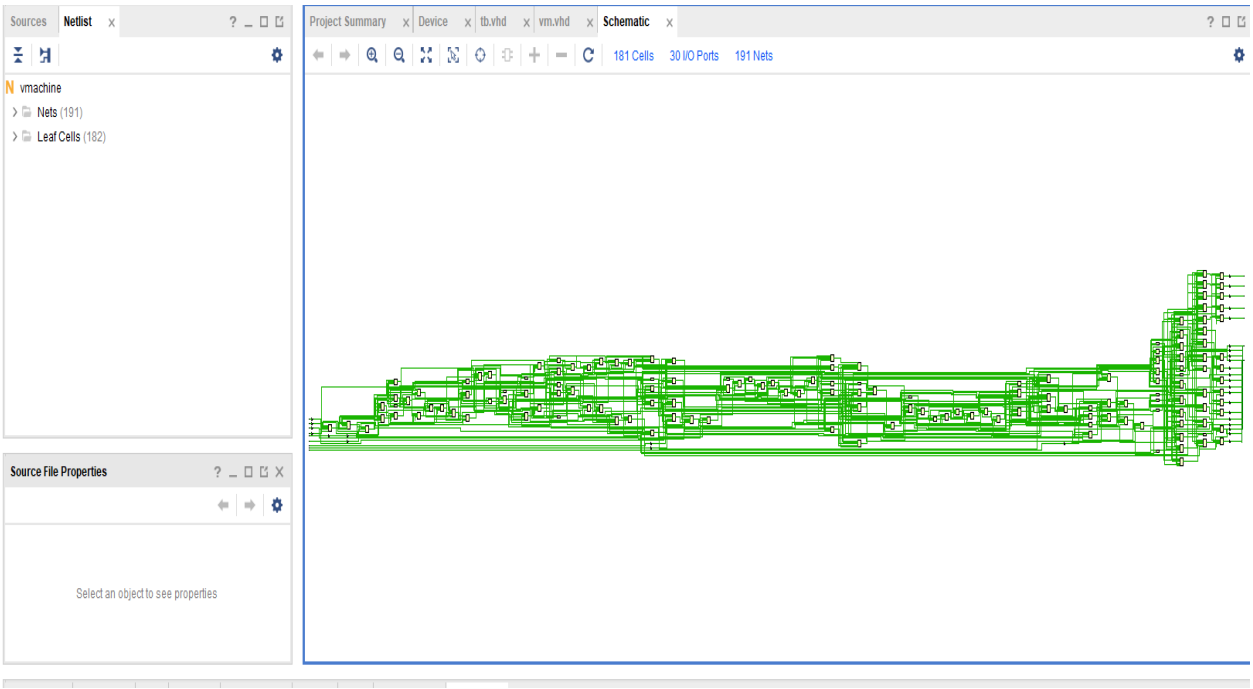
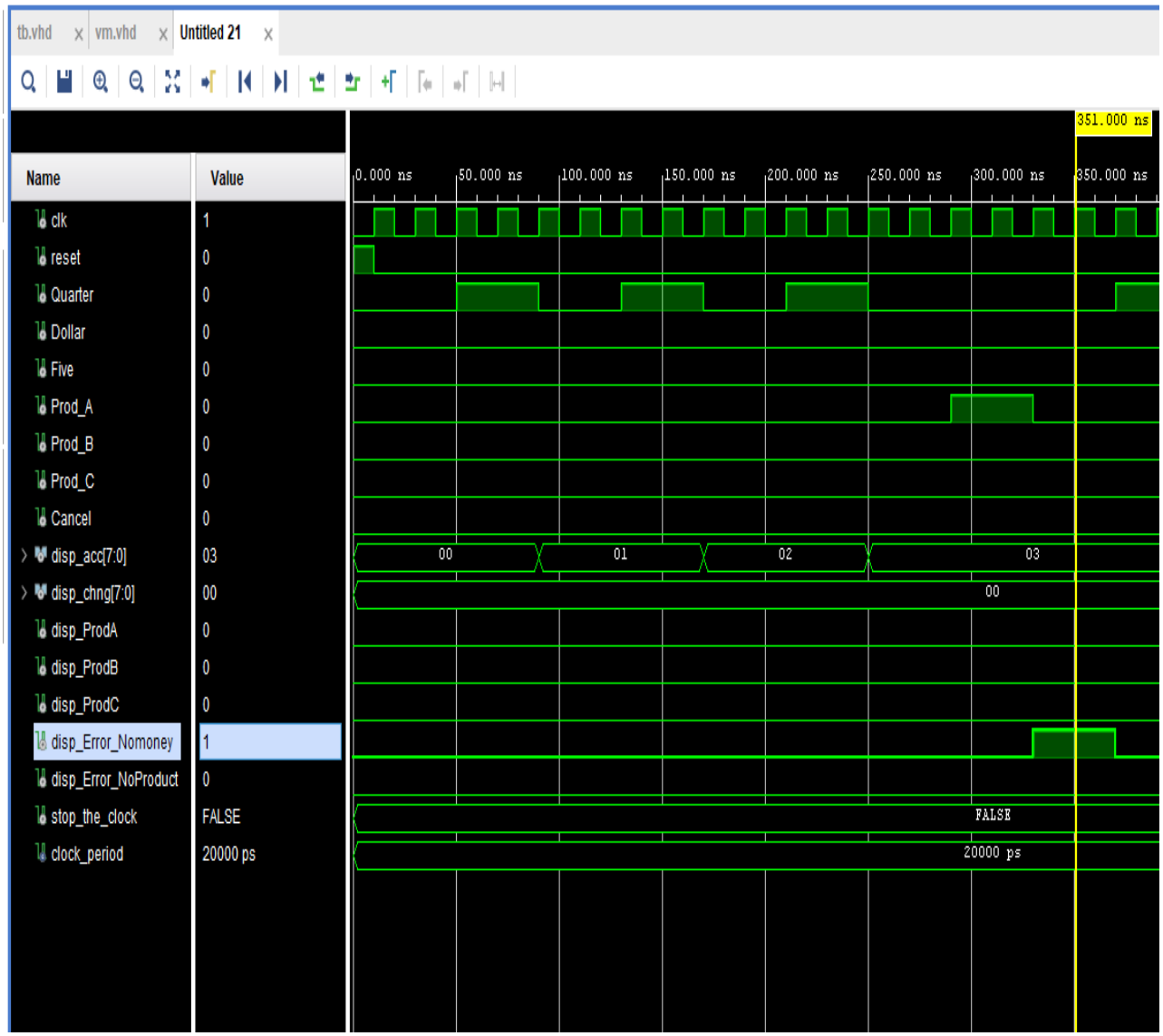
Final report on Vending Machine:
Sakar Poudel
W978017

