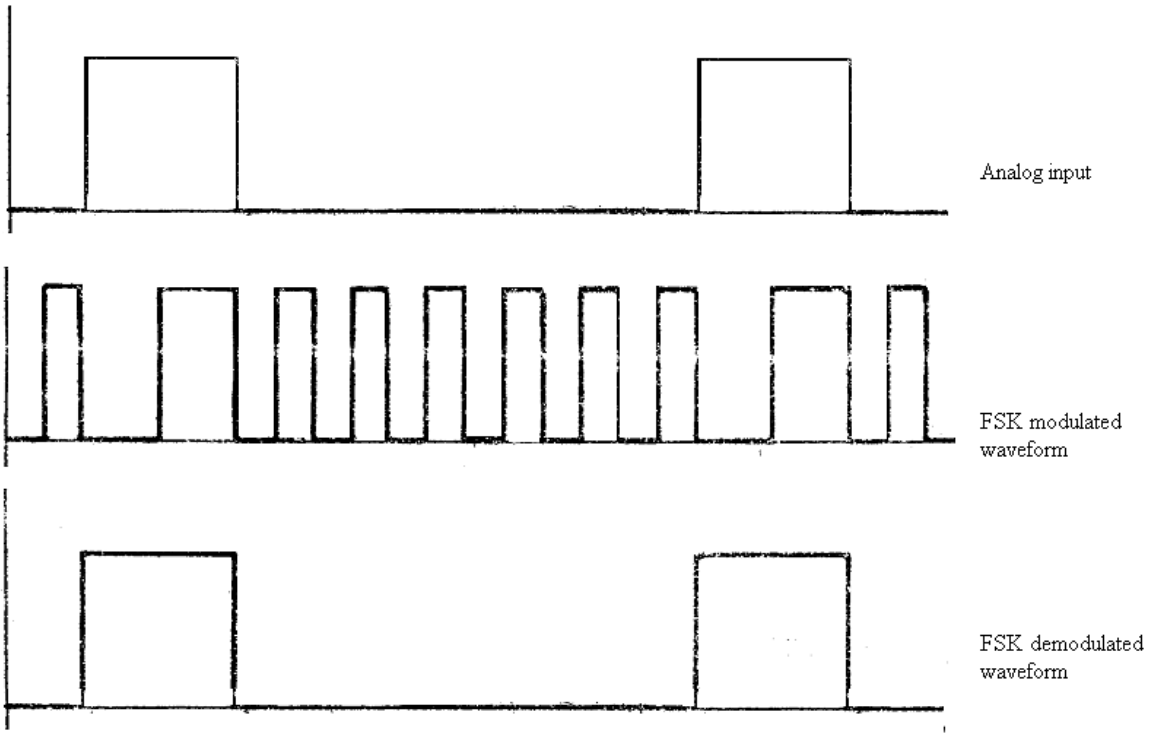
Precautions:

3. Keep the connections tight.
4. Do not lift the IC pins. Check if the ICs of the kit are in tact.

Kit diagram:


FSK MODULATOR AND DATA GENERATORS
FSK DEMODULATOR
DATA INPUT
FSK OUTPUT
7400
7490
74163
555
BASIC CLOCK
TL084
DATA OUTPUT
FSK INPUT
±12V, +5V POWER SUPPLY
POWER
ON

Model waveforms:



4. Generation and Detection of PSK

Aim: To study the operation of Phase Shift Keying modulation and demodulation with the help of kit.

Apparatus: PSK kit, CRO and connecting probes

Theory:

Phase Shift Keying is a digital modulation Technique. A cosinusoidal carrier of a fixed amplitude and frequency is taken. The digital data of 1's and 0's is converted to $s(t) = A_C \cos(2\pi f_c t)$ $s(t) = A_C \cos(2\pi f_c t + \pi)$ respectively. In the kit, phase shift keying is obtained using an OP-AMP circuitry and a switch. If incoming bit is 0, the output is same as the carrier; if it is zero, the output is 90^0 phase shifted version of the carrier signal. For demodulation a coherent detector is used. It has 3 parts, a multiplier, and integrator and decision device. The operation can be analyzed from the circuit.

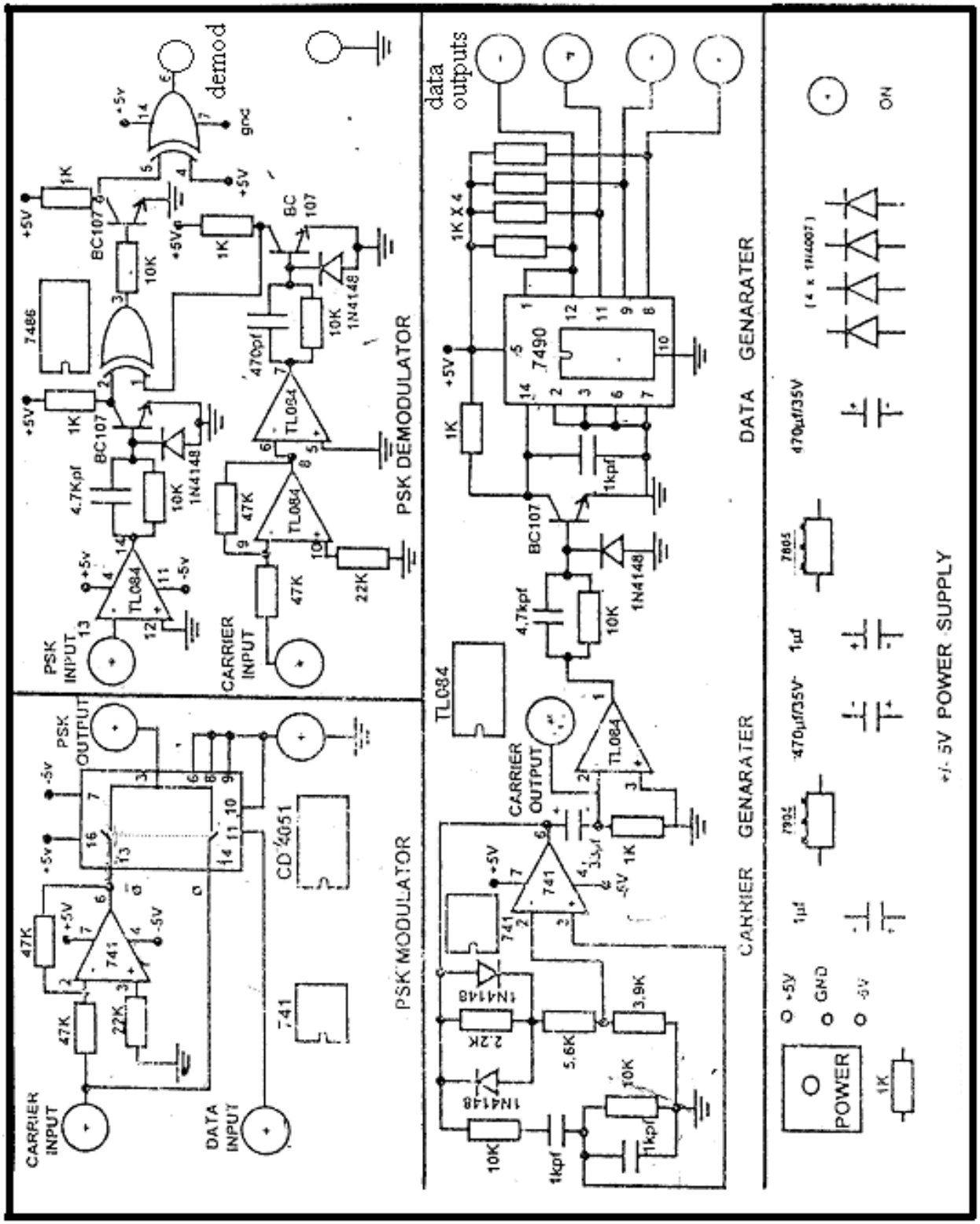
Procedure:

1. Power on the kit. Apply carrier signal to the input of the modulator.
2. Apply the modulating data signal from one of the decade counter's outputs. Observe the modulated signal in the CRO.
3. Apply the PSK modulator output to the input of demodulator. Also apply the carrier signal to the demodulator.
4. Observe the demodulator output on the CRO.
5. Compare the two signal amplitudes.

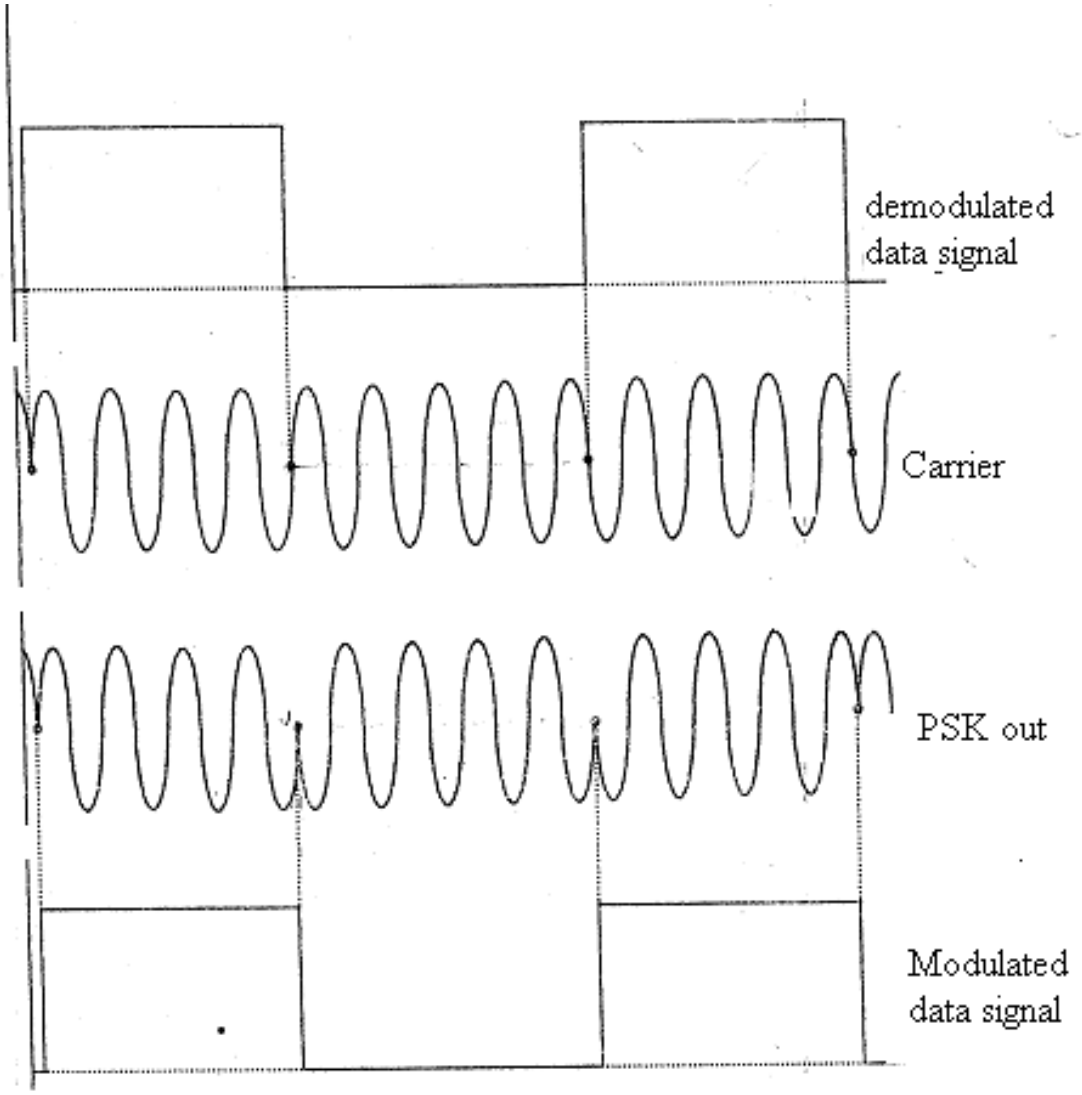
Precautions:

