Drinks Vending Machine

Team 01
07.07.2023

Team members

- Jaouaher Belgacem
- Jasmeet Singh
- Evrard Leuteu Feukeu

Introduction

This project outlines the development of vending machines for drinks with the use of FPGAs (Field-Programmable Gate Arrays) and VHDL (VHSIC Hardware Description Language). A vending machine is a device designed to dispense drinks or other products automatically when users insert coins or tokens into the machine.

FPGAs are integrated circuits that can be programmed after manufacturing to perform a specific function, making them highly customizable. VHDL is a hardware description language used to design digital circuits. Together, FPGAs and VHDL provide a powerful toolset for designing complex digital systems, such as vending machines.

In the case of vending machines for drinks, FPGAs, and VHDL are used to implement the control logic for the machine. This includes tasks such as detecting and verifying the coins or tokens inserted by the user, dispensing the correct drink based on the user's selection, and providing feedback to the user about the status of the machine.

Overall, the flexibility and programmability of FPGAs make them an ideal choice for vending machine design, as they can be tailored to meet the specific requirements of the machine. VHDL is used to describe the behavior of the machine's digital circuits, which can then be implemented in the FPGA.

Concept description

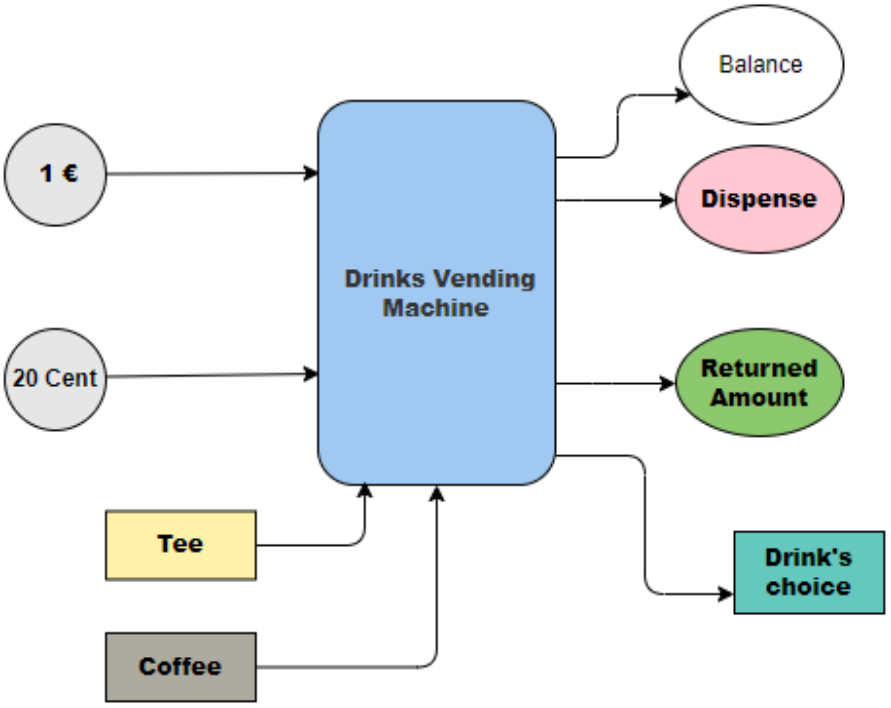
The vending machine concept that we are proposing, consists of inserting specific coins and requesting a drink type, and according to the inserted money, the machine will dispense the requested drink if the balance is enough and will return money in case the inserted money is higher than the drink’s price.
The developed concept of the vending machine accepts only two types of coins which are:

- 20 cents
- 1 €

Moreover, the machine offers only two types of drinks which are:

- Tea: 40 cents
- Coffee: 60 cents

Therefore, for 1€ inserted coin, the user can choose to have one drink or two (only if the chosen drinks are Tea and Coffee).



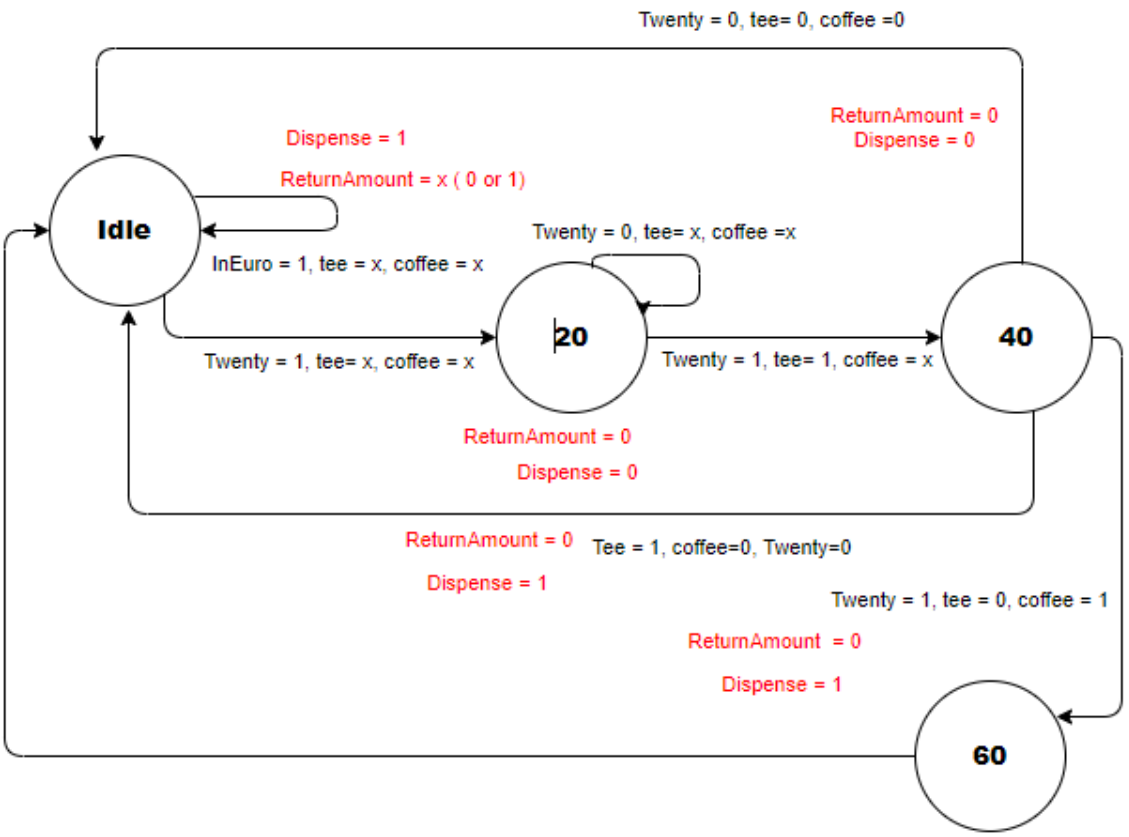
Block Diagram of Vending Machine

As shown in the block diagram, the vending machine has three inputs and two outputs which are respectively, €1, 20 cents, drink's choice and it will result in, dispensing of the drinks and returned amount.
It's important to note that the coin reader of the €1 and 20 Cents are separate.

1. Finite State Machine

The proposed finite state machine diagram of our vending machine consists of four different states which are:

- **Idle**: it is the start state of the system, where it is ready to receive money and requests. Besides it could be achieved by inserting €1 or after reaching the amount of 60 cents. Hence the output of this state is a dispense of one or two drinks and it may or may not outcome a returned money as it depends on the requested drinks (Tea + Coffee)
- **20**: in order to reach this state it depends on whether the inserted coin is equal to 20 or not and in this stage there will be no dispense nor a money return possible as the balance is less than the minimum price required (40 cents for Tea).
- **40**: Similar to the previous state, it's fulfilled, once a 20 cent is inserted. However, the only drink that is available for dispense is a Tee with no return. However, in the case of choosing coffee, both outputs will receive 0. If the user did not insert money or choose any drink then it will be redirected to the idle state automatically and the balance will be then donated.
- **60:** if another 20 cents is inserted, the balance will reach 60 cents. Hence, it will return to an idle state automatically if the coffee is chosen. Thus, the machine will dispense coffee and return no money.



Finite State Machine Diagram

As a result, we chose to use Mealy FSM as the states depend on the input value.

2. Truth table

The truth table of the Finite state machine will be described as follows:

Current State	euroIn	twentyCent	Tea	Coffee	Next state	Dispense	Returned Amount
Idle	0	0	X	X	Idle	0	0
Idle	1	0	1	0	Idle	1	1
Idle	1	0	0	1	Idle	1	1
Idle	1	0	1	1	Idle	1	0
Idle	0	1	X	X	20	0	0
20	0	1	1	0	40	0	0
20	0	1	0	1	40	0	0
20	0	1	1	1	40	0	0
40	0	0	0	0	idle	0	0

40	0	0	1	0	idle	1	0
40	0	1	0	1	60	0	0
60	0	0	0	1	idle	1	0

Project/Team management

As a team, we chose to work with the scrum project management model. As it is easier to break down the life cycle of the project into several iterations. The tasks are gradually assigned to reach the ultimate task of the project. This leads to periodic meetings to check the progress status of the tasks of each team member.
To manage this project we used several tools:

- Github: to upload files
- WhatsApp group: to discuss some points briefly and send meetings information
- Google Meet: to meet and discuss the progress, tasks, and problems we have faced.