In this class I have designed and simulated a vending machine which accepts quarters, dollars and five-dollar bills and dispenses three products of different prices. The price of product A is 1.75, the price of product B is 2.50 and the price of product C is 3.25. The design takes account of total money inserted, the product selected and dispenses the product and change accordingly. The design is tested using testbench file. All the results are posted below:

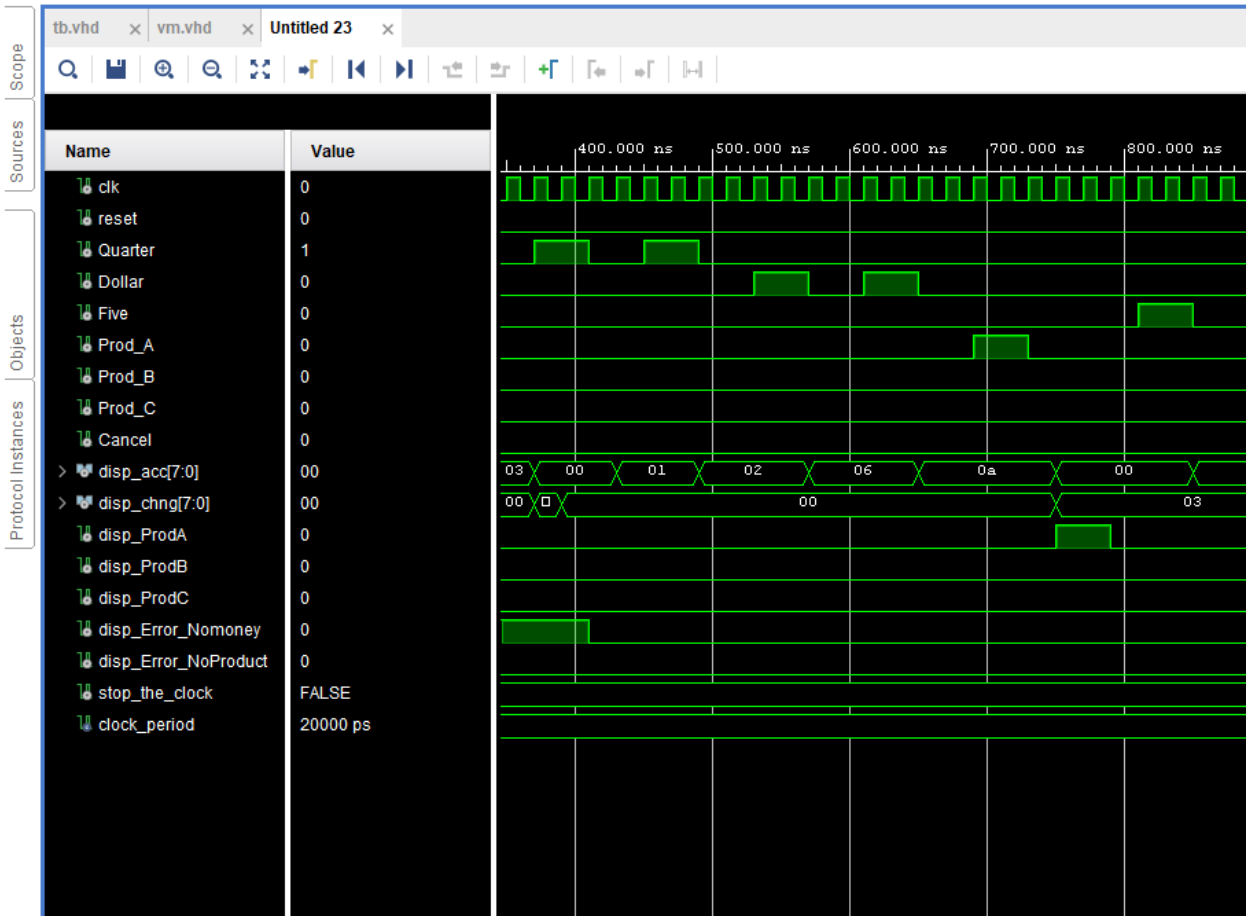
The schematic of the implemented design is:



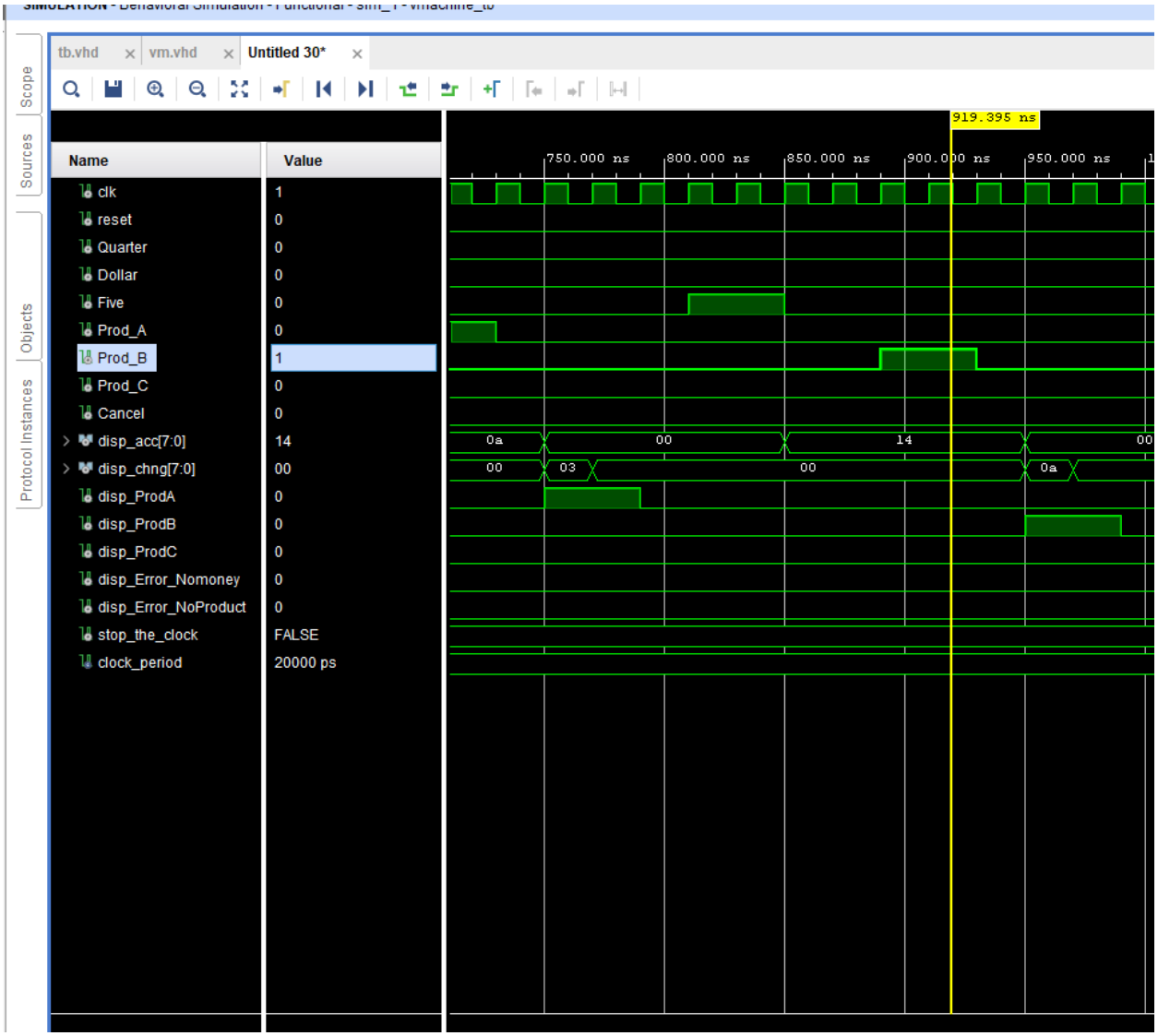
1) **Insert 3 quarters and request prod_A => error message (Insufficient fund)**



