EEE 304 : DIGITAL ELECTRONICS LABORATORY

Project Report

Name of Project : Smart Booth for Payment

Submitted By -

M Zafir Sadik Khan

Student ID : 1606135

Nur Nabila Sadia

Student ID: 1606138

Toufiqur Rahman Shuvo

Student ID : 1606159

Video Link :
https://drive.google.com/file/d/1OSeZ_mHHvYu1nYnRGfFQvclaFHNnSI5l/view?usp=sharing

Objective :

The objective of this project is to calculate the cost and the return money of a customer. There are three products (A, B & C) with predefined price. The customer will give input for the quantity of each product and the money he will give. The system will calculate the total cost of products and it will also calculate the money to be returned to the customer.

Design Procedure :

1. First we made a decimal to binary converter. For this we used simulino (Arduino uno), a number pad and an LCD display in Proteus.
2. The inputs were stored in a register. There are four inputs (Amount of money given, quantity of A, quantity of B and quantity of C). These four inputs were stored in four different registers.
3. In this step we multiplied the quantity of each product with their respective prices. The prices were set as A = 5, B = 10, C = 15. Three multipliers were used.
4. In this step the total prices of A, B and C were added. For this we used two adders. The first adder added price of A and price of B. The second adder added the output from first adder and the price of C. The output from the second adder is the total cost of the customer.
5. In this step we used a subtractor to subtract the total cost from the amount of money given by the customer.
6. In the last step a binary to BCD converter is designed. Using BCD we showed our outputs in digital display in decimal.

Verilog Code :

```
module DLD_PROJECT (a,b,c,Tk,C,R);
        parameter n = 16;
        input [n-1:0]a,b,c,Tk;          //a,b,c are the number of products A,B,C respectively;
                                        // Tk = Input money
        output reg [n-1:0]C,R;          //C = Total cost, R = Return money
        parameter pa = 8'b0101;                 //Price of A = 5 tk
        parameter pb = 8'b1010;                 //Price of B = 10 tk
        parameter pc = 8'b1111;                 //Price of C = 15 tk
        wire [n-1:0] PA,PB,PC;
        reg [n-1:0]r;


        multiply step1(PA,pa,a);
        multiply step2(PB,pb,b);
        multiply step3(PC,pc,c);

        always@(PA,PB,PC)
                begin
                C = PA + PB + PC;
                r = ~C + 1;
                R = Tk + r;
                end

endmodule



module multiply(product,multiplier,multiplicand);
  input  [15:0]multiplier, multiplicand;
  output  [15:0]product;
  reg  [15:0]product;
  integer  i;

  always @( multiplier or multiplicand )
    begin
       product = 0;
       for(i=0; i<8; i=i+1)
           if( multiplier[i] == 1'b1 ) product = product + ( multiplicand << i );
    end
endmodule
```

Explanation of Code :

In this code there are four inputs and two outputs.

a = Quantity of product A; pa = Price of product A

b = Quantity of product B; pb = Price of product B

c = Quantity of product C; pc = Price of product C

Tk = The money given by the customer

C = Total cost of the customer

R = Return money of the customer

There are three wires PA, PB, PC

PA = a*pa

PB = b*pb

PC = c*pc

We calculated the values of PA, PB and PC by using a module named multiply.

This module multiplies two numbers. Here, both the inputs of the module multiply are of 8 bits. So the output of this module will be of 16 bits. This module multiplies by shifting and then adding.

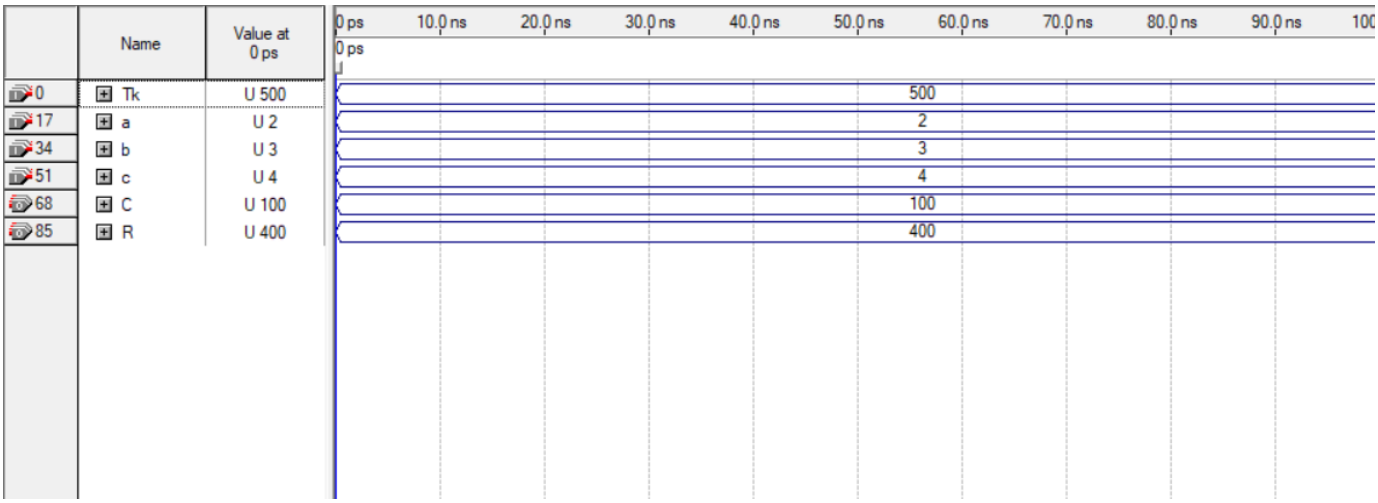
In the always block the total cost (C) is calculated. Then the return money (R) is calculated by subtracting C from the input money.

Timing Diagram :

For example we take a = 2, b = 3, c = 4 and Tk = 500.

So the total cost should be, C = 2*5 + 3*10 + 4*15 = 100.

And the return money should be, R = 500 – 100 = 400.



From the timing diagram,

C = 100

R = 400

So our code works properly.

Proteus Circuit Diagram :

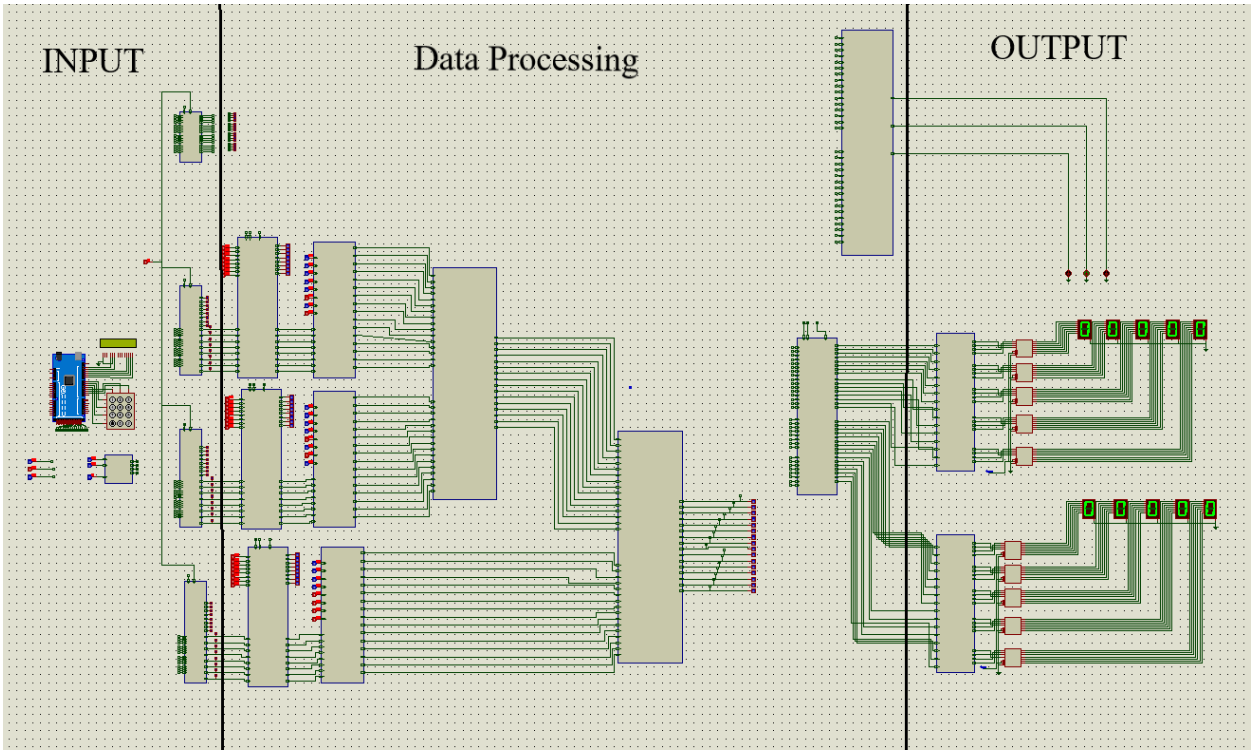


Figure - 1

There are mainly three portions in this circuit :

i. Input
ii. Data Processing and calculation
iii. Output

In the input part an Arduino, number pad and lcd display is used to give input the numbers (Input money, quantity of A,B,C) in decimal. Then they are stored in registers.

In the data processing part, multiplicator is used to multiply the product quantity with their prices. Here we fixed prices as A = 5 tk, B = 10 tk, C = 15 tk per unit. Then the values are added with adders. The total cost is then subtracted form the input money. The output from the subtractor is the remaining balance or return money.

In the output part, binary to BCD converter is designed. From BCD output we used proteus built in program to show those BCD values in decimal in a 7-segment display.

PART 1: DECIMAL TO BINARY CONVERTER

For the purpose of our SMART BOOTH project, we used Arduino MEGA for the conversion. As we have to send various values (let number of product, Balance) as input, it is more helpful for converting decimal to binary, as well as for displaying this data.(link for the code is below the figure)

***Instruction was not to use Arduino but it was impossible for us to take decimal inputs using logic gates. In that case we had to give binary inputs manually.

