Drinks Vending Machine

Team 01

16.06.2023

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

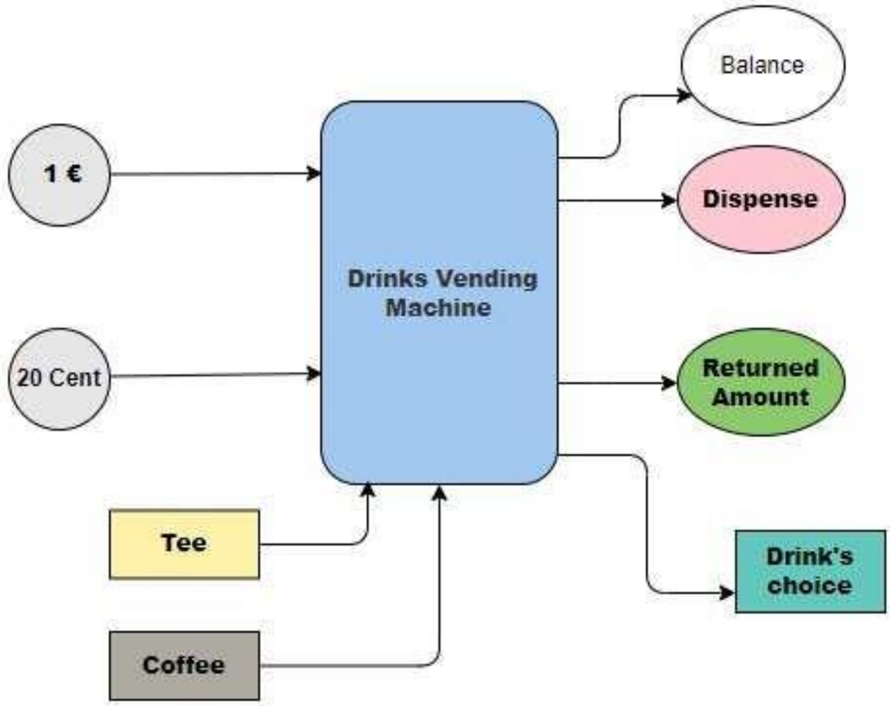
The vending machine concept that we are proposing, consists of inserting specific coins and requesting a drink type, and according to the inserted money, the machine will dispense the requested drink if the balance is enough and will return money in case the inserted money is higher than the drink’s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



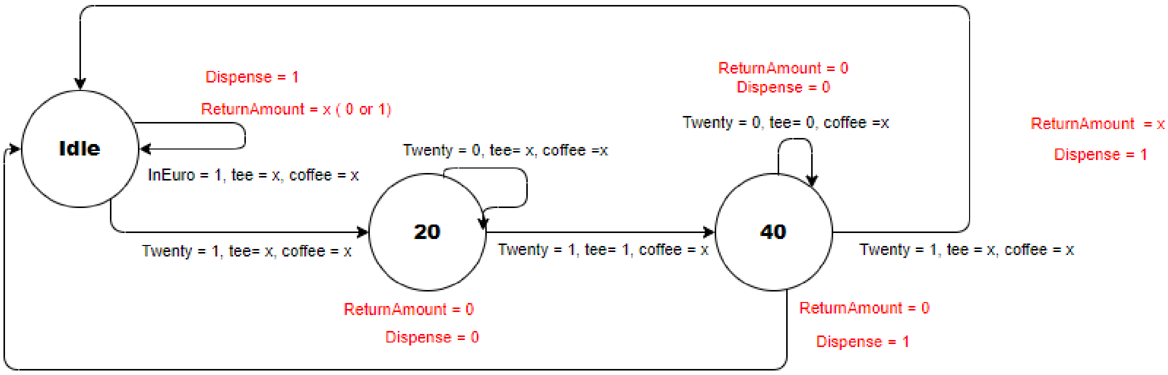
Block Diagram of Vending Machine

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. Thus for both cases dispense = x as it could be 0 or 1 depending on the requested drink. However, if the user inserted another 20 cents, the balance will reach 60 cents. Hence, it will return to an idle state. The dispensed drink could be coffee or tea and in case of choosing tea, the returned amount will be 20 cents. Otherwise, no return. Thus, it's important to note that for this state, the user can choose tea or coffee to receive the drink.



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	1	0
20	0	1	0	1	40	0	0
20	0	1	1	1	40	0	0
40	0	0	0	x	40	0	0
40	0	1	1	0	idle	1	1
40	0	1	0	1	idle	1	0
40	0	1	1	1	40	0	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). For the documentation, I wrote the concept (Block diagram + FSM diag and truth table), VHDL implementation, Project management, VHDL, and FPGA implementations, and 7-segment display.
- **Jasmeet**: Developed the earlier stage concept diagram, InputOutput for VHDL Code (7-Segment Display), and PCB Design. For documentation: PCB and VHDL input/outputs part and integration of the FSM with Display and Future/Further Other Way of Implementation.
- **Evrard**: responsible for the testbench of the VHDL code. For documentation: Introduction

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 different states of the FSM we used D flipflop for storing 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 if 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.

(The simulation is not perfect yet as the testbench is not delivered yet)

2. Testbench

1)

- Objective: This project is to be implemented in VHDL, using the main clock of the Digilent Board to drive the state machine. The layout is as follows.
- The vending machine has the following inputs:

Clock: A clock signal used to drive the finite state machine (FSM).
Reset: A signal used to reset the FSM to its initial state.
euroIn: An input signal indicating the insertion of a euro coin.
twentyCent: An input signal indicating the insertion of a twenty-cent coin.
tee: An input signal indicating the selection of tea as a drink.
coffee: An input signal indicating the selection of coffee as a drink.

- The vending machine has the following outputs:

returnAmount : a signal that goes high when a single quarter has been accepted.
Balance: a signal that goes high when a drink has been dispensed
drinksChoice: A 2-bit vector output representing the chosen drink(s).