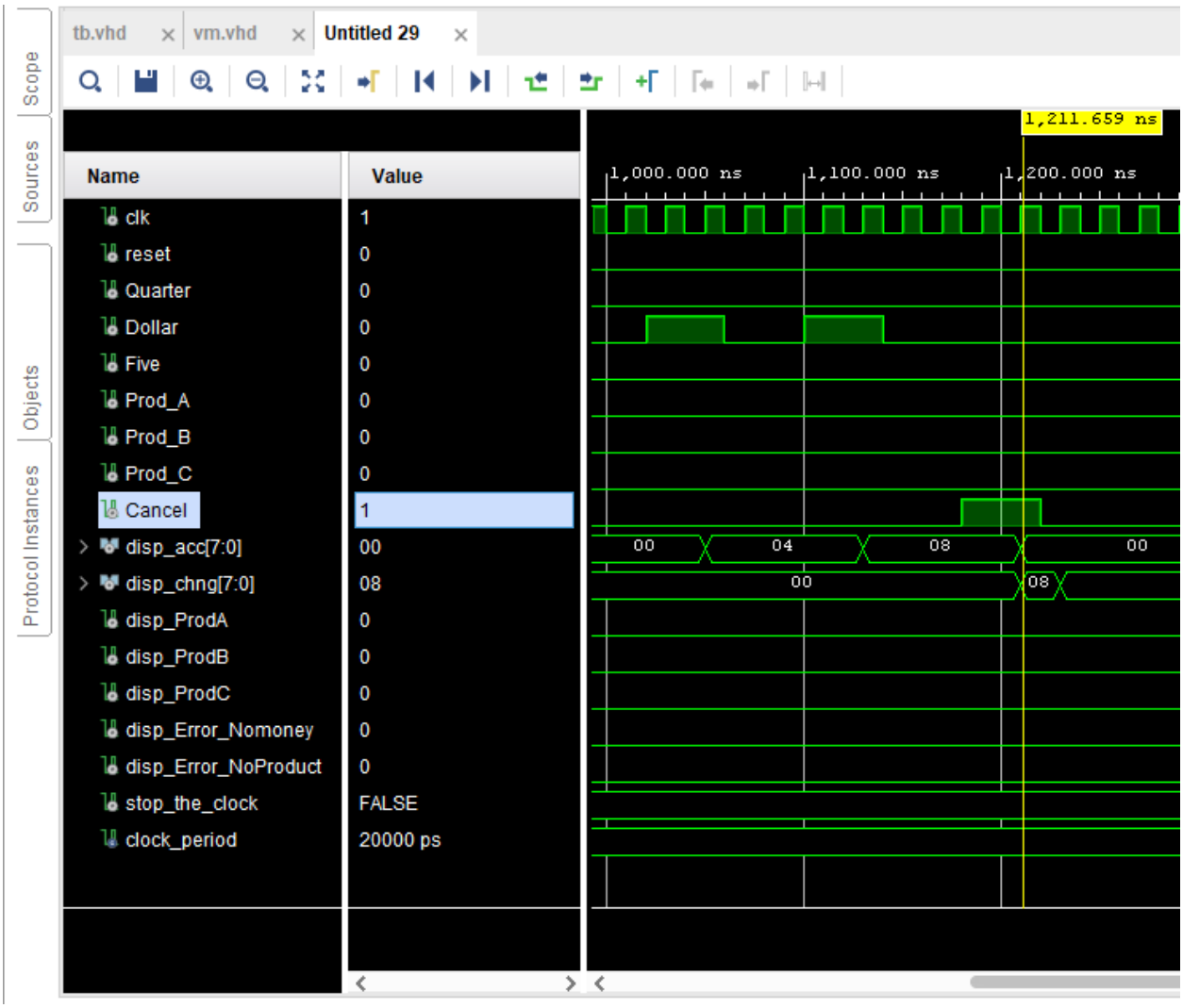
2) **insert 2 quarters and 2 Dollars and request prod_A => Disp prod_A and Change = $0.75 (Prod_A is reduced by one)**