1. Keep the connections tight.
2. Do not lift the IC pins. Check if the ICs of the kit are in tact.

Kit diagram:


PSK MODULATOR
PSK DEMODULATOR
DATA GENARATER
CARRIER GENARATER
+/- 5V POWER SUPPLY
CARRIER INPUT
DATA INPUT
PSK OUTPUT
PSK INPUT
CARRIER INPUT
CARRIER OUTPUT
data outputs
demod
CD 4051
TL084
7490
7486
BC107
BC 107
1N4148
1N4007
POWER
ON
GND

Model waveforms:



5. Generation and Detection of TDM

Aim: To study the operation of Time Division Multiplexing and demultiplexing with the help of kit.

Apparatus: TDM kit, CRO and connecting probes

Theory:

TDM is the multiplexing technique in time domain. If there are more users who wish to communicate with another set of many users, they are provided a single channel over which sequential communication can happen. Each user is provided a time slot and the user sticks to his slot. The data of each user is sampled. In the channel, the first sample of user 1, then user 2.., are sent for a pre-described set of users. Then again the second sample of each user is sent in the same order. At the receiver synchronization of timing circuits allows, each user's information to be delivered to the correct destination.

In the kit four signals are analogous to 4 users. Each signal is given to the input of the multiplexer. The transmission of signal on the channel is controlled using a switch matrix controlled by a decoder circuit. The 2 by 4 decoder selection lines are driven by divide by 5, divide by 10 outputs. The corresponding output is from 0 to 3. The switch of the corresponding channel closes and the signal's sample is passed on to the channel. This can be seen from the kit diagram. At the receiver, the clock used for selection is synchronous with the transmitter clock. This ensures that the outputs arrive at the correct order. A duty cycle selector is there on the transmitter side of the circuit, which ensures the mode of operation. The synchronization can be achieved either by direct method or by using a PLL. Both are available on the kit.

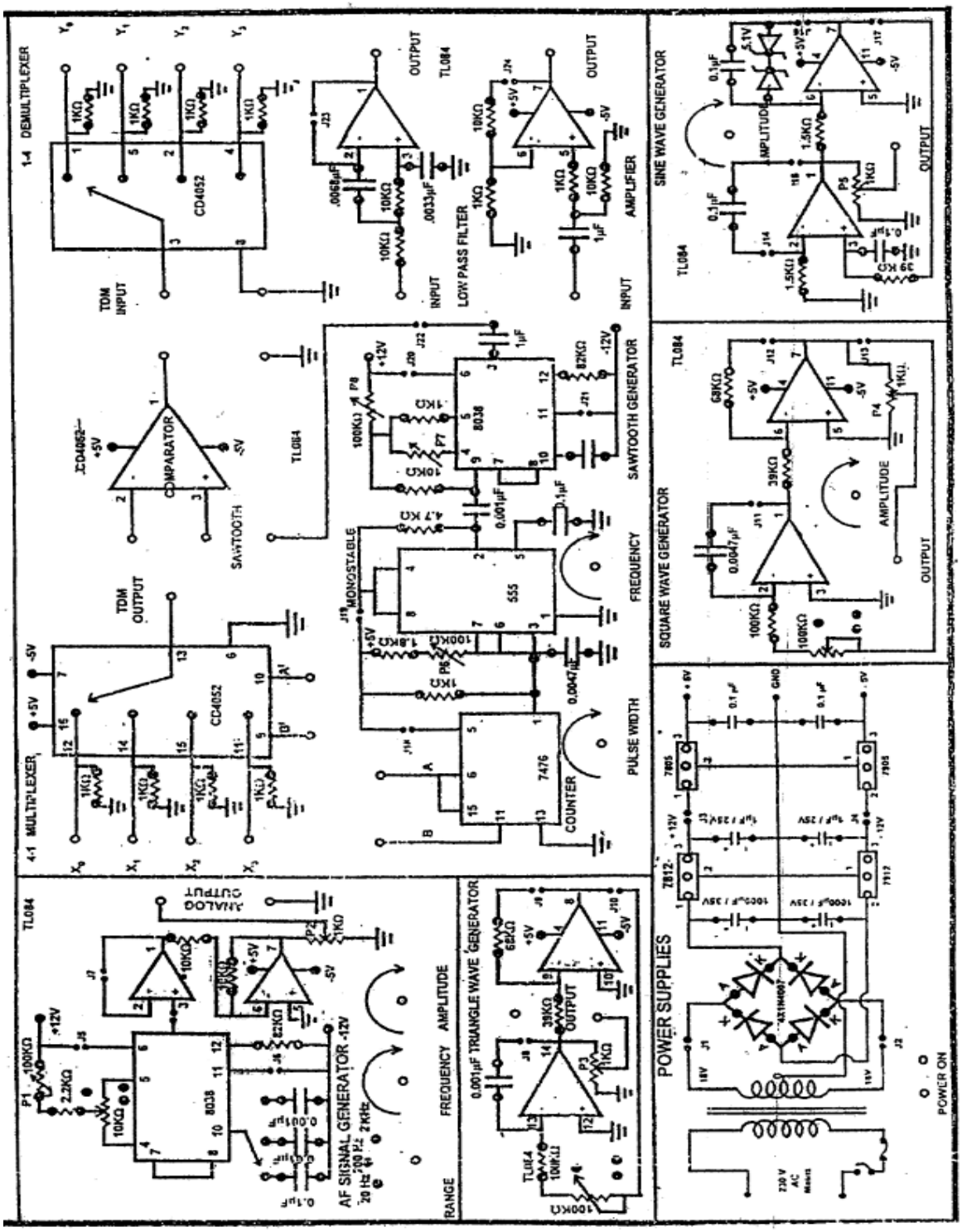
Procedure:

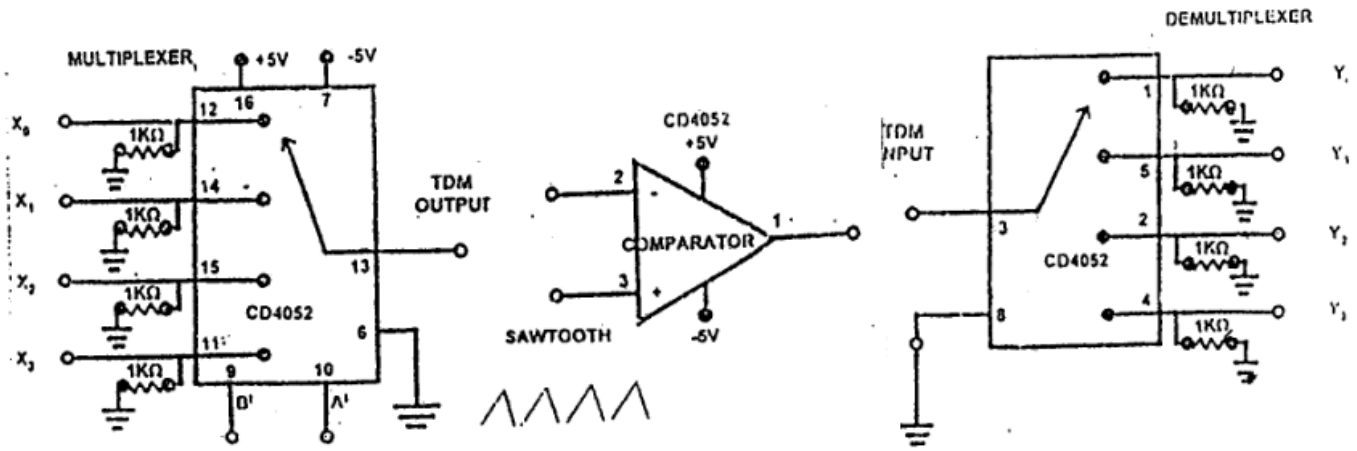
1. Turn on the power to the kit. Check the clock signal from the transmitter timing logic block. (64KHz)
2. Set the duty cycle selector switch to position 5.
3. Give the function generator outputs signals as input to the transmitter block. Adjust their amplitudes to known values using a CRO before giving to transmitter input.
4. Connect the Tx clock and Ch 0 reference of transmitter to the Rx clock and Ch 0 reference of Receiver block. This ensures that Ch 0 at Tx is same as Tx 0 at Rx.
5. Check the outputs of the demultiplxer or receiver and note the amplitudes. Compare the waveforms with those of sent waveforms. Both must be identical.
6. Now change the duty cycle selector from 5 to 11, position. Use the PLL circuit for synchronization. Connect the Tx clock to the PLL input and the Rx clock also to PLL circuit in the specified position. The Rx clock will now be synchronized to Tx clock. Repeat the steps 3 to 5 and collect the output waveforms.
7. Compare the input and output signals in both the synchronization methods.