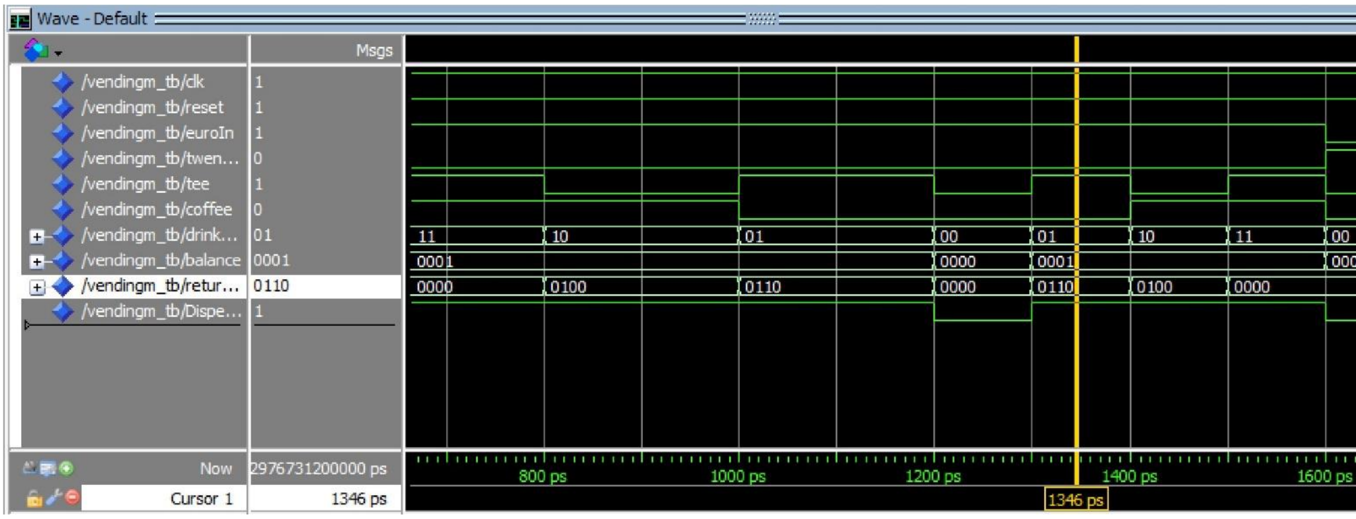
2)Cases

```
    -- Test case 1: Choose tea (40 cents)
euroIn <= '1';
twentyCent <= '0';
tee <= '1';
coffee <= '0';
wait for 20 ns;

-- Test case 2: Choose coffee (60 cents)
euroIn <= '1';
twentyCent <= '0';
tee <= '0';
coffee <= '1';
wait for 20 ns;
```

```
-- Test case 3: Choose tea (40 cents) and coffee (60 cents)
euroIn <= '1';
twentyCent <= '0';
tee <= '1';
coffee <= '1';
wait for 20 ns;

-- Test case 4: Insert 20 cents (no drink chosen)
euroIn <= '0';
twentyCent <= '1';
tee <= '0';
coffee <= '0';
wait for 20 ns;

-- Test case 5: Choose tea (40 cents) with insufficient funds
euroIn <= '0';
twentyCent <= '0';
tee <= '1';
coffee <= '0';
wait for 20 ns;

-- Test case 6: Choose coffee (60 cents) with exact funds
euroIn <= '1';
twentyCent <= '1';
tee <= '0';
coffee <= '1';
wait for 20 ns;

-- Test case 7: Insert 50 cents (no drink chosen)
euroIn <= '0';
twentyCent <= '1';
tee <= '1';
coffee <= '0';
wait for 20 ns;

-- Test case 8: Insert 60 cents and choose coffee (exact funds)
euroIn <= '0';
twentyCent <= '1';
tee <= '0';
coffee <= '1';
wait for 20 ns;

-- Test case 9: Choose tea (40 cents) and receive 20 cents change
euroIn <= '1';
twentyCent <= '0';
tee <= '1';
```

```
coffee <= '0';
wait for 20 ns;
```