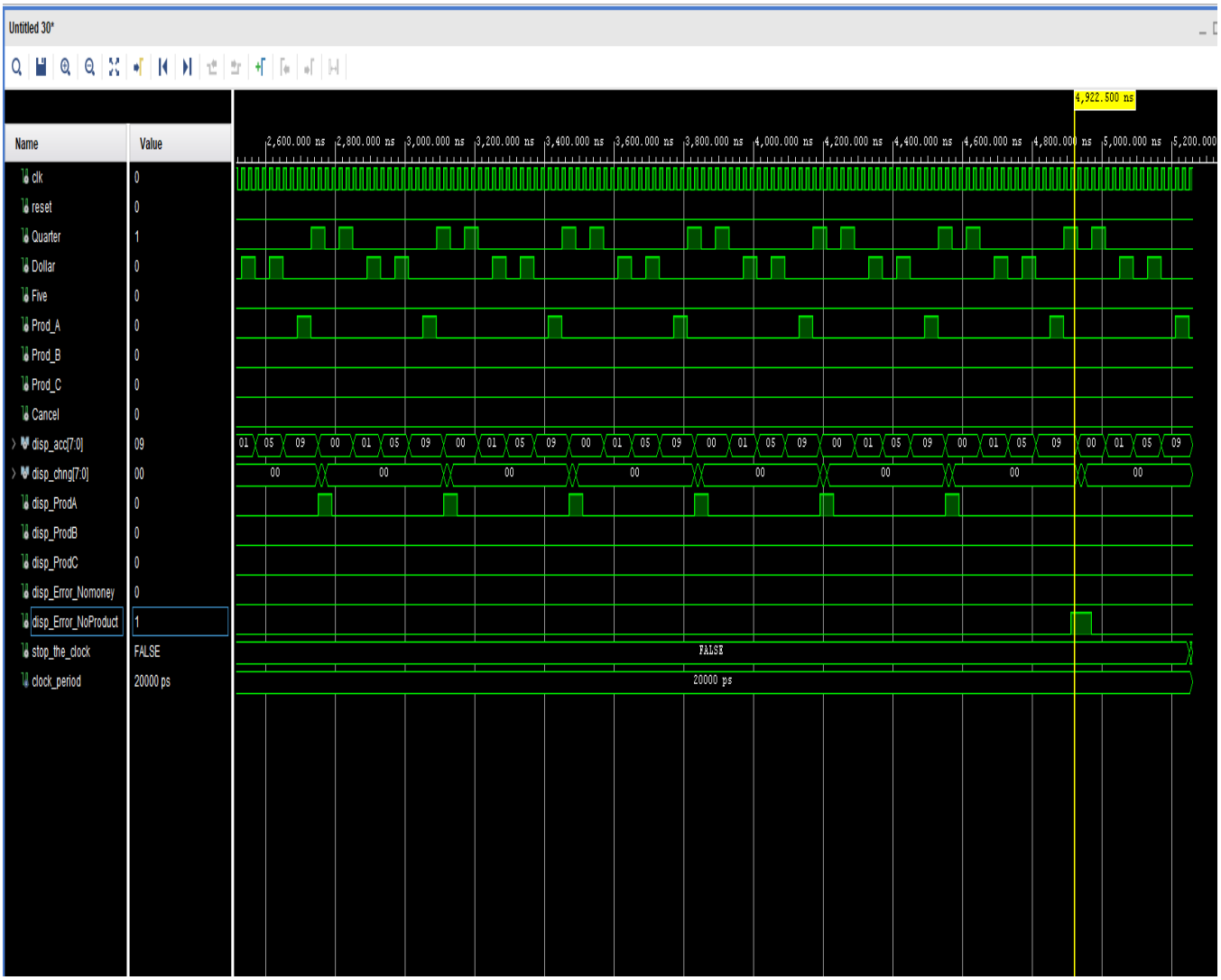
3)insert 5 Dollars and request prod_B => Disp prod_B and Change = $1.50 (Prod_B is reduced by one)



