End Project Report on

# "Inventory Manager"


Submitted for end project of FPGA Based System Design Lab:


| | |
|---|---|
| Harshith Ranga | 21ECB0A47 |
| Vikram Dev | 21ECB0A48 |
| Rohan Gunjal | 21ECB0A49 |
| Somil Maldhani | 21ECB0A72 |


## Supervised By:


- Dr. T.V.K Hanumantha Rao
- Dr. Ch. V. Rama Rao

Department of Electronics and Communication Engineering,
National Institute of Technology, Warangal

# Aim:

To implement Inventory Management system using FPGA Artix – 7 4DDR and perform necessary operations to display working of the project.

# Software and Hardware Used:

Vivado Xilinx 2014.2 Version/ 2021 Version as EDA Tool

FPGA Artix – 7 ADDR (XC7A100TCSG324-1) as Hardware Tool

# Theory:

## Introduction to Verilog:

Verilog is a hardware description language (HDL) used to represent electrical systems. At the register-transfer level of abstraction, it is most widely employed in the design and verification of digital circuits. Today, Verilog is the most popular HDL used and practiced throughout the semiconductor industry.

HDL designs are technology-independent, highly straightforward to create and debug, and are typically more helpful than schematics, especially for complex circuits. HDL was created to help engineers improve the design process by allowing them to define the intended hardware's functionality and then have automation tools turn that behaviour into actual hardware pieces such as combinational gates and sequential logic.

## Introduction to FPGA:

FPGA stands for field-programmable gate array. At its core, an FPGA is an array of interconnected digital subcircuits that implement common functions while also offering very high levels of flexibility. But getting a full picture of what an FPGA is requires more nuance. This article introduces the concepts behind FPGAs and briefly discuss what logic gates are, how to program an FPGA, and what makes an FPGA different from a microprocessor in design.

It consists of three main parts: Configurable Logic Blocks — which implement logic functions. Programmable Interconnects — which implement routing. Programmable I/O Blocks — which connect with external components. FPGAs can be used as co-processors to accelerate data analysis and processing in research and production environments. Another commonly customised FPGA based networking device are Network Interface Cards (NIC). A lot of processing and reaction latency can be saved by moving trading algorithms inside the NIC and closer to the exchange

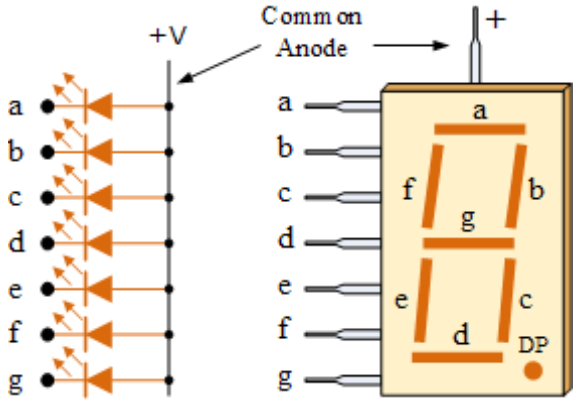
## Seven Segment Display:

A Seven-Segment Display is an indicator commonly used by FPGA designers to show information to the user. Code to convert from binary to seven-segment display compatible can be done easily in VHDL and Verilog.

Both the VHDL and Verilog code work the same way. They simply use a Look-Up Table to do the decoding from the hexadecimal input to the 7-segment output. The individual segments are listed in the table below.

7-Segment Locations

| Name | Location |
|---|---|
| A | Top Middle |
| B | Top Right |
| C | Bottom Right |
| D | Bottom Middle |
| E | Bottom Left |
| F | Top Left |
| G | Middle |
| DP | Decimal Point7-Segment Labelled Indicator |




To drive each segment to a different value, the enables (AN[7:0]) and segment values (CA…CG) must be driven sequentially, at a rapid enough speed that our eyes don't detect the flicker. For example, to drive display 0 and 1 to the values 3 and 9, we drive CA…CG to display the value 3, and then drive AN[0] LOW, then we drive CA…CG to display the value 9 and drive AN[1] LOW. If we refresh each digit about every 2 ms, our eyes can't detect any flicker.

Note that the code below will only drive one seven-segment display and therefore can only display decimal numbers 0-9. Any application requiring a number higher than nine will need to first convert the number to Binary Coded Decimal (BCD). The linked code shows how this is done. If this is your first time using a seven-segment display you should just try counting 0-9 first, then work your way up to a 2-digit display.