Precautions:

1. Keep the connections tight.
2. Do not lift the IC pins. Check if the ICs of the kit are in tact.

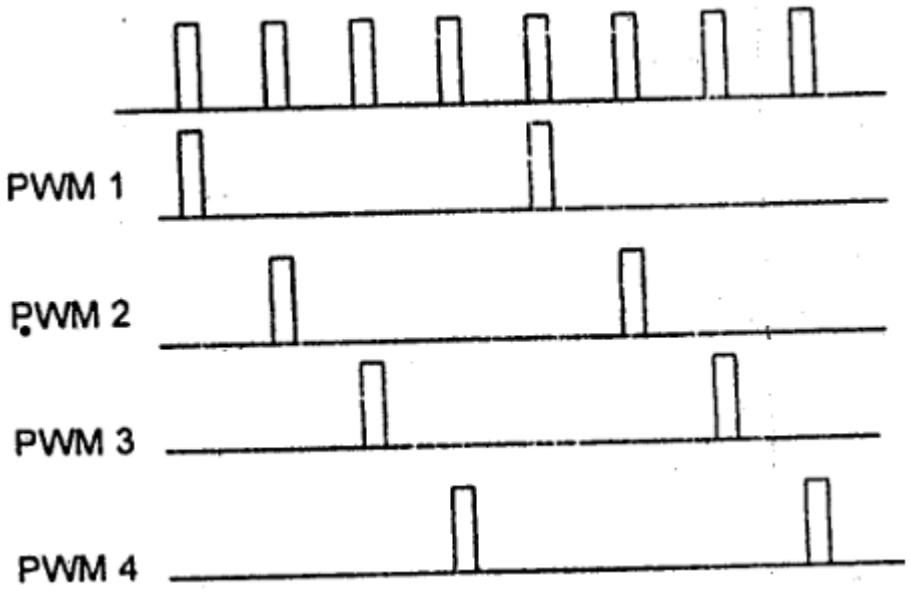
Kit diagram:





Model waveforms:

For Channel selection:



6. Generation and Detection of DPSK

Aim:
To study the various steps involved in generating the differential binary signal and differential phase shift keyed signal at the modulator and recovering the binary signal from the received DPSK signal.
Apparatus: Cathode ray oscilloscope, Probes, Patch chords

Theory:
Digital communication become important with the expansion of the use of computers and data processing ,and have continued to develop into a major industry providing the interconnection of the computer peripherals and transmission of data between distant sites. Phase shift keying is a relatively new system, in which the carrier may be phase shifted by =90 degrees for mark, and by -90 degrees for space. PSK has a number of similarities to FSK in many aspects; frequency of the carrier is shifted in accordance with the modulating square wave.

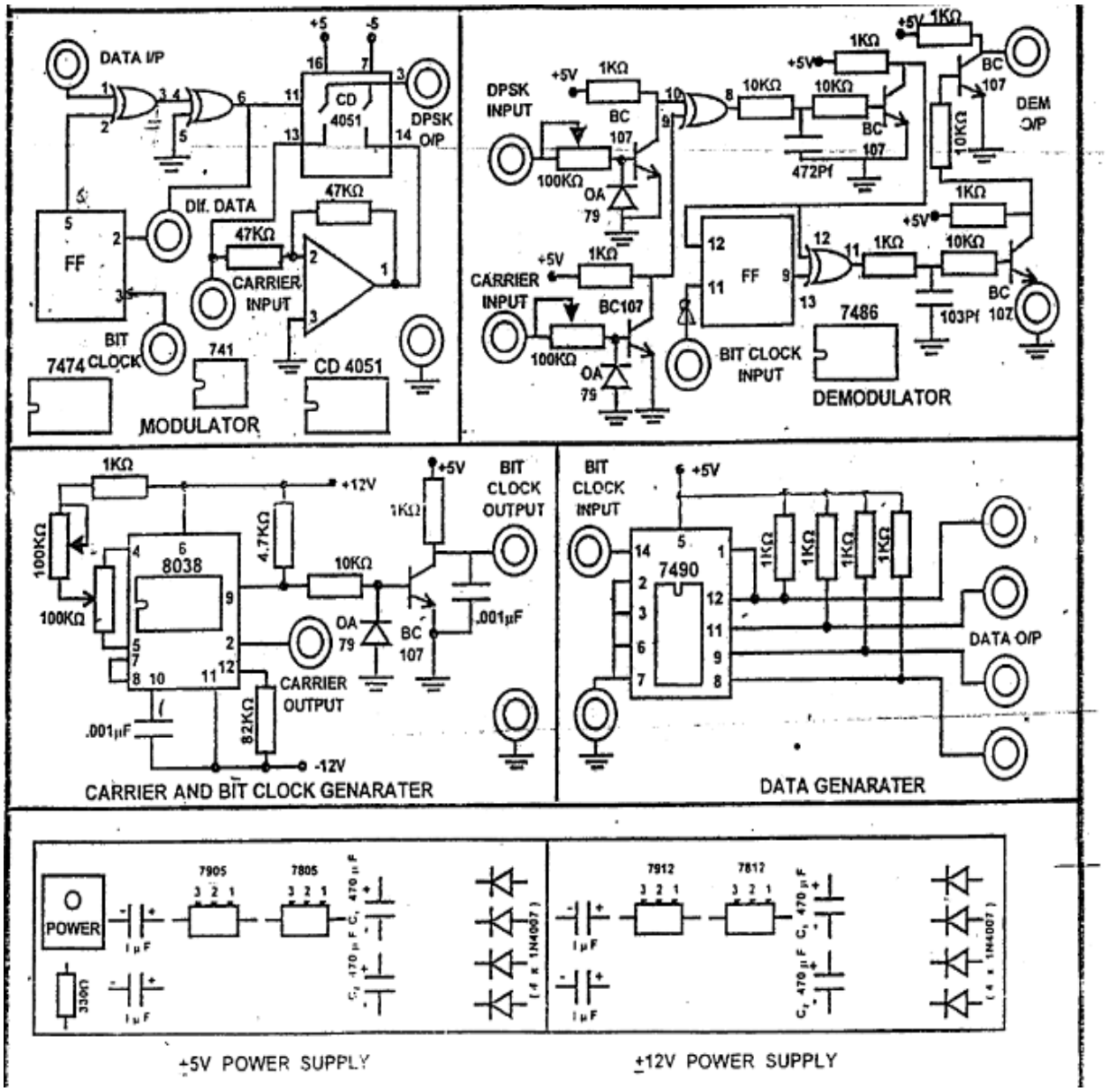
Fig-1 shows the circuit diagram of differential phase shift key modulation and de modulation .In this IC 8038 is a basic wave form generator which generates sine, square, triangle waveforms. The sine wave generated by 8038 IC is used as a carrier signal to the system. The square wave generated by 8038IC is +/_ 12V level. so this is converted into a +5V signal with the help of a transistor and diodes as shown in fig 1.this square wave is used as a clock input to a decade counter which generates the modulating data outputs.

The differential signal to the modulating signal is generated using an Exclusive-OR gate and a 1-bit delay circuit. CD 4051 is an analog multiplexer to which carrier is applied with and with out 180 degrees phase shift(created by using an operational amplifier connected in inverting mode)to the two inputs of the IC 741.Differential signal generated by Ex-OR gate (IC 7486)is given to the multiplexers control signal input. depending upon the level of the control signal, carrier signal applied with or with out phase shift is steered to the output.1-bit delay generation of differential signal to the input is created by using a D-flip flop(IC7474).During the demodulation ,the DPSK signal is converted into a +5V square wave using a transistor and is applied to one input of the Ex-OR gate. To the second input of the gate ,carrier signal is applied after conversion into a +5V signal. So the Ex-OR gate output is equivalent to the differential signal of the modulating data. This differential data is applied to the one input of the Ex-OR gate and to the second input, after 1-bit delta the same signal is given. So the output of this Ex-OR gate is modulating signal.

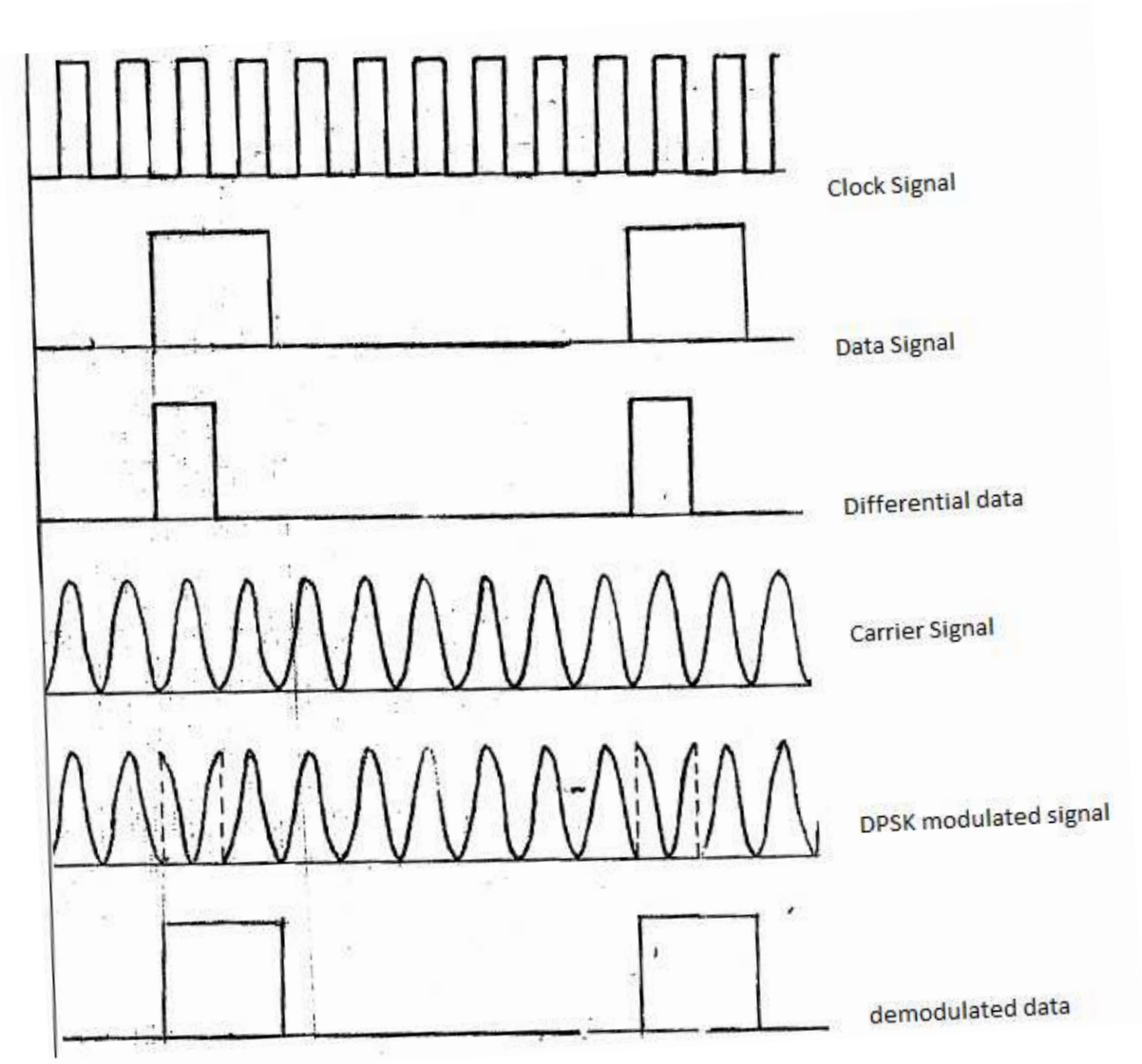
Procedure:
1. Switch on the experimental board.
2. Check the carrier signal and the data generator signals initially.
3. Apply the carrier signal to the carrier input of the DPSK modulator and give the data generator to the data input of DPSK modulator and bit clock output to the input of DPSK modulator and bit clock o/p to bit clock input of modulator.
4. Observe the DPSK modulating output with respect to the input data generated signal of dual trace oscilloscope, and observe the DPSK signal with respect to differential data also.