4) insert 2 Dollars and cancel the transaction => Change = $2.0



5) **repeat 2) ten times and the in the last time => error message (Prod_A_not_avail)**



All these simulations gave the correct result. The source code and the testbench code is posted below

VHDL Source Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
entity vmachine is
Port (
    clk : in std_logic;
    reset : in std_logic;
    Quarter,Dollar,Five: in std_logic; --to track if someone inserted quarter, dolllar or five dollar bill
    Prod_A,Prod_B,Prod_C  : in std_logic ;
    Cancel:in std_logic;
    disp_acc:out std_logic_vector (7 downto 0);
    disp_chng:out std_logic_vector (7 downto 0);

disp_ProdA,disp_ProdB,disp_ProdC,disp_Error_Nomoney,disp_Error_NoProduct: out std_logic
 );
end vmachine;


architecture Behavioral of vmachine is
type state_type is (
    idle_state,
    Q_insert,
    D_insert,
    F_insert,
```

```
    Q_wait,
    cancel_state,
    cancel_wait,
    reset_state,
    error_insertmore,
    error_notavailable,
    check_A,
    check_B,
    check_C,
    drop_A,
    drop_B,
    drop_C,
    wait_drop
     );
signal state_reg: state_type;
signal count_reg,change_reg: unsigned(7 downto 0):="00000000";
signal ProdA_reg,ProdB_reg,ProdC_reg :unsigned (3 downto 0):="0000";
signal disp_ProdAsig,disp_ProdBsig,disp_ProdCsig: std_logic:='0';
signal error_Nomoneysig,error_NoProductsig: std_logic:='0';
begin
Process(clk,reset,state_reg)
begin
if (reset='1') then
    state_reg<=reset_state;
elsif(clk'event and clk='1')then
```

```
case state_reg is
when reset_state=>
   count_reg<="00000000";
   change_reg<="00000000";
   ProdA_reg<="1010";--10 items of A, B and C at first
   ProdB_reg<="1010";
   ProdC_reg<="1010";
   state_reg<=idle_state;
when idle_state=>
   disp_ProdAsig<= '0'; disp_ProdBsig <= '0'; disp_ProdCsig <= '0';error_Nomoneysig<='0'; error_NoProductsig<='0';
   if (Quarter='1') then
      state_reg<=Q_insert;---Quarterer inserted
   elsif (Dollar='1') then
      state_reg<=D_insert;--dollar inserted
   elsif (Five='1') then
      state_reg<=F_insert;--five dollar bill inserted
   elsif (Cancel='1') then
      state_reg<=cancel_state;
   elsif (Prod_A='1') then --if product a is selected then check if count_reg has enough money for transaction
      if (to_integer(count_reg) > 6 ) then -- A[1.75] == 7 Quarter counts == 00000111
         state_reg<=check_A;
      else          -- if money is not enough ask for more
         state_reg<=error_insertmore;
```