Tasks and Roles

The project is divided into three main parts:

- VHDL Code
- TestBench
- FPGA Implementation
- KiCad (schematic, PCB layout, and 3D view)
- Documentation

Every team member worked on a different task:

- **Jaouaher**: developed the VHDL source code of the vending machine and did the FPGA implementation (xdc file setup), earlier integrated I/O + one section of the 7-segment display on FPGA. For the documentation, I wrote the concept (Block diagram + FSM diag and truth table), VHDL implementation, Project management, VHDL, and FPGA implementations.
- **Jasmeet**: Developed the earlier stage concept diagram, input/output for VHDL Code (7-Segment Display), Integration of FSM and Debugging of the VHDL code, and PCB Design. For documentation: PCB and VHDL input/output parts, integration of the FSM with the display, and Future/Further Other ways of implementation
- **Evrard**: Developed the VHDL testbench of the VHDL source code as well as the test cases. For documentation: Introduction of the project method (*FPGAs and VHDL*) and project breakdown.

Technologies

- VHDL
- FPGA
- KiCAD

VHDL and FPGA Implementation

The implementation of the vending machine is divided into two phases:

1. VHDL

The VHDL code is composed of three processes:

- As we need to update the FSM's different states, we used D flipflop to store the data.
- The second process is composed of the logic of the state machine. Where we are setting the different conditions for transitioning from one state to another.
- The last process is about the output itself. where we are assigning the values of the dispense and returned money outputs with respect to all the inputs such as the inserted coin and drinks choice.

2. Testbench

2.1) Description

- **Stimulus Process**

This process is in charge of creating stimulus signals that will test the functionality of the vending machine state machine. The process commences with a wait period of 100ns, after which we initiate a machine reset by setting the reset high. Post another 100ns delay, the reset is released by setting it back to low. All remaining signals, which include euroIn, twentyCent, tee, and coffee, are then set to zero.

- **Test cases**

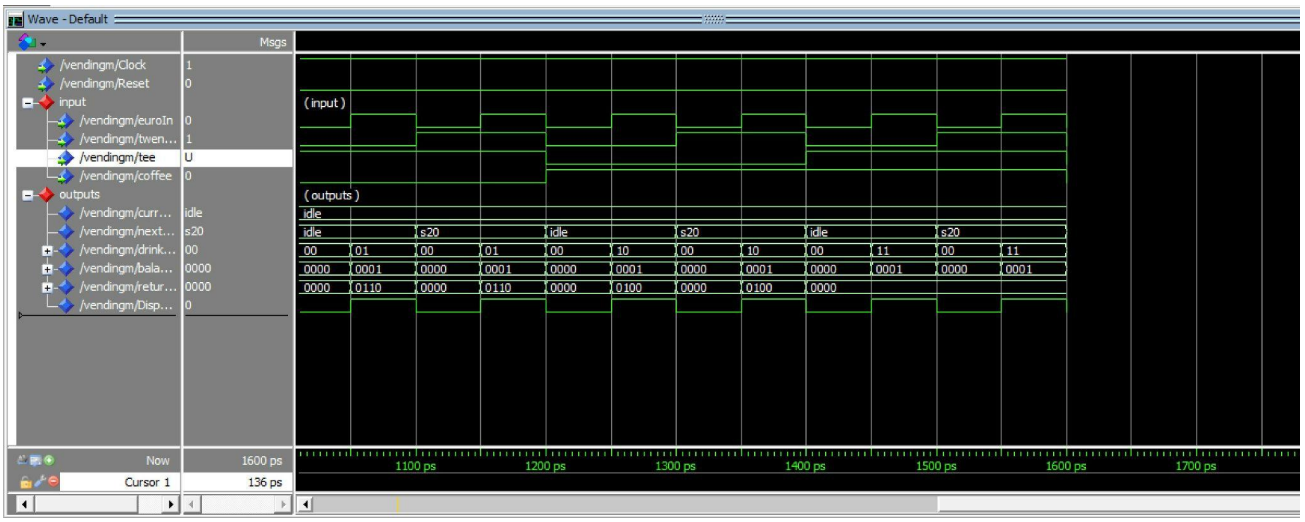
The process continues to define a series of test cases, each with a unique set of input values and a corresponding expected output.

- **End process**

The architecture ends with a wait statement with no condition, resulting in an infinite wait period. This essentially marks the end of our test as there are no further actions to be executed.

In conclusion, this test bench offers a sequence of tests that verify the expected behaviour of your vending machine when faced with varying conditions, such as the insertion of coins and the selection of drinks. This will assist in determining if your design can accurately respond to diverse scenarios.

2.2) Simulation



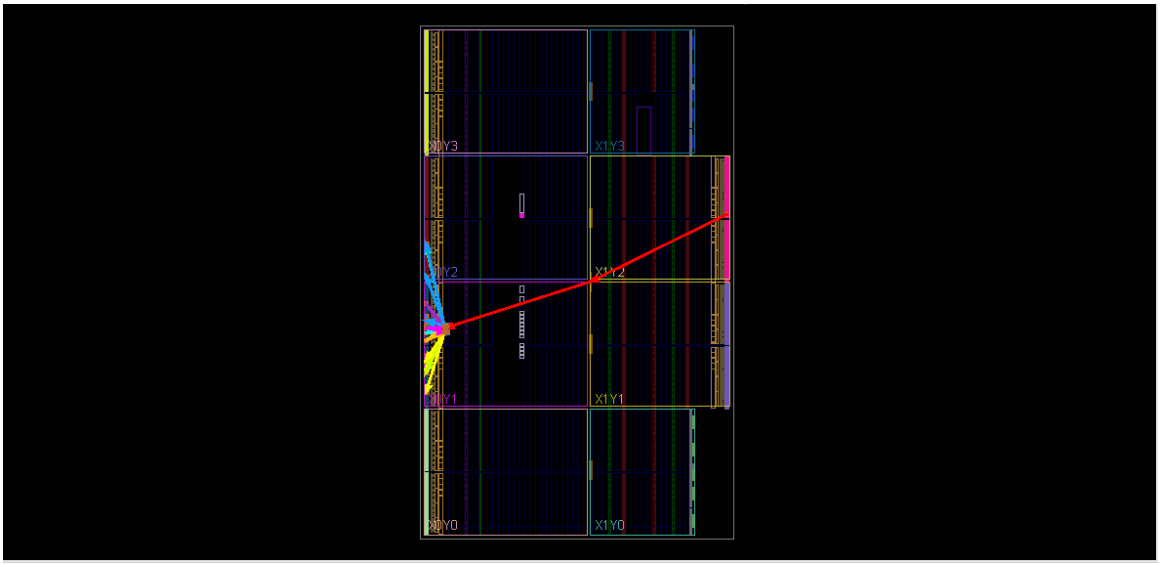
3. VHDL Input/Output

1) The Data flow Design of the 7 Segment Display on the FPGA is different than the regular segment display. on FPGA there are two units of 4-seven segment display attached together therefore they share the same data line on the FPGA evaluation kit making it a little complex to program them.
2) The code takes the input from the FSM and displays it accordingly.

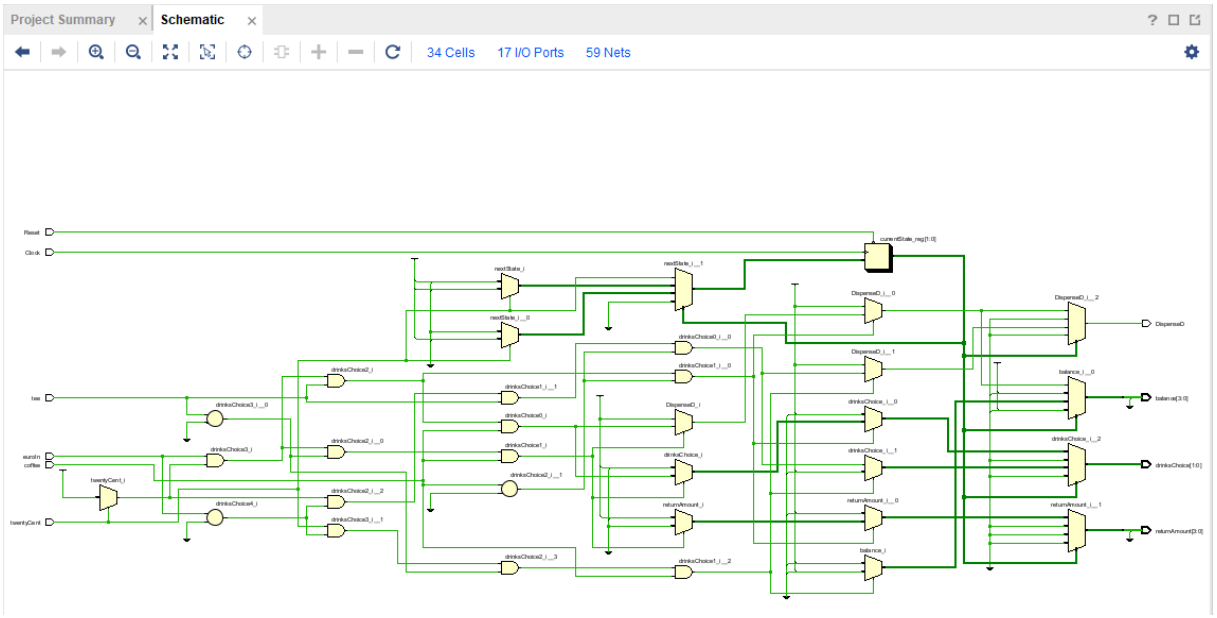
4. Integration of FMS and 7-Segment Display