Circuit diagram:

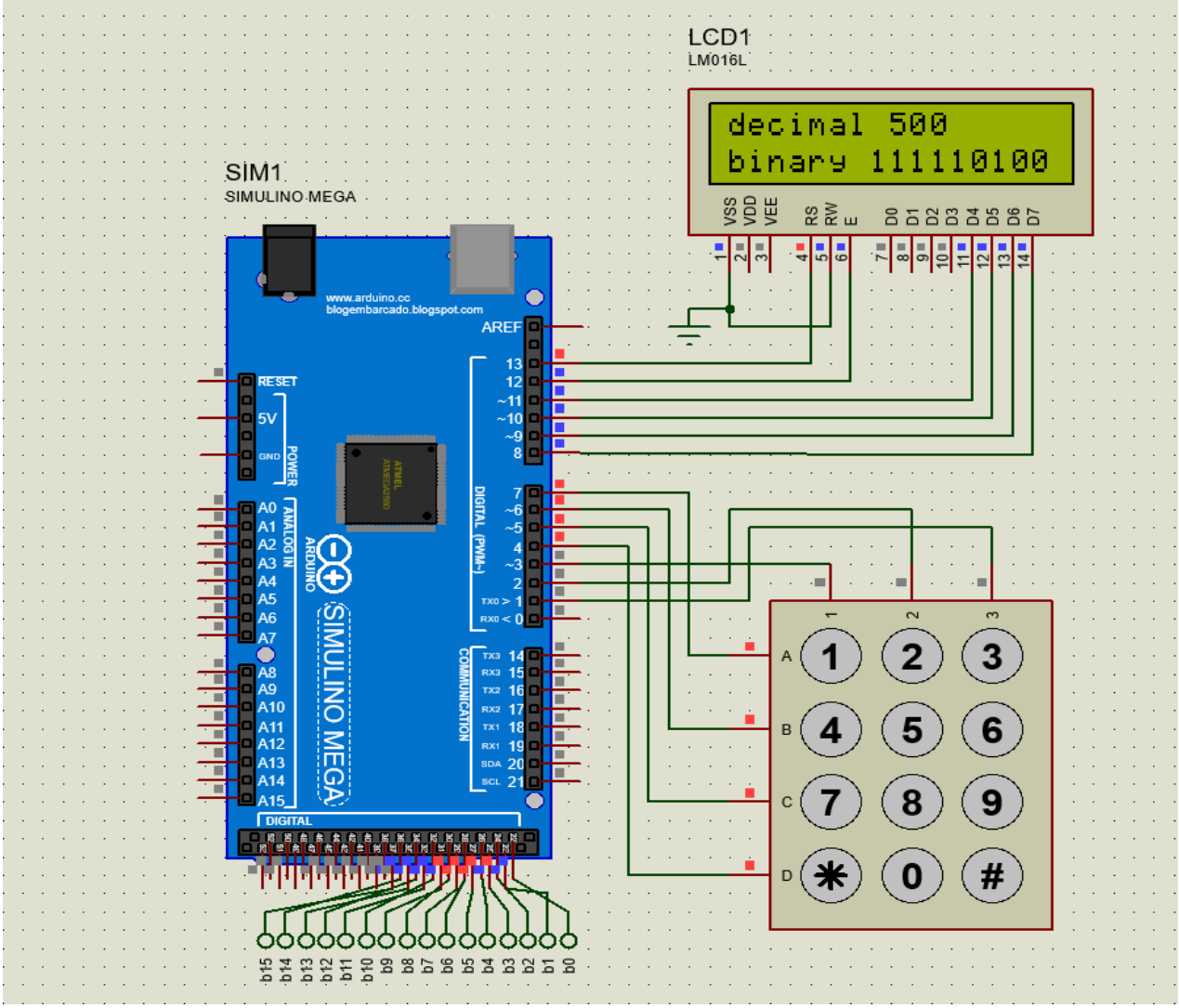


Figure - 2

https://drive.google.com/file/d/1SzZCk5t1JqDNpUqXwxke9Xek7IHNNO6z/view?usp=sharing

PART 2 : DATA STORE

INPUT:

1) Balance in customer's account (max. 16 bit)

2) How much Product-A is demanded (max 8 bit)

3) How much Product-B is demanded (max 8 bit)

4) How much Product-C is demanded (max 8 bit)

For storing this data we need to use register (Figure - 3)

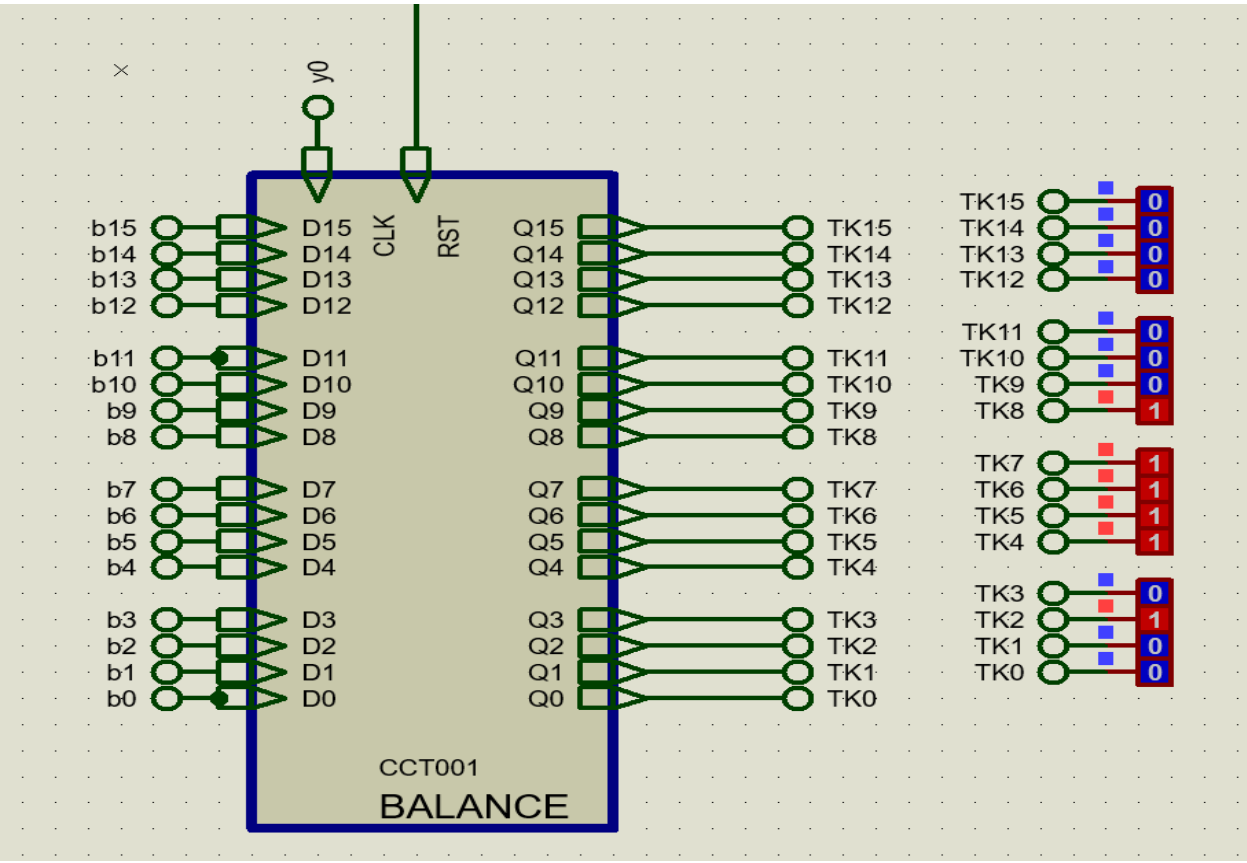


Figure - 3

Storing algorithm:

As there are four input, we need four 16-bit register. For passing the data in various register, we use a 2-to-4 decoder. The output of decoder is connected with the clock of positive edge triggered flip flop (Figure - 5).

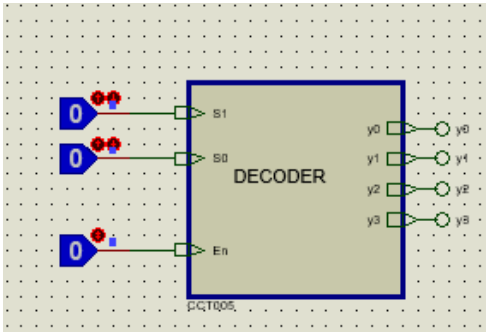


Figure - 4

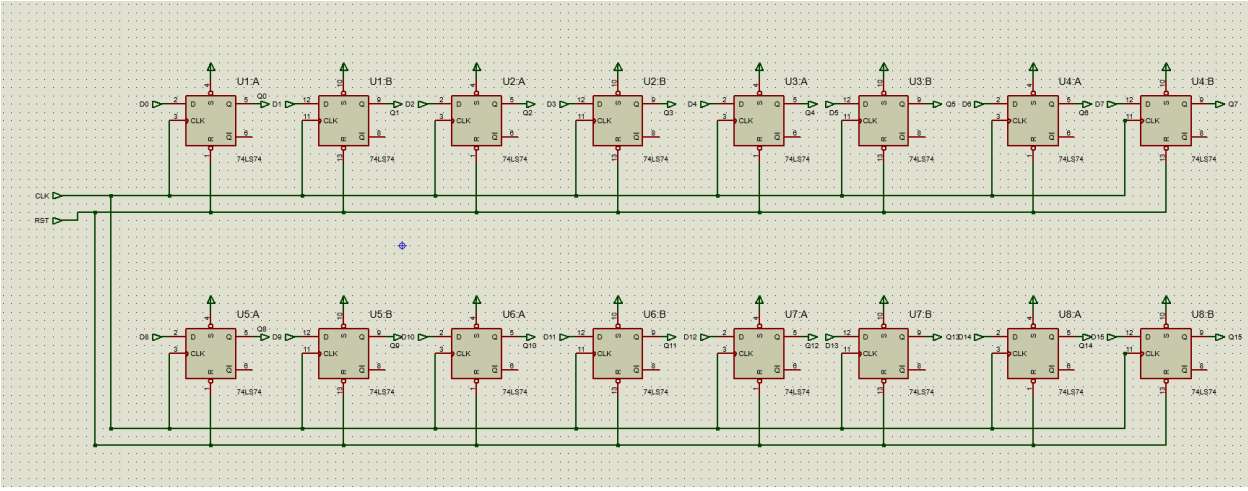


Figure – 5

For (S1 S0)

00 , Y0=EN & EN=TOOGLE FROM 0 T0 1, which means register for BALANCE is active & it will store the 16 bit binary value.