3)Simulation

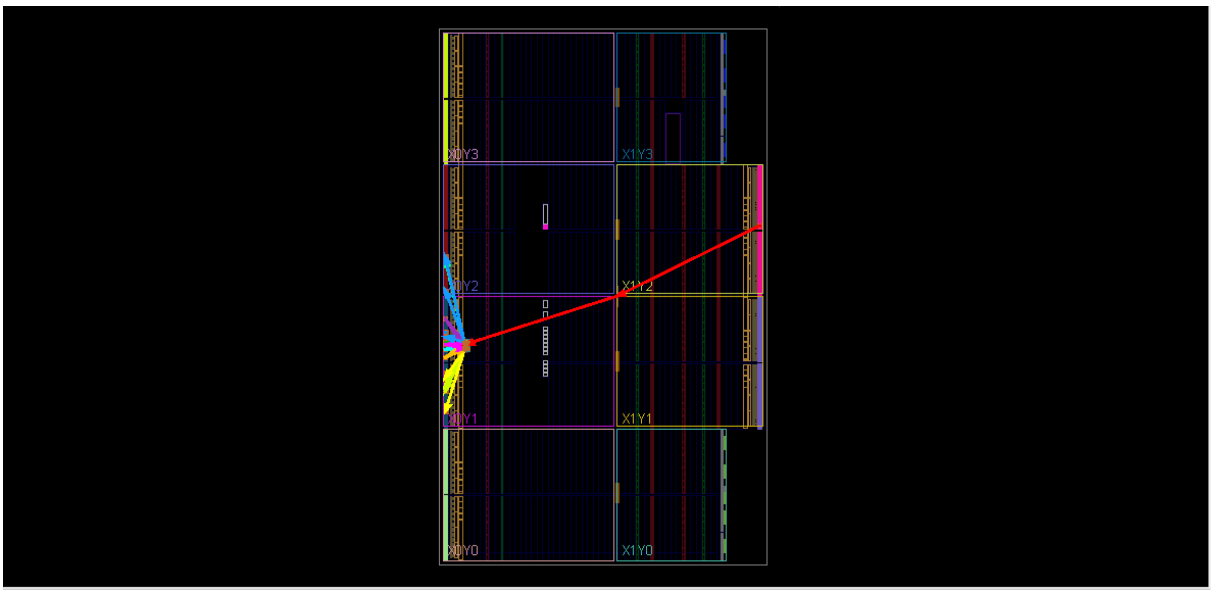


3. VHDL InputOutput

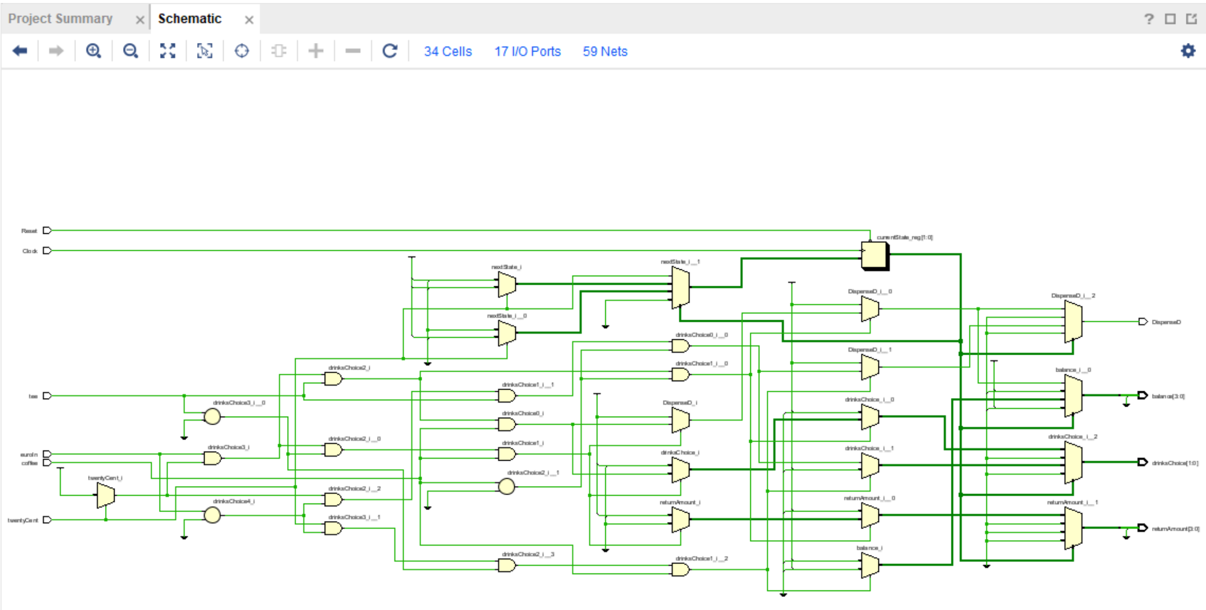
1) The Dataflow Design of the 7 Segment Display on the FPGA is different than the regular segment display. on fpga there are two units of 4-seven segment display attached together therefore they share the same data line on the fpga evaluation kit making it a little complex to program them.
2) The code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