To integrate the FSM with 7-Segment Display, we decided to have few variable outputs in FSM, such as dispense, which indicates if the selected drink is dispensed, return amount, which indicates the amount to return; and balance, which indicates how much user has added to the machine. As we are using FSM, which changes state using 20 cents as an input, it makes it easier to output the desired result. We have a display process running parallel to FSM; therefore, as soon as there are some changes in the FSM output, one can see them on the Display. On BCD display, we have the availability of eight seven-segment displays from 1 to 7, so we have allocated specific displays for specific purposes such as first and second Seven segments are used to display drink choice as there is an option to select both drinks while using 1 euro, and the third and fourth are used to display the total value of the drinks The fifth is used to display if the selected drinks are dispensing or not; the sixth is no display; the seventh is for inserted amount; and the eighth is for return amount.

5. FPGA



The I/O configuration for the Inputs/Outputs to the FPGA board's pins.



FPGA Schematic

Both FPGA screenshots are representing 6 different output cases where the 7-segment display is used for displaying information and the LEDs for indicating the states of the FSM. This is the I/O port mapping from VHDL inputs/outputs to FPGA pins is represented as follows:

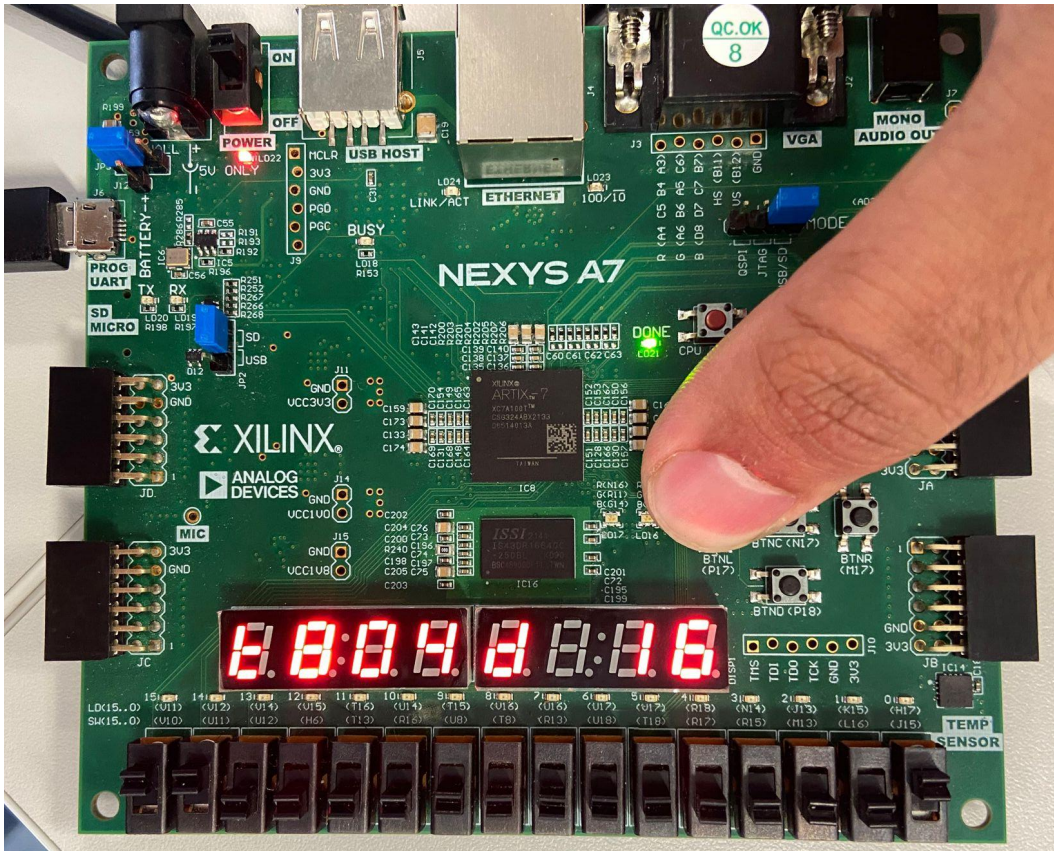
For drinks choice, switches (J15 and L16) are used if:

- 01: Tea is chosen
- 10: Coffee is chosen
- 11: Both are selected

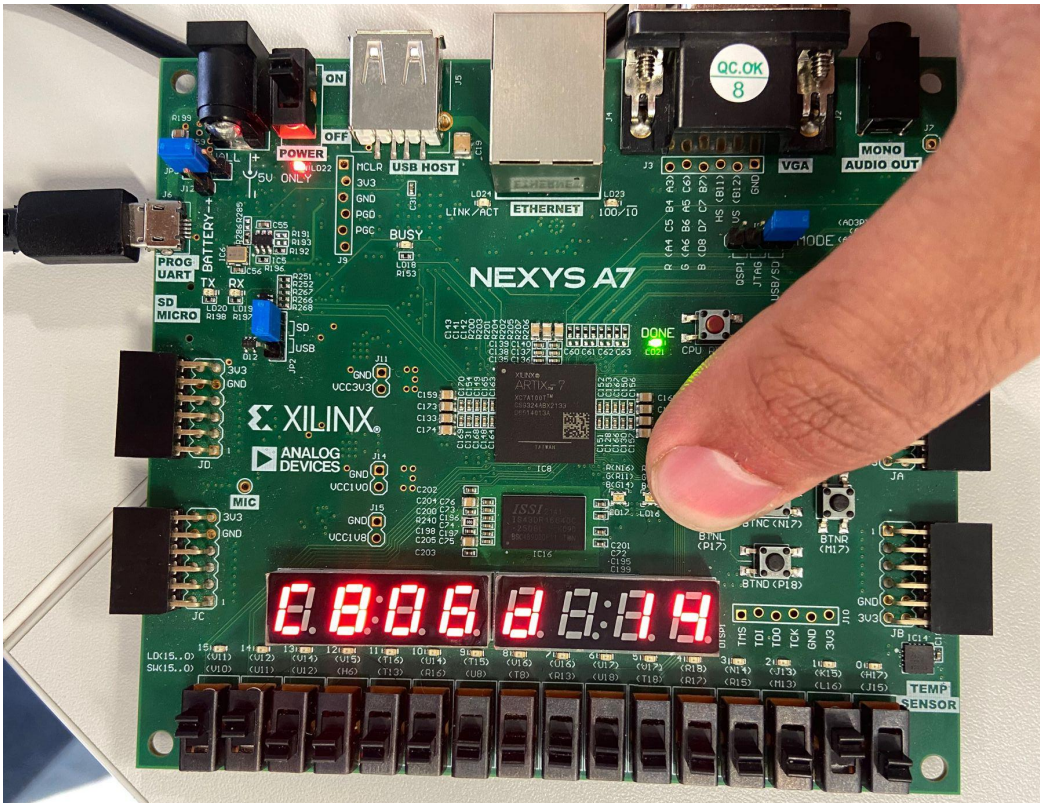
For Inserting money, buttons are used:

- €1: pressing the P17 button
- 20 cents: pressing M17 (each press is equal to 20 cents)