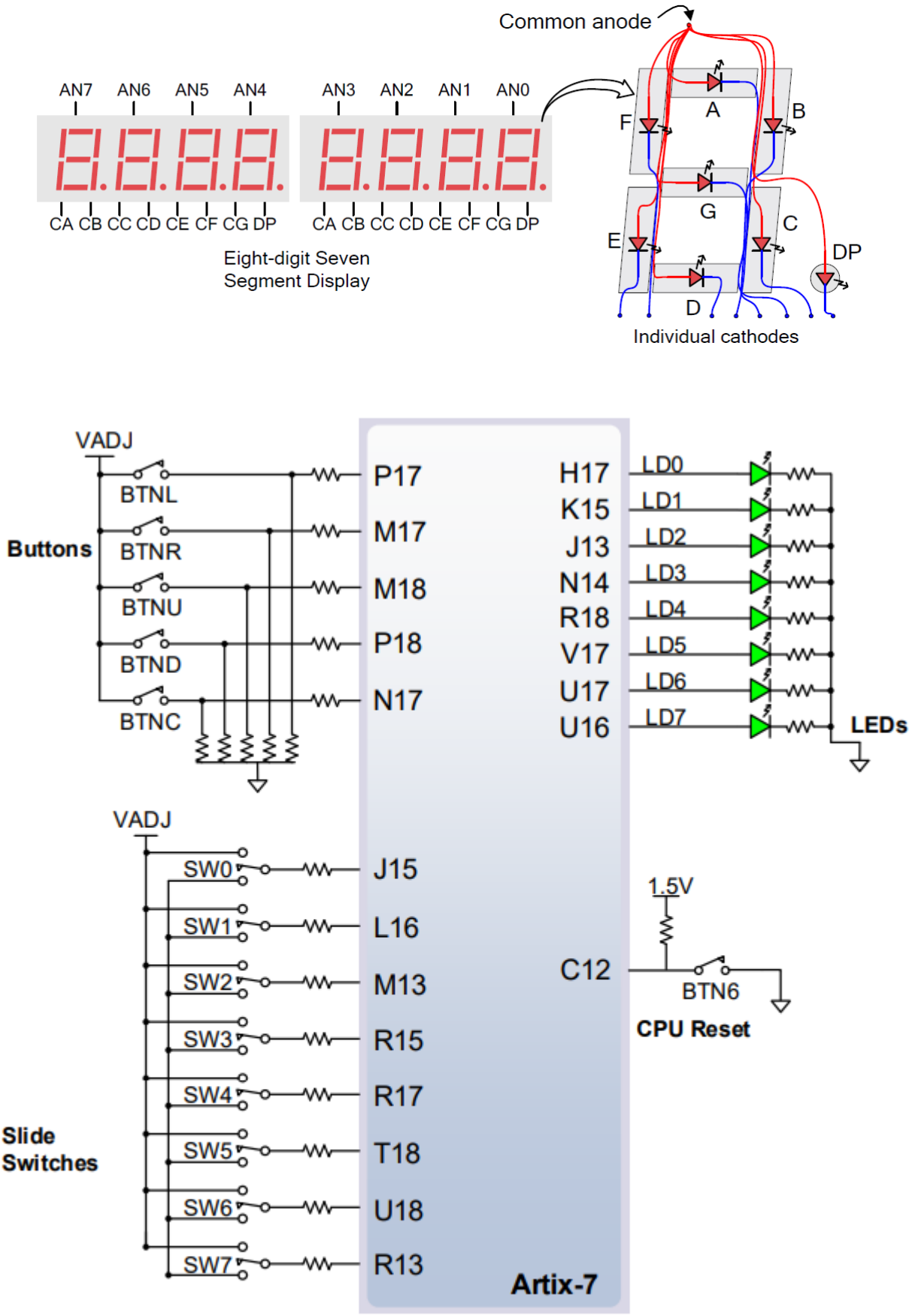



*Figure 11. General purpose I/O connections.*

## Project Introduction:

This project uses direct hardware interference to address the challenges of space management and classification. During the current economic expansion, both the quantity and scale of new commercial warehouses have increased dramatically; e-commerce, or online shopping, has been a driving factor of this upsurge.

Every quarter, more and larger inventories are put up, and as a result, their administration has become an important aspect of the planning for these warehouses. Using Verilog, A quicker and more effective way to organise the inventory is guaranteed. The FPGA will be a low-cost, less power-hungry storage organiser alternative to software, making it a competitive product in the market.

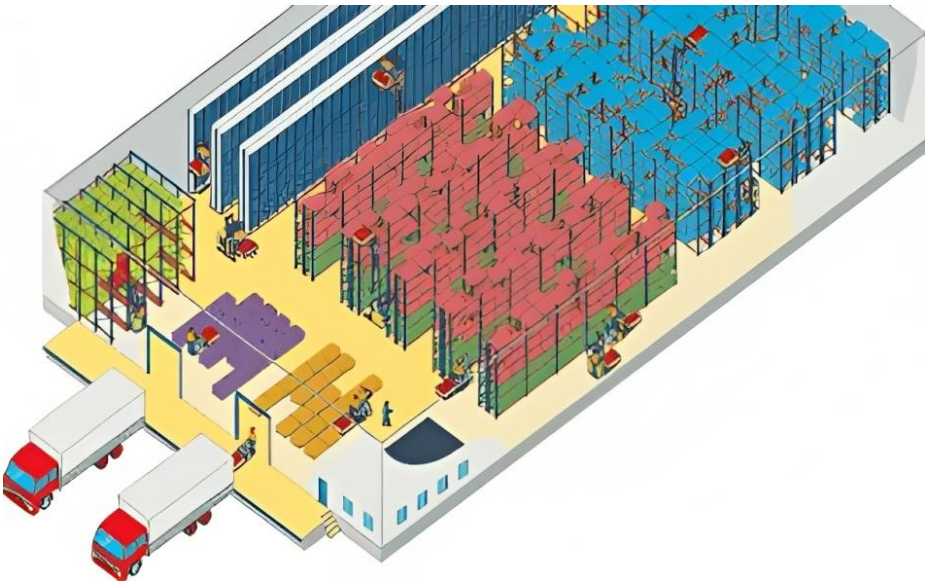
Considering the fact that administration of the warehouses is crucial, we have implemented simple yet efficient methodologies for better outcomes and sustainable development. Three levels of abstraction have been used while developing. The project first encounters a password checking module, followed with product ID encoder and then Location Finder with displaying the count of the units of a unique product ID.

To maintain the integrity of the warehouse management system, security must be unbreakable. Hence, any user must verify their identify with 8 Bit long decimal number password. On the account of failure after trying thrice, the respective authorities are alarmed and unless a Master Password is entered and verified, management system is not granted access.

To avail a unique product its designated location, its product ID is entered into the module and its respective floor is allotted based on its priority and followed by its section, sub section and shelf. These operations are performed in the Product ID Encoder Module.

After the location of the product is accessed, one can start adding or subtracting items based on the number of orders. Since priority is given to certain items these items are added and removed from the shelfs primarily and then algorithm is continued for the other products. After updating the current values of the quantity of the products, these are displayed on the console and both the underflowing and overflowing cases are also defined for extreme cases. These operations are dealt in Location Finder, Display Modules.

# Warehouse Representation:



The image above is an accurate representation of the warehouse this project is handling. According to the warehouse parameters, the warehouse essentially contains three floors with the bottom most floor given more priority for removal or addition of items for easier access. In each floor there are 5 Sections which are further into 5 Sub Sections. In each of the 5 Sub Sections, there are 5 Shelfs and, in each shelf, it has 5 containers. Hence, a shelf can withhold a total of 75 units of a unique product.

By these calculations, the full capacity of the warehouse is to store 125 Unique Products and 9375 total units of all the products.

Considering the quantity of the number of products, warehouse operations can be performed on huge scale. Addition or Subtraction of the products from their respective locations needs to be undergone smoothly with less power consuming tactics for a sustainable development.