```
        end if;
    elsif (Prod_B='1') then    --if product b is selected then check if count_reg has enough money for transaction
        if (to_integer(count_reg) > 8) then      -- B[2.50] == 10 Quarter counts == 00001010
            state_reg<=check_B;
        else          --if money is not enough ask for more
            state_reg<=error_insertmore;
        end if;
    elsif (Prod_C='1') then     --if product c is selected then check if count_reg has enough money for transaction
        if (to_integer(count_reg) > 12) then      -- C[3.25] == 13 Quarter counts == 00001101
            state_reg<=check_C;
        else          -- if money is not enough ask for more
            state_reg<=error_insertmore;
        end if;
    end if;
when Q_insert=>
    count_reg<=to_unsigned((to_integer(count_reg)+1),8); --increase the count register by 1 if Quarterer is inserted; we are counting in Quarterers
    state_reg<=Q_wait;

when D_insert=>
    count_reg<=to_unsigned((to_integer(count_reg)+4),8);--increase the count register by 4; 1 dollar=4*25c
    state_reg<=Q_wait;
```

```
when F_insert=>
   count_reg<=to_unsigned((to_integer(count_reg)+20),8);--increase the count register by 20;$5=20*25C
   state_reg<=Q_wait;
when cancel_state=>
   change_reg<=count_reg;
   count_reg<="00000000";
   state_reg<=cancel_wait;
when cancel_wait=>
   change_reg<="00000000";
   count_reg<="00000000";
   state_reg<=idle_state;
when Q_wait=>
   if (Quarter='0') then
      state_reg<=idle_state;
   elsif (Dollar='0') then
      state_reg<=idle_state;
   elsif (Five='0') then
      state_reg<=idle_state;
   end if;
when check_A=>
   if(to_integer(ProdA_reg)>0) then
      state_reg<=drop_A;
   else
      state_reg<=error_notavailable;
   end if;
```

```
when check_B=>
    if(to_integer(ProdB_reg)>0) then
        state_reg<=drop_B;
    else
        state_reg<=error_notavailable;
    end if;
when check_C=>
    if(to_integer(ProdC_reg)>0) then
        state_reg<=drop_C;
    else
        state_reg<=error_notavailable;
    end if;
when error_insertmore=>
     error_Nomoneysig<='1';
    if (Prod_A='0') and (Prod_B='0')and (Prod_C='0') then
        state_reg<=cancel_state;
    end if;
when error_notavailable=>
    error_NoProductsig<='1';
    if (Prod_A='0') and (Prod_B='0')and (Prod_C='0')then
        state_reg<=cancel_state;
    end if;
when drop_A=>
    disp_ProdAsig<= '1';
    change_reg<=to_unsigned((to_integer(count_reg)-7),8);
```

```
    count_reg<="00000000";
    ProdA_reg<=to_unsigned((to_integer(ProdA_reg)-1),4);
    state_reg<=wait_drop;
when drop_B=>
    disp_ProdBsig  <= '1';
    change_reg<=to_unsigned((to_integer(count_reg)-10),8);
    count_reg<="00000000";
    ProdB_reg<=to_unsigned((to_integer(ProdB_reg)-1),4);
    state_reg<=wait_drop;
when drop_C=>
    disp_ProdCsig <= '1';
    change_reg<=to_unsigned((to_integer(count_reg)-13),8);
    count_reg<="00000000";
    ProdC_reg<=to_unsigned((to_integer(ProdC_reg)-1),4);
    state_reg<=wait_drop;
when wait_drop=>
    if(Prod_A='0')and (Prod_B='0') and (Prod_C='0') then
        change_reg<="00000000";
        state_reg<=idle_state;
    end if;
when others=>
    state_reg<=idle_state;
end case;
end if;
end Process;
```