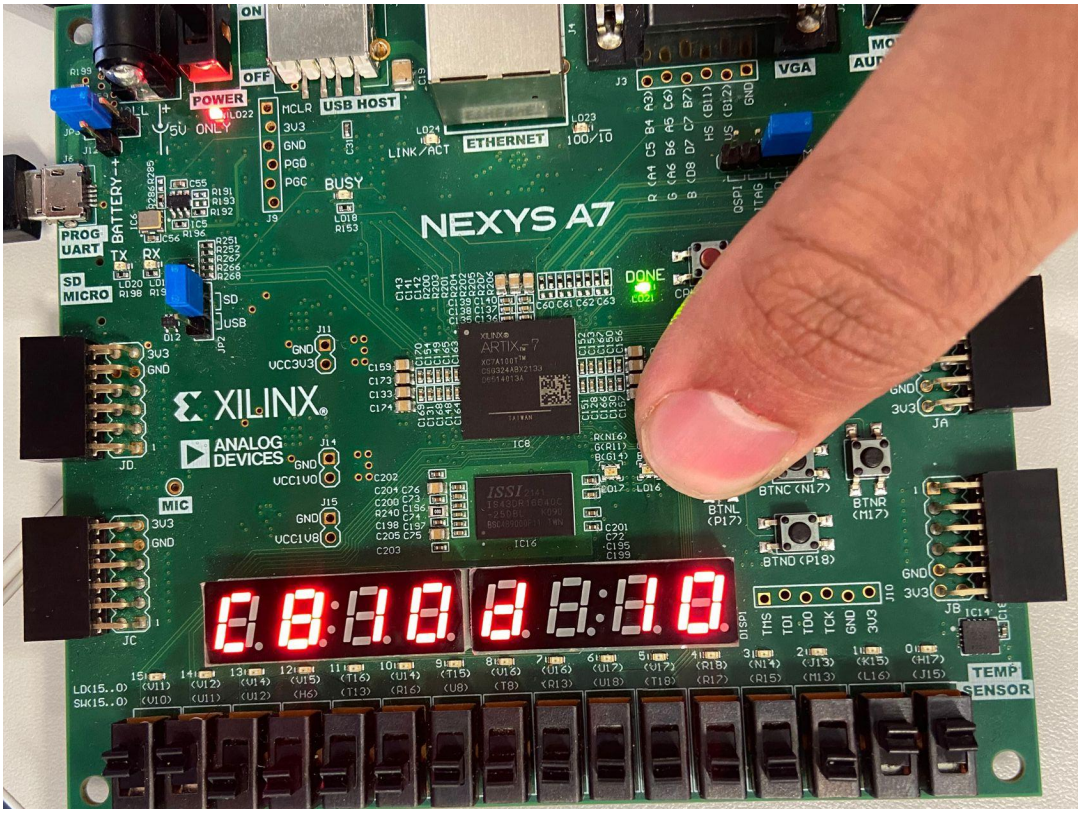
- **Case1**: €1 is inserted, tee is chosen, dispense (d), balance= €1, price =40 cents, Return amount = 60 cents



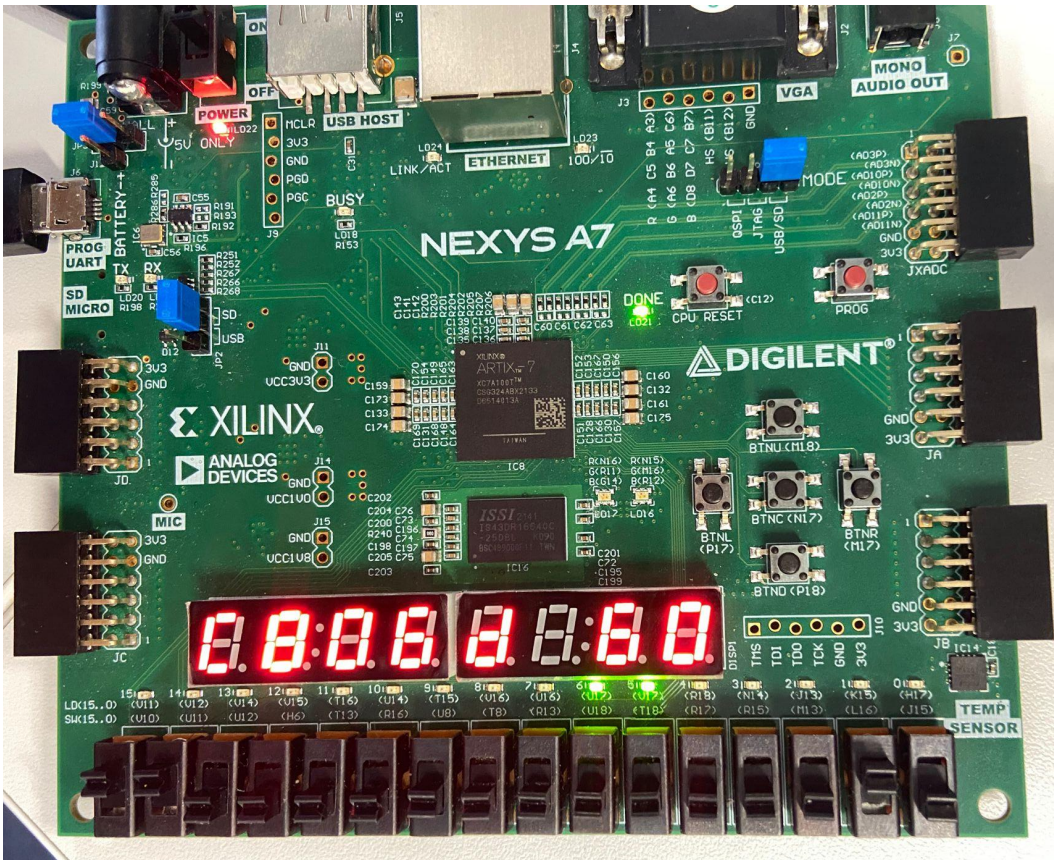
- **Case2**: €1 is inserted, coffee is chosen, dispense (d), balance= €1, price =60 cents, Return amount = 40 cents



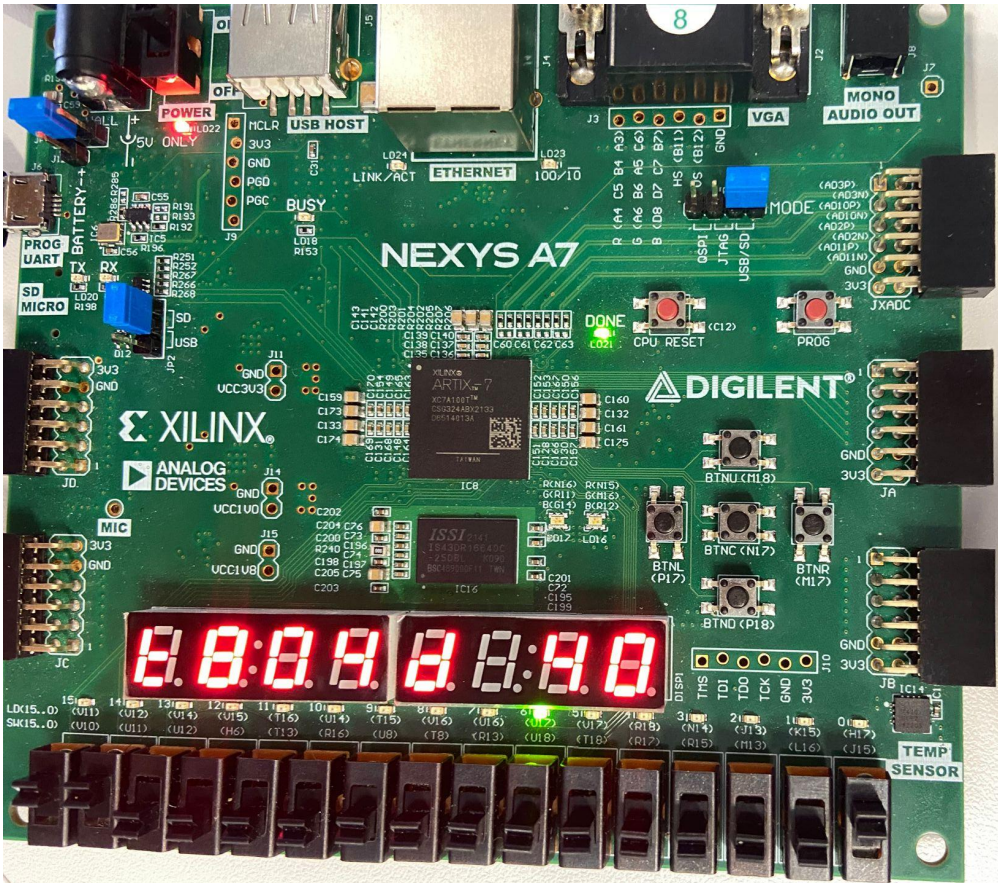
- **Case3**: €1 is inserted, both drinks are chosen, dispense (d), balance= €1, price =€1, Return amount = 0