5. Give the output of the DPSK modulator signal to the input of the demodulator, give the bit clock output to the bit clock input to the demodulator and also give the carrier output to the carrier input of demodulator.
6. Observe the demodulator output with respect to data generator signal.

Kit Diagram:



Model Waveforms:



7. Generation and Detection of Delta Modulation

Aim: To study the operation of delta modulation and demodulation with the help of kit.

Apparatus: DM kit, CRO and connecting probes

Theory:

Delta modulation is the DPCM technique of converting an analog message signal to a digital sequence. The difference signal between two successive samples is encoded into a single bit code. The block and kit diagrams show the circuitry details of the modulation technique. A present sample of the analog signal m(t) is compared with a previous sample and the difference output is level shifted, i.e. a positive level (corresponding to bit 1) is given if difference is positive and negative level (corresponding to bit 0) if it is negative. The comparison of samples is accomplished by converting the digital to analog form and then comparing with the present sample. This is done using an Up counter and DAC as shown in block diagram. The delta modulated signal is given to up counter and then a DAC and the analog input is given to OPAMP and a LPF to obtain the demodulated output.

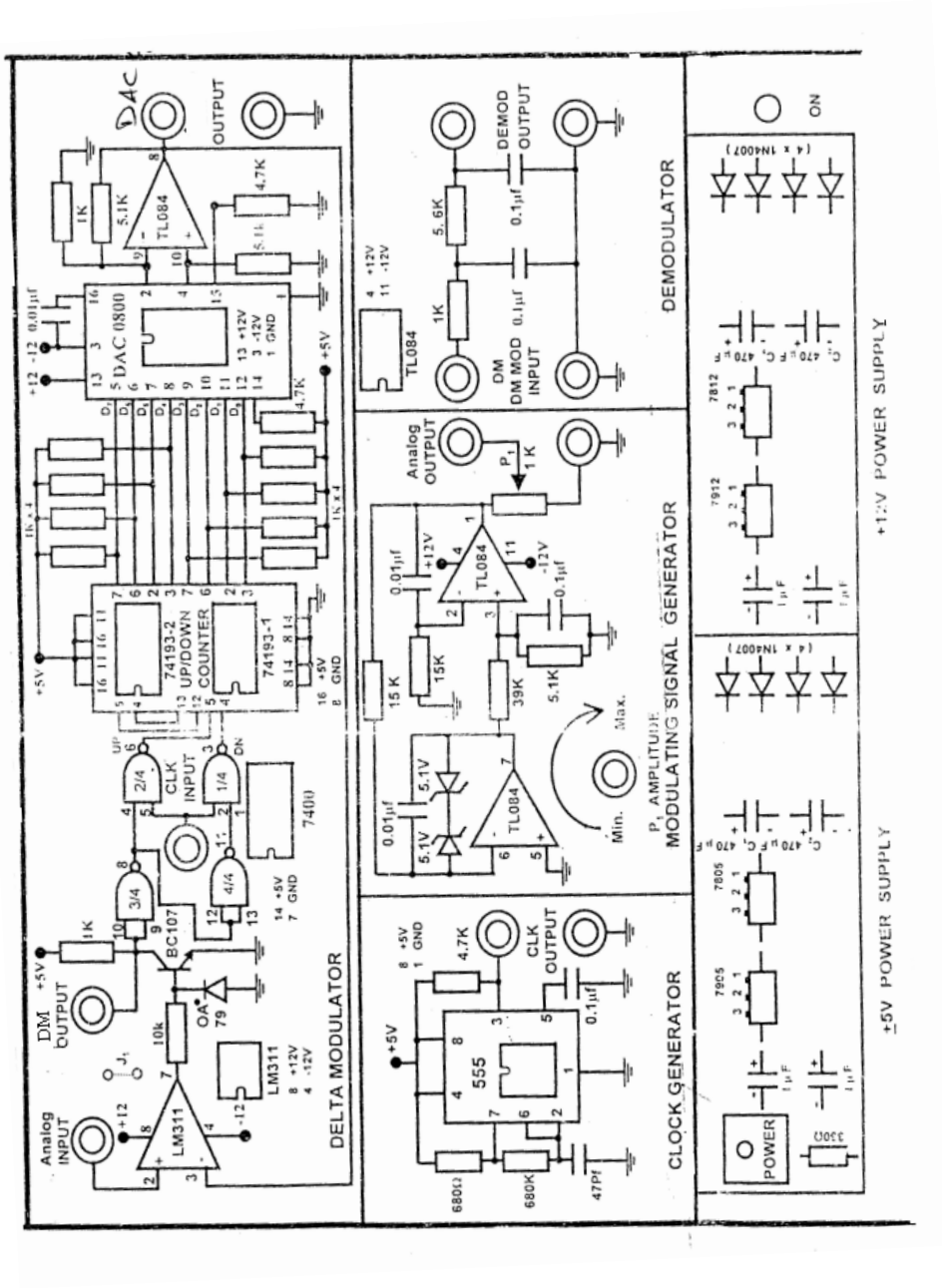
Procedure:

1. Switch on the kit. Connect the clock signal and the modulating input signal to the modulator block. Observe the modulated signal in the CRO.
2. Connect the DM output to the demodulator circuit. Observe the demodulator output on the CRO.
3. Also observe the DAC output on the CRO.
4. Change the amplitude of the modulating signal and observe the DAC output. Notice the slope overload distortion. Keep the tuning knob so that the distortion is gone. Note this value of the amplitude. This is the minimum required value of the amplitude to overcome slope overload distortion.
5. Calculate the sampling frequency required for no slope overload distortion. Compare the calculated and measured values of the sampling frequency.

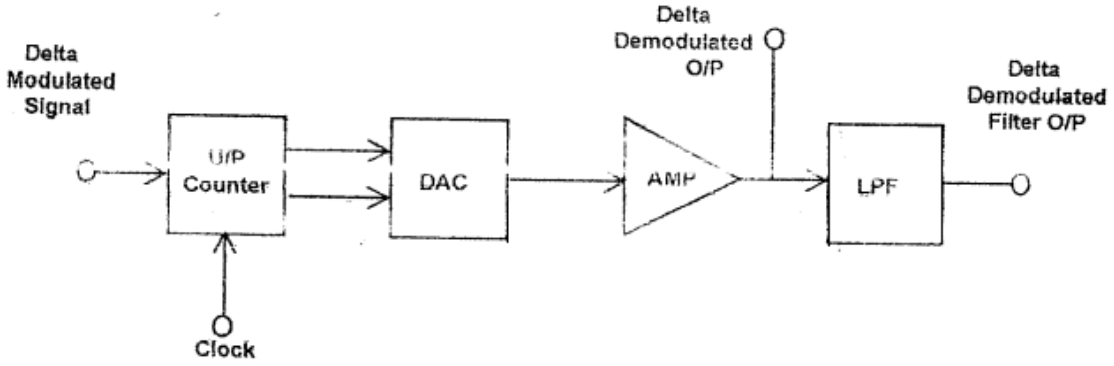
Precautions:

1. Keep the connections tight.
2. Do not lift the IC pins. Check if the ICs of the kit are in tact.

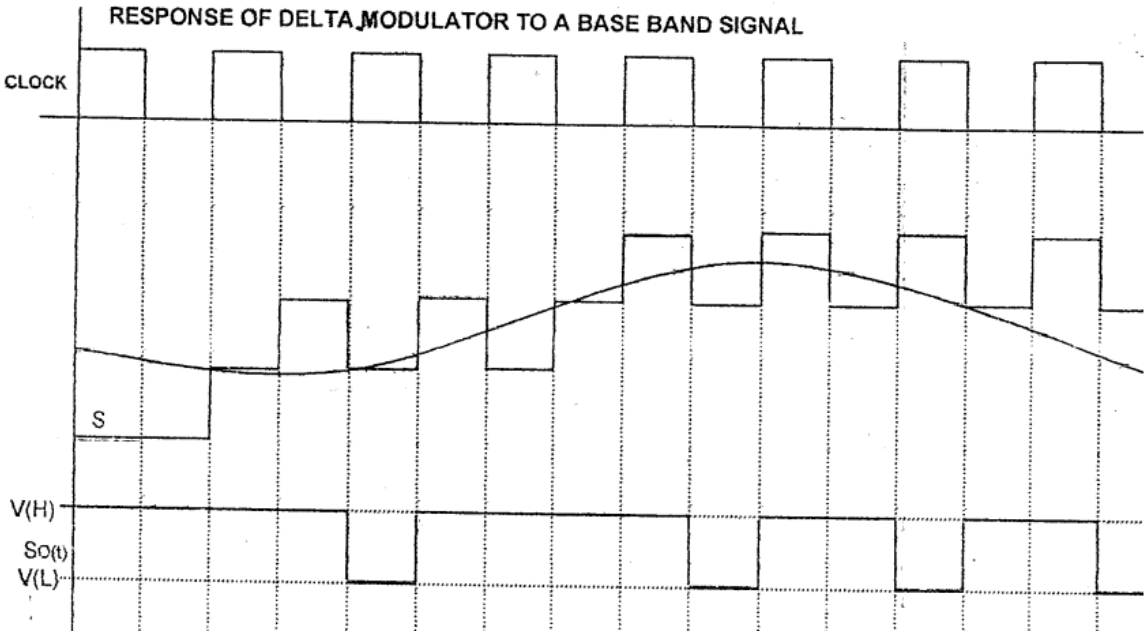
Kit diagram:


DELTA MODULATOR
CLOCK GENERATOR
MODULATING SIGNAL GENERATOR
DEMODULATOR
±5V POWER SUPPLY
±12V POWER SUPPLY
DAC
OUTPUT
DAC 0800
TL084
LM311
74193-2
UP/DOWN
COUNTER
74193-1
7400
555
BC107
OA 79
CLK
INPUT
CLK
OUTPUT
Analog
INPUT
DM
OUTPUT
Analog
OUTPUT
DEMOD
OUTPUT
DM
DM MOD
INPUT
Max.
Min.
POWER
ON
7805
7905
7812
7912
(4 x 1N4007)

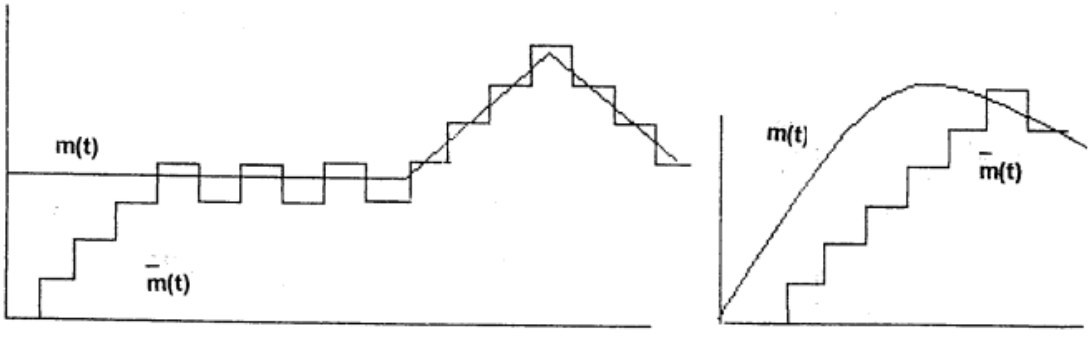
Block Diagram



Model Waveforms:



RESPONSE LIMITATIONS OF DELTA MODULATION



8.LOGIC GATES

8.1 AND GATE

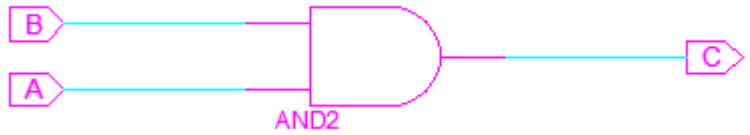
Aim:

To implement And Gate using VHDL.