4. Integration of FMS and 7 Segment Display

5. FPGA

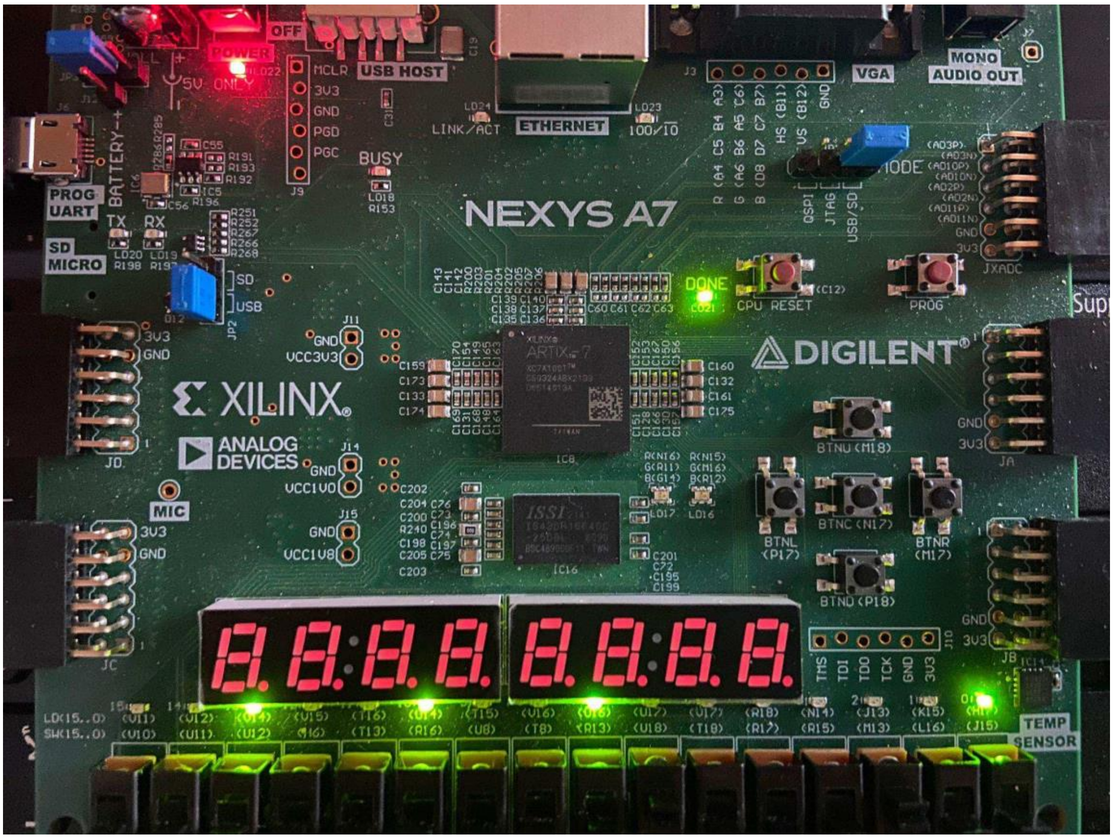


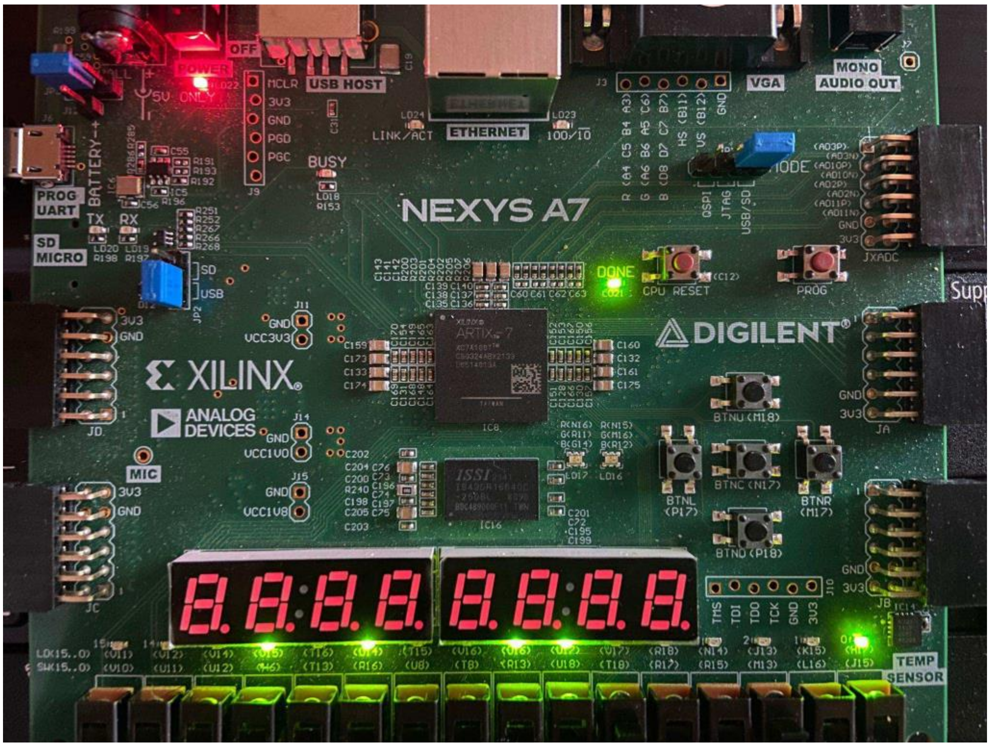
The I/O configuration for the Inputs/Outputs to the FPGA board's pins.



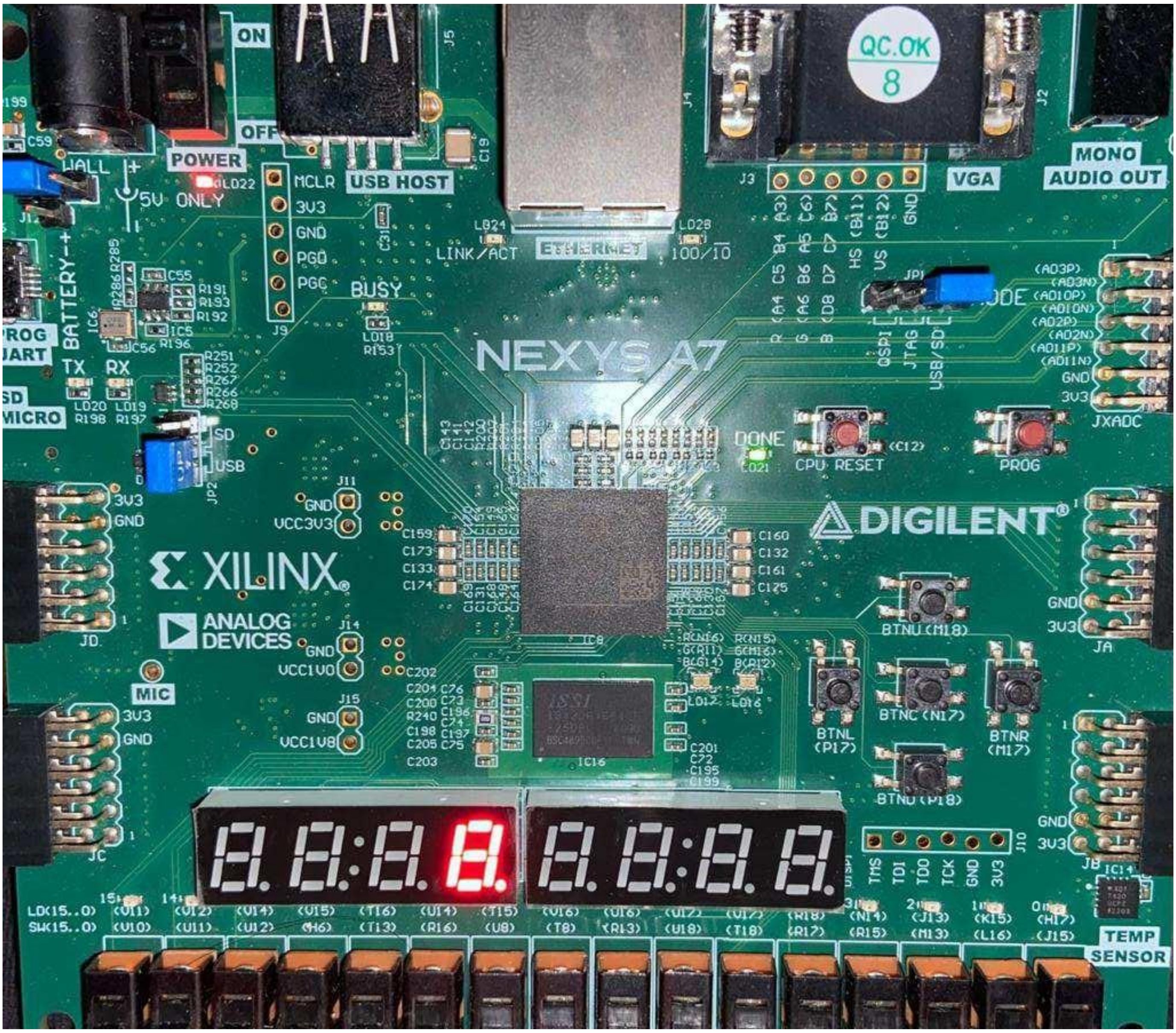
FPGA Schematic

For both FPGA screenshots, both are for inserting 1 euro but one for choosing the tea and the second for choosing the coffee as a drink. However, for the outputs, they are shown according to the LEDs.





This is the I/O port mapping from VHDL inputs/outputs to FPGA pins.