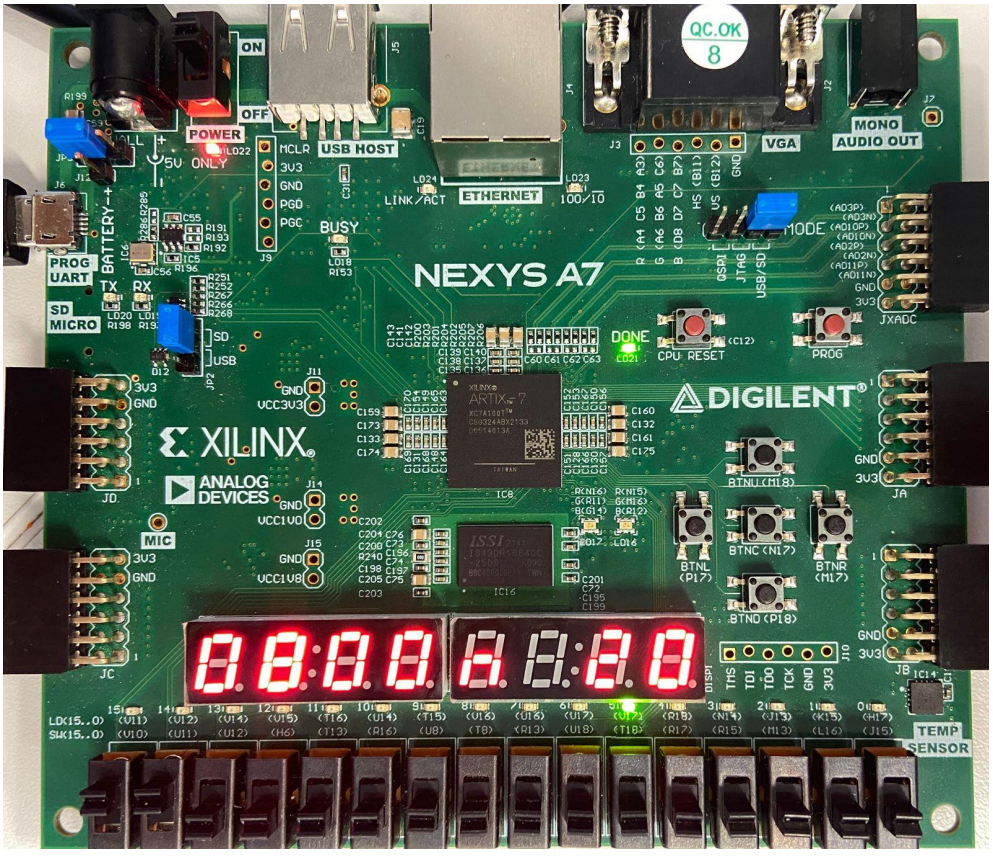
- **Case4**: 60 Cents is inserted, coffee is chosen, dispense (d) , balance= 60 , price =60 cents, Return amount =0



- **Case5**: 40 Cents is inserted, tee is chosen, dispense (d), balance= 40, price =40 cents, Return amount =0



- **Case6**: 20 Cents is inserted, no drink is chosen, no dispense (n), balance= 20, price =0 cents, Return amount =0



Constraint File:

```
set_property IOSTANDARD LVCMOS33 [get_ports {An0[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {An0[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {An0[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {An0[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {An0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {An0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {An0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {An0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[0]}]
set_property DRIVE 12 [get_ports {An0[7]}]
set_property DRIVE 12 [get_ports {An0[6]}]
set_property DRIVE 12 [get_ports {An0[5]}]
set_property DRIVE 12 [get_ports {An0[4]}]
set_property DRIVE 12 [get_ports {An0[3]}]
set_property DRIVE 12 [get_ports {An0[2]}]
set_property DRIVE 12 [get_ports {An0[1]}]
set_property DRIVE 12 [get_ports {An0[0]}]
set_property PACKAGE_PIN U13 [get_ports {An0[7]}]
set_property PACKAGE_PIN K3 [get_ports {An0[6]}]
set_property PACKAGE_PIN T14 [get_ports {An0[5]}]
set_property PACKAGE_PIN P14 [get_ports {An0[4]}]
set_property PACKAGE_PIN J14 [get_ports {An0[3]}]
set_property PACKAGE_PIN T9 [get_ports {An0[2]}]
set_property PACKAGE_PIN J18 [get_ports {An0[1]}]
set_property PACKAGE_PIN J17 [get_ports {An0[0]}]
set_property PACKAGE_PIN T10 [get_ports {Display[7]}]
set_property PACKAGE_PIN R10 [get_ports {Display[6]}]
set_property PACKAGE_PIN K16 [get_ports {Display[5]}]
set_property PACKAGE_PIN K13 [get_ports {Display[4]}]
set_property PACKAGE_PIN P15 [get_ports {Display[3]}]
set_property PACKAGE_PIN T11 [get_ports {Display[2]}]
set_property PACKAGE_PIN L18 [get_ports {Display[1]}]
set_property PACKAGE_PIN H15 [get_ports {Display[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports Clock]
set_property PACKAGE_PIN E3 [get_ports Clock]
set_property PACKAGE_PIN M17 [get_ports twentyCent]
set_property PACKAGE_PIN P17 [get_ports euroIn]
set_property IOSTANDARD LVCMOS33 [get_ports twentyCent]
set_property IOSTANDARD LVCMOS33 [get_ports euroIn]

set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED_idle }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { LED_s20 }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED_s40 }]; #IO_L14N_T2_SRCC_14 Sch=led[9]

set_property PACKAGE_PIN R15 [get_ports Reset]
set_property IOSTANDARD LVCMOS33 [get_ports Reset]

set_property PACKAGE_PIN L16 [get_ports coffee]
set_property PACKAGE_PIN J15 [get_ports tee]
set_property IOSTANDARD LVCMOS33 [get_ports coffee]
set_property IOSTANDARD LVCMOS33 [get_ports tee]
```