# FLOWCHART:

## 1. Password Checker

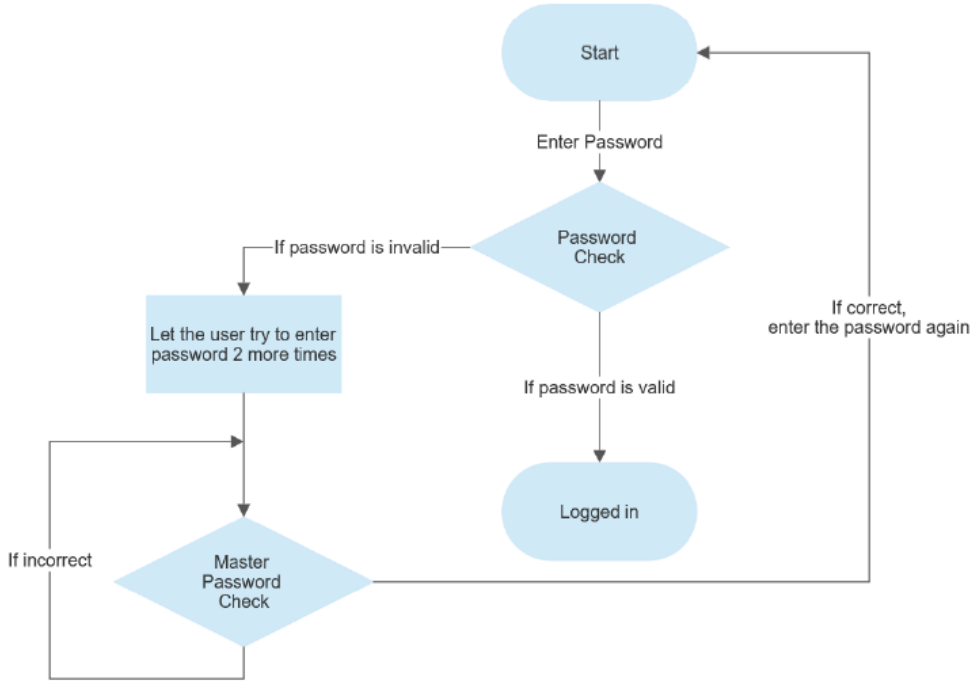



*Figure 1: Password Check*

## 2. Inflow and Outflow Working of the Warehouse

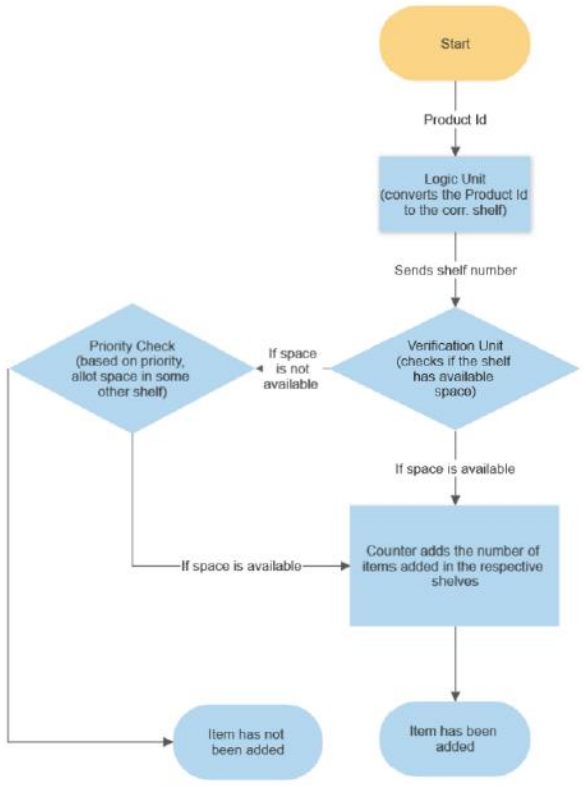



*Figure 2: The inflow workflow of the Warehouse*

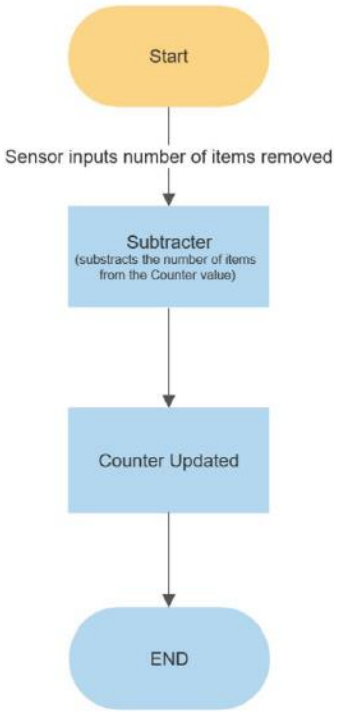



*Figure 3: The outflow workflow of the Warehouse*

# MODULE: MAIN PROJECT

## Design Sources:

```
`timescale 1ns / 1ps

module projectMain(clk, passInput, masterInput, productId, Enter, Enter2, Enter3,
DomItems, SubItems, rst, Alarm, Total, shelfTot, Enabled, cathode, anode);

input clk;

input [31:0] passInput,masterInput;

input [5:0] productId;

input Enter, Enter2, Enter3, rst;

input [3:0] DomItems,SubItems;

output reg Alarm;

output wire [3:0] anode;

output wire [7:0] cathode;

output reg [6:0] shelfTot;

output reg [12:0] Total;

output reg Enabled ;

//////////////////////////////Password-Registers//////////////////////////////////////////

reg Enable;

reg chk1,chk2;

reg reset;

reg[1:0] count;

//////////////////////////////ProductId-Registers//////////////////////////////////////////

reg [3:0] section , subSection , shelf;

//////////////////////////////Add/Sub-Registers//////////////////////////////////////////

reg [3:0] addItems, subItems;  // 15-> max items to add or sub at a time

reg [6:0] totalItems;  //

//////////////////////////////Data-Registers//////////////////////////////////////////

integer data[2:0][2:0][2:0][2:0];

integer i,j,k,l;

//////////////////////////////Main-Registers//////////////////////////////////////////

//reg [3:0] selected [4:0];

reg overflow, underflow;
```

```
integer cnt;
////////////////////////////////BCD-Registers/////////////////////////////////////////////////
reg [12:0] t;
reg [3:0] d1, d2, d3, d4;
////////////////////////////////Initialization/////////////////////////////////////////////////
initial begin
$display("Init\n");
Enable <= 1'b0; count <= 2'd00; Alarm <= 1'b0; t <= 0;
totalItems = 7'd0;
Total = 13'd0;
shelfTot = 7'd0;
for(l=0;l<3;l=l+1) begin
   for(k=0;k<3;k=k+1) begin
      for(j=0;j<3;j=j+1) begin
         for(i=0;i<3;i=i+1) data[l][k][j][i] = 0;
      end
   end
end
end
////////////////////////////////Enter-Password/////////////////////////////////////////////////
always @ (posedge Enter or posedge Enter2 or posedge Enter3 or posedge rst) begin
$display("Enable %d Enter %d Enter2 %d Enter3 %d",Enable ,Enter,Enter2,Enter3);
if(rst == 0) begin
//    $display("Enable %d Enter %d",Enable ,Enter);
   if (Enable == 1'b0 & Enter == 1'b1) begin
   //Password = 32'h60306779;//masterPass = 32'h30805262;
   chk1 = (32'h60306779 == passInput);
   chk2 = (32'h30805262 == masterInput);
   $display("chk1 %d | chk2 %d | count %d \n", &chk1, &chk2,&count);
   if(count==2'b11)
      begin
         if(chk2==1'b1)
```

```
            begin
              Alarm = 1'b0;
              count=2'b00;
            end
        end
      else
        begin
          if(chk1==1'b1)
            Enable = 1'b1;
          else
          begin