01 , Y1=EN & EN=TOOGLE FROM 0 T0 1, which means register for PRODUCT A is active & it will store the 8 bit binary value.

10, Y2=EN & EN=TOOGLE FROM 0 T0 1, which means register for PRODUCT B is active & it will store the 8 bit binary value.

11, Y3=EN & EN=TOOGLE FROM 0 T0 1,which means register for PRODUCT C is active & it will store the 8 bit binary value.

PART 3 : PRODUCT CHECKER & UPDATEING VALUES (8 bit)

The product checker submodule makes a comparison between initial available product & demanded product.

Algorithm:

If available product > demanded product :

1. Product checker submodule passes the value of demanded product to the next submodule (see, part 4).Here op7,op6,op5,op4,op3,op2,op1,op0 are 8 bit output from the product checker.(figure 1.5)
2. Then the checker updates it value (Figure 6) by subtracting.

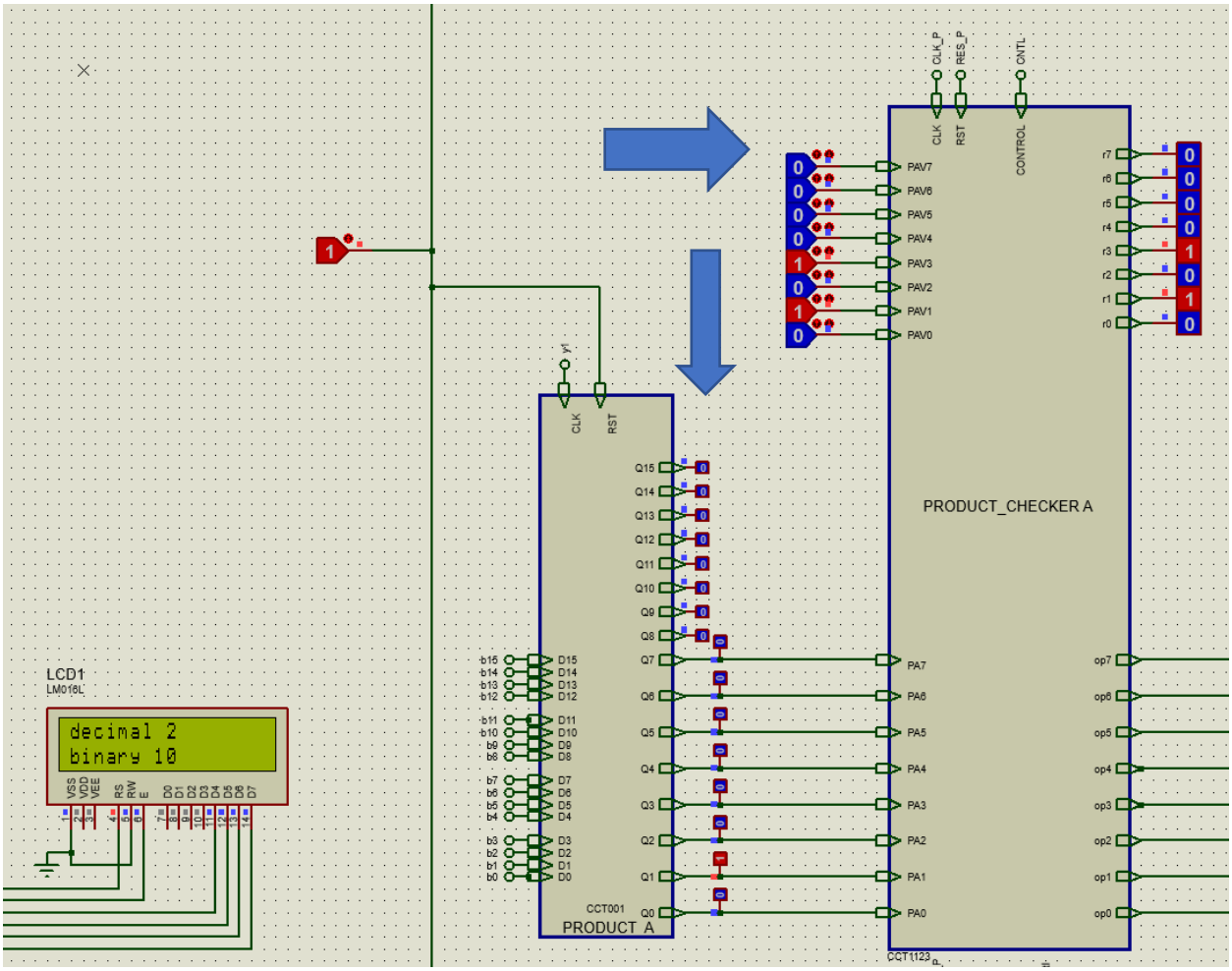


Figure - 6

From figure - 6 we can see, the number of demanded product is 2 (b'10). Available product is 10 (b'1010). When CLK is toggle from 0 to 1 (positive edge), it updates the value of available product (Figure 7) & it will be 8(1000).

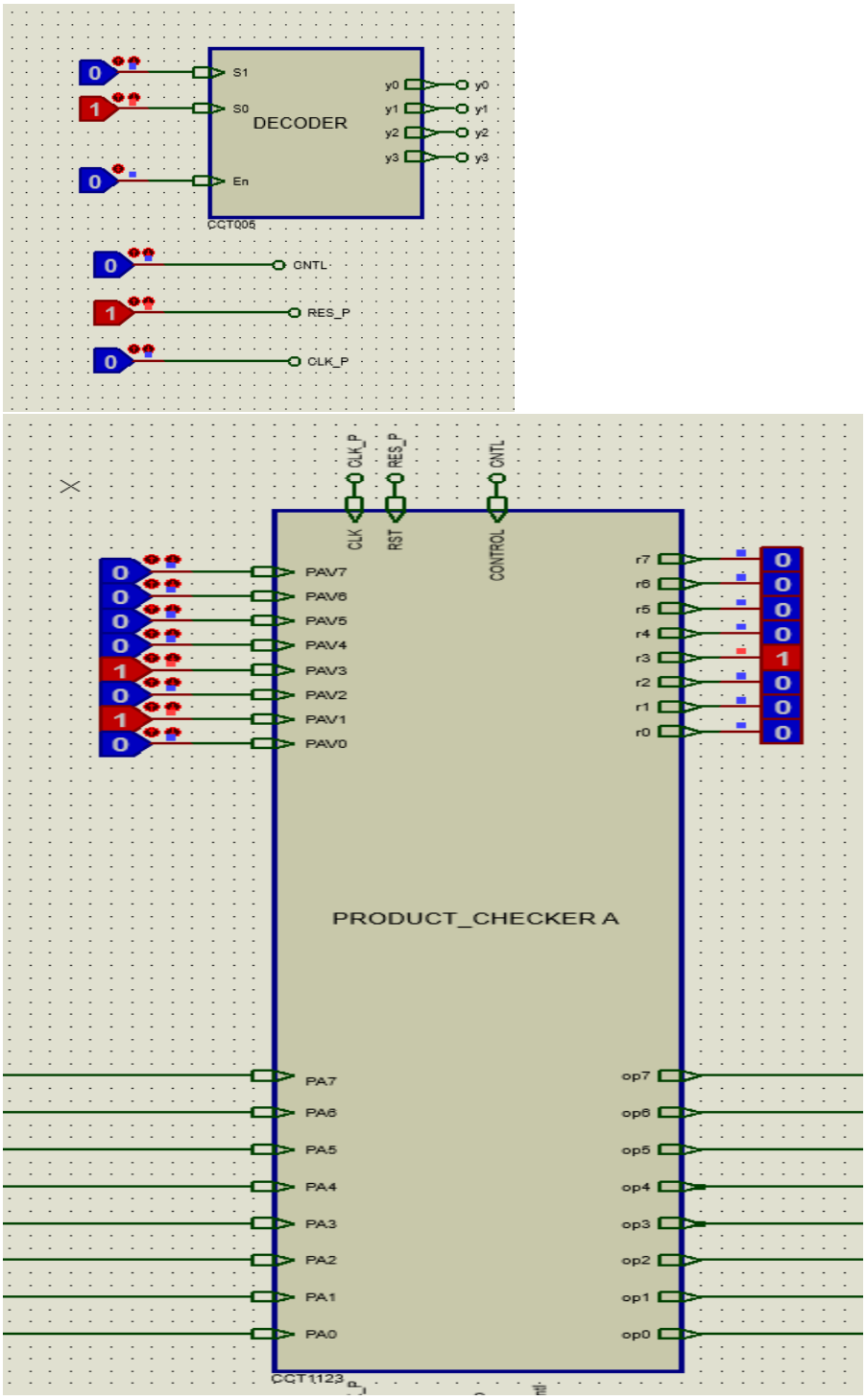


Figure – 7

So after customer1, when customer2 will come & demand a new quantity let 3(b’11), the new updated available product will be 5(b’101) (Figure 7) & so on.