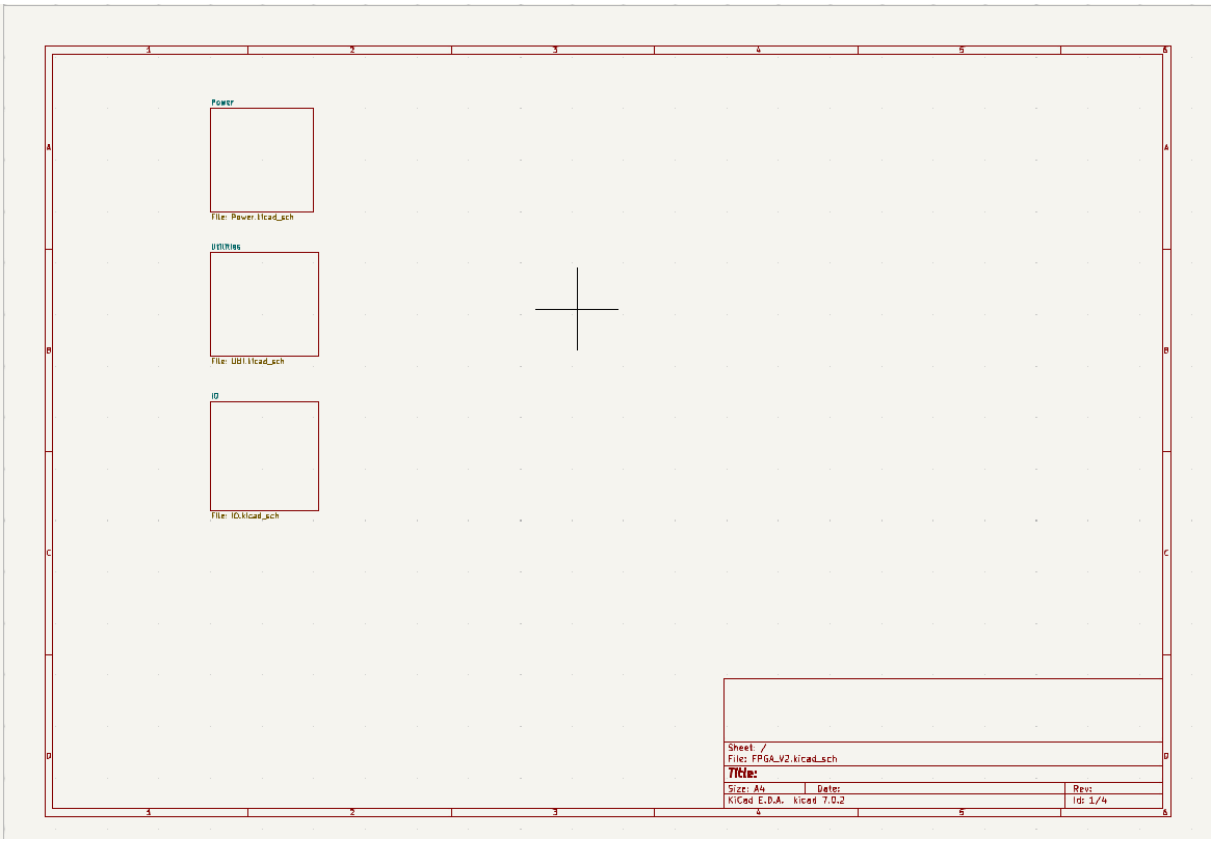
PCB Design

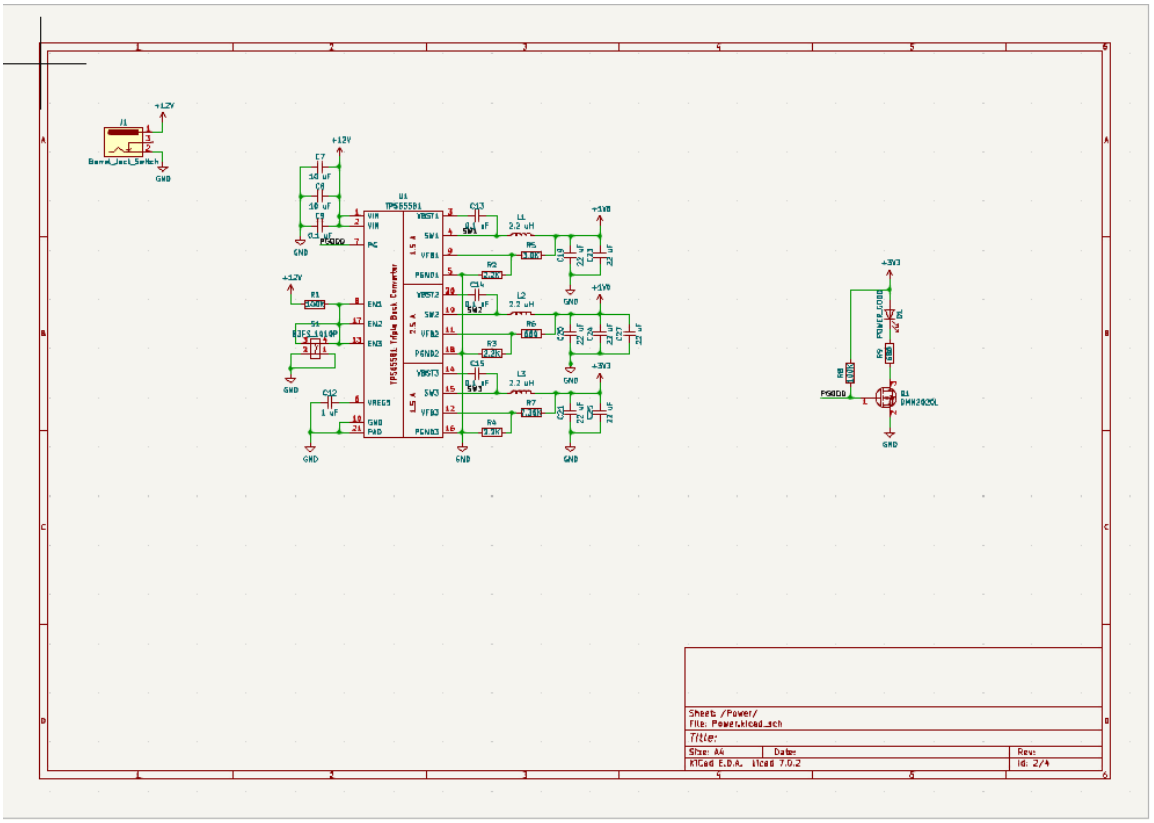
For PCB Design - Initially, Nexys A7 FPGA was used to create the PCB solution; however, due to the complexity of the FPGA and the simplicity of the project, the design was shifted to use Spartan 7 FPGA; however, the unfinished FPGA design for Nexys 7 is available on GitHub.

1) Schematic

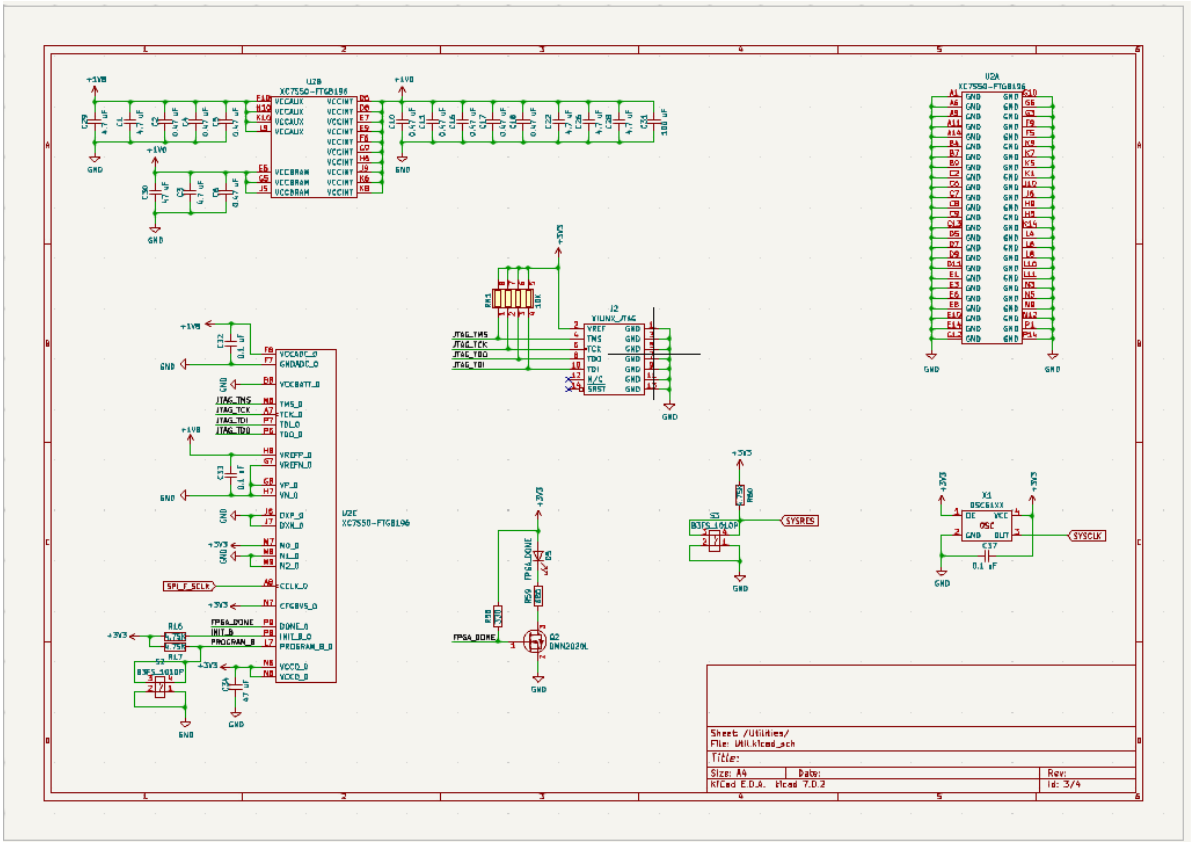
a) Power Schematic for providing power to the component on the PCB board, In the power schematic - a buck converter is used as different components take different voltages, and sometimes the same component needs different voltages, such as the FPGA and input for power.
b) Utility Schematic for connecting different
c) I/O schematic for Input and output
d) https://forum.kicad.info/t/erc-failed-to-read-simulation-model-from-fields-kicad-v7-0-2/41781 There is an error with the current version of Kicad that may result in problems while using auto-routing.