            if(count==2'b10)
              Alarm = 1'b1;
            count = count+1;
          end
        end
      end
      if(Enable == 1'b1 & Enter2 == 1'b1) begin
        $display("productId obtained %d\n", &productId);
        section = productId[5:4];
        subSection = productId[3:2];
        shelf = productId[1:0];
        $display("%b %b %b sec subsec shelf pre", &section, &subSection, &shelf);
      end
    //   $display("%d %d",&Enable, &Enter3);
      if(Enable == 1'b1 & Enter3 ==1'b1)
      begin
        $display("%d %d %d sec subsec shelf", &section, &subSection, &shelf);
        $display("inside main\n");
        addItems = DomItems;
        subItems = SubItems;
        underflow = 0;
```

```
    overflow = 0;
    cnt = 0;
    ////////////////////////////////MAIN////////////////////////////////////////////////////
  //    for(i = 0; i < 5; i = i+1) selected[i] = data[section][subSection][shelf][i];
    //////////Add//////////
    if(addItems > 0) begin
      for(i = 0; i < 3; i = i+1) begin
        if(data[section][subSection][shelf][i] < 10) begin
          if(data[section][subSection][shelf][i] + addItems > 10) begin
            addItems = addItems - (10 - data[section][subSection][shelf][i]);
            data[section][subSection][shelf][i] = 10;
          end
          else begin
            data[section][subSection][shelf][i]= data[section][subSection][shelf][i] +
addItems;
            addItems = 0;
          end
        end
        $display("selected add: %d | ", &data[section][subSection][shelf][i]);
      end
      if(addItems > 0) begin
        $display("No more space left in the shelf!\n");
        overflow = 1;
      end
    end
    ////////Sub//////////
    if(subItems > 0) begin
      for(i = 0; i < 3; i = i+1) begin
        if(data[section][subSection][shelf][i] > 0) begin
          if(data[section][subSection][shelf][i] - subItems < 0) begin
            subItems = subItems - data[section][subSection][shelf][i];
            data[section][subSection][shelf][i] = 0;
          end
```

```
            else begin
               data[section][subSection][shelf][i] = data[section][subSection][shelf][i] -
subItems;
               subItems = 0;
            end
         end
         $display("selected sub: %d | ", &data[section][subSection][shelf][i]);
      end
      if(subItems > 0) begin
         $display("No more Items left to remove!\n");
         underflow = 1;
      end
   end
   /////////final?/////////
   Total = 0; t = 0;
   for(l=0;l<3;l=l+1) begin
      for(k=0;k<3;k=k+1) begin
         for(j=0;j<3;j=j+1) begin
            for(i=0;i<3;i=i+1) begin
               Total = Total + data[l][k][j][i];
               t = t + data[l][k][j][i];
            end
         end
      end
   end
   totalItems = 0;
   for(i = 0; i < 3; i = i+1) totalItems = totalItems + data[section][subSection][shelf][i];
   shelfTot = totalItems;
  //    for(i = 0; i < 5; i = i+1) data[section][subSection][shelf][i] = selected[i];
  end
end
else begin
shelfTot = 0; Total = 0; t = 0;
```