NEXYS A7
XILINX
ANALOG DEVICES
DIGILENT
QC.OK
8
POWER
ON
OFF
USB HOST
ETHERNET
VGA
MONO AUDIO OUT
CPU RESET
PROG
DONE
BUSY
MIC
TEMP SENSOR
JA
JC
JD
JXADC

```
# Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { Clock }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
#ucreate_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];


#Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { Reset }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { euroIn }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { twentyCent }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { tee }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { coffee }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { balance[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { balance[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { balance[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { balance[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { returnAmount[0] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { returnAmount[1] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { returnAmount[2] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { returnAmount[3] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { DispenseD }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { drinksChoice[0] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { drinksChoice[1] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
```

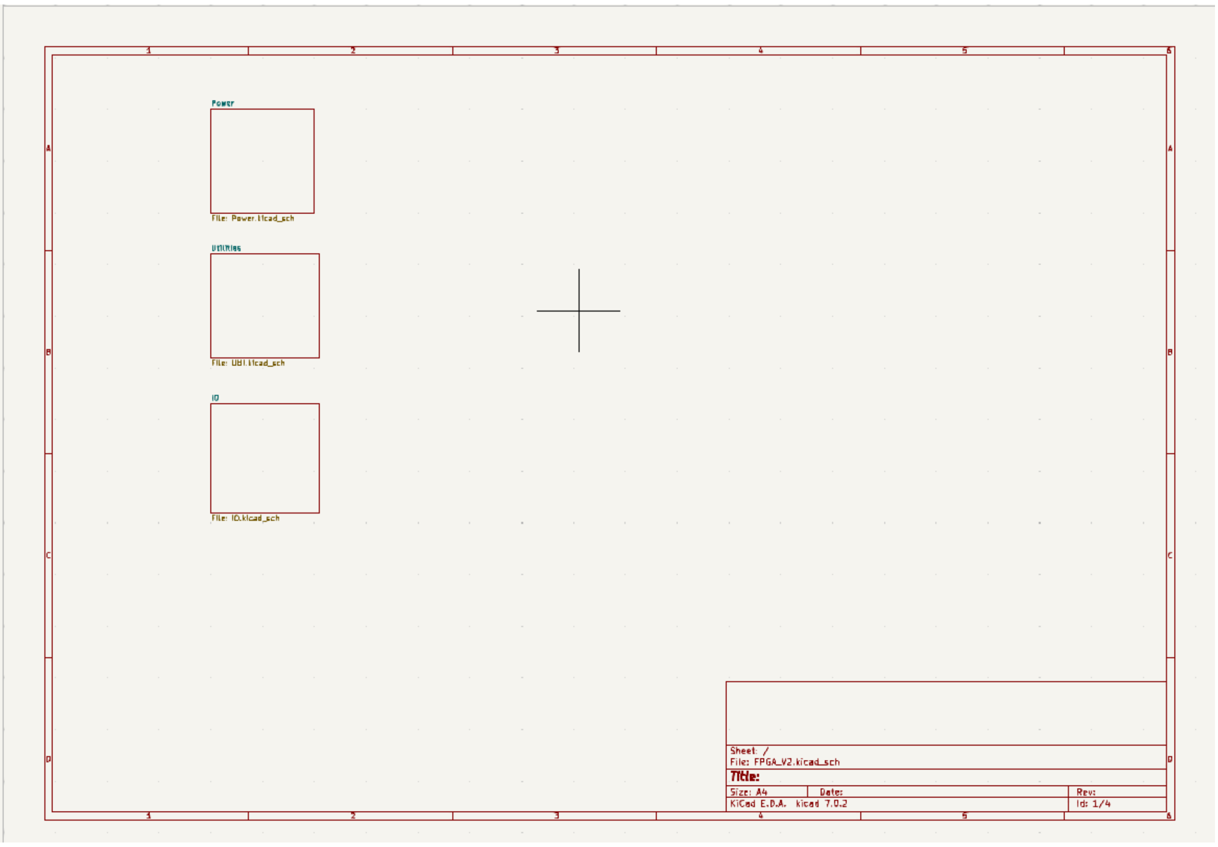
PCB Design

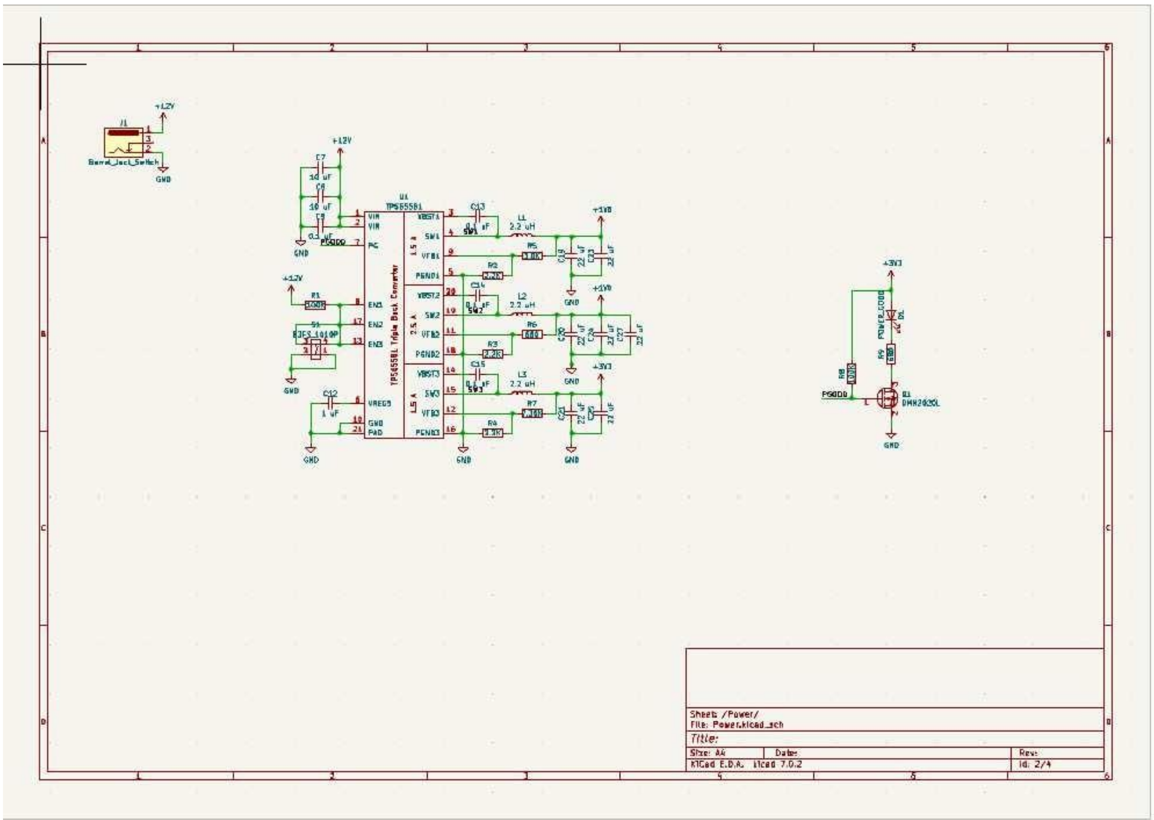
For PCB Design - Initially, Nexys A7 FPGA was used to create the PCB solution; however, due to the complexity of the FPGA and the simplicity of the project, the design was shifted to use Spartan 7 FPGA; however, the unfinished FPGA design for Nexys 7 is available on GitHub.