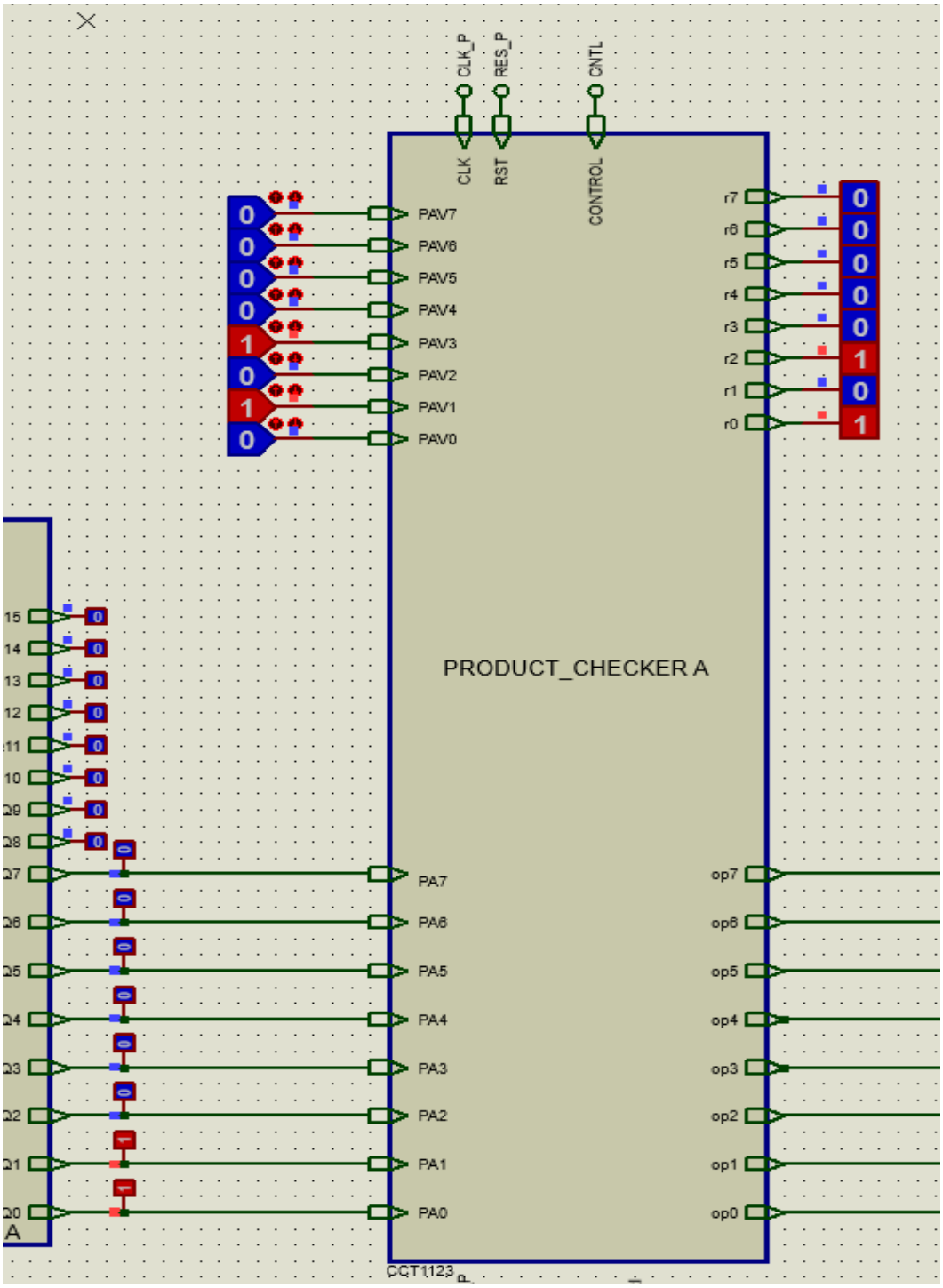


Figure - 7

When available product < demanded product :

The product checker will not pass the values to the next stage (part 4, we will discuss).

Circuit diagram inside the checker :

It’s one kind of sequential circuit.

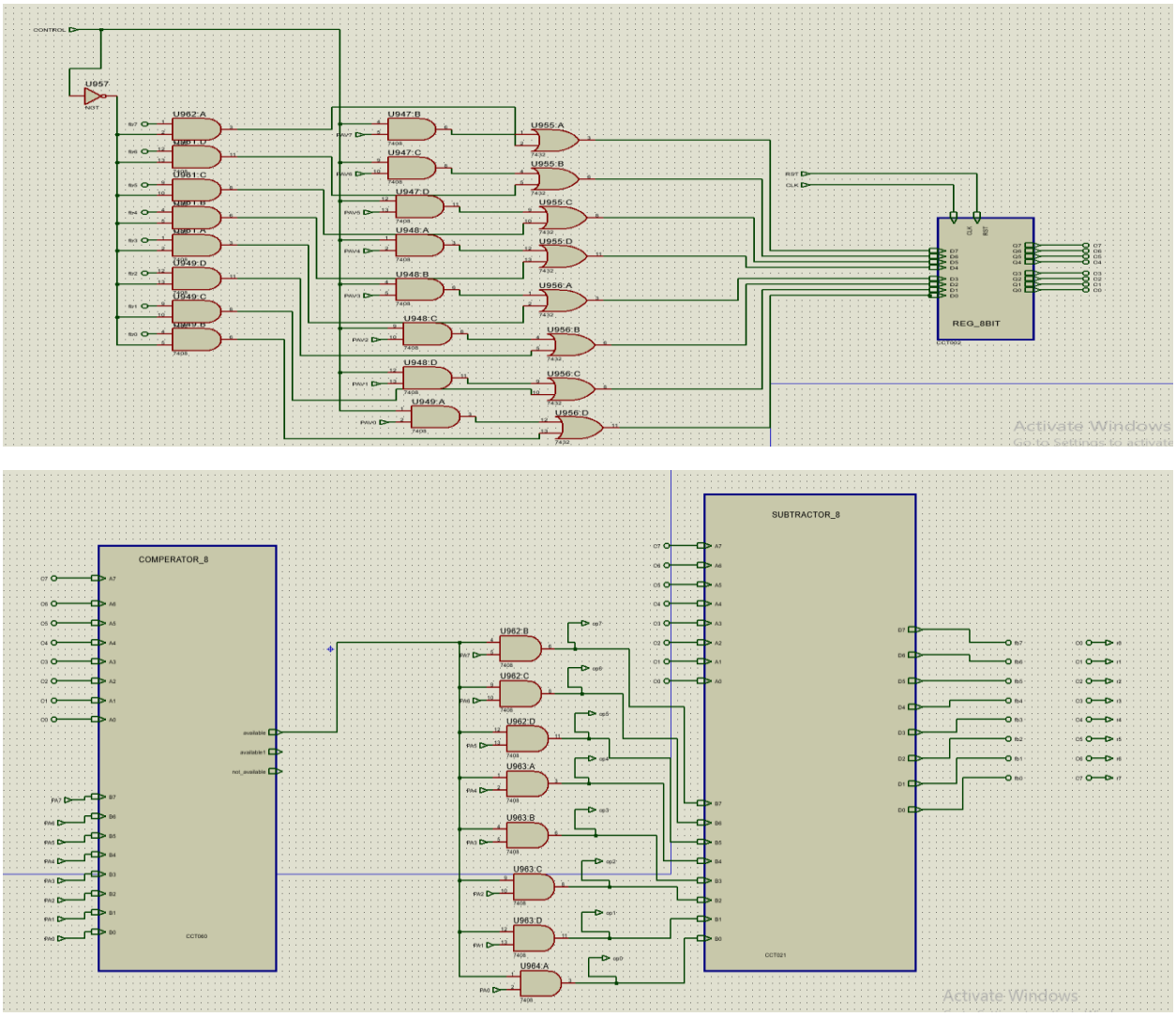


Figure – 8

Part 4 : Multiplication

Two 8-bit numbers are multiplied in the multiplication block. One number is fixed (Unit cost of a product) and another number comes from the register according to the given input (Figure – 9).

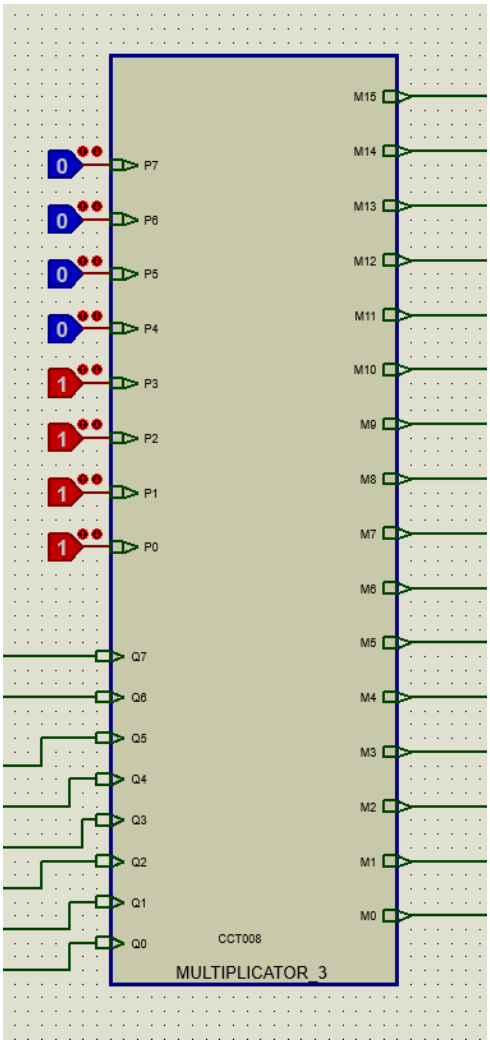


Figure – 9

If we multiply two 8-bit numbers, the output will be a 16-bit number. So the multiplier has 16-bit output.