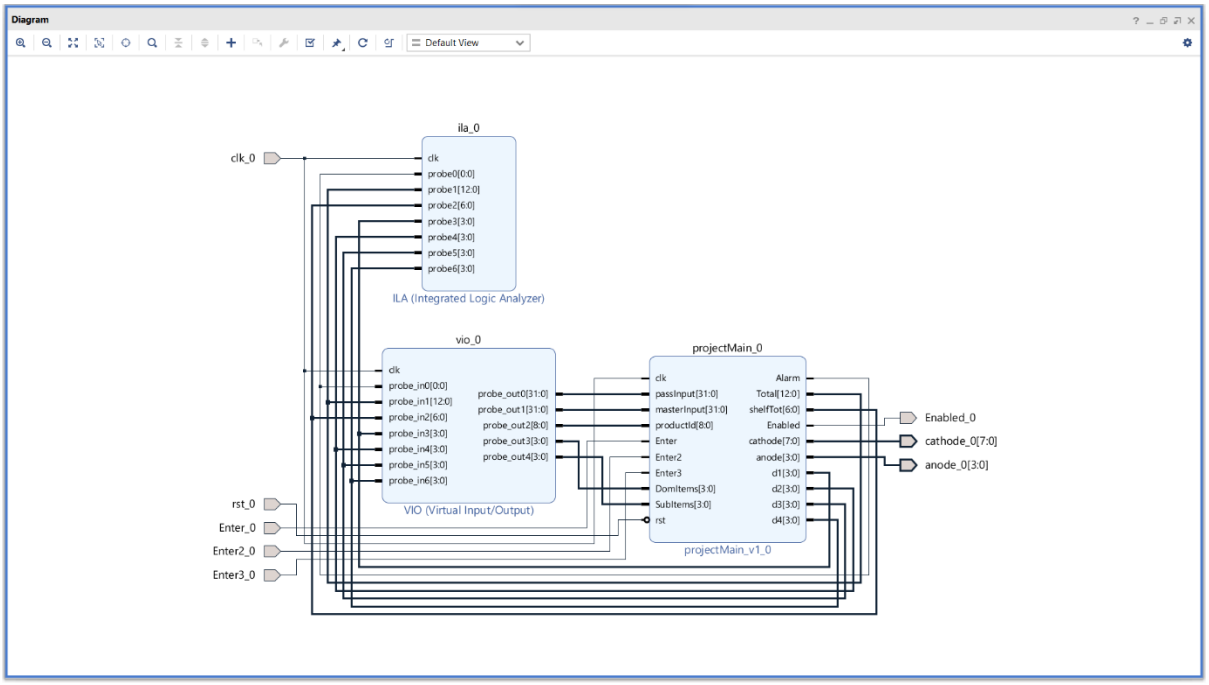
```
for(l=0;l<3;l=l+1) begin
   for(k=0;k<3;k=k+1) begin
      for(j=0;j<3;j=j+1) begin
         for(i=0;i<3;i=i+1) data[l][k][j][i] = 0;
      end
   end
end
end
Enabled = Enable;
//$display("%d", &t);
d1 = t % 10;
d2 = (t/10)%10;
d3 = (t/100)%10;
d4 = (t/1000)%10;
end
//////////////////////////////////////////////////////////////////////////////////
converter conv(d1, d2, d3, d4, clk, anode, cathode);
endmodule
//////////////////////////////////////////////////////////////////////////////////
module converter(input[3:0] d1, d2, d3, d4, input clk, output wire [3:0] anode, output wire
[7:0] cathode);
wire div;
wire [1:0] ref_counter;
clock_divider cd(clk, div);
refreshcounter rc(div, ref_counter);
anode_control ac(ref_counter, anode);
wire [3:0] ONE_Dig;
BCD_control bc(d1, d2, d3, d4, ref_counter, ONE_Dig);
BCD_to_Cathodes btc(ONE_Dig,cathode);
endmodule
//////////////////////////////////////////////////////////////////////////////////
module refreshcounter(input refresh_clock, output reg [1:0] refreshcounter = 0);
always @ (posedge refresh_clock) refreshcounter <= refreshcounter + 1;
```

```
endmodule
//////////////////////////////////////////////////////////////////////////////////
module anode_control (input [1:0] refreshcounter, output reg [3:0] anode = 0);
always@ (refreshcounter)
begin
case (refreshcounter)
2'b00: anode = 4'b1110;
2'b01: anode = 4'b1101;
2'b10: anode = 4'b1011;
2'b11: anode = 4'b0111;
endcase
end
endmodule
//////////////////////////////////////////////////////////////////////////////////
module BCD_control(
input [3:0] digitl, //right digit // ones
input [3:0] digit2, // tens
input [3:0] digit3, // hundreds
input [3:0] digit4, // thousands
input [1:0] refreshcounter,
output reg [3:0] ONE_DIGIT = 0
);

always@ (refreshcounter)
begin
case (refreshcounter)
2'd0: ONE_DIGIT = digitl; // digit 1 ON
2'd1: ONE_DIGIT = digit2; // digit 2 ON
2'd2: ONE_DIGIT = digit3; // digit 3 ON
2'd3: ONE_DIGIT = digit4; // digit 4 ON
endcase
end
```

```
endmodule
//////////////////////////////////////////////////////////////////////////////////
module BCD_to_Cathodes ( input [3:0] digit, output reg [7:0] a_to_g =0);
always@ (digit)
begin
case (digit)
   0: a_to_g = 8'b10000001;
   1: a_to_g = 8'b11001111;
   2: a_to_g = 8'b10010010;
   3: a_to_g = 8'b10000110;
   4: a_to_g = 8'b11001100;
   5: a_to_g = 8'b10100100;
   6: a_to_g = 8'b10100000;
   7: a_to_g = 8'b10001111;
   8: a_to_g = 8'b10000000;
   9: a_to_g = 8'b10000100;
   default: a_to_g = 8'b11111111;
endcase
end
endmodule
//////////////////////////////////////////////////////////////////
module clock_divider(input clk, output reg div);
integer cnt = 0;
initial div = 0;
always @ (posedge clk) begin
   if(cnt == 9999) begin
      div = 1 ^ div;
      cnt = 0;
   end
   else cnt = cnt + 1;
end
endmodule
```

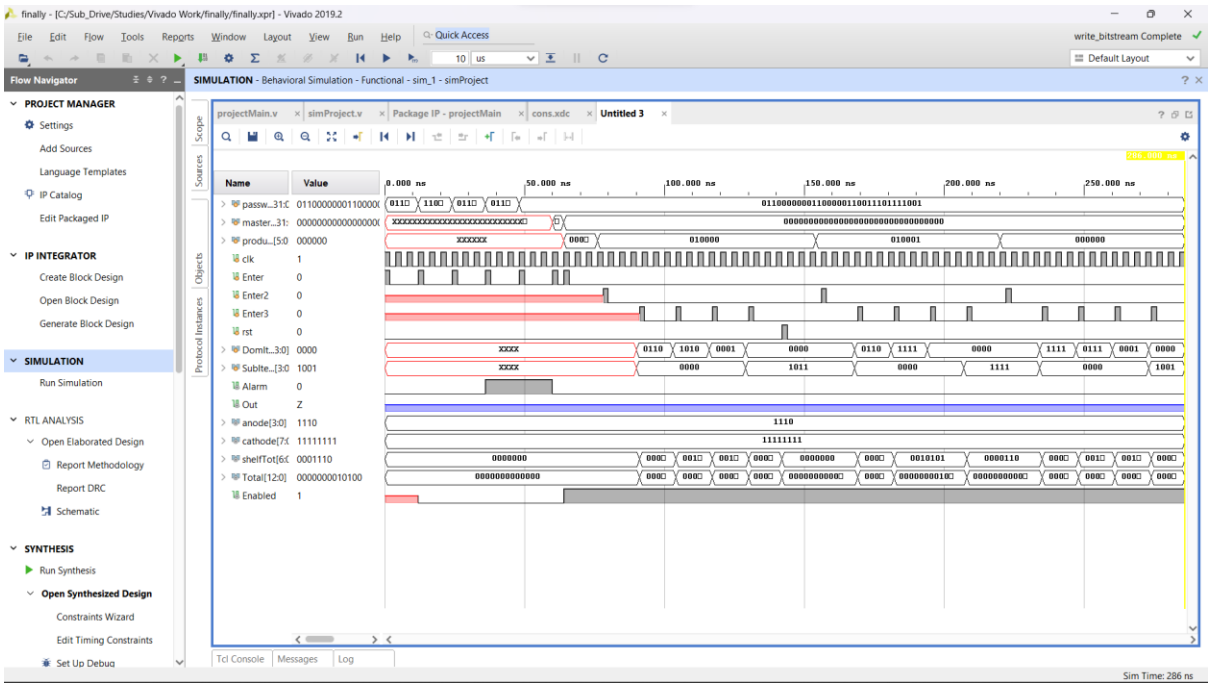
# Block Diagram:




# Simulation Sources:

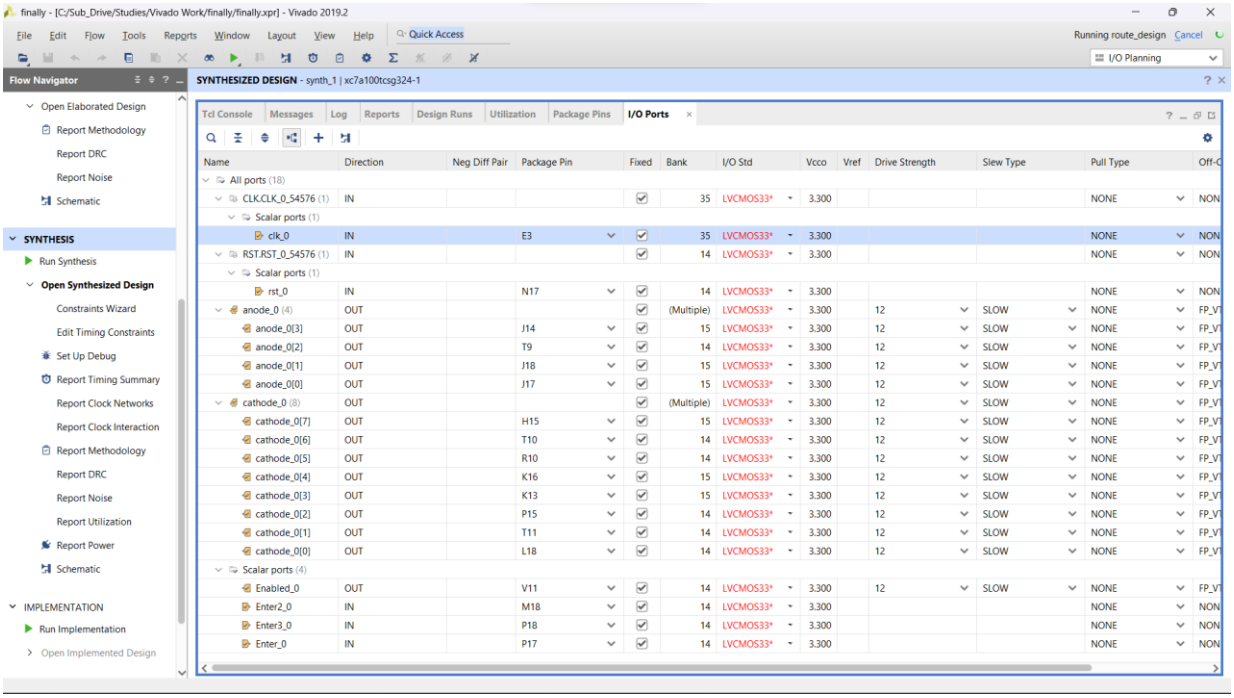
```
module simProject();
reg [31:0] password,masterInput;
reg [5:0] productId ;
reg clk;
reg Enter, Enter2, Enter3, rst;
reg [3:0] DomItems, SubItems;
wire Alarm, Out;
wire [3:0] anode;
wire [7:0] cathode;
wire [6:0] shelfTot;
wire [12:0] Total;
wire Enabled;
projectMain UUT(clk, password, masterInput, productId, Enter, Enter2, Enter3, DomItems, SubItems, rst, Alarm, Total, shelfTot, Enabled, cathode, anode);
always #2 clk = ~clk; initial begin
rst = 0; clk = 1;
password = 32'b01101000001100000110011101111011; Enter = 1'b1; #2 Enter = 1'b0;
#10 password = 32'b01100100001100000110011011110101; Enter = 1'b1; #2 Enter = 1'b0;
#10 password = 32'b01100000001100000110011101111011; Enter = 1'b1; #2 Enter = 1'b0;
#10 password = 32'b01100000000100000110011101111011; Enter = 1'b1; #2 Enter = 1'b0;
#10 password = 32'h60306779; Enter = 1'b1; #2 Enter = 1'b0;
#10 masterInput = 32'h30805262;  Enter = 1'b1; #2 Enter = 1'b0;
#2 password = 32'h60306779;productId = 9'd0 ;masterInput = 32'd0; Enter = 1'b1; #2 Enter = 1'b0;
#10 productId = 6'b010000; #2 Enter2 = 1'b1; #2 Enter2 = 1'b0;
```

```
#10 DomItems = 4'd6; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd10; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd1; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd0; SubItems = 4'd11; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 rst = 1'b1; #2 rst = 1'b0;
#10 productId = 6'b010001; #2 Enter2 = 1'b1; #2 Enter2 = 1'b0;
#10 DomItems = 4'd6; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd15; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd0; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd0; SubItems = 4'd15; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 productId = 6'b000000; #2 Enter2 = 1'b1; #2 Enter2 = 1'b0;
#10 DomItems = 4'd15; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd7; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd1; SubItems = 4'd0; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 DomItems = 4'd0; SubItems = 4'd9; #1 Enter3 = 1'b1; #2 Enter3 = 1'b0;
#10 $finish;
end
endmodule
```