1) Schematic

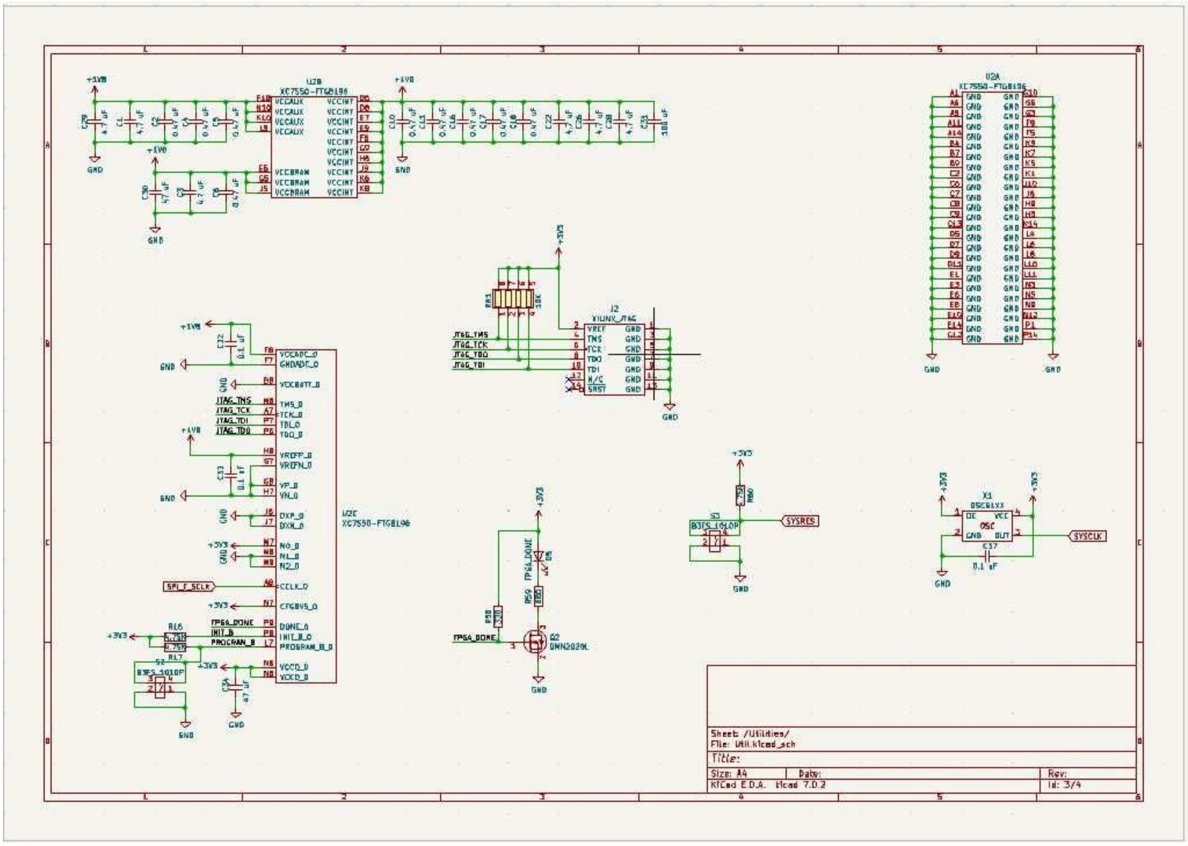
a) Power Schematic for providing power to the component on the PCB board, In the power schematic - a buck converter is used as different components take different voltages, and sometimes the same component needs different voltages, such as the FPGA and input for power.
b) Utility Schematic for connecting different