Algorithm :

Multiplier is mainly constructed by doing AND operation the multiplicand with a bit of multiplicator and then shifting it. After all the shifting, they are added.

```
    11011011
    00110011
    11011011
   110110110
  0000000000
 00000000000
110110110000
1101101100000
00000000000000
000000000000000
0010101110100001
```

Here each bit in multiplicand is ANDed by each bit of multiplicator. Then they are shifted left by 0 padding in the right. After all the shifting, they are added. This is obtained by the circuit below (Figure-10).

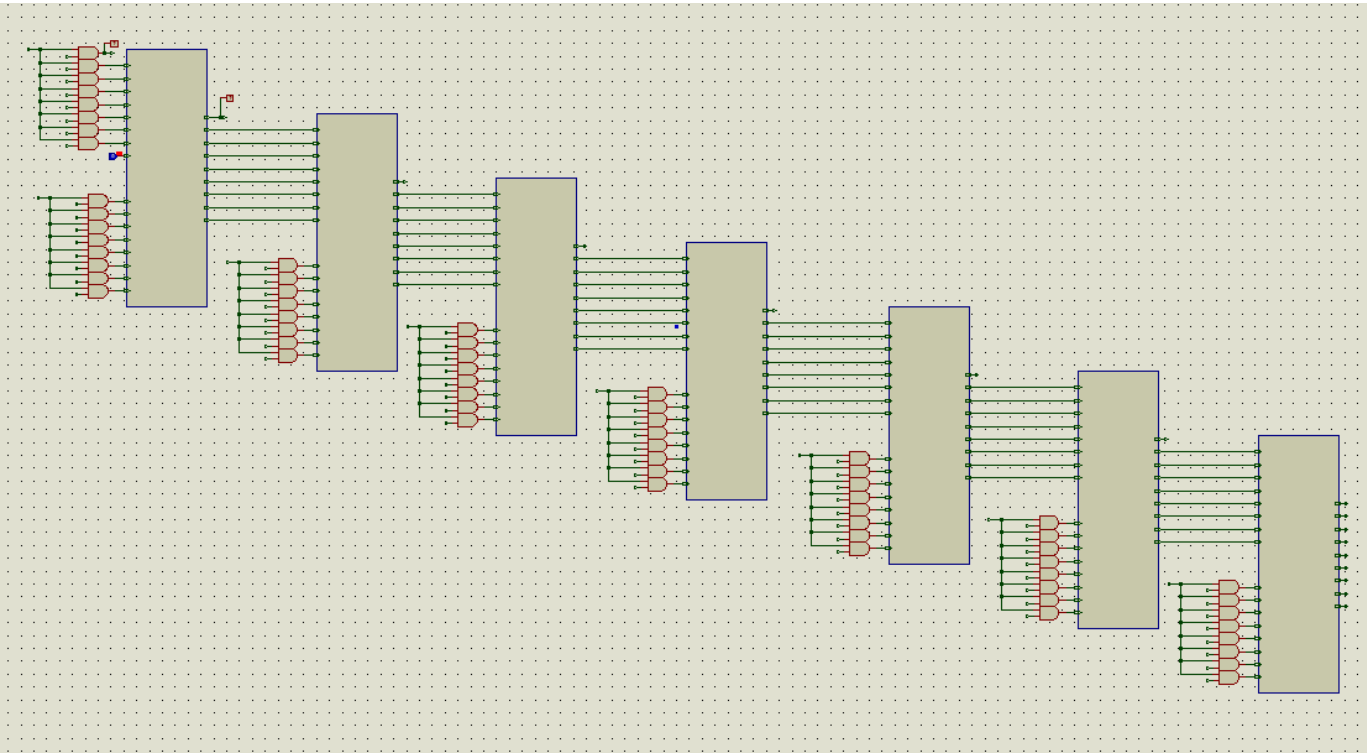


Figure - 10

In the circuit, the addition is done between two consecutive bit-stream. So, for an 8-bit multiplier, 7 8-bit adders are needed.

Part 5 : Addition

After multiplication, the cost of each product should be added. So we used 16-bit adder (Figure – 11).

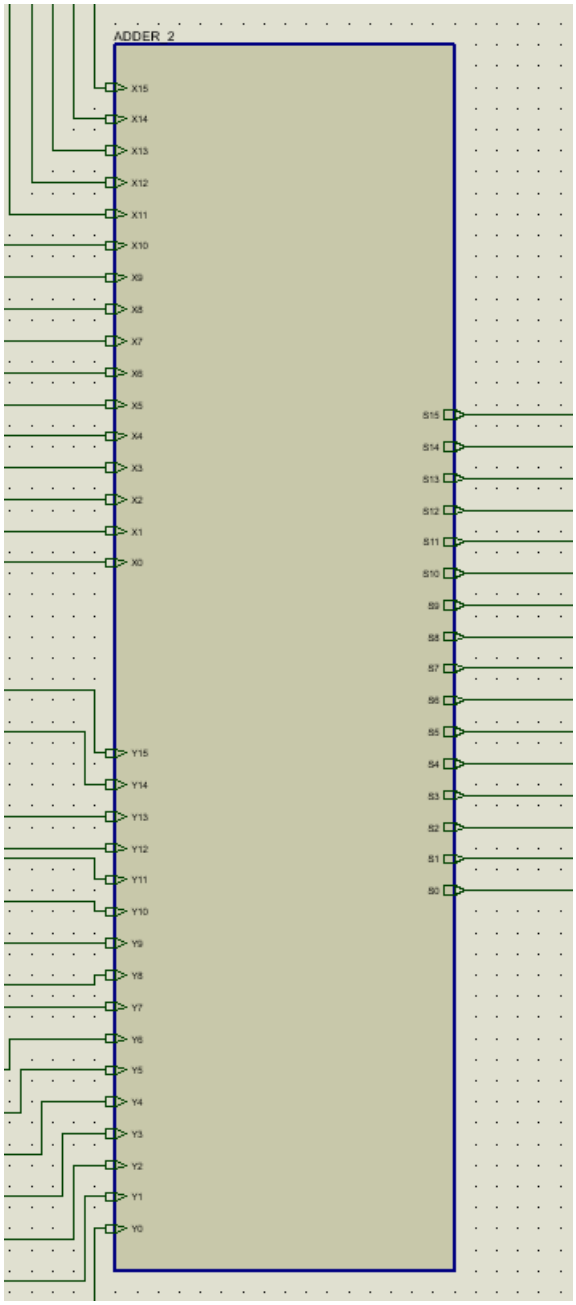


Figure – 11

We have to add three 16-bit numbers. So we used two adders. Adder1 will add the cost of Product-A and Product-B. Adder2 will add the Product-C and the output from Adder1.

Inside the adder block :

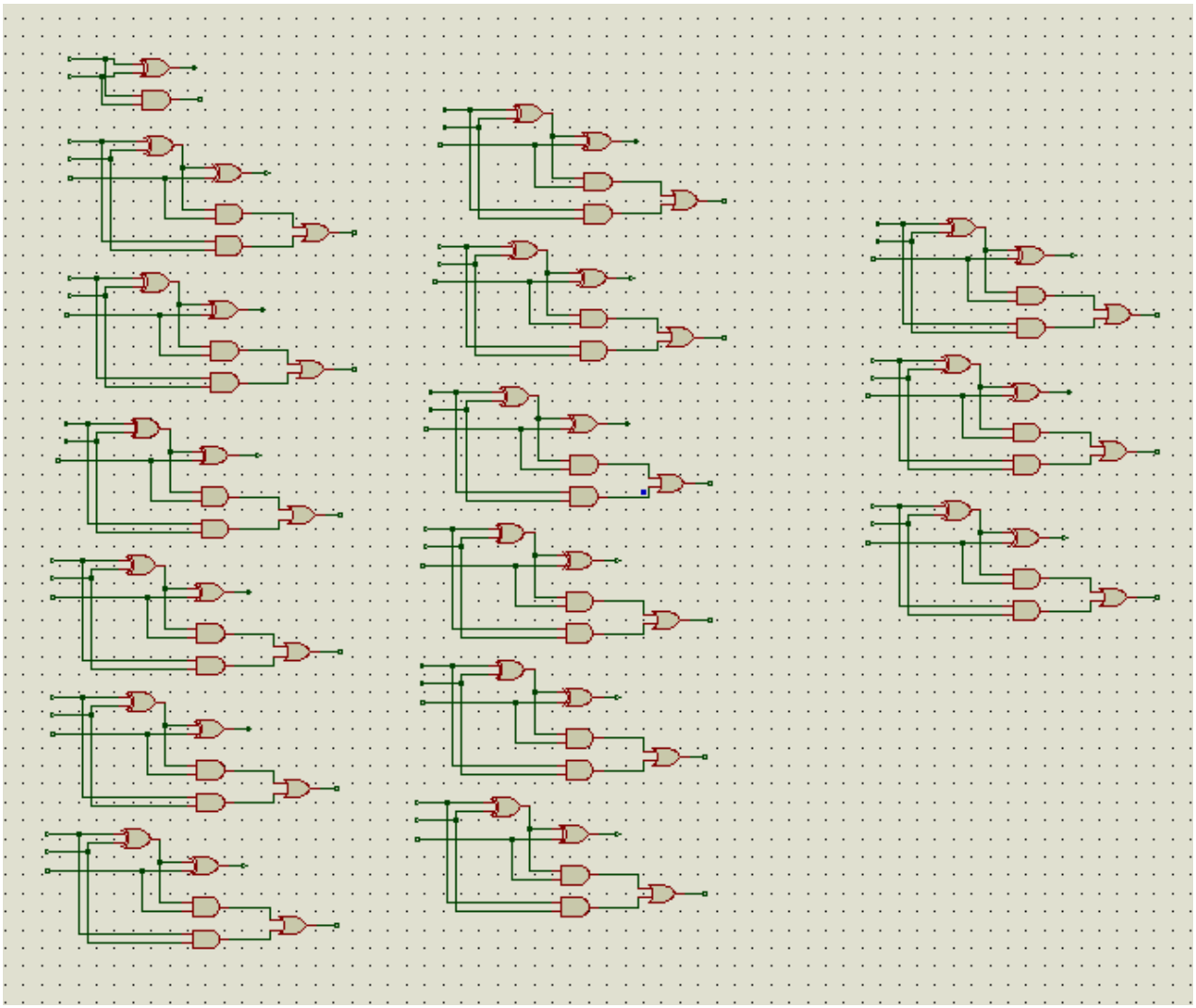


Figure – 12

This is a simple 16-bit adder. 16 full adders are used here to construct this circuit.