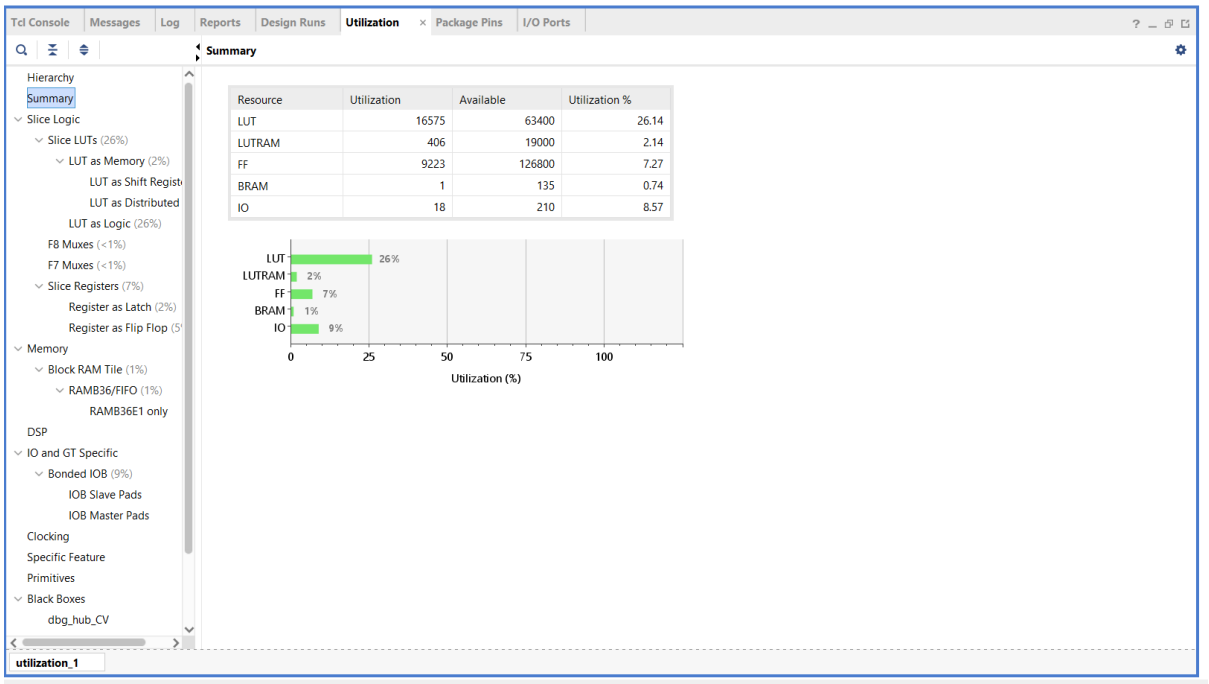



Simulation Waveforms

# I/O Ports Planning:




# Utilisation Report:



## Modules Explanations

# PASSWORD CHECKER

For a warehouse or a storage unit should maintain its integrity and security, the access should be limited. Manipulation or altering the items in the storage unit is the utmost priority while keeping the warehouse safe. So, to equip an efficient security system, we have thought to use 8-digit password in decimal number system. Now, the input is entered into the user interface. Since the password length is of 8 digits, bit size equals to 32 bits.

Once the password is entered, algorithm checks whether it is correct. If it is correct, access to storage organiser is availed. If the entered password is wrong, the user gets to have three attempts to correctly enter the right password. Once the count goes three, and user hasn't entered the right password yet, the respective authorities are alerted accordingly and an alarm goes off. After this stage, password can't be entered anymore and to override and acquire access to the system, one has to enter a Master Password.

If the entered Master Password is still wrong, alarm will be still going on. Until and unless the entered Master Password is correct, the alarm will be still going on, and access isn't given to the user. Once, the Master Password entered is correct, alarm will stop and enable is given for the next implementation of the design. Now, access is availed and a user can continue with further operations in the storage unit.

The input is given and it is converted into binary format. This top module includes a module which acts as a decoder based on rows and columns and accordingly it decodes the given input value. This is converted into binary format which then is given as final password/master password. Starting from MSB, the first four bits tell us about the row. The next four bits tell us about the column, and the last bit the end bit.

# PRODUCT ID ENCODER

The warehouse has a capacity to maintain and withhold a humongous capacity of products. To properly organize all such unique products with each to its own quantity, each product is allotted a designated location in the warehouse and a 9 Bit Unique Code in Binary Number System.

The warehouse we are handling with contains one floors having 5 Sections, with each Section containing 5 Sub Sections. In each of the 5 Sub Sections a total of 5 shelfs are present. This gives us the location of the designated shelf of the unique product ID entered.

From the encoded 9 Bit Product ID, the first 3 bits (starting from MSB) are allotted to the Section. The next 3 bits after section's are allotted to the Sub Section in a certain Section. The next 3 bits gives us the Shelf of the unique product ID.

# Main ALU

## Theory:

This is the brain of the entire Inventory Manager. This module handles input output of the passwords, product ID, number of items to be added or subtracted and the seven-segment display control. The number of items in each container is stored inside the data register and the values can be altered by giving the unit the product id and the number of items to be added or removed.

This module tells the user whether there is enough space to add the items or if there are enough items to be removed. This will be notified in the terminal. The next thing the module tells is the total size of the total inventory, which will be displayed on the seven-segment display. This is done by converting the number into 4 digits and scanning them one by one at a frequency of 10kHz. High frequency tricks our brain to think that the number is actually static and an immediate response is obtained.

## Observations:

Observing the waveforms here, since the first password is entered correctly, i.e., (60306779) access is allotted and management of storage unit is equipped. In the second input, (74518051) password in entered incorrectly so the access isn't granted yet. A user only has 3 attempts to get it correct. Similarly in third (15511220) and fourth (61051797) inputs are entered incorrectly.

Hence, to maintain integrity of the system and not to compromise the security, further attempts aren't permitted. Since there is someone who isn't authorised to access the system, the necessary members are Alarmed. Unless a Master Password is entered which is correct and verified (30805262), access to the management system is not allotted. Once the correct Master Password is entered, the Alarmed is Turned OFF and count is reset back to zero

Observing the waveforms generated above, we can see that from input 00100111100001000111, the first four bits 0010 is the section value, the next four bits 0111 is the sub section in its respective section, the further four bits are 1000 which denote the shelf corresponding to. After these the next 7 bits which gives us the days of storage given by: 0100011 and necessary floor is allotted accordingly. The last bit of the input i.e., unique product ID is priority bit which will be further discussed in the next module and it is used to allot the containers in the shelf accordingly.