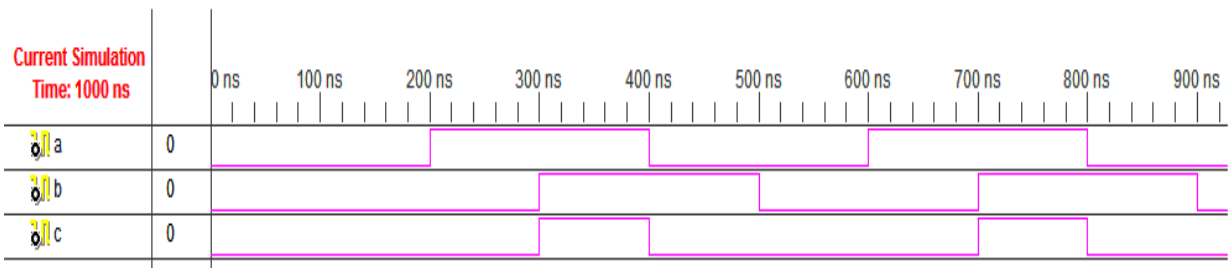
Code:

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

entity and1 is
     port(A,B : in std_logic;
             C : out std_logic);
end and1;

architecture archi of and1 is
begin
     C <= A and B;
end archi;
```

RTL SCHEMATIC:



Simulation Waveforms



8.2 OR GATE

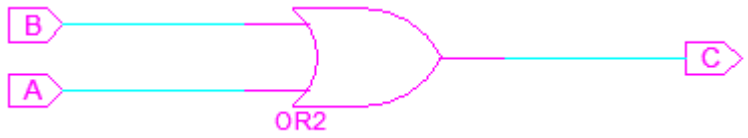
Aim:

To implement And Gate using VHDL.

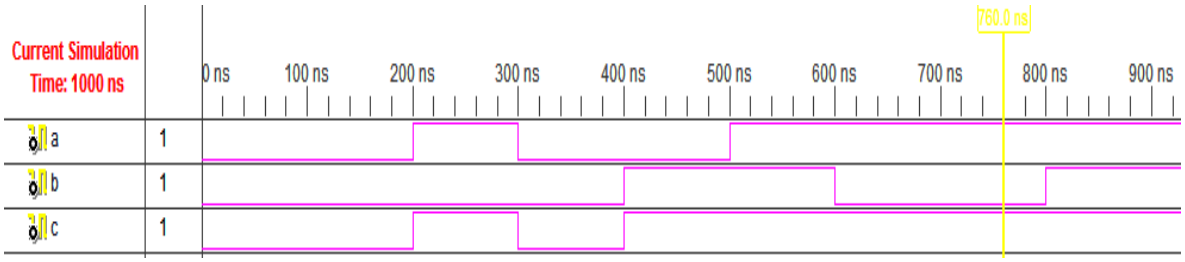
Code:

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

entity or1 is
     port(A,B : in std_logic;
             C : out std_logic);
end or1;

architecture archi of or1 is
begin
     C <= A or B;
end archi;
```

RTL Schematic:



Simulation Waveforms:



8.3 NAND GATE

Aim:

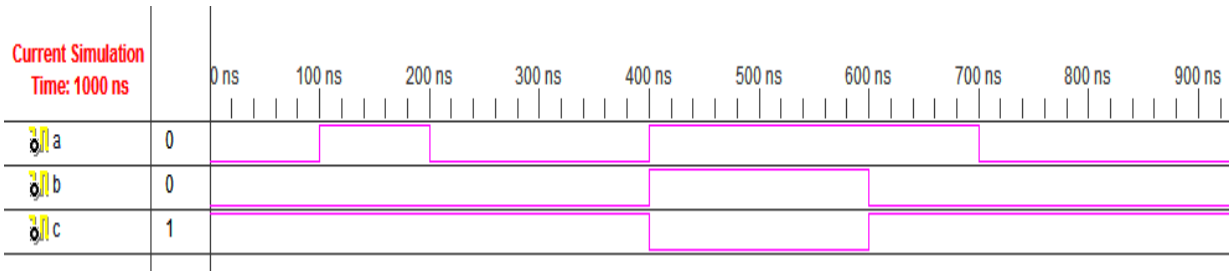
To implement Nand Gate using VHDL.

Code:

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

entity nand1 is
     port(A,B : in std_logic;
             C : out std_logic);
end nand1;

architecture archi of nand1 is
begin
     C <= A nand B;
end archi;
```

RTL Schematic:

Simulation Waveforms:



8.4 EX-OR GATE

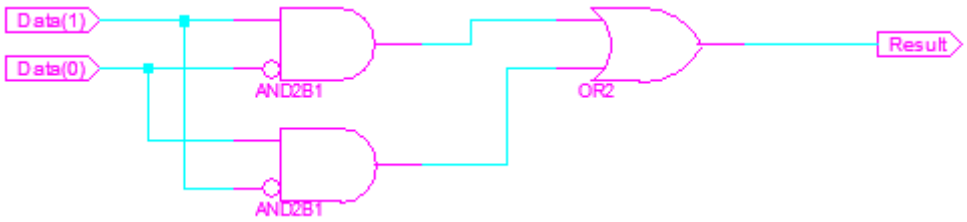
Aim:

To implement Nand Gate using VHDL.

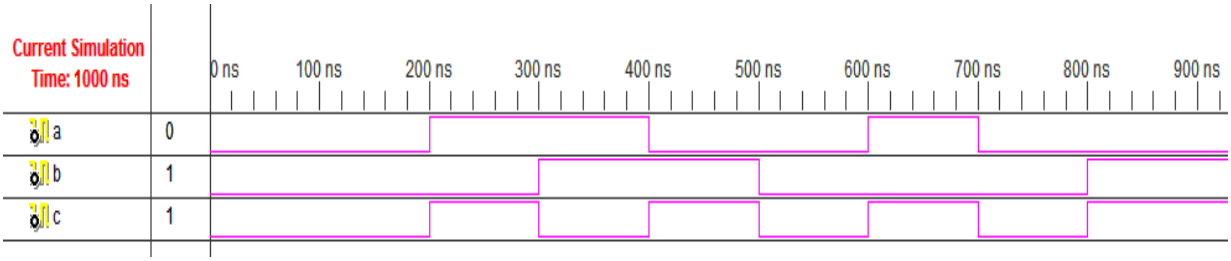
Code:

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

entity exor1 is
     port(A,B : in std_logic;
             C : out std_logic);
end exor1;

architecture archi of exor1 is
begin
     C <= A xor B;
end archi;
```

RTL Schematic:



Simulation Waveforms:



8.5 NOT GATE

Aim:

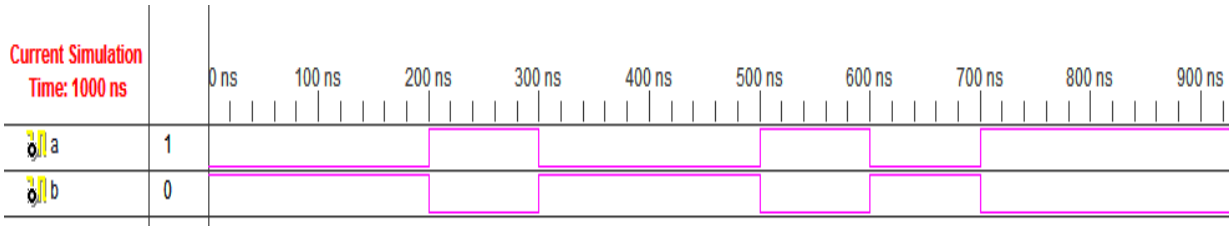
To implement Not Gate using VHDL.

Code:

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

entity exor1 is
     port(A : in std_logic;
          B : out std_logic);
end exor1;

architecture archi of exor1 is
begin
     B<=  not A;
end archi;
```

RTL Schematic:

Simulation Waveforms:



8.6 HALF ADDER

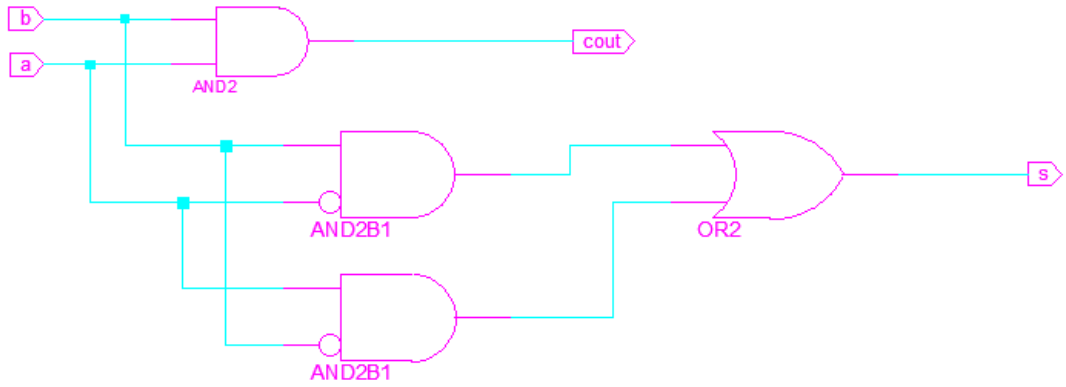
Aim:

To implement Half Adder using VHDL.

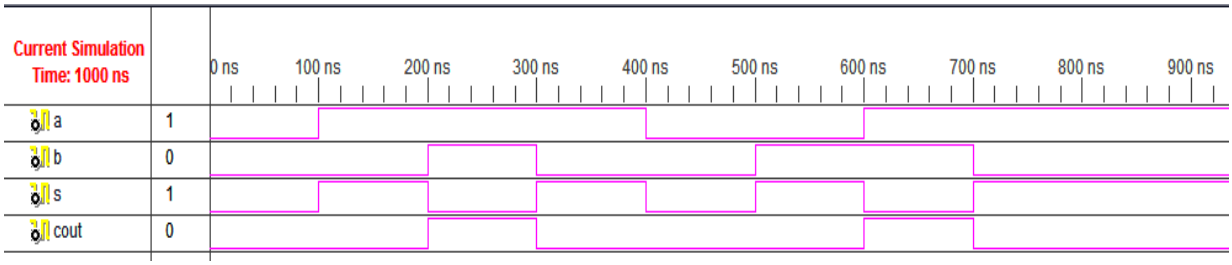
Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity halfadder is
    port(     a    : in  std_logic;
              b    : in  std_logic;
              s    : out std_logic;
              cout : out std_logic);
end halfadder;

architecture Behavioral of halfadder is

begin
  s <= a xor b;
  cout<= a and b;

end Behavioral;
```