Part 6 : Balance Checker & Subtraction

The balance checker has two inputs. One is the input money or balance from the customer which was stored in register 1. Another is the total cost calculated by the two adders. This module subtracts the cost from the balance and gives output the remaining balance or return money.

This module also works exactly the same if the customer wants to make another purchase after the first purchase decision.

There is a feedback loop in this module (Figure-14) by which it can store memory. It stores the remaining balance and subtracts from it if another purchase is made.

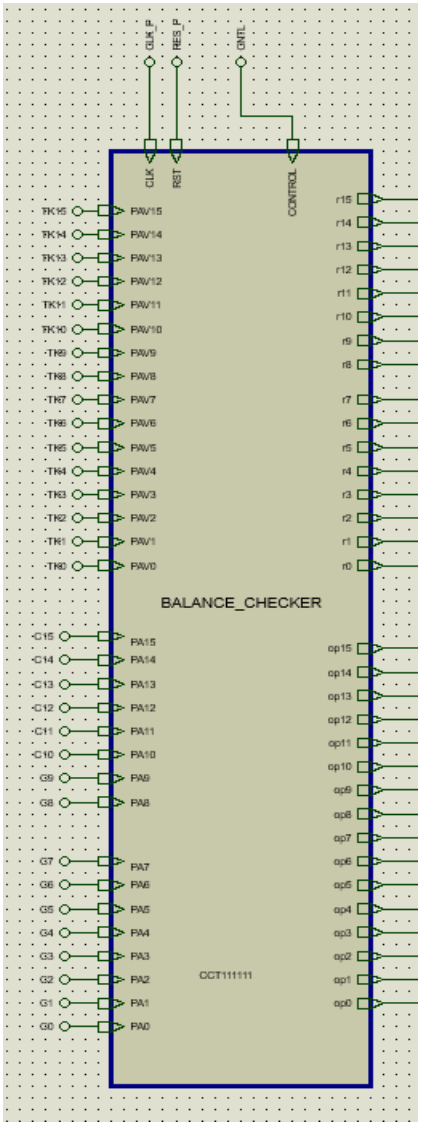


Figure – 13

Inside the balance checker module :

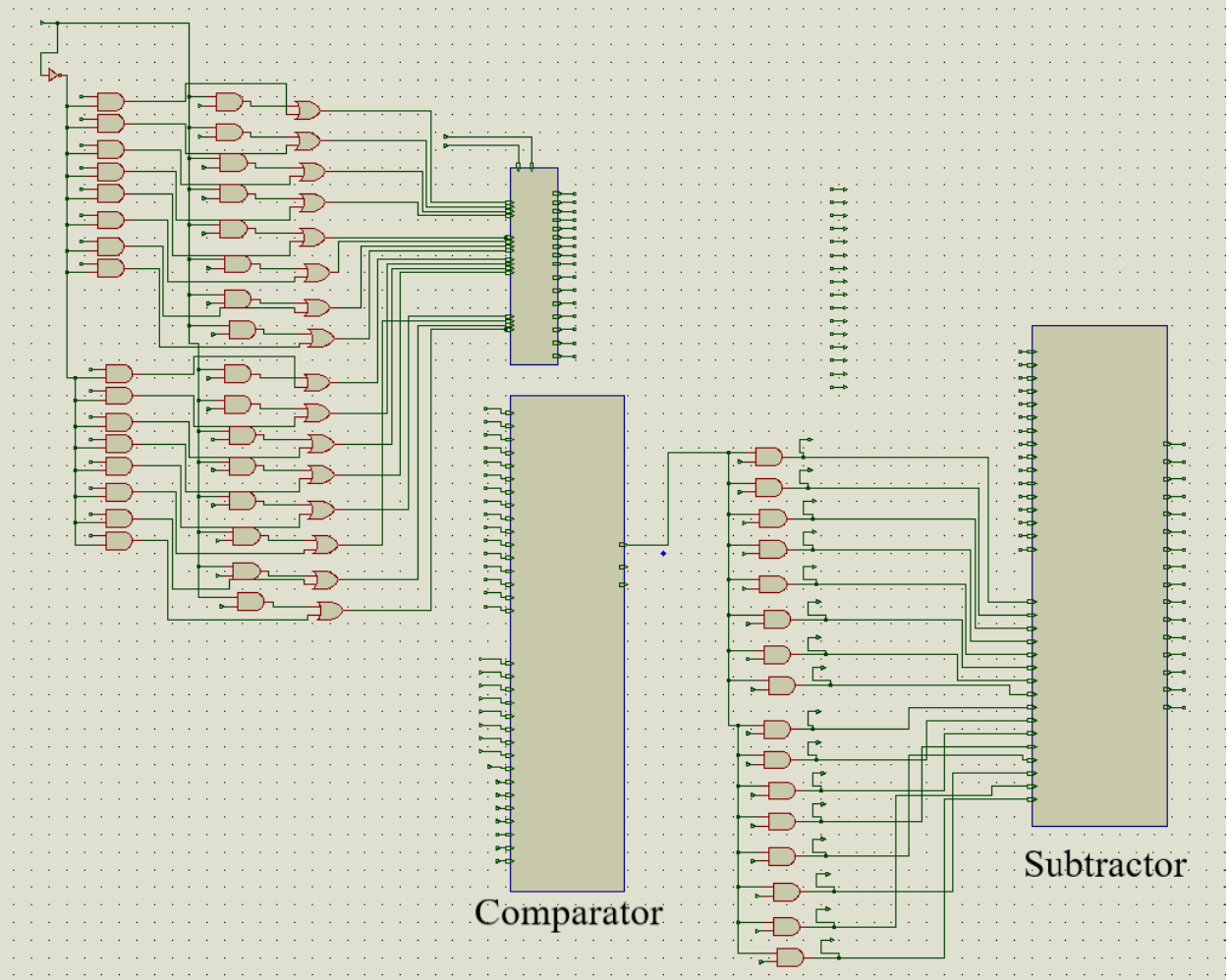


Figure - 14

There is a comparator and subtractor block in the balance checker module. The comparator compares the balance and the cost. If balance > cost; then the subtractor will subtract the cost from the balance.

The circuit inside the subtractor :

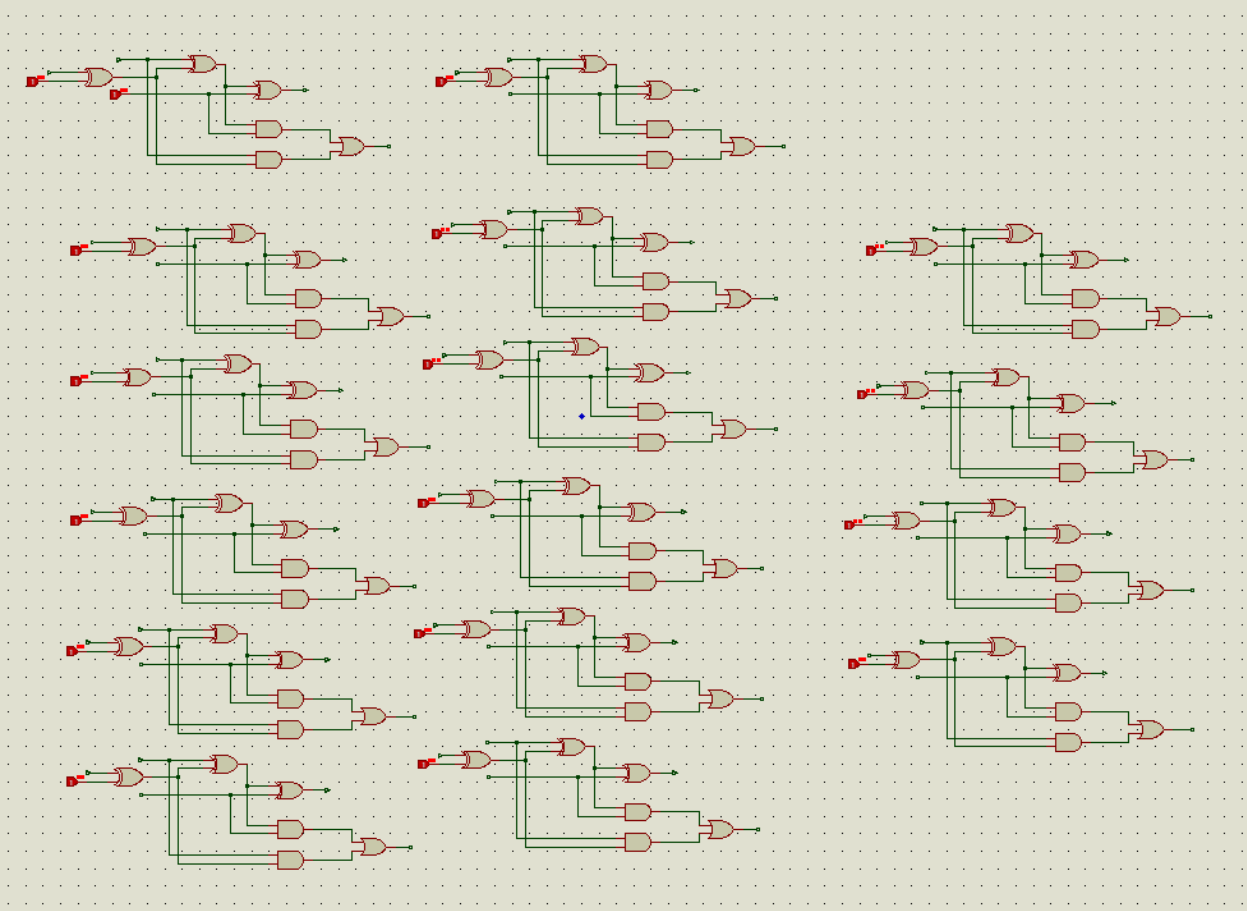


Figure - 15

The circuit inside the comparator :