```
disp_acc<=std_logic_vector(count_reg);
disp_chng<=std_logic_vector(change_reg);
disp_Error_Nomoney<=std_logic(error_Nomoneysig);
disp_Error_NoProduct<=std_logic(error_NoProductsig);
disp_ProdA<=std_logic(disp_ProdAsig);
disp_ProdB<=std_logic( disp_ProdBsig);
disp_ProdC<=std_logic(disp_ProdCsig);
end Behavioral;
```

VHDL TESTBENCH CODE:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity vmachine_tb is
end;

architecture bench of vmachine_tb is

  component vmachine
  Port (
     clk : in std_logic;
     reset : in std_logic;
     Quarter,Dollar,Five: in std_logic;
     Prod_A,Prod_B,Prod_C  : in std_logic ;
     Cancel:in std_logic;
     disp_acc:out std_logic_vector (7 downto 0);
     disp_chng:out std_logic_vector (7 downto 0);

disp_ProdA,disp_ProdB,disp_ProdC,disp_Error_Nomoney,disp_Error_NoProduct:
out std_logic
   );
  end component;

  signal clk: std_logic;
```

```
  signal reset: std_logic;
  signal Quarter,Dollar,Five: std_logic;
  signal Prod_A,Prod_B,Prod_C: std_logic;
  signal Cancel: std_logic;
  signal disp_acc: std_logic_vector (7 downto 0);
  signal disp_chng: std_logic_vector (7 downto 0);
  signal
disp_ProdA,disp_ProdB,disp_ProdC,disp_Error_Nomoney,disp_Error_NoProduct:
std_logic ;

  constant clock_period: time := 20 ns;
  signal stop_the_clock: boolean;

begin

  uut: vmachine port map ( clk              => clk,
                           reset            => reset,
                           Quarter          => Quarter,
                           Dollar           => Dollar,
                           Five             => Five,
                           Prod_A           => Prod_A,
                           Prod_B           => Prod_B,
                           Prod_C           => Prod_C,
                           Cancel           => Cancel,
                           disp_acc         => disp_acc,
                           disp_chng        => disp_chng,
```

```
                disp_ProdA          => disp_ProdA,
                disp_ProdB          => disp_ProdB,
                disp_ProdC          => disp_ProdC,
                disp_Error_Nomoney   => disp_Error_Nomoney,
                disp_Error_NoProduct => disp_Error_NoProduct );

  stimulus: process
  begin

    -- Put initialisation code here
Quarter<='0';
Dollar<='0';
Five<='0';
Prod_A<='0';
Prod_B<='0';
Prod_C<='0';
Cancel<='0';
reset<='0';

reset<='1';
wait for 10ns;
reset<='0';
wait for 40ns;

    -- Put test bench stimulus code here
```

```
Quarter<='1';
wait for 40ns;
Quarter<='0';
wait for 40ns;
Quarter<='1';
wait for 40ns;
Quarter<='0';
wait for 40ns;
Quarter<='1';
wait for 40ns;
Quarter<='0';
wait for 40ns;
Prod_A<='1';
wait for 40ns;
Prod_A<='0';
--2) insert 2 quarters and 2 Dollars and request prod_A => Disp prod_A and Change = $0.75 (Prod_A is reduced by one)
wait for 40ns;
Quarter<='1';
wait for 40ns;
Quarter<='0';
wait for 40ns;
Quarter<='1';
wait for 40ns;
Quarter<='0';
wait for 40ns;
```

```
Dollar<='1';
wait for 40ns;
Dollar<='0';
wait for 40ns;
Dollar<='1';
wait for 40ns;
Dollar<='0';
wait for 40ns;
Prod_A<='1';
wait for 40ns;
Prod_A<='0';
wait for 40ns;

--3) insert 5 Dollars and request prod_B => Disp prod_B and Change = $1.50 (Prod_B is reduced by one)
wait for 40ns;
Five<='1';
wait for 40ns;
Five<='0';
wait for 40ns;
Prod_B<='1';
wait for 40ns;
Prod_B<='0';
wait for 40ns;
--4) insert 2 Dollars and cancel the transaction => Change = $2.0
wait for 40ns;
```

```
Dollar<='1';
wait for 40ns;
Dollar<='0';
wait for 40ns;
Dollar<='1';
wait for 40ns;
Dollar<='0';
wait for 40ns;
Cancel<='1';
wait for 40ns;
Cancel<='0';
wait for 40ns;
--5) repeat 2) ten times and the in the last time => error message (Prod_A_not_avail)
wait for 40ns;
for I in 0 to 10 loop
Quarter<='1';
wait for 40ns;
Quarter<='0';
wait for 40ns;
Quarter<='1';
wait for 40ns;
Quarter<='0';
wait for 40ns;
Dollar<='1';
wait for 40ns;
```

```
Dollar<='0';
wait for 40ns;
Dollar<='1';
wait for 40ns;
Dollar<='0';
wait for 40ns;
Prod_A<='1';
wait for 40ns;
Prod_A<='0';
end loop;
stop_the_clock <= true;
   wait;
 end process;

 clocking: process
 begin
   while not stop_the_clock loop
     clk <= '0', '1' after clock_period / 2;
     wait for clock_period;
   end loop;
   wait;
 end process;

end;
```