RTL Schematic:



Simulation Waveforms:



8.7 FULL ADDER

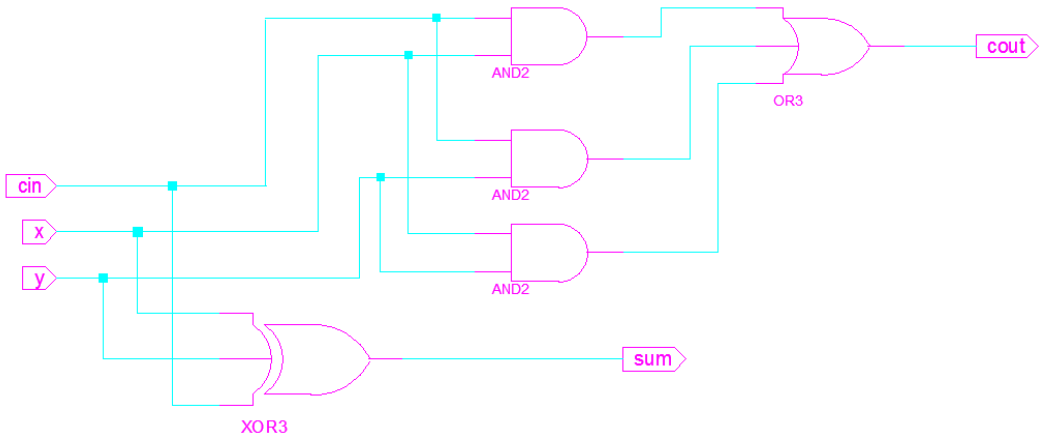
Aim:

To implement Full Adder using VHDL.

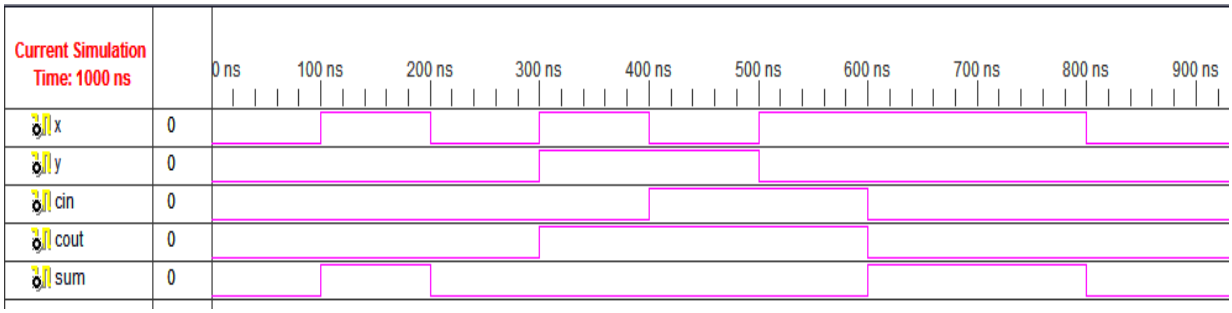
Code:

```
LIBRARY  ieee ;
USE ieee.std_logic_1164.all ;

ENTITY full_adder IS
   PORT(x:    IN   std_logic ;
        y:    IN   std_logic ;
        cin:  IN   std_logic ;
        cout: OUT  std_logic ;
        sum:  OUT  std_logic
   ) ;
END full_adder ;

ARCHITECTURE equations OF full_adder IS
BEGIN
   sum  <= x XOR y XOR cin ;
   cout <= (x AND y) OR (x AND cin) OR (y AND cin) ;
END ;
```

RTL Schematic:



Simulation waveforms:



9.MULTIPLEXERS/DEMULTIPLEXERS

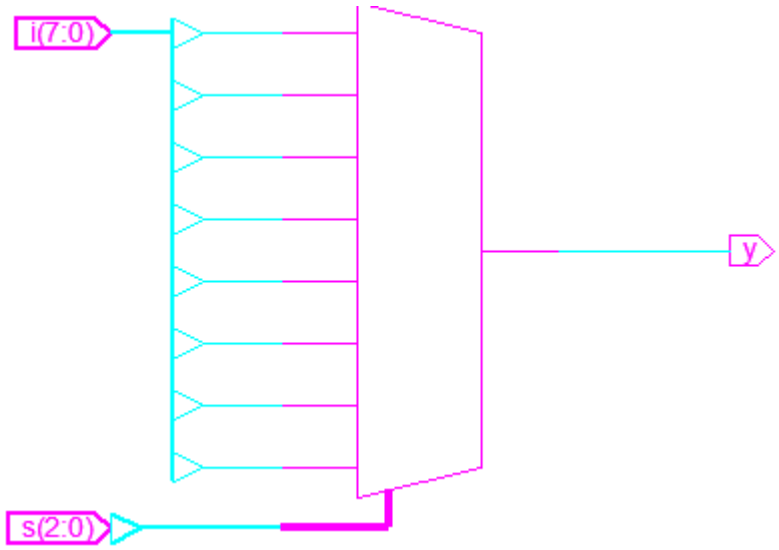
Aim:

To implement Multiplexer 8x1using VHDL.

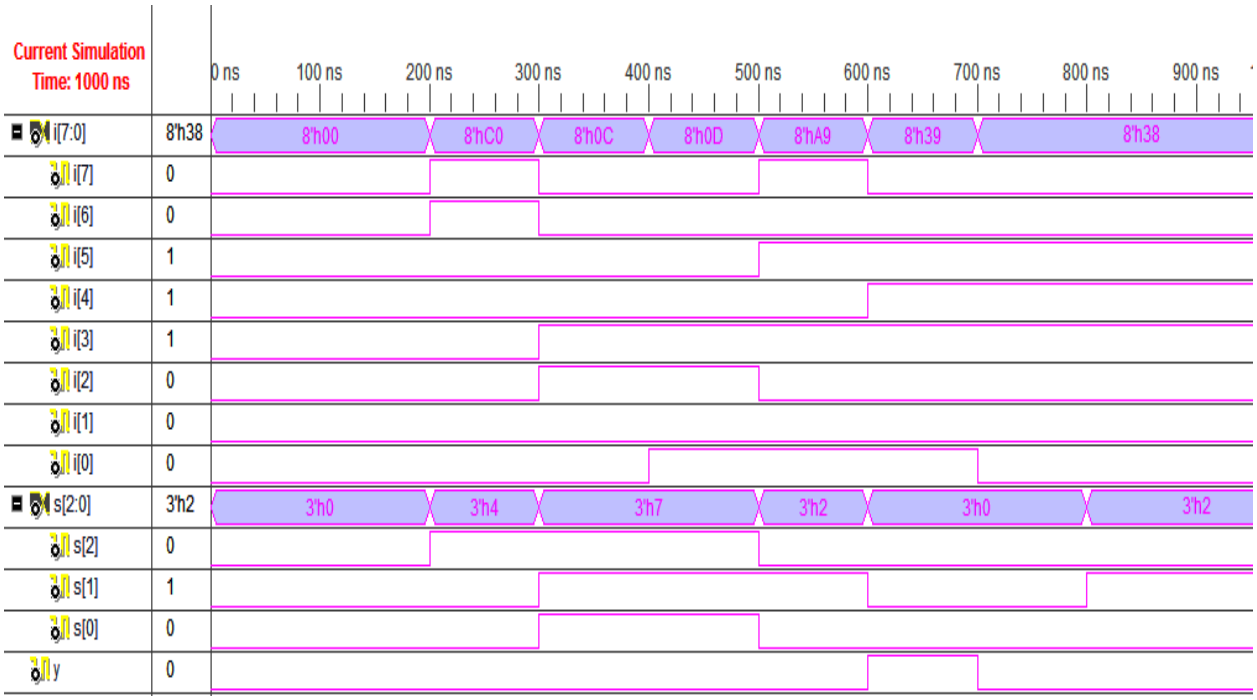
Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity mux8x1 is
              Port (
              i : in std_logic_vector(7 downto 0);
                 s : in std_logic_vector (2 downto 0);
                 y : out std_logic
                );

end mux8x1;

architecture Behavioral of mux8x1 is

begin

              process (i,s)
              begin
                   if (s = "000") then y <= i(0);
                   elsif (s = "001") then y <= i(1);
                   elsif (s = "010") then y <= i(2);
                   elsif (s = "011") then y <= i(3);
                   elsif (s = "100") then y <= i(4);
                   elsif (s = "101") then y <= i(5);
                   elsif (s = "110") then y <= i(6);
                   else y <= i(7);
                   end if;
                   end process;