Root of the Schematic

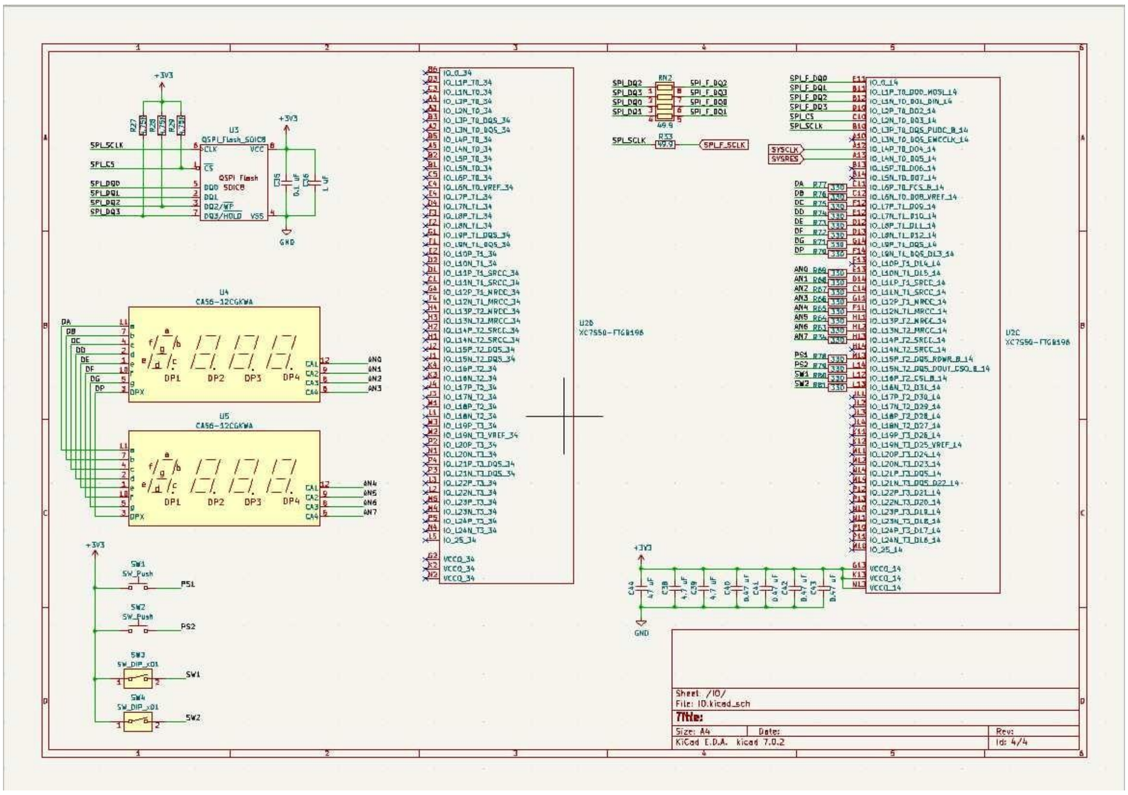




Power Management



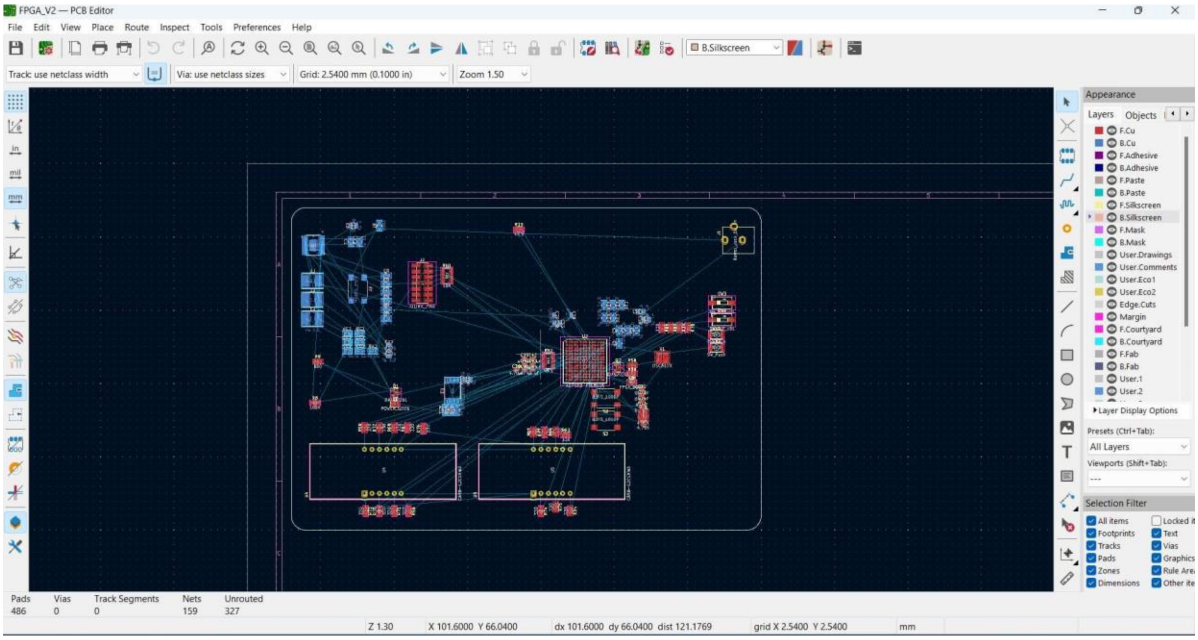
Utility management

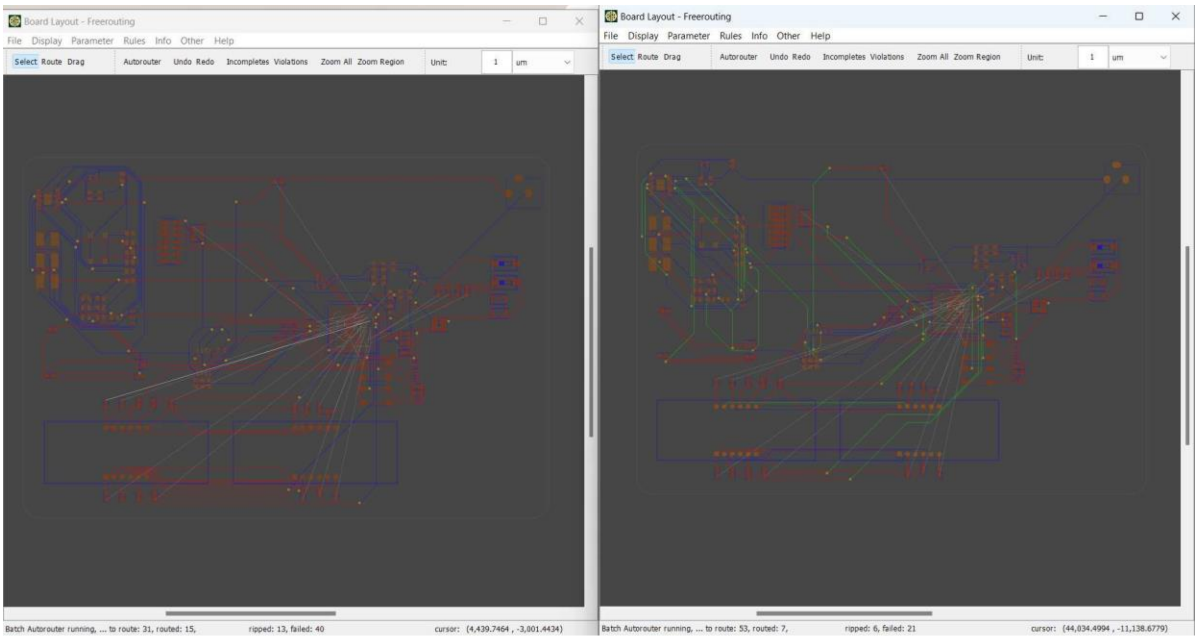


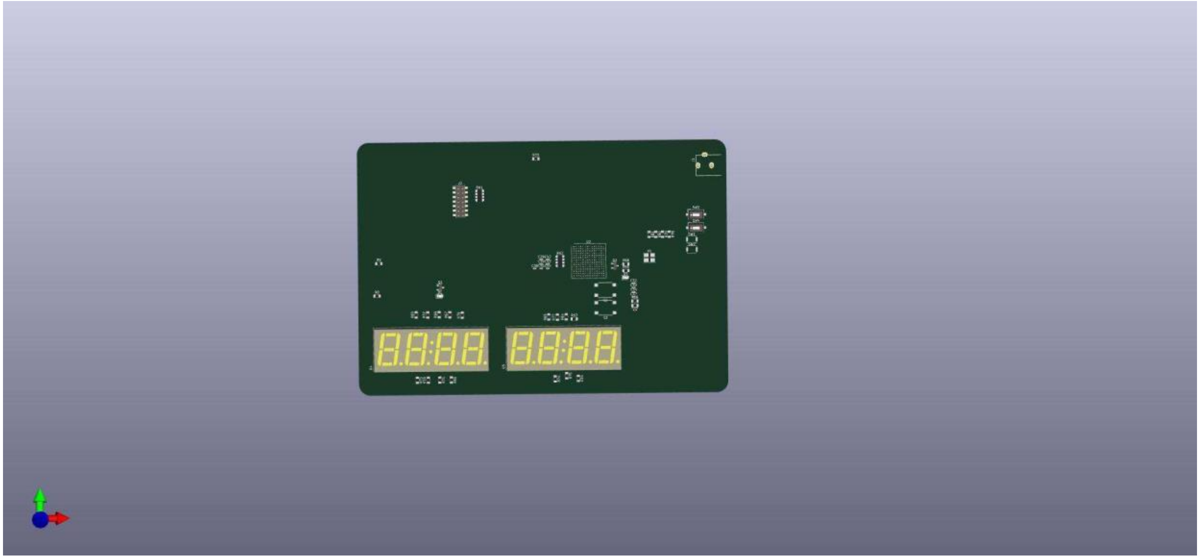
IO connected to the IC

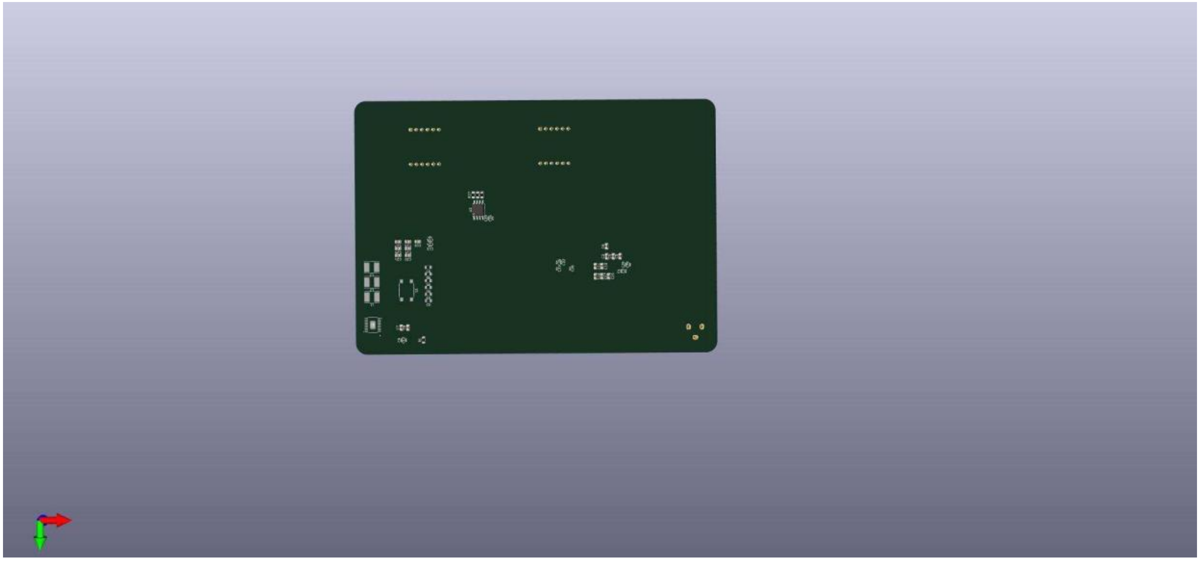
AutoRouting and components placement

Team 01





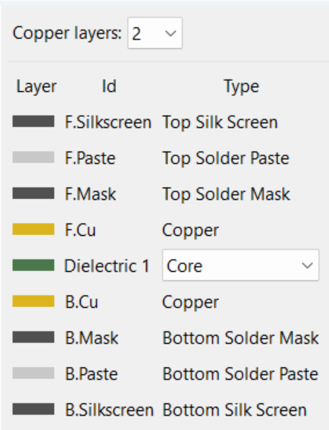




PCB Layer

For the following schematics and design we had two option for routing either to go with basic 2 layer or to have more layer. but we decided to to do autorouting with both option available and to compare the result therefore for four layer system all the layers have mixed property for autorouting to routes the track.

2 Copper Layer

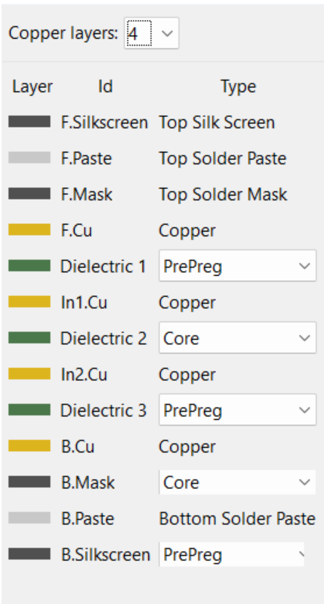

Copper layers: 2

Layer	Id	Type
	F.Silkscreen	Top Silk Screen
	F.Paste	Top Solder Paste
	F.Mask	Top Solder Mask
	F.Cu	Copper
	Dielectric 1	Core
	B.Cu	Copper
	B.Mask	Bottom Solder Mask
	B.Paste	Bottom Solder Paste
	B.Silkscreen	Bottom Silk Screen

Both Front copper and Back Copper are used for signals.

F.cu	Signals
B.cu	Signals

4 Copper Layer


Copper layers: 4

Layer	Id	Type
	F.Silkscreen	Top Silk Screen
	F.Paste	Top Solder Paste
	F.Mask	Top Solder Mask
	F.Cu	Copper
	Dielectric 1	PrePreg
	In1.Cu	Copper
	Dielectric 2	Core
	In2.Cu	Copper
	Dielectric 3	PrePreg
	B.Cu	Copper
	B.Mask	Core
	B.Paste	Bottom Solder Paste
	B.Silkscreen	PrePreg

Here all F.cu and B.cu i.e front and back copper layers are for signal and internal copper layer 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

Further/Future/Another Way of Implementation (optional)

In this Section we will talk about different ways on implementing the same system such as creating a system that works completely on Input without the use of FSM

Sources/References

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