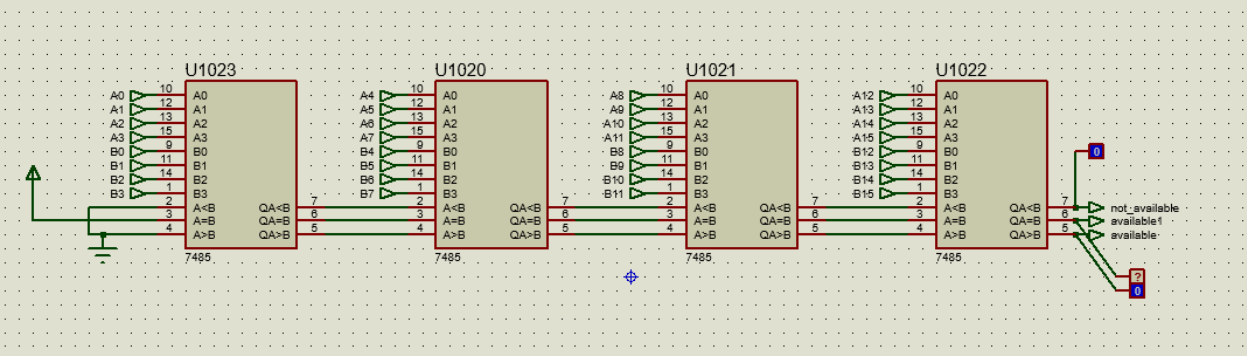


Figure – 16

Part 7 : Binary to BCD Converter

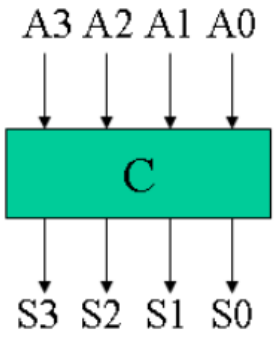
We converted binary into BCD by shift and add-3 algorithm :

1. Shift the binary number left one bit.
2. If 8 shifts have taken place, the BCD number is in the *Hundreds*, *Tens*, and *Units* column.
3. If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
4. Go to 1.

Example :

Operation	Tens	Units	Binary
HEX			**E**
Start			1 1 1 0
Shift 1		1	1 1 0
Shift 2		1 1	1 0
Shift 3		1 1 1	0
Add 3		1 0 1 0	0
Shift 4	1	0 1 0 0	
BCD	**1**	**4**	

Truth table for ADD-3 module :



A3	A2	A1	A0	S3	S2	S1	S0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Using this truth table we designed the C block as :

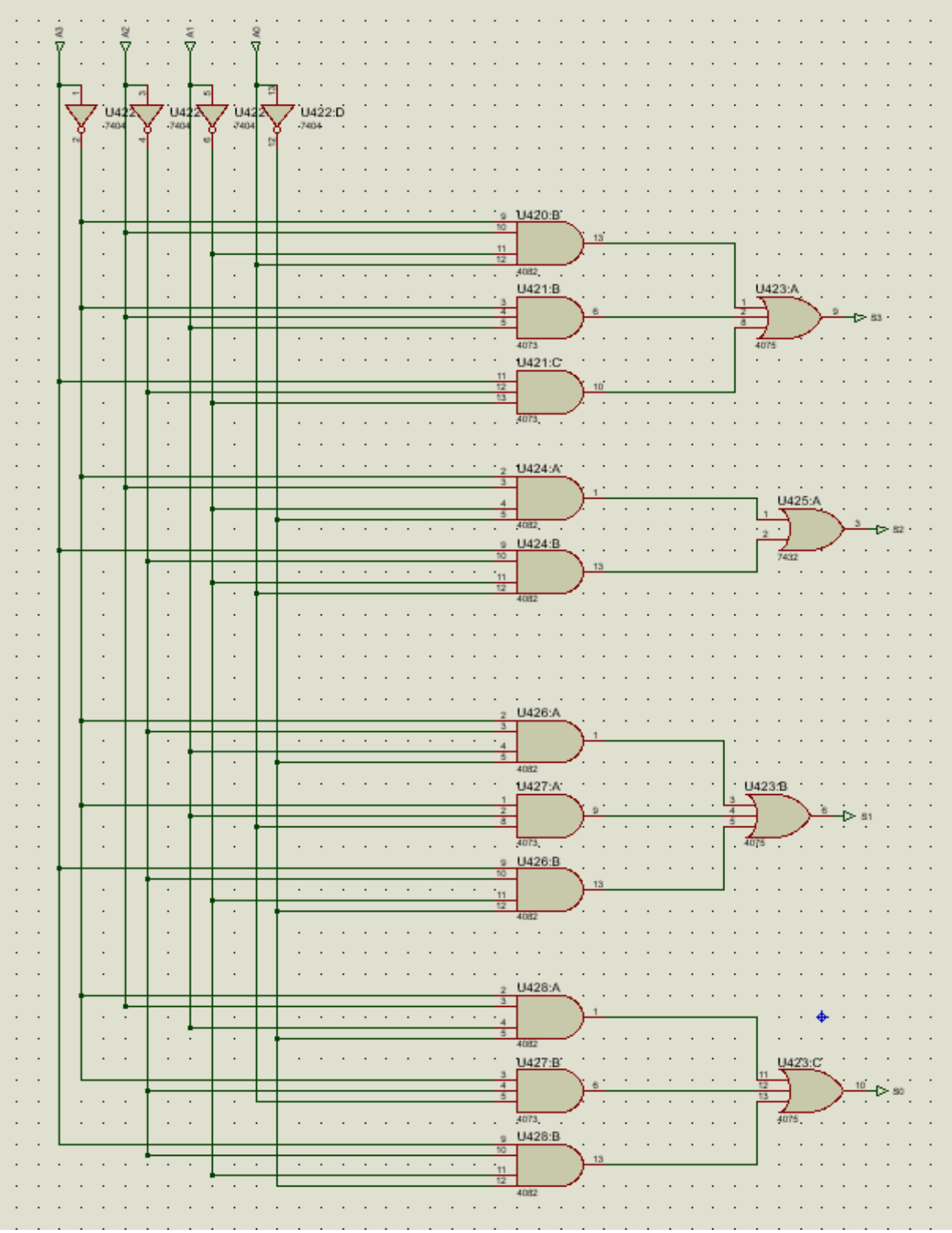
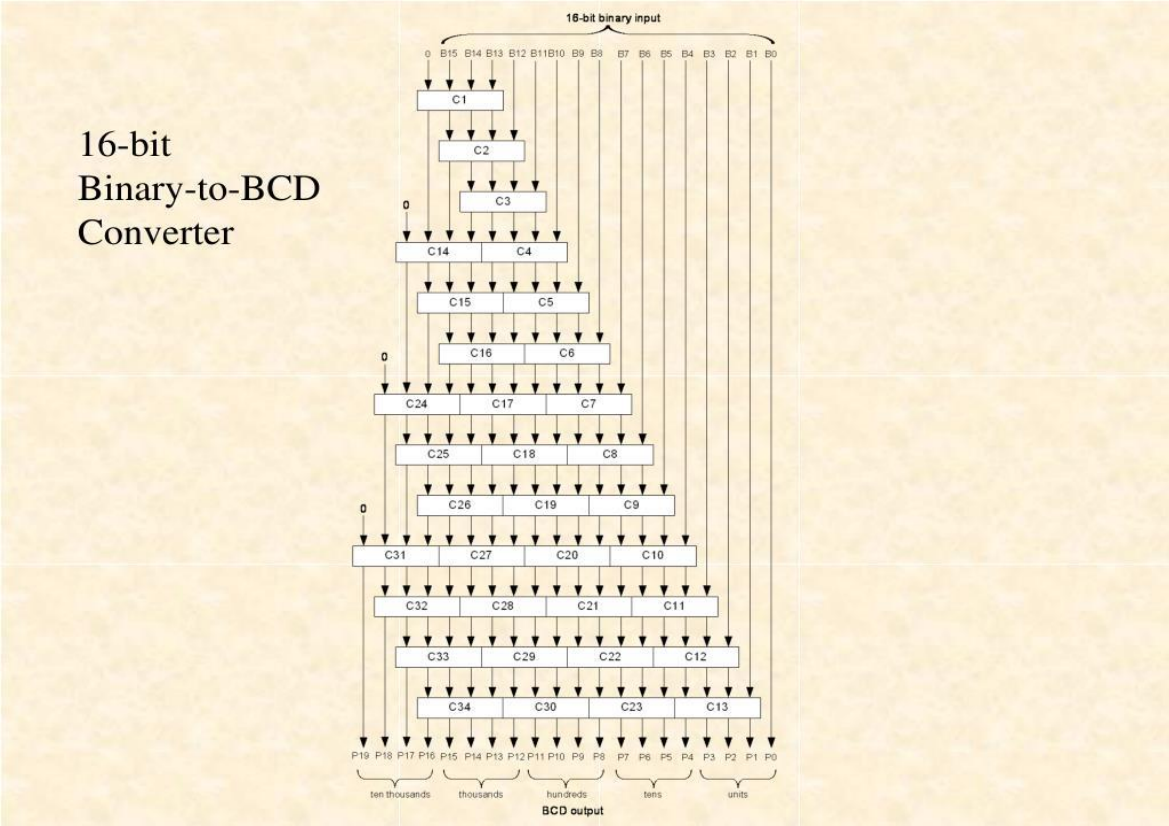