The root of the Schematic

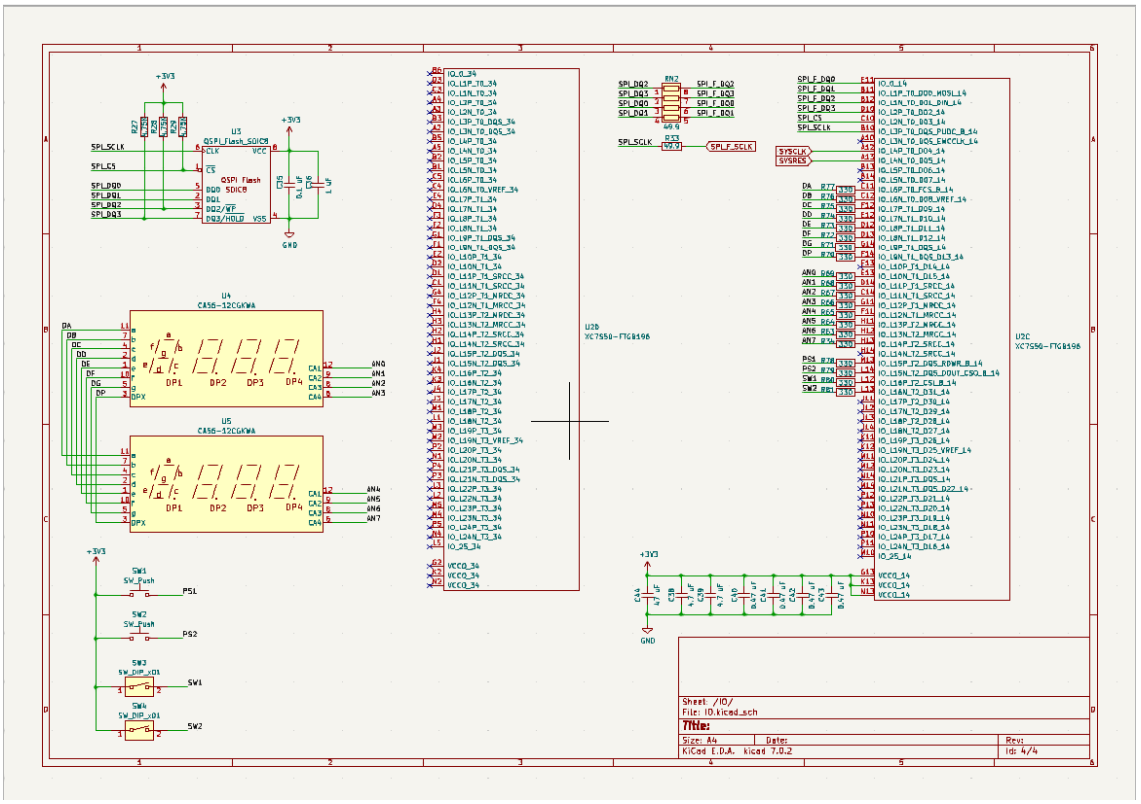




Power Management

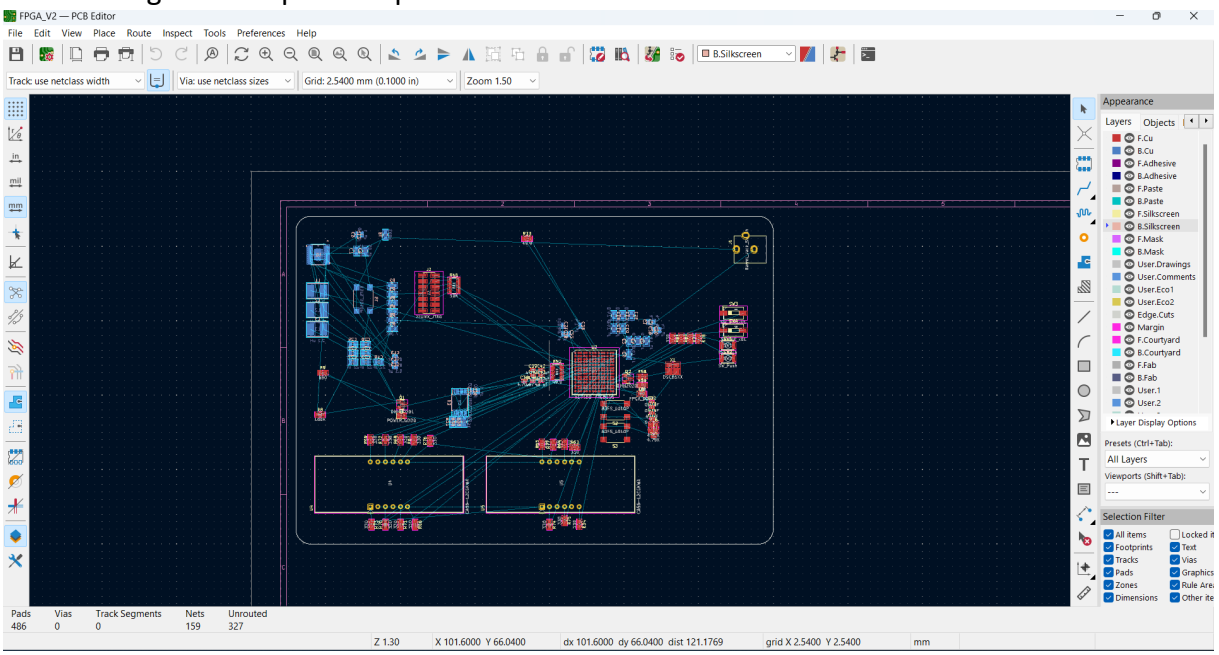


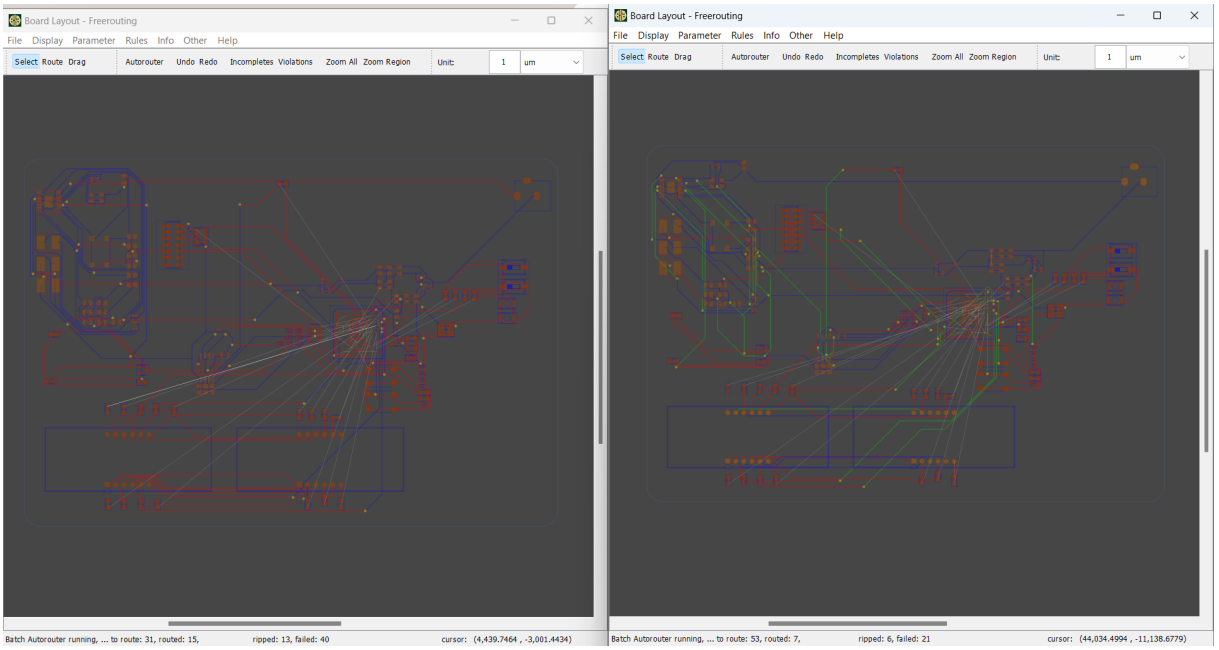
Utility management



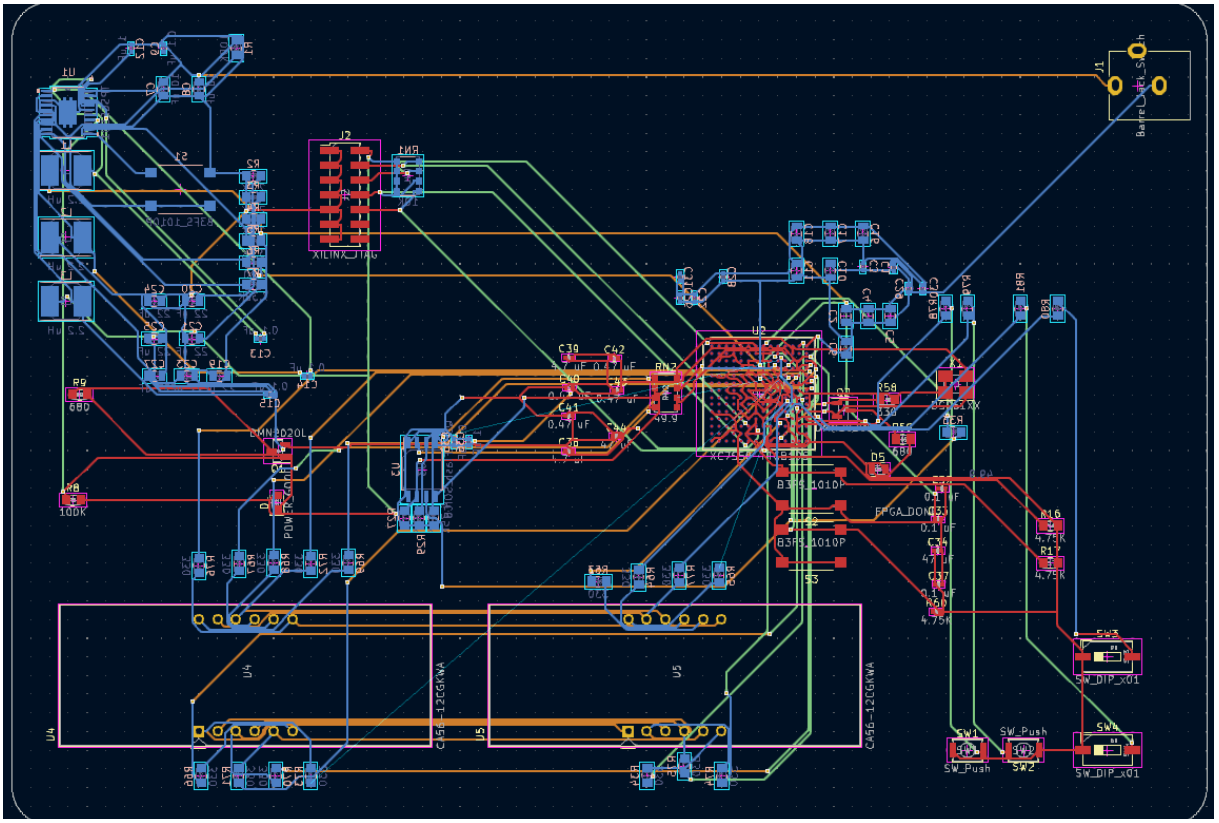
IO connected to the IC

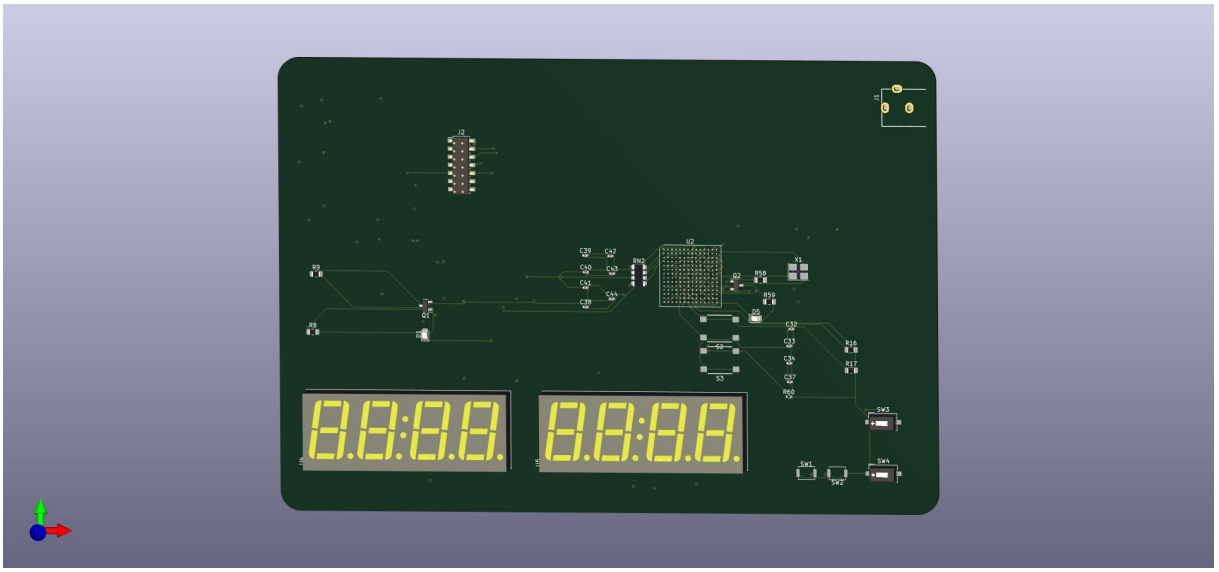
AutoRouting and components placement

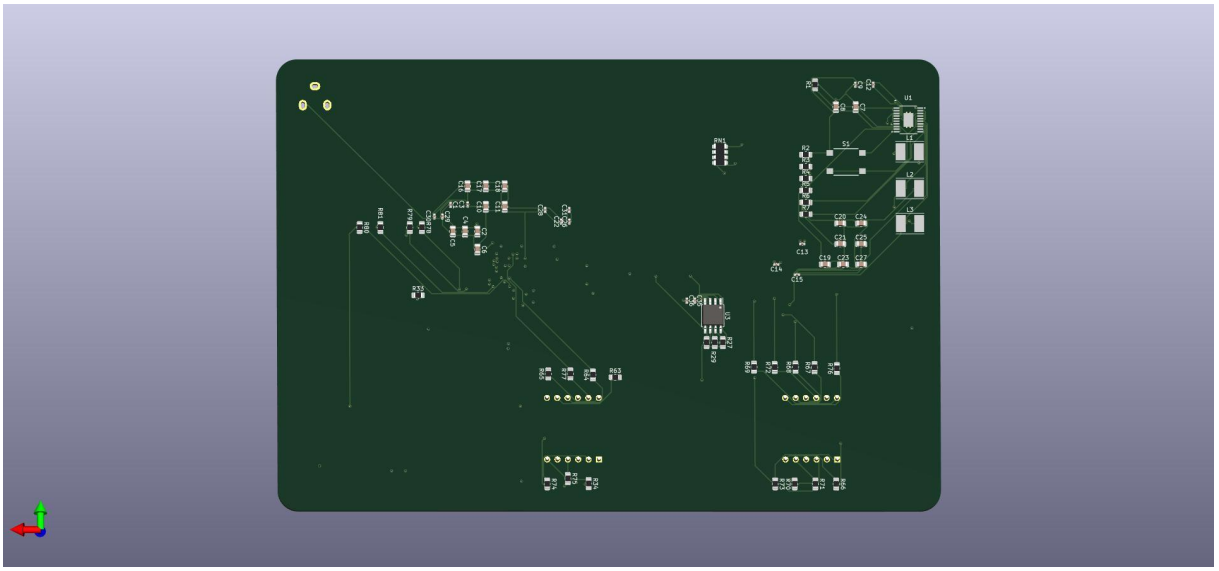




After Completion of FreeRouting 4 Layer



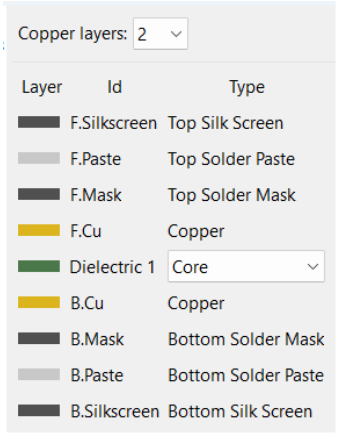




PCB Layer

For the following schematics and design, we had two options for routing either to go with the basic two layers or to have more layers. but we decided to do auto-routing with both options available and to compare the results; therefore, for a four-layer system, two layers have mixed properties and two layers have signal properties for autorouting to route the track.

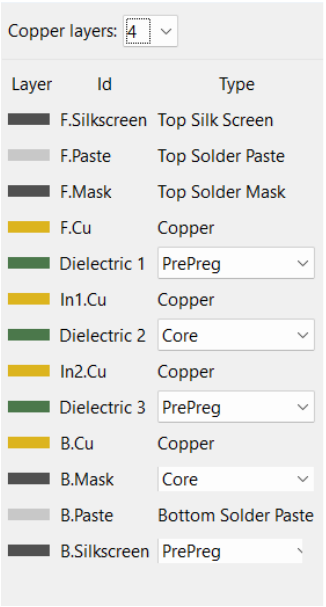
2 Copper Layer



Both Front copper and Back Copper are used for signals.

F.cu	Signals
B.cu	Signals

4 Copper Layer



Here all F.cu and B.cu i.e. front and back copper layers are for signal and internal copper layers 1 and 2 are for mixed categories i.e. they can be ground, Vcc, or signals, etc.

F.cu	Signals
In1.cu	Mixed
In2.cu	Mixed

B.cu	Signals

Another Way of Implementation

In this Section, we will talk about different ways of implementing the same system, such as creating a system that works completely on Input without the use of FSM.
One way to create such a system is to use switches for tea and coffee, and we can have dedicated switches for money. For example, there can be four switches representing 10 cents, 20 cents, 50 cents, and 1 euro as the currency we can see in the market. As for initial development, we don't need memory; we can start with only having a fixed number of inputs available, which can be further upgraded with push buttons and memory; we will talk about it later.
Let's start talking about a fixed number of inputs, so in total we have six switches working together. We can also add one push button as an enter, such as when the user has added the amount he needs to add using switches, he/she can press the button that executes the task for checking the drink choice, available money, and required money. Depending on this, the system can either dispatch drinks with or without a return or just return the money. In this scenario, one will use switch cases in one process that executes during the pressing of the enter button, and there will be one process for the output that is displayed on it The system can either dispatch drinks with or without return, or just return money. In this scenario, one will use switch cases in one process that executes during the pressing of enter button, and there will be one process for the output that is displayed on the BCD array.
In such an implementation, one has to put switches back to their original positions as a reset, which is not implementation worthy but one can use four buttons for the input of the amount, but what if the user presses the same button again what about the extra input? As mentioned earlier, this system can only handle a fixed number of inputs. For such an implementation, one needs to do an arithmetic implementation with memory. We could not implement this way due to the time limitation of the available hardware, but it still sounds fun to implement a vending machine that mimics the real-time scenario where there can be n number of drinks and n number of inputs in terms of money.

Sources/References

https://github.com/JaouaherBelgacem/Hw_and_AES-projects/tree/main/AES%20Project