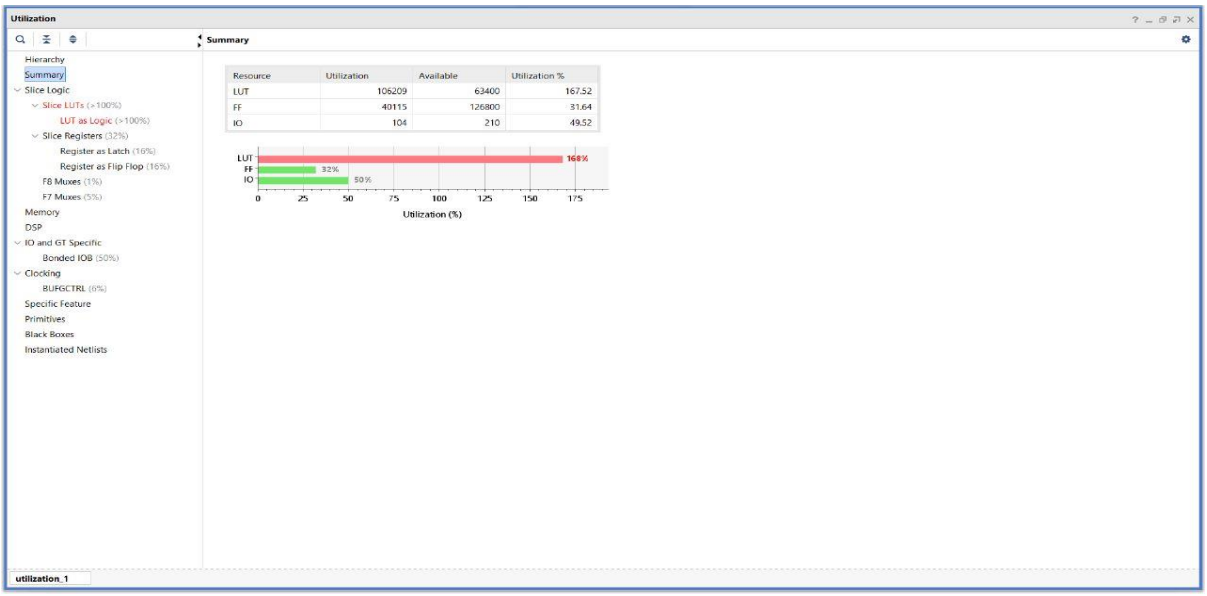
Similarly, the next two product ID given: (01000101100000000100), (10010011100001000110) also perform as expected giving their designated locations correctly. Hence the product ID encoder module is appropriately performing its operations.

# **Conclusion:**

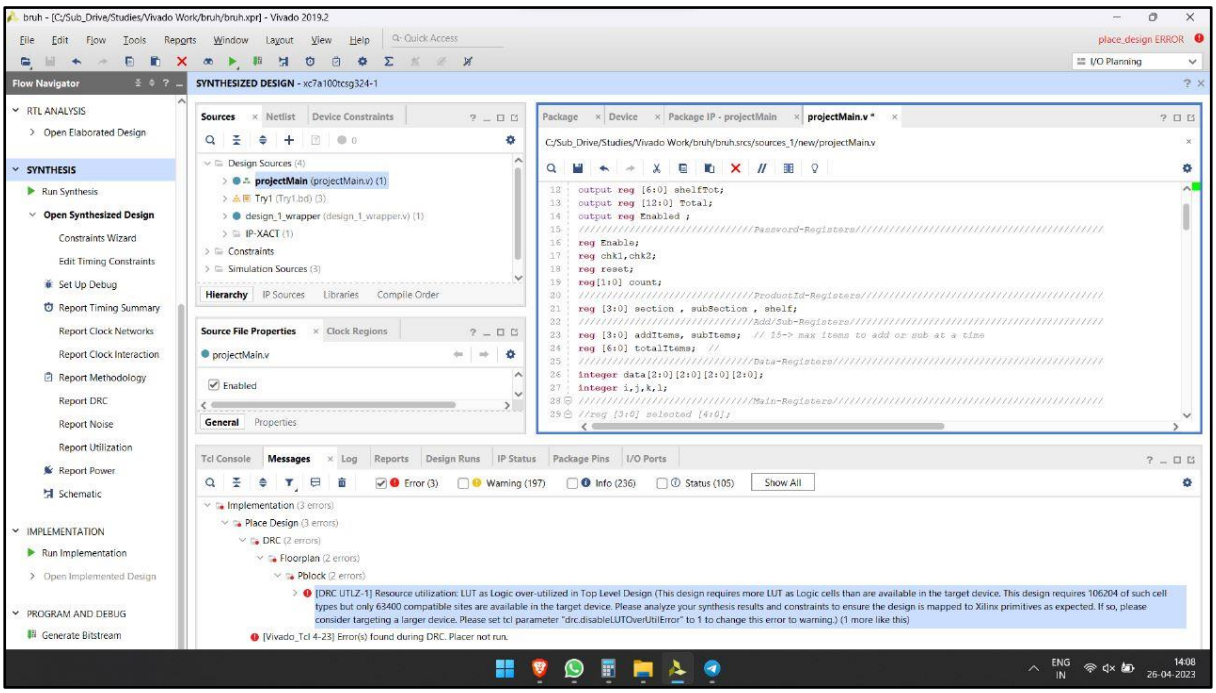
## FPGA Limitations:

FPGA Artix – 7 ADDR (XC7A100TCSG324-1)

This hardware which was used during implementation of the main project module doesn't meet our requirements as per the number of LUTs available. Our project module requires 68% more LUTs Than those available with the Board FPGA Artix – 7 ADDR (XC7A100TCSG324-1), due the complex logic, hence we had to scale down our module logic in order to implement on




FPGA Artix – 7 ADDR (XC7A100TCSG324-1).



Hence, we can conclude that the project has been successfully implemented with all of the aims achieved with maximum accuracy. The waveforms obtained are consistent with the expected results and the simulations are accordingly intact. The schematic diagrams are in consistent with modules. Even though all the modules are used as individuals, all the three modules can be cascaded to perform operations as a whole singular module.

The password checker module works as expected and performs its operations accordingly. The product ID Encoder gives the location as expected and the third module ROMLINK grants access to the units of the unique product ID from which the priority of the products can be accordingly adjusted and addition or subtraction of the products from the warehouse can be performed accordingly.