end Behavioral;
```

RTL Schematic:



Simulation Waveforms:



10.DESIGN OF ALU

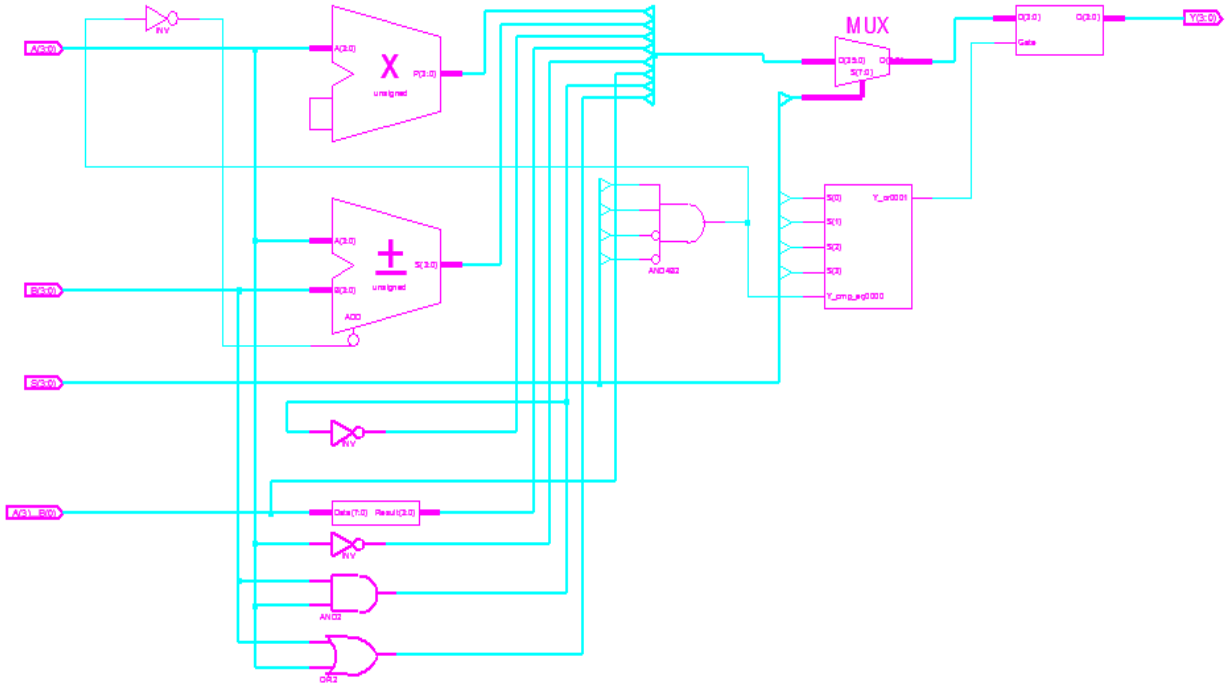
Aim:

To implement 4-Bit ALU using VHDL.

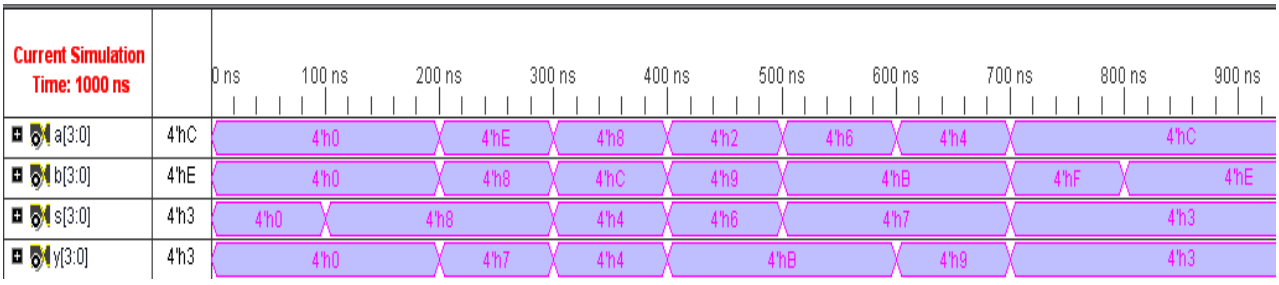
Code:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;


entity LC2_ALU is
   port( A: in std_logic_vector (3 downto 0);
      B: in std_logic_vector (3 downto 0);
      S: in std_logic_vector (3 downto 0);
      Y: out std_logic_vector (3 downto 0));
end LC2_ALU;


architecture bhv of LC2_ALU is
begin
   process(A, B, S)
   begin

   case S is
      when "0000" => Y <= A or B;
      when "0001" =>  Y<= A and B;
      when "0010" => Y <= A;
      when "0011" => Y <= not A;
      when "0100" => Y <= A xor B;
      when "0101" => Y <= A nand B;
      when "0110" => Y <= A + B;
      when "0111" => Y <= A - B;
      when "1000" => Y <= A*B;
      when "1001" => Y <= A/B;
      when others => null;
   end case;

end process;
end bhv;
```

RTL Schematic:



Simulation Waveforms:



11. 4-BIT MAGNITUDE COMPARATOR

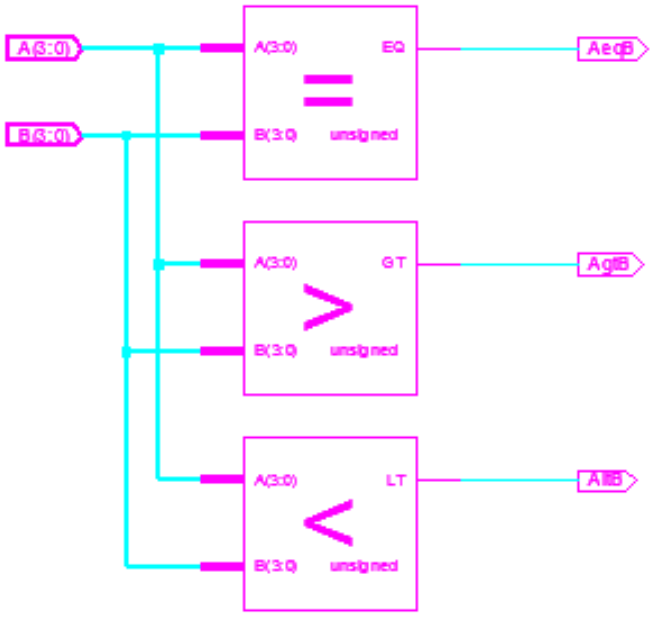
Aim:

To implement 4-Bit Magnitude Comparator using VHDL.

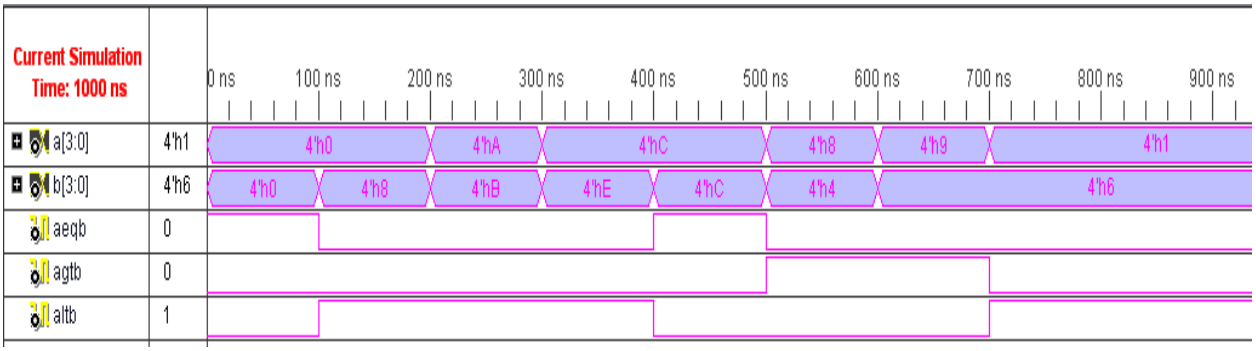
Code:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY compare IS
     PORT ( A, B : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          AeqB, AgtB, AltB : OUT STD_LOGIC ) ;
END compare ;


ARCHITECTURE Behavior OF compare IS
BEGIN
     AeqB <= '1' WHEN A = B ELSE '0' ;
     AgtB <= '1' WHEN A > B ELSE '0' ;
     AltB <= '1' WHEN A < B ELSE '0' ;
END Behavior ;
```

RTL Schematic:



Simulation Waveforms:



12. JK, D & T Flip-Flops

12.1 J-K Flip-Flop

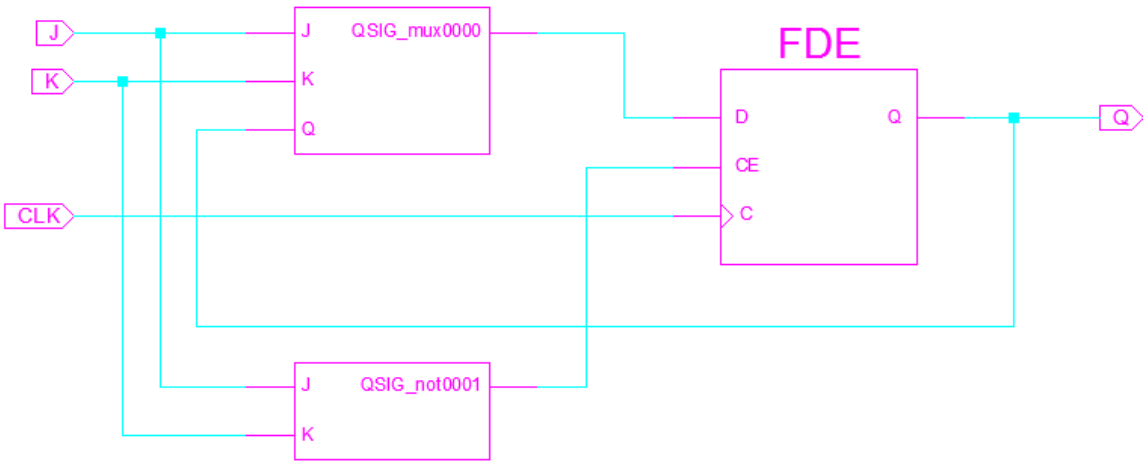
Aim:

To implement J-K Flip-Flop using VHDL.

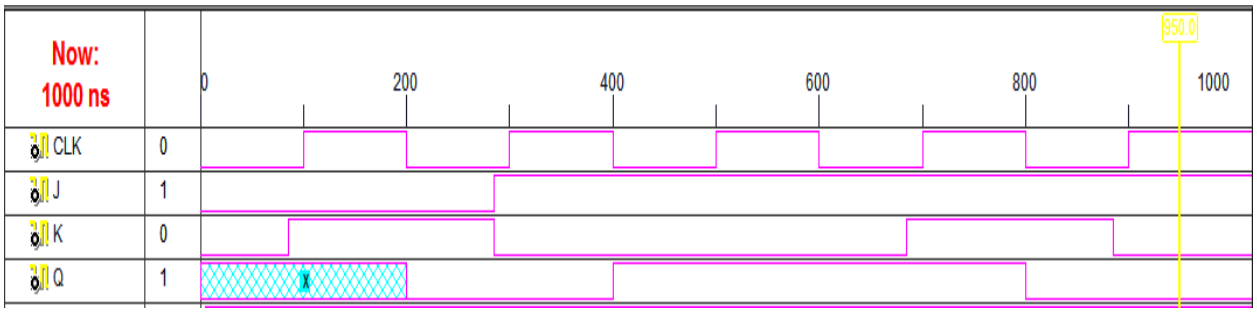
Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity jkflip is
        Port (CLK, J , K : in std_logic ;
                Q  : out std_logic ) ;

end jkflip;

architecture Behavioral of jkflip is
                signal QSIG : std_logic;
begin

    process(CLK,J,K)
    begin
        if (CLK'event and CLK = '1') then
            if (J = '0' and K = '0') then
                    QSIG <= QSIG;
                elsif (J = '1' and K = '0') then
                    QSIG <= '1';
                elsif (J = '0' and K = '1') then
                    QSIG <= '0';
                elsif (J = '1' and K = '1') then
                    QSIG <= not(QSIG);
                end if;
        end if;
            Q<=QSIG;
    end process;

end Behavioral;
```

RTL Schematic:



Behavioral Simulation:



12.2 D Flip-Flop

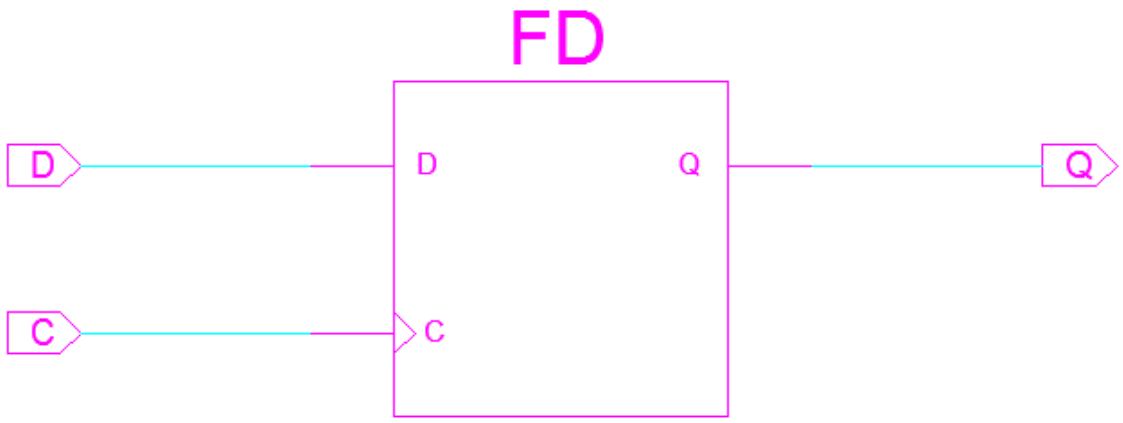
Aim:

To implement D Flip-flop Gate using VHDL.