Figure - 17

Using this C block the 16-bit binary to 20-bit BCD conversion algorithm will be :



We constructed this circuit in proteus :

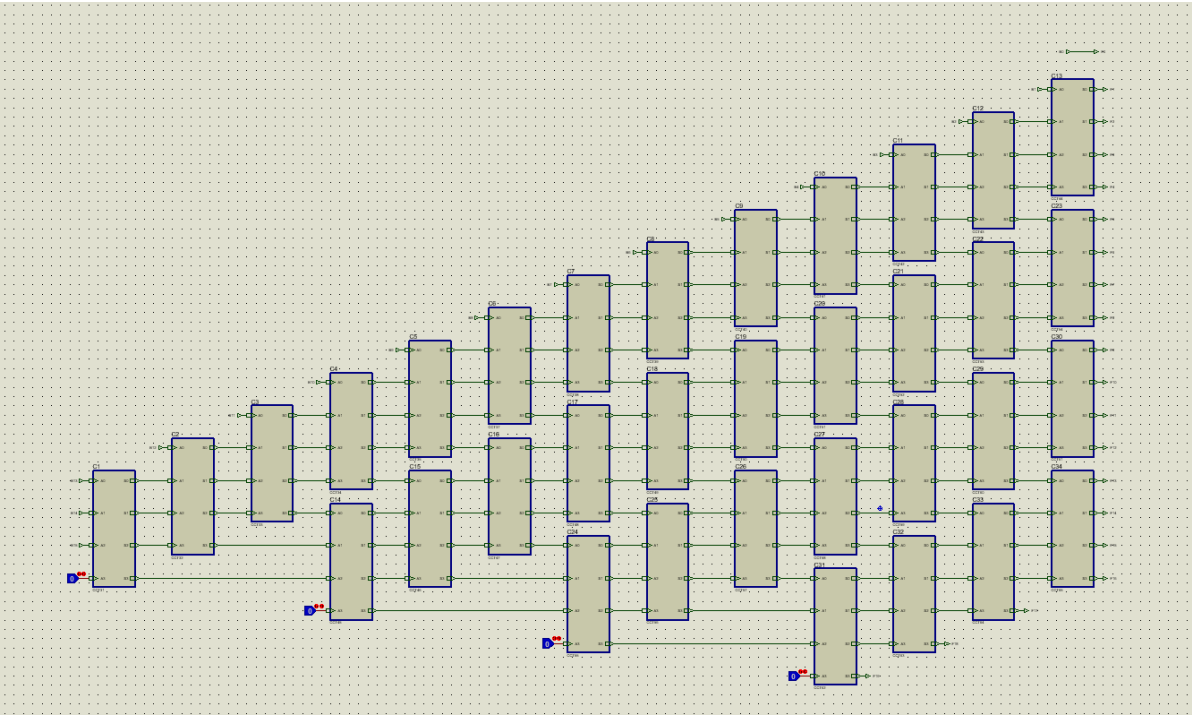


Figure – 18

Thus our output circuit is like this :

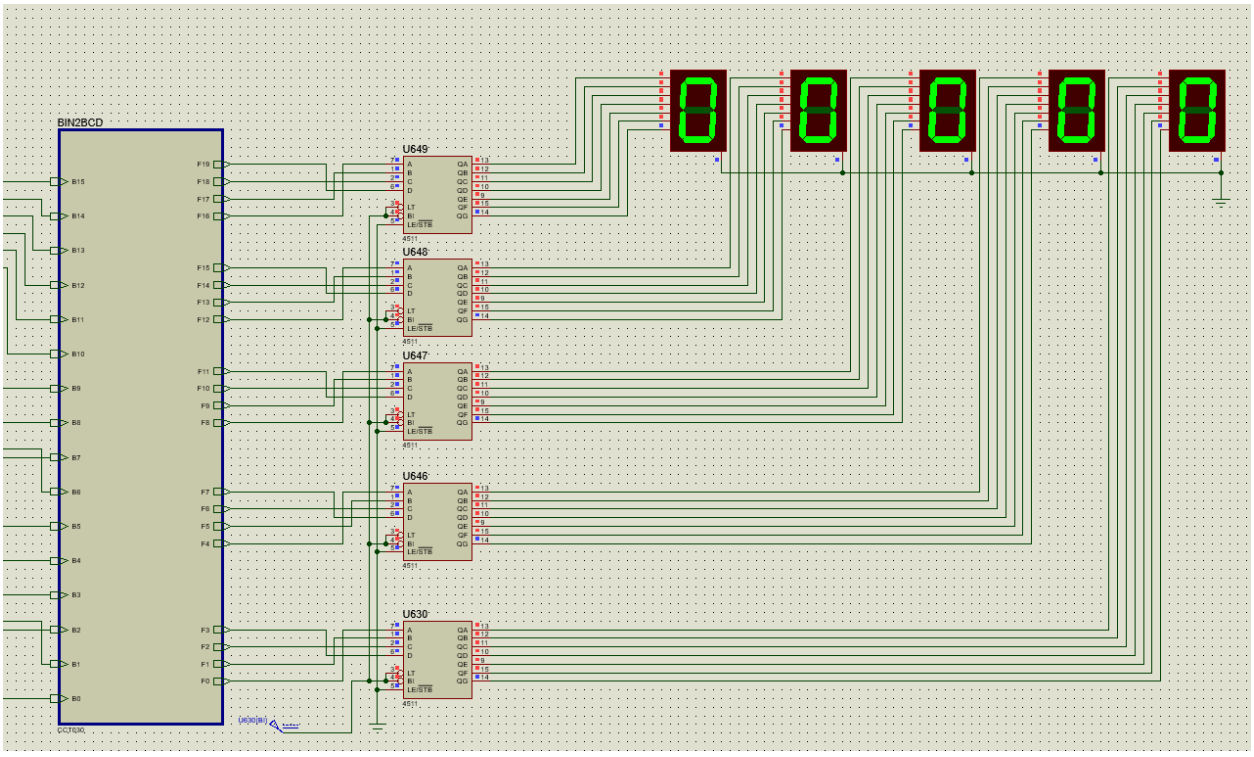


Figure - 19

We also three LED to indicate if the balance is enough to purchase the required products. A comparator is used to control these LEDs.

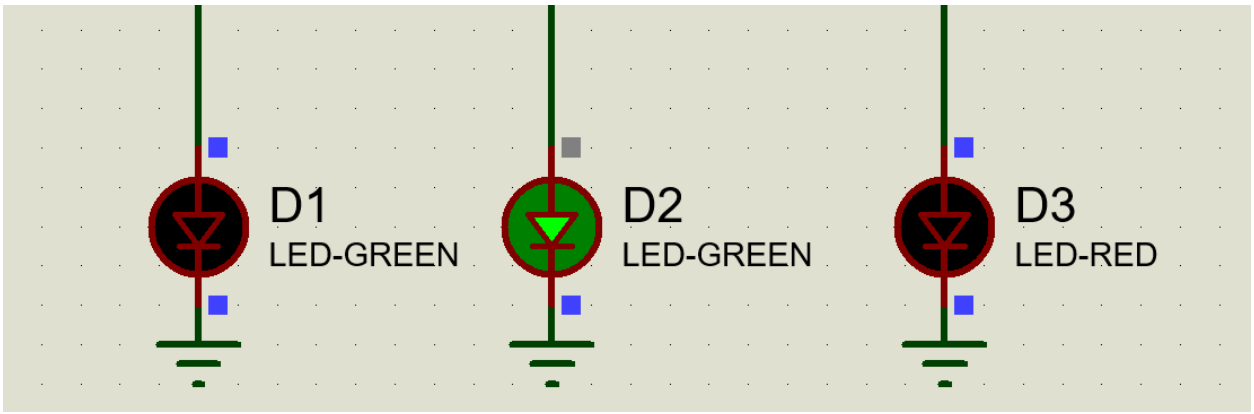


Figure – 20

** When Balance > Cost ➔ D1 turns green

** When Balance = Cost ➔ D2 turns green

** When Balance < Cost ➔ D3 turns red

Limitations :

There are some limitations in this project that needs to be discussed.

As we are using 16 bits for input money; maximum 2^{16}-1 = 65535 taka can be given as input. Same thing goes for quantity of products. For taking input for products, we are using 8 bits. So maximum 2^{8}-1 = 255 products can be selected per item for one purchase. If a customer wants to buy more item, he will have to make purchase in more than one step.

Discussion :

In this "**Smart Booth**" project we have done the data analysis, calculation part & the output part without using microcontroller. We used basic logic gates. The mechanical implementation is not done here. Further, it is possible to design a mechanical system (motor, sensors etc.) in the Proteus platform & make simulation. For this purpose, we have to design stepper motor controller system.

Future aspects:

1. As the system is designed for '16 bit basis' for balance & '8 bit design' for product, further we have to plan the system for 32 bit design & design our mechanical system.

2. Marketing plan:

- Improve the design & making PCB design (using microcontroller if necessary).

- Create central monitoring system (An IoT based system, done, in 3-1 communication project) & developing own server.

- Making agreement with Bkash, Rocket or Sonali Bank or whatever for the transmission of balance & making a business scheme properly. Initially, it is complicated to make an agreement with this company. So, initially, we have a plan to install a system which can detect inputed money (let, you give two

50tk, the system detects this balance & takes input 100tk. And let, your cost is tk 80. So, the system returns back tk 20 likely to ATM booth system)

- Finally, install the system.

Contribution :

1606135 :

Constructing the multiplier, adder, subtractor, balance checker block, binary to bcd converter, comparator, output display system.

1606138 :

Writing the code in Verilog.

1606159 :

Constructing the input giving system, writing code in Arduino, decimal to binary conversion, constructing registers, product checker block.