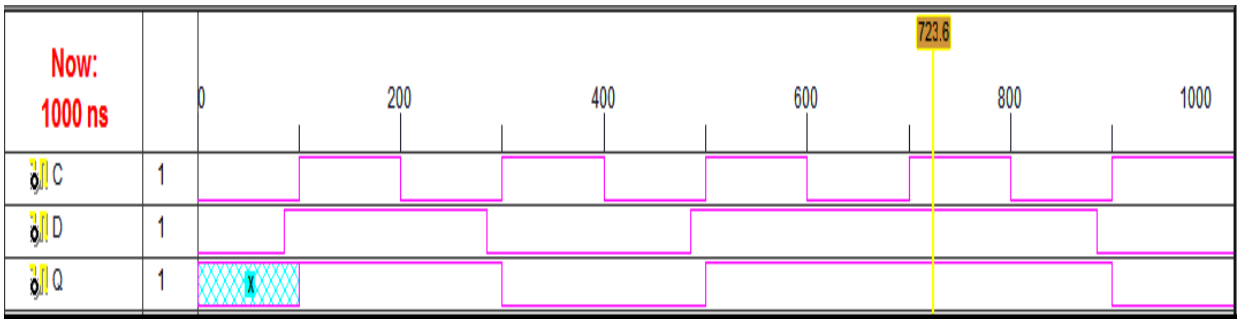
Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity dflip is

                port(C, D : in std_logic;
                                Q : out std_logic);
end dflip;

architecture Behavioral of dflip is

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

RTL Synthesis:



Behavioral Simulation:



12.3 T Flip-Flop

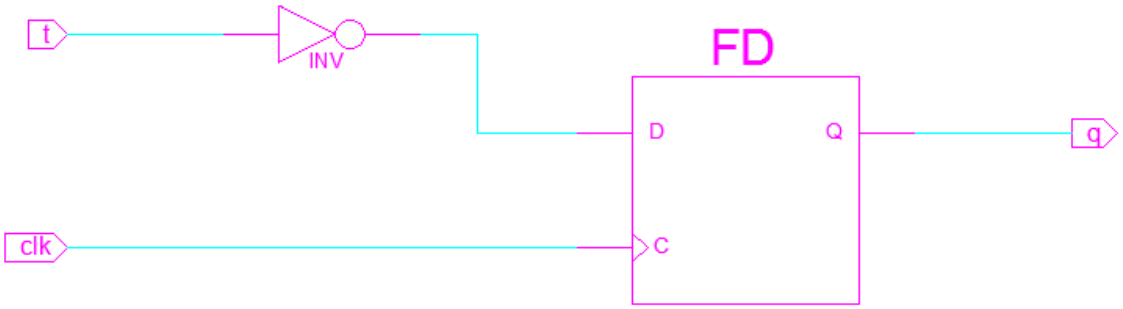
Aim:

To implement T Flip-Flop using VHDL.

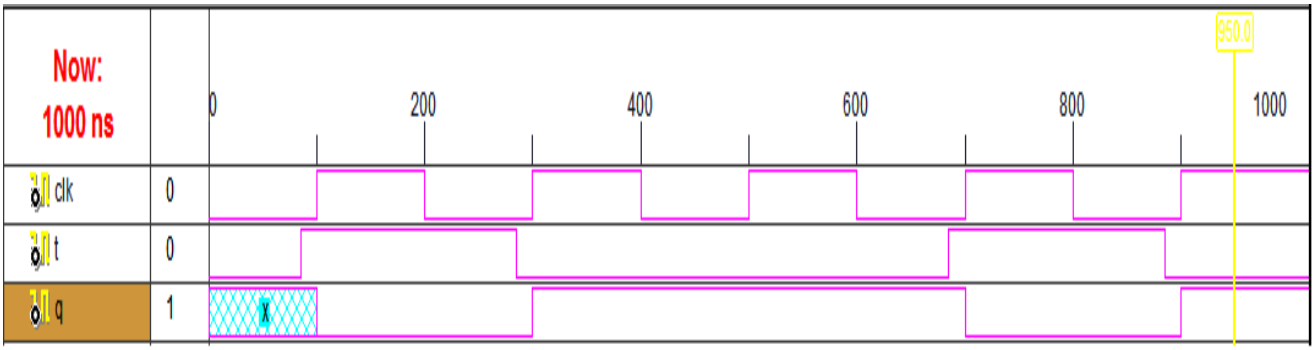
Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tff is
   port (
      t  :in  std_logic;
      clk   :in  std_logic;
      q     :out std_logic

   );
end tff;

architecture rtl of tff is
begin
   process (clk)
         begin
               if (clk'event and clk='1') then
           q <= not t;
         end if;
   end process;

end architecture;
```

RTL Synthesis:



Behavioral simulation:



13. Synchronous counter

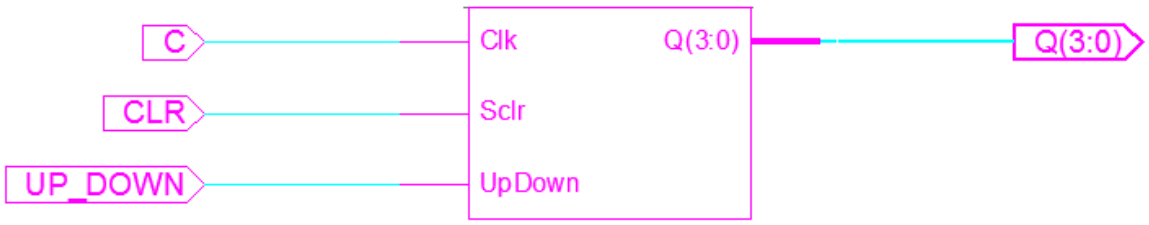
Aim:

To implement Synchronous counter using VHDL.

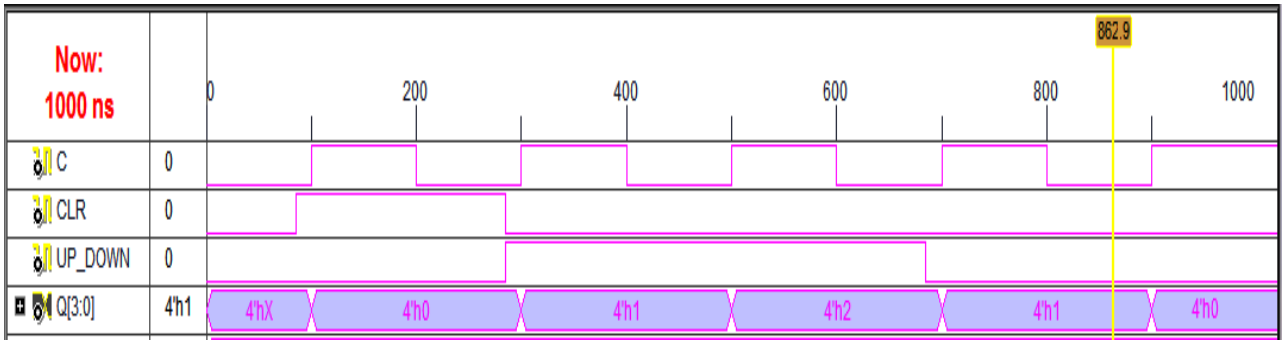
Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sync_counter is
        port(C, CLR, UP_DOWN : in std_logic;
                                        Q : out std_logic_vector(3 downto 0));
end sync_counter;

architecture archi of sync_counter is
        signal tmp: std_logic_vector(3 downto 0);
begin
        process (C, CLR)
        begin
                if (C'event and C='1') then
                    if (CLR='1') then
                                        tmp <= "0000";
                         else
                                        if (UP_DOWN='1') then
                                                tmp <= tmp + 1;
                                        else
                                                tmp <= tmp - 1;
                                        end if;
                           end if;
                  end if;
        end process;
Q <= tmp;
end archi;
```

RTL Schematic:



Behavioral Simulation:



14. Asynchronous counter

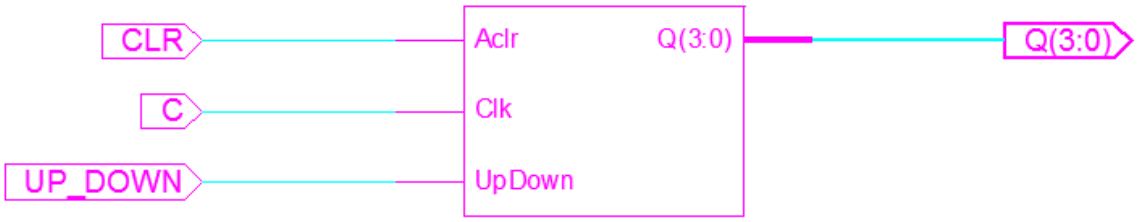
Aim:

To implement Asynchronous counter using VHDL.

Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity async_counter is
	port(C, CLR, UP_DOWN : in std_logic;
					Q : out std_logic_vector(3 downto 0));
end async_counter;

architecture archi of async_counter is
	signal tmp: std_logic_vector(3 downto 0);
begin
	process (C, CLR)
	begin
		if (CLR='1') then
			tmp <= "0000";
		elsif (C'event and C='1') then
			if (UP_DOWN='1') then
				tmp <= tmp + 1;
			else
				tmp <= tmp - 1;
			end if;
		end if;
	end process;
Q <= tmp;
end archi;
```

RTL Schematic:



Behavorial Simulation: