Complete a simple FSM project:

LLM vs Human

Team 6 HW 4

Made by : He Zicheng, Liu Kaixin, Yang Runkang

* MENU
* Introduction
  + Brief description of the project's goals
  + Overview of the role of LLM in the project
* LLM written code
  + Using prompt-based interactions to guide the LLM's responses
  + Design Phase
  + Evaluation of LLM generated code
* Handwritten code
* State transition
* System block
* Evaluation of handwritten code
* Conclusion
* Appendix
  + Verilog Code for the basic vending machine
  + Testbench for the basic vending machine
  + Verilog Code for handwritten vending machine
  + Testbench for the handwritten vending machine

Introduction

Brief description of the project's goals

The project embarked upon was an ambitious venture to design and simulate an automatic vending machine using digital circuitry, specifically through the application of Verilog in a simulated environment provided by EDA Playground. The primary objective was to create a functional model of a vending machine that could handle tasks such as accepting coins, dispensing products, and returning change. We approached this project in two different ways: the first and crucial component of the project was the integration of a large language model (LLM), which served as a vital tool in aiding the design and simulation processes. In the second part we did this project on our own without the help of any big language models. Thus, a set of comparisons are obtained to analyze the advantages and disadvantages of large language models for the development of digital circuit projects.

Overview of the role of LLM in the project

The LLM's contribution was multifaceted, providing not only technical guidance and Verilog code examples but also assisting in refining the logic behind the vending machine's operation. This interaction was critical in bridging the gap between conceptual design and practical implementation, enabling a streamlined approach to tackling the complexities associated with digital circuit design.

In sum, the document aims to showcase the symbiotic relationship between human ingenuity and artificial intelligence. It highlights how large language models can extend their utility beyond simple question-and-answer formats to actively participating in complex engineering projects. This document will detail each phase of the project—from initial conception through to testing and final evaluations—emphasizing the iterative and collaborative nature of this modern design process.

**Summary of Conclusions**

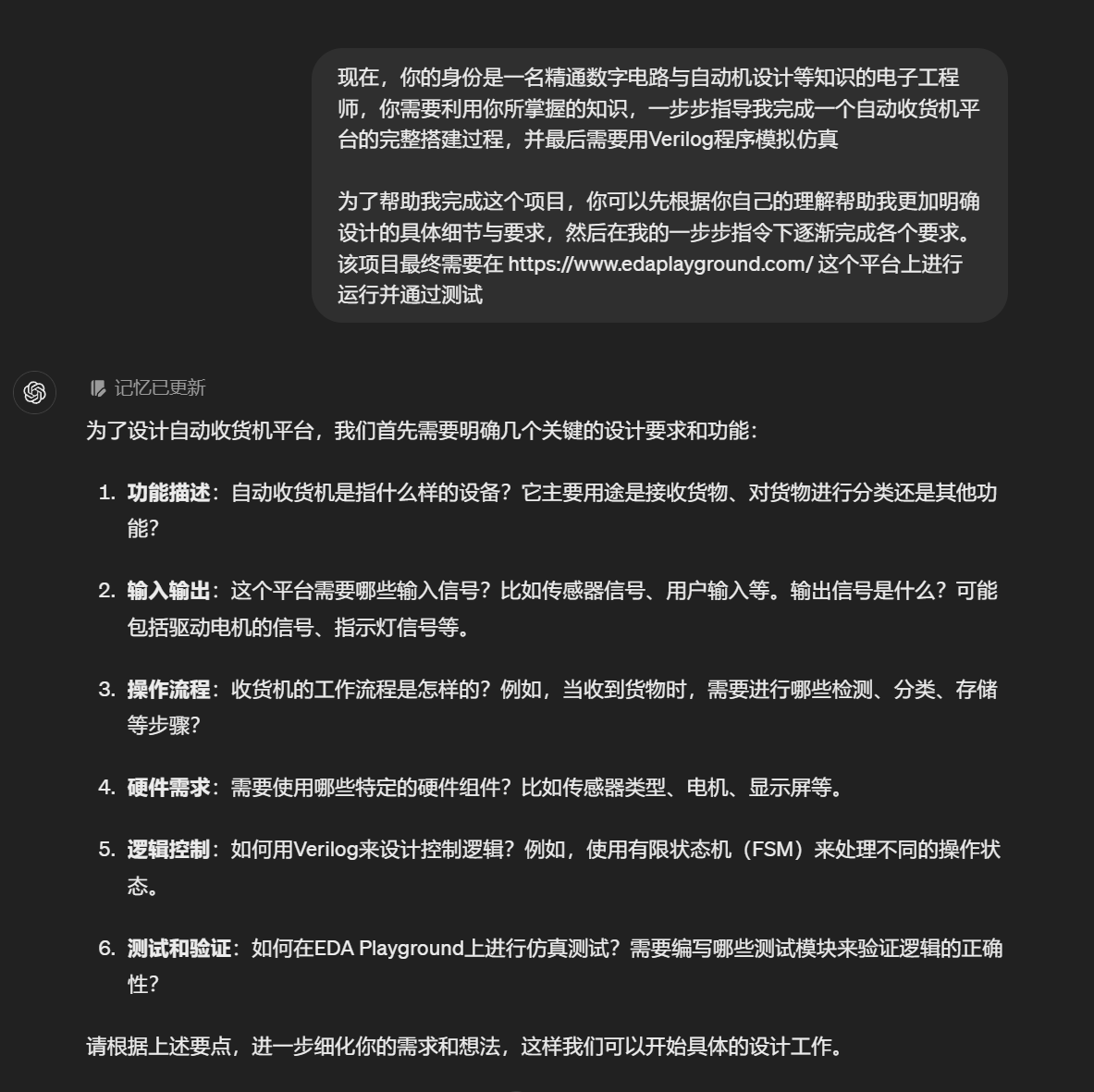
**The quality of the code written by LLM is only enough to accomplish the most basic tasks, and it focuses too much on logic to realize the functionality of the actual use case.**

***But, LLM can help zero-base users write code that performs basic functions. For experienced hardware engineers, LLM can also significantly improve productivity.***

LLM written code

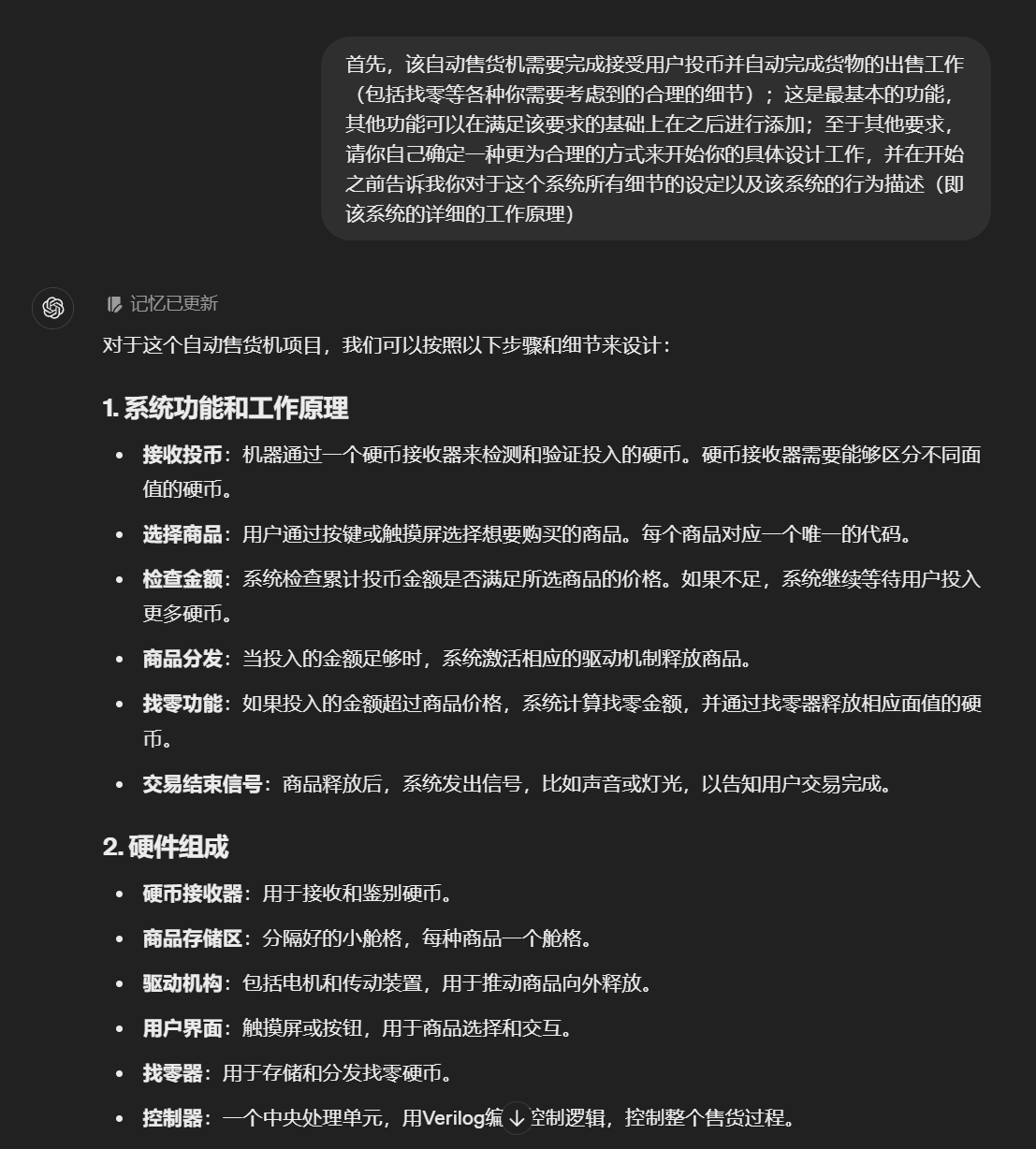
Initial interactions of the project requirements

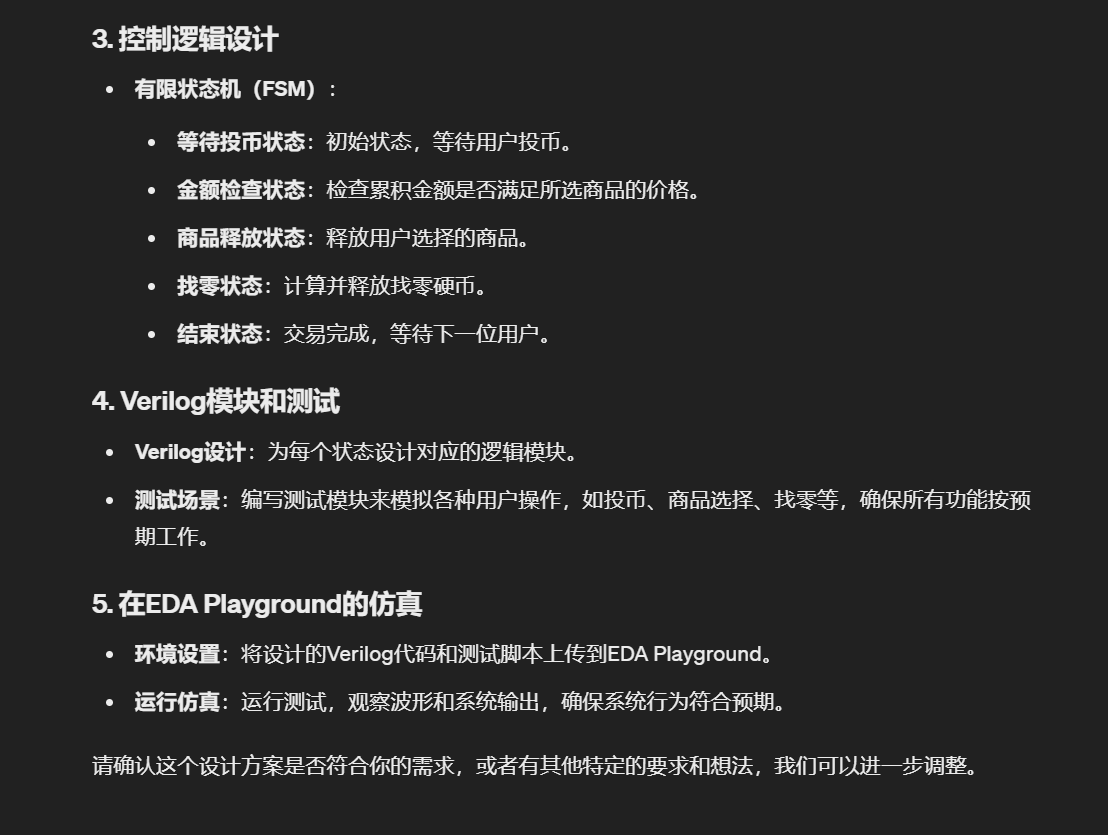
The initiation of the project began with defining the core objectives and functionalities of the automatic vending machine. The primary function identified was to enable the machine to accept coins, select products, dispense the chosen items, and provide change when necessary. I approached the LLM with these basic requirements, using specific prompts to refine the concept and explore potential designs.



Using prompt-based interactions to guide the LLM's responses

The LLM's role was instrumental in helping to shape the initial framework of the vending machine's design. I posed questions about the best practices for implementing finite state machines (FSM) in Verilog, to which the LLM responded with detailed explanations. These interactions were crucial in laying the groundwork for the subsequent development phases. Each response from the LLM was tailored to the context of the project, providing not only technical solutions but also clarifying complex concepts related to digital circuit design.

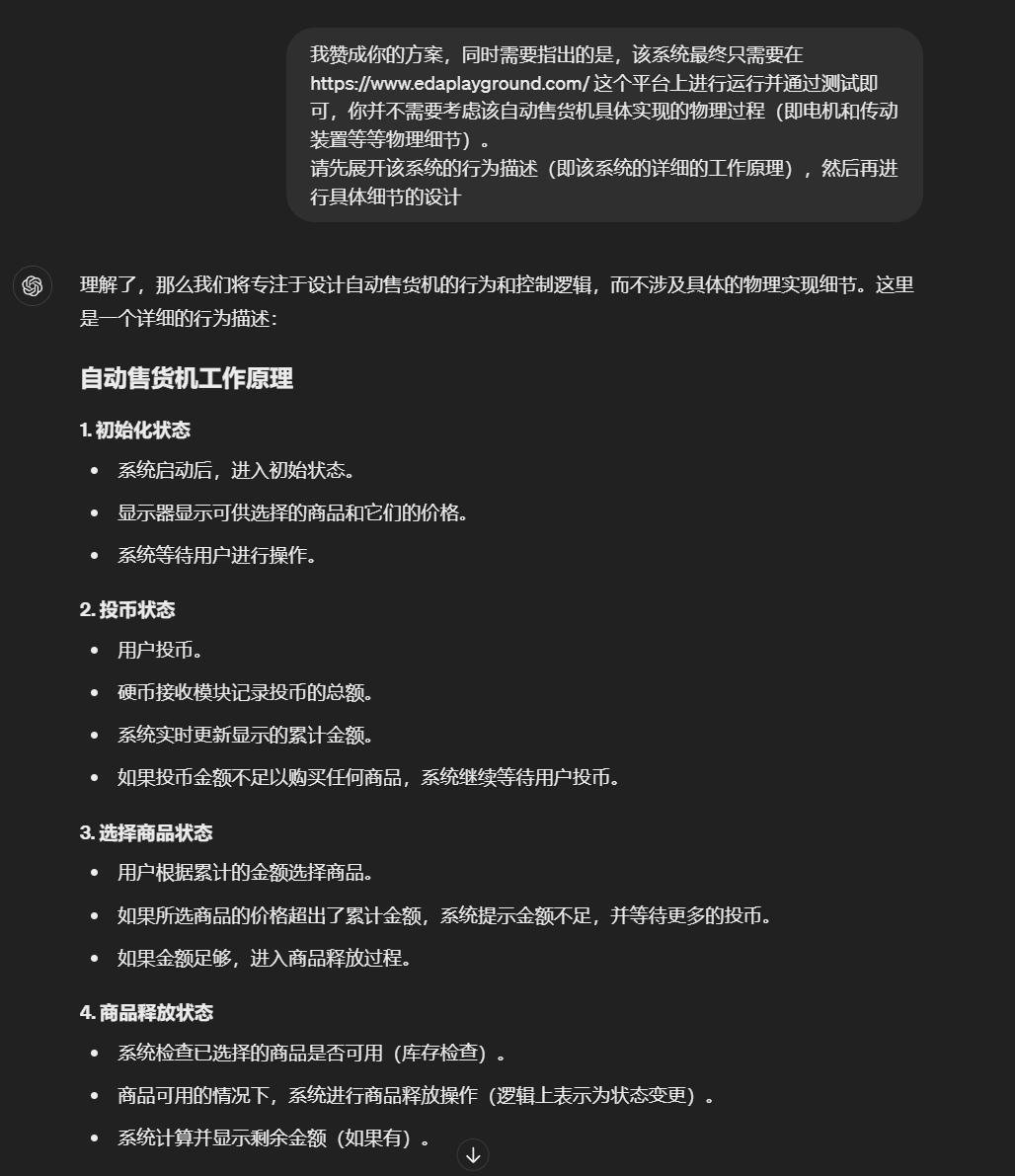




Design Phase

The automatic vending machine's operational logic

Once the initial requirements and functionalities were set, the design phase focused on developing a detailed digital model of the vending machine. I engaged the LLM with prompts that sought advice on structuring the FSM, defining state transitions, and implementing the logic for accepting inputs and triggering outputs.

The LLM provided a structured approach to the Verilog implementation, suggesting modular design for ease of testing and maintenance. The LLM also recommended strategies for simulation purposes and for ensuring that the design would be scalable and adaptable to future enhancements.

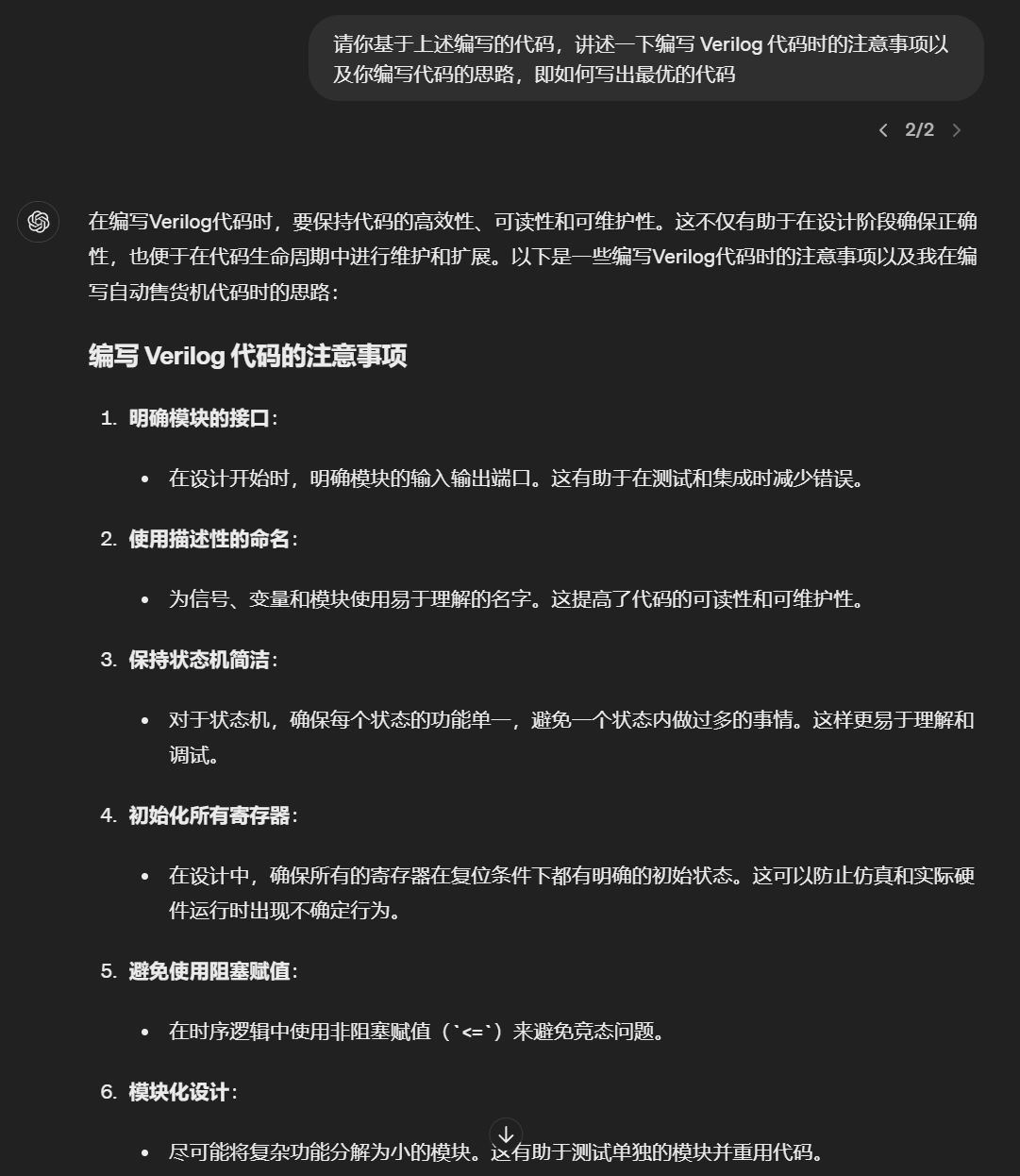
Code implementation of the Vending Machine System

Transitioning from the design phase, the development stage involved the actual creation and coding of the vending machine's digital circuit in Verilog. This was a meticulous process where each function of the vending machine was translated into a module within the Verilog code.



*See the code in the* ***appendix***

The LLM provided detailed Verilog examples for each core component of the system, such as coin acceptance, product dispensation, and change return mechanisms. These examples were critical in helping me understand how to structure the code effectively. The model's responsiveness to coins, product selection, and the computation of change were implemented as separate modules, allowing for easier debugging and testing.



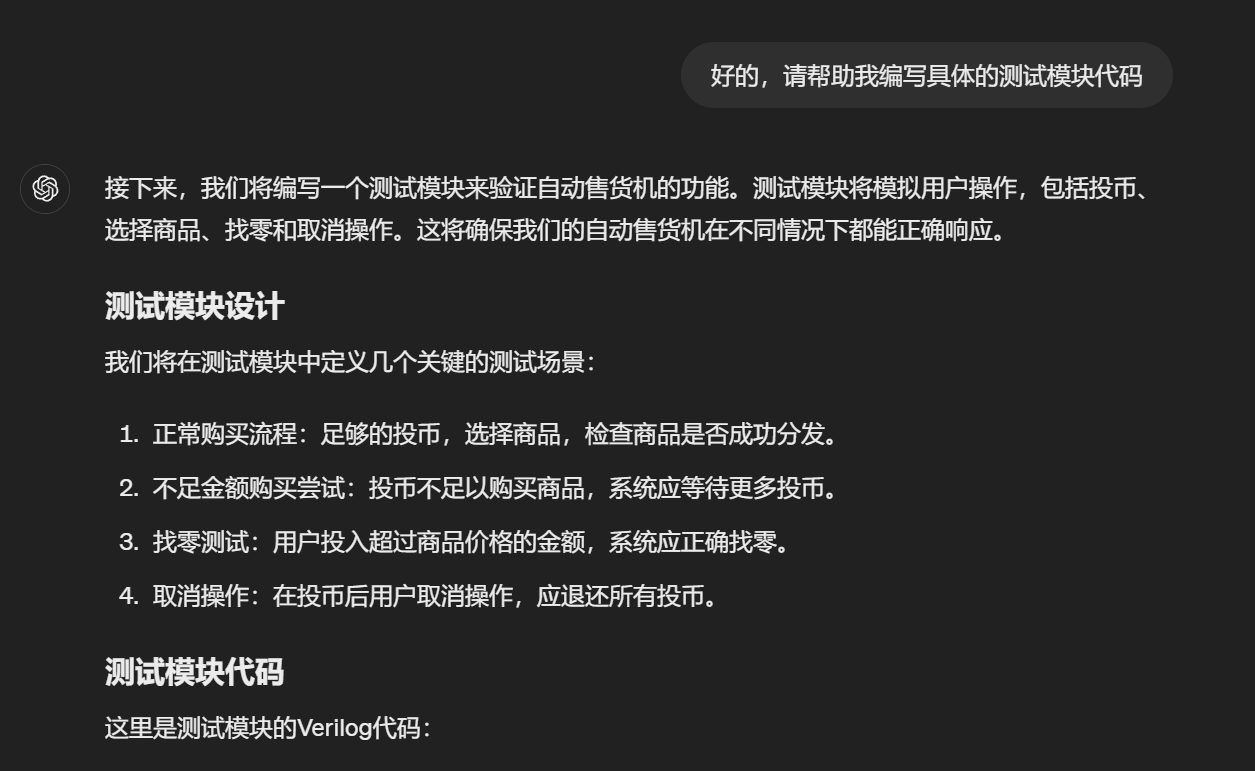
Additionally, the LLM played a crucial role in advising on best practices for writing maintainable and scalable Verilog code. This included tips on using always blocks effectively, managing state transitions, and ensuring that all potential edge cases were covered. The LLM's guidance was invaluable in refining the system's architecture to ensure it operated efficiently and reliably under all expected operational scenarios.

Testing and Simulation

Once the development of the Verilog code was sufficiently advanced, the next step involved testing and simulation to ensure that the vending machine operated as intended. The LLM helped in drafting comprehensive test cases that simulated a variety of user interactions with the vending machine. These interactions included inserting coins, selecting products, cancelling transactions and so on.



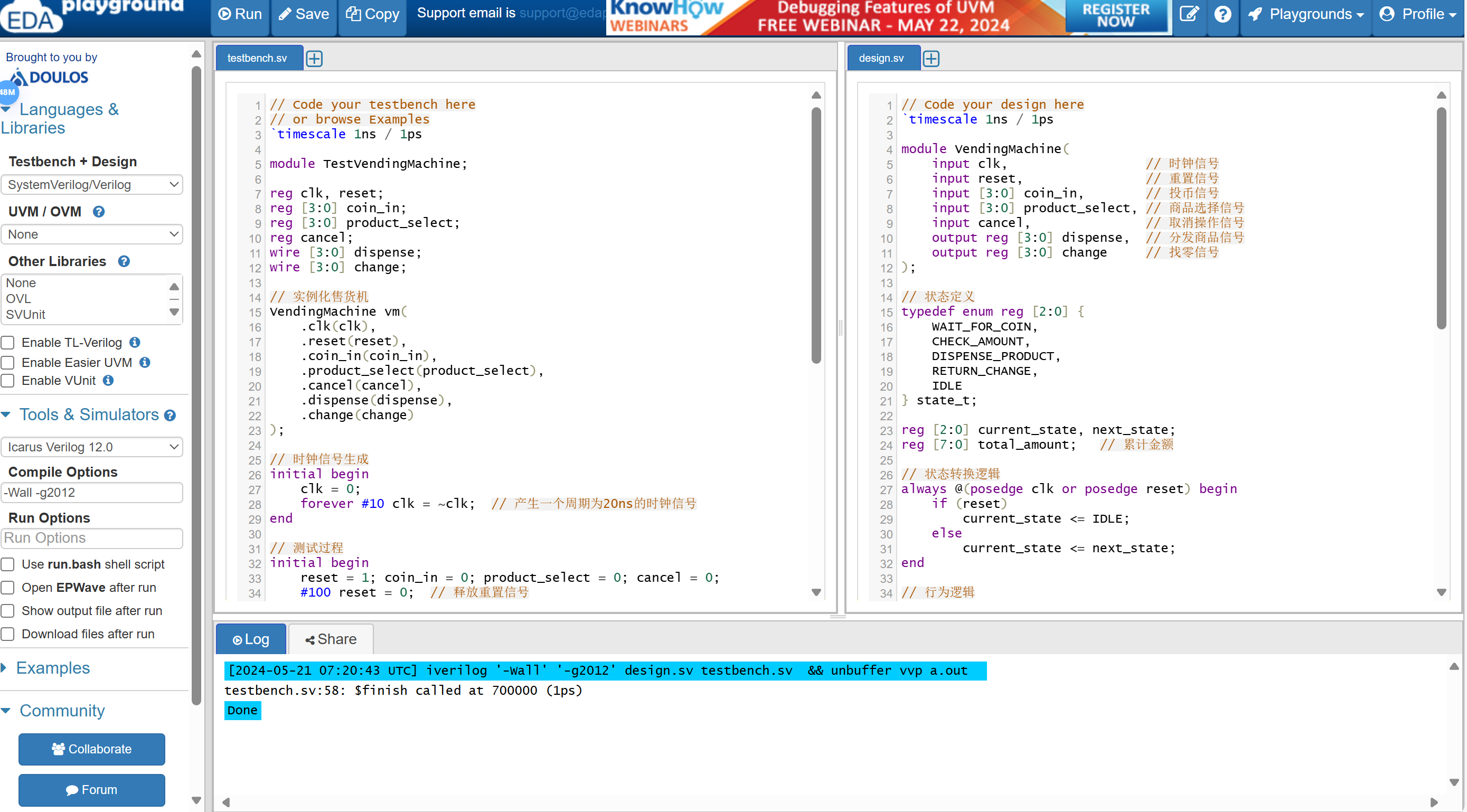
The LLM provided templates and examples of how to structure the test benches in Verilog. This included how to generate clock signals, simulate button presses for selecting products, and insert coin values. I used these templates to create a robust testing environment in EDA Playground, where each function of the vending machine was tested extensively.



*See the code in the* ***appendix***

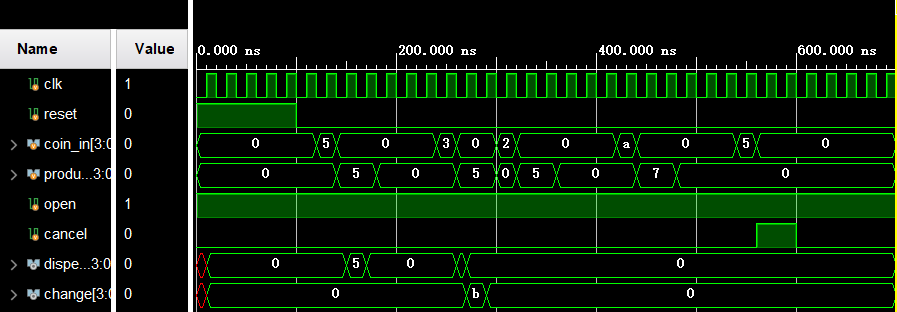
Evaluation of LLM generated code

The evaluation phase involved a critical assessment of the vending machine's performance based on the simulation results.

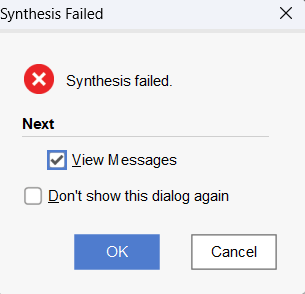


Simulation results were meticulously reviewed, and the LLM offered suggestions on how to interpret the outcomes and adjust the code based on the observed behaviors. The testing phase shows that the code written by the LLM is **not correct.** The programing didn’t consider sequential logic during generating the code. Since the test bench will only update the value while a rising edge occurs, the condition switch process written in the code is the a fixed time cycle for 20ns each status , which needs the very accurate value input at exactly the wanted situation. This doesn’t in line with the actual situation, so the design’s basic logic is wrong. Moreover, some code habits used in the code doesn’t meet the standards.

LLM’s code can pass behavioral simulation.



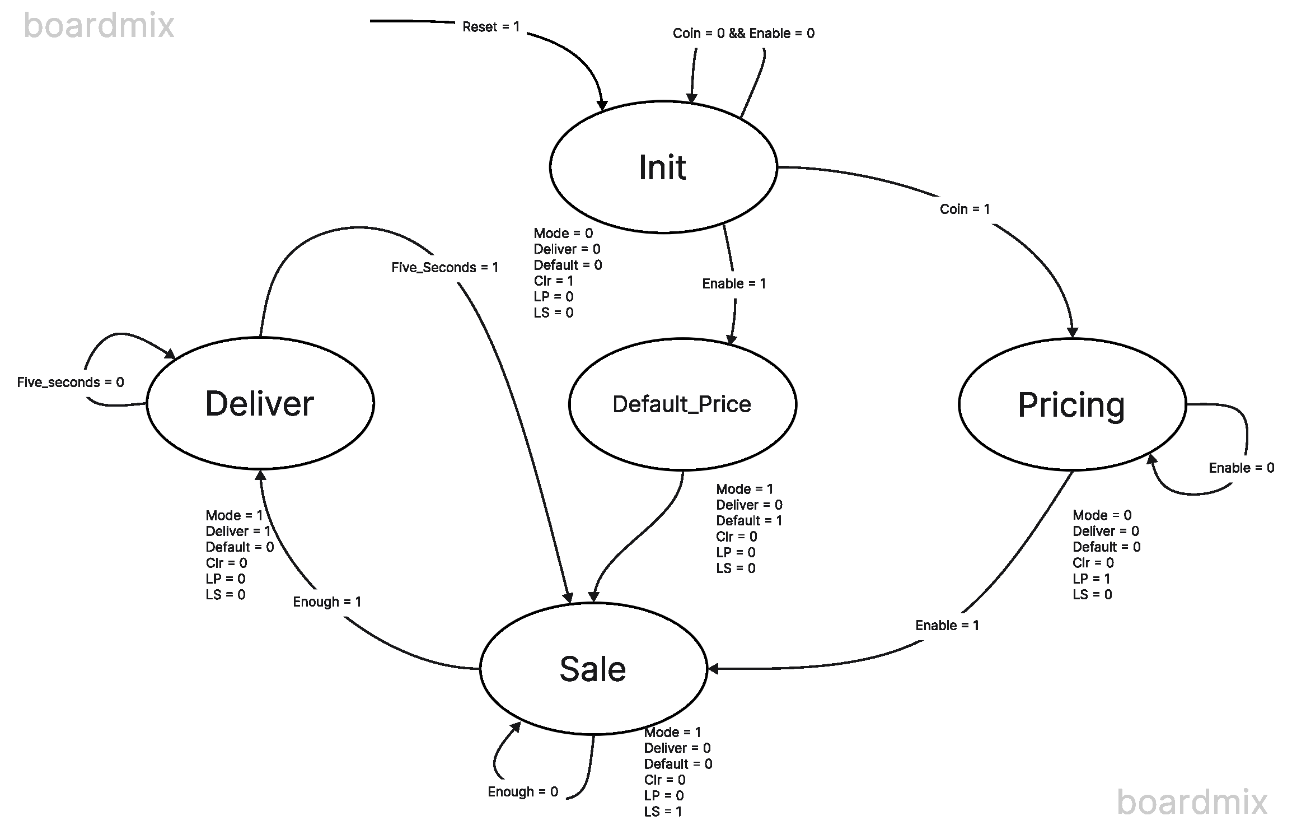
But it cannot be synthesized



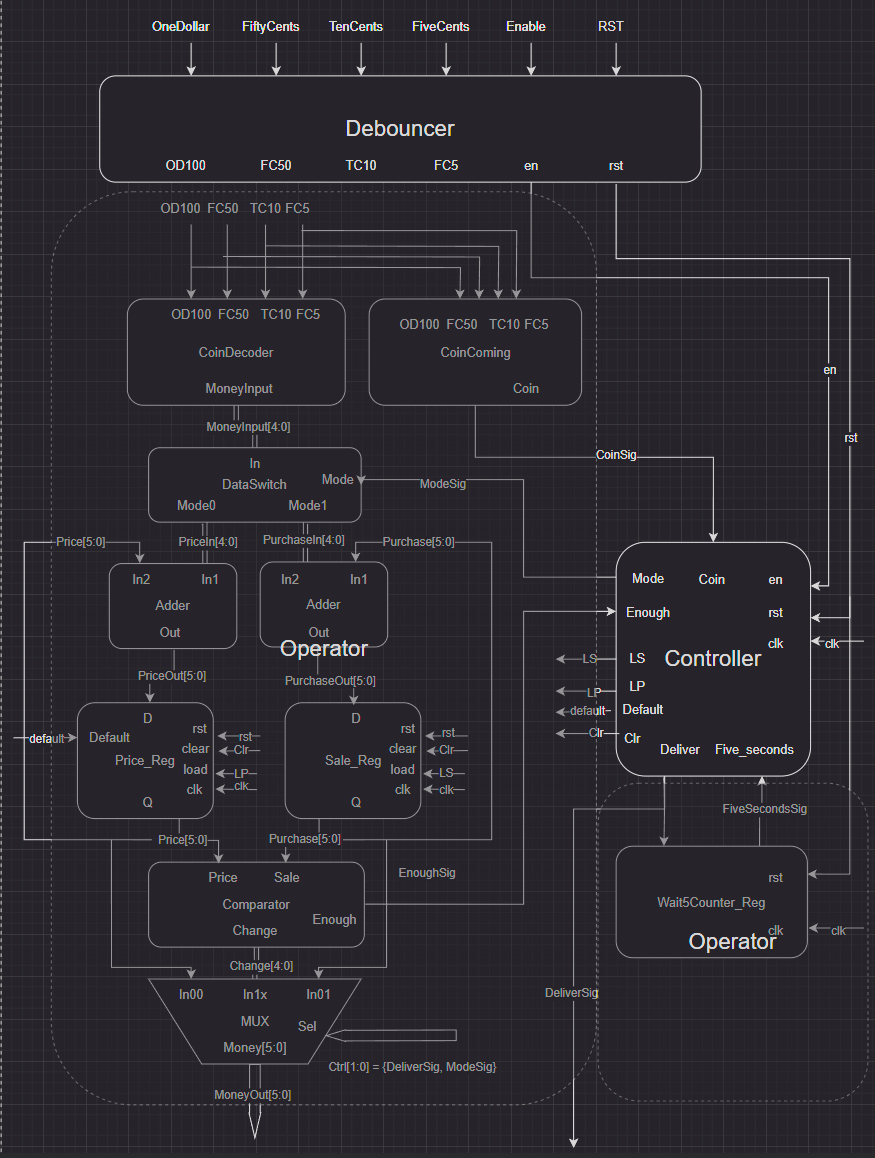
Handwritten code

**State transition**

We consider that the vending machine, which only sell on item at one time, has 5 states, including initial state, pricing state, default pricing state, sale state and deliver state. The transition of each state need to meet some conditions. First, we stay in the initial state until the coins come in or the sale is begin. When the coins first come in, we set the price at the number of the coins that comes in and wait for the sale. When the sale begin before the price is set, it will enter default price state first and turn to sale state immediately. In the sale state, we will only turn to deliver state when the input coins is enough for buying the product. The deliver state turns back to the sale state after the item is delivered. The setting of price can only be down after resetting the machine.

****

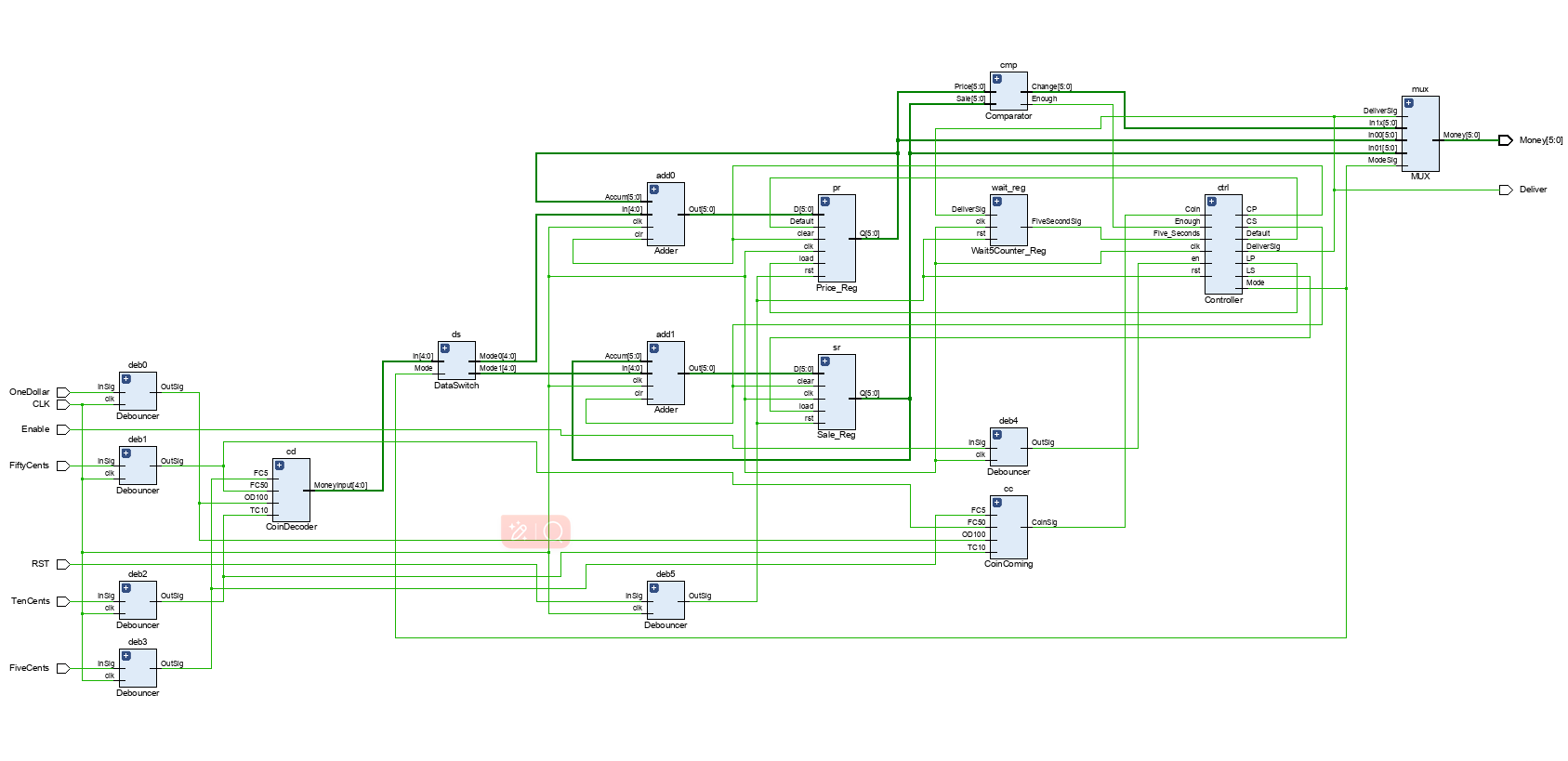
**System block**

To achieve the final effects, we need to change these states into blocks in vivado. The whole structure is shown below:

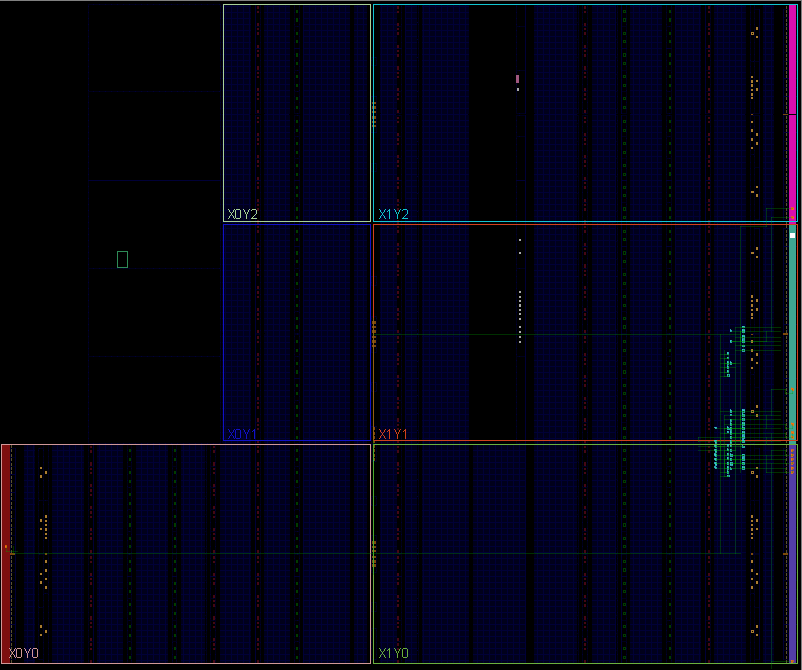
The code of each part is listed in the appendix.

**Evaluation of handwritten code**

This is the schematic of the human written code, as you can see, different functions are put into different modules, which greatly increases the robustness of the whole system and the ability to expand the function.

****

The graph below shows the hardware footprint of the hand-written code after placement and routing (the hardware in use is in the light).

****

**Comparing the codes: LLM versus Human**

By comparing the results written by chat-gpt with those written by humans, we can conclude that:

1, LLM can complete the most basic requirements of the project in a very short time, while manual handwriting and debug took 3 to 4 times as long as LLM.

2, The final code given by LLM is done in the same module, which greatly reduces the possibility of extending its functionality. Manually written code separates the temporal logic from the combinational logic, which makes it highly robust and extensible.

3, The code written by LLM only focuses on the logic in the digital circuit and does not take into account the details of the use of this circuit in real scenarios. For example, it is not considered in LLM that the keys of a vending machine need the function of shaking elimination, and the change between states is in clock cycles, which means that we need to complete the purchase operation within a few nanoseconds.

4, The code written by LLM only considers the function implementation and is reflected in its high hardware occupancy on FPGA, while the code written by manual can take into account the requirement of saving hardware deployment.

Conclusion

The project's completion marked a significant achievement in utilizing a large language model to assist in the complex process of digital circuit design. The collaboration with the LLM proved to be exceptionally beneficial, from the initial conceptualization to the final testing and evaluation stages. The document has detailed how, through iterative and prompt-based interactions, a user can effectively guide an LLM to produce tailored and practical responses that significantly contribute to the success of a technical project.

This is great for users who have zero background in hardware design. LLM can help users write a code that accomplishes basic functions in a short period of time. Although we can see that there is still a gap between the quality of the code and the results achieved by people with experience in hardware system design. Ultimately, however, we believe that the quality of the code generated by LLM is related to the quality of the prompts used to train LLM, which has a lot to do with the hardware knowledge base and the familiarity of the trainer with the project.

This experience not only highlights the potential of large language models in enhancing engineering design processes but also sets a precedent for future projects where similar collaborative approaches can be employed to tackle challenging technical tasks.

**The final conclusion is that LLM can help zero-base users write code that performs basic functions. For experienced hardware engineers, LLM can also significantly improve productivity.**

Appendix

The appendix section contains the code written by the LLM and by ourselves (human) throughout the document. These resources provide a tangible reference to the theoretical and practical aspects covered in the document, offering an invaluable resource for anyone looking to understand or replicate the project's methodology.

LLM written code

`timescale 1ns / 1ps

module VendingMachine(

    input clk,                  // 时钟信号

    input reset,                // 重置信号

    input [3:0] coin\_in,        // 投币信号

    input [3:0] product\_select, // 商品选择信号

    input cancel,               // 取消操作信号

    output reg [3:0] dispense,  // 分发商品信号

    output reg [3:0] change     // 找零信号

);

// 状态定义

typedef enum reg [2:0] {

    WAIT\_FOR\_COIN,

    CHECK\_AMOUNT,

    DISPENSE\_PRODUCT,

    RETURN\_CHANGE,

    IDLE

} state\_t;

reg [2:0] current\_state, next\_state;

reg [7:0] total\_amount;   // 累计金额

// 状态转换逻辑

always @(posedge clk or posedge reset) begin

    if (reset)

        current\_state <= IDLE;

    else

        current\_state <= next\_state;

end

// 行为逻辑

always @(\*) begin

    case (current\_state)

        IDLE: begin

            if (coin\_in > 0)

                next\_state = WAIT\_FOR\_COIN;

            else

                next\_state = IDLE;

        end

        WAIT\_FOR\_COIN: begin

            total\_amount += coin\_in;  // 累计投币金额

            next\_state = CHECK\_AMOUNT;

        end

        CHECK\_AMOUNT: begin

            if (total\_amount >= product\_select)  // 检查金额是否足够

                next\_state = DISPENSE\_PRODUCT;

            else if (cancel)

                next\_state = RETURN\_CHANGE;

            else

                next\_state = WAIT\_FOR\_COIN;

        end

        DISPENSE\_PRODUCT: begin

            dispense = product\_select;  // 输出商品

            total\_amount -= product\_select;  // 更新剩余金额

            next\_state = RETURN\_CHANGE;

        end

        RETURN\_CHANGE: begin

            change = total\_amount;  // 输出找零

            total\_amount = 0;       // 重置金额

            next\_state = IDLE;

        end

        default: begin

            next\_state = IDLE;

        end

    endcase

end

endmodule

Testbench written by LLM

`timescale 1ns / 1ps

module TestVendingMachine;

reg clk, reset;

reg [3:0] coin\_in;

reg [3:0] product\_select;

reg cancel;

wire [3:0] dispense;

wire [3:0] change;

// 实例化售货机

VendingMachine vm(

    .clk(clk),

    .reset(reset),

    .coin\_in(coin\_in),

    .product\_select(product\_select),

    .cancel(cancel),

    .dispense(dispense),

    .change(change)

);

// 时钟信号生成

initial begin

    clk = 0;

    forever #10 clk = ~clk;  // 产生一个周期为20ns的时钟信号

end

// 测试过程

initial begin

    reset = 1; coin\_in = 0; product\_select = 0; cancel = 0;

    #100 reset = 0;  // 释放重置信号

    // 测试1: 正常购买流程

    #20 coin\_in = 5;  // 投币5单位

    #20 coin\_in = 0; product\_select = 5;  // 选择价格为5的商品

    #40 product\_select = 0;  // 清除选择

    // 测试2: 不足金额购买尝试

    #60 coin\_in = 3;  // 投币3单位

    #20 coin\_in = 0; product\_select = 5;  // 尝试购买价格为5的商品

    #40 product\_select = 0; coin\_in = 2;  // 再次投币以满足价格

    #20 coin\_in = 0; product\_select = 5;

    #40 product\_select = 0;

    // 测试3: 找零测试

    #60 coin\_in = 10;  // 投币10单位

    #20 coin\_in = 0; product\_select = 7;  // 购买价格为7的商品，应找零3

    #40 product\_select = 0;

    // 测试4: 取消操作

    #60 coin\_in = 5;  // 投币5单位

    #20 coin\_in = 0; cancel = 1;  // 取消操作，应退币5单位

    #40 cancel = 0;

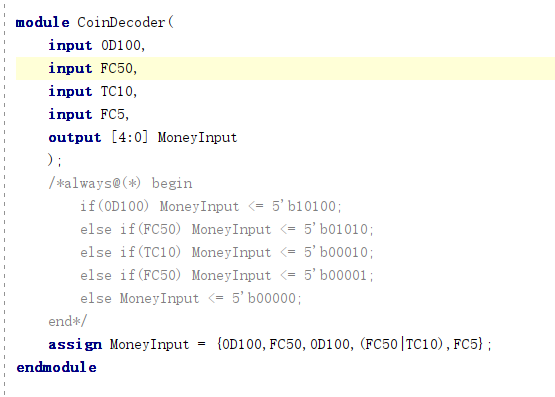
    #100 $finish;  // 结束仿真

end

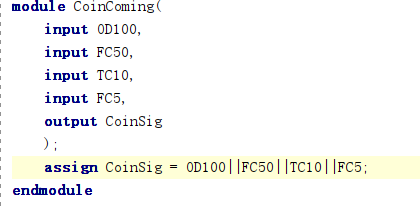
endmodule

Hand written code

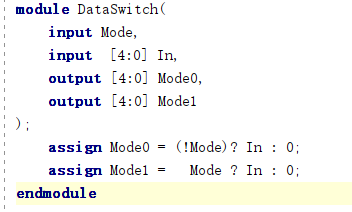
1. Coin decoder:

Translate the input coins into specific money.

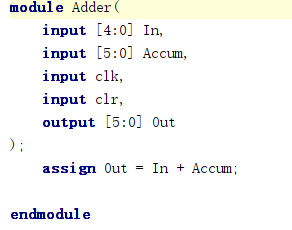
1. Coin coming:

Store the price in register for later the use in later states.

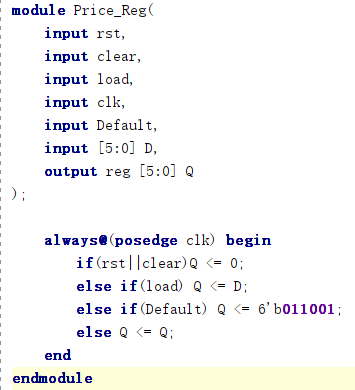
1. Data switch:

Decide whether its price setting or sale state:

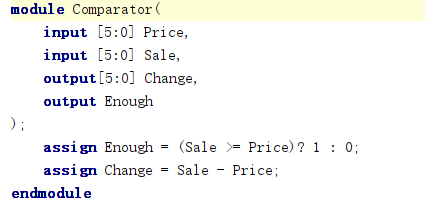
1. Adder:

Calculate the total value:

1. Price/Sale\_Reg:

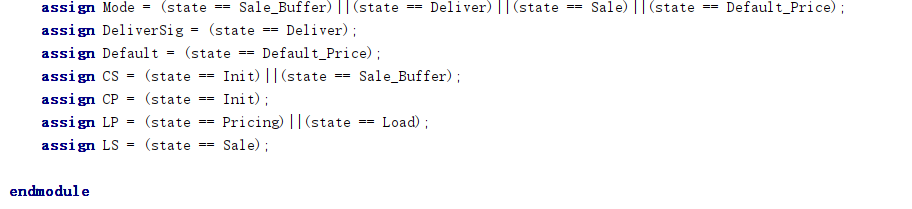
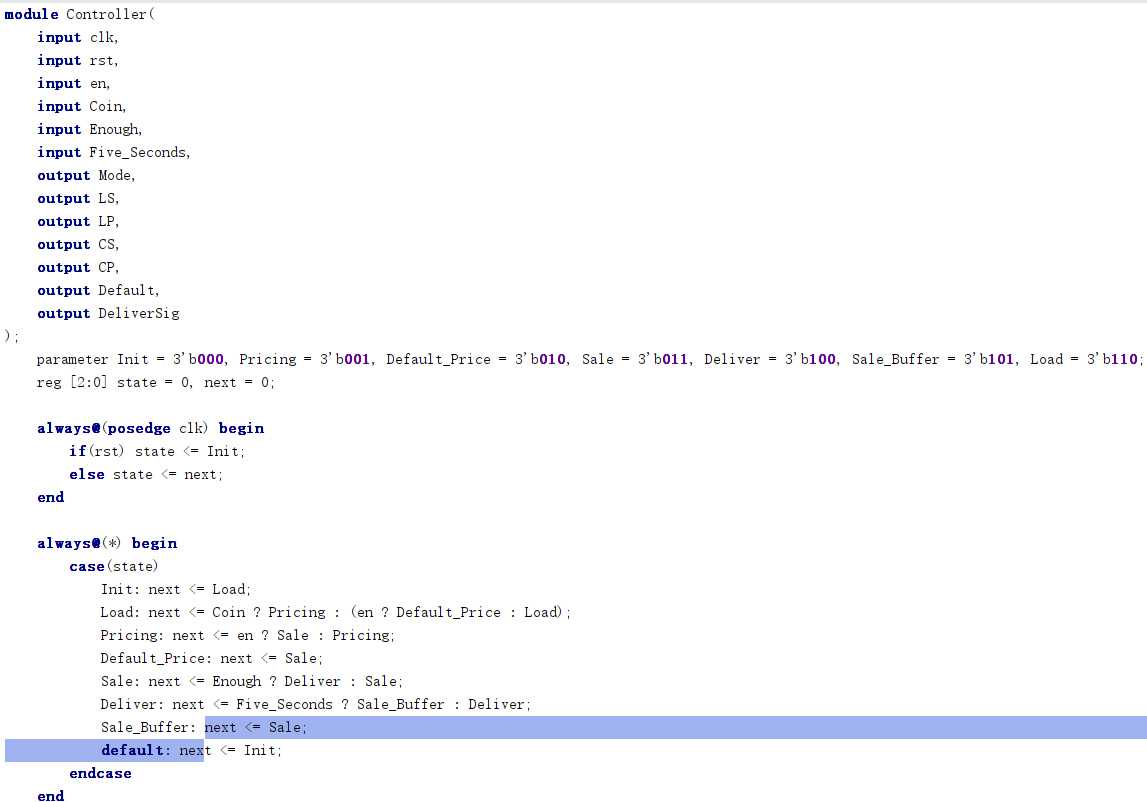
Calculate the total value:

1. Comparator:

To see whether the money is enough for the purchase:

1. Controller:

To see whether the money is enough for the purchase:



Testbench written by Hand

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/04/12 16:28:41

// Design Name:

// Module Name: VENDING\_MACHINEtb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module VENDING\_MACHINEtb(

);

reg en = 0,

rst = 0,

clk = 0,

od100 = 0,

fc50 = 0,

tc10 = 0,

fc5 = 0;

wire deliver;

wire [5:0] money;

VENDING\_MACHINE vm(.Enable(en),.RST(rst),.CLK(clk),.OneDollar(od100),.FiftyCents(fc50),.TenCents(tc10),.FiveCents(fc5),.Deliver(deliver),.Money(money));

always #5 clk <= ~clk;

initial begin

#100 rst = 1;

#2 rst = 0;

#1 rst = 1;

#100 rst = 0;

#2 rst = 1;

#1 rst = 0;

//set price to 165 (21)

#100 fc5 = 1;

#2 fc5 = 0;

#1 fc5 = 1;

#100 fc5 = 0;

#2 fc5 = 1;

#1 fc5 = 0;

#100 fc50 = 1;

#2 fc50 = 0;

#1 fc50 = 1;

#100 fc50 = 0;

#2 fc50 = 1;

#1 fc50 = 0;

#100 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#100 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

#100 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#100 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

//sale mode, change should be 0

#100 en = 1;

#4 en = 0;

#2 en = 1;

#100 en = 0;

#2 en = 1;

#1 en = 0;

#100 fc5 = 1;

#2 fc5 = 0;

#1 fc5 = 1;

#100 fc5 = 0;

#2 fc5 = 1;

#1 fc5 = 0;

#100 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#100 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

#100 fc50 = 1;

#2 fc50 = 0;

#1 fc50 = 1;

#100 fc50 = 0;

#2 fc50 = 1;

#1 fc50 = 0;

#100 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#100 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

//sale mode again, change should be 95(13)

#100 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#100 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

#100 fc50 = 1;

#2 fc50 = 0;

#1 fc50 = 1;

#100 fc50 = 0;

#2 fc50 = 1;

#1 fc50 = 0;

#100 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#100 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

#100 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#100 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

//reset to test default-price mode

#100 rst = 1;

#2 rst = 0;

#1 rst = 1;

#100 rst = 0;

#2 rst = 1;

#1 rst = 0;

//default mode

#100 en = 1;

#4 en = 0;

#2 en = 1;

#100 en = 0;

#2 en = 1;

#1 en = 0;

//sale mode, change should be 40 (8)

#100 fc50 = 1;

#2 fc50 = 0;

#1 fc50 = 1;

#100 fc50 = 0;

#2 fc50 = 1;

#1 fc50 = 0;

#100 fc5 = 1;

#2 fc5 = 0;

#1 fc5 = 1;

#100 fc5 = 0;

#2 fc5 = 1;

#1 fc5 = 0;

#100 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#100 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

#100 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#100 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

//reset

#100 rst = 1;

#2 rst = 0;

#1 rst = 1;

#100 rst = 0;

#2 rst = 1;

#1 rst = 0;

end

endmodule

/\*

always #3 clk <= ~clk;

initial begin

//use default price

//saling:

#100 en = 1;

#4 en = 0;

#2 en = 1;

#60 en = 0;

#2 en = 1;

#1 en = 0;

#100 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#60 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

#100 fc50 = 1;

#2 fc50 = 0;

#1 fc50 = 1;

#60 fc50 = 0;

#2 fc50 = 1;

#1 fc50 = 0;

//Deliver: change should be 25 cents

//we should wait long enough to sale again

//Sale again

#100 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#60 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

#100 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#60 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

#100 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#60 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

#100 fc5 = 1;

#2 fc5 = 0;

#1 fc5 = 1;

#60 fc5 = 0;

#2 fc5 = 1;

#1 fc5 = 0;

//Deliver: change should be 0 cents

//we should wait long enough to operate again

//reset to test pricing mode

#100 rst = 1;

#2 rst = 0;

#1 rst = 1;

#40 rst = 0;

#2 rst = 1;

#1 rst = 0;

//Pricing

#100 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#40 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

#20 fc50 = 1;

#2 fc50 = 0;

#1 fc50 = 1;

#40 fc50 = 0;

#2 fc50 = 1;

#1 fc50 = 0;

#20 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#40 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

#20 fc5 = 1;

#2 fc5 = 0;

#1 fc5 = 1;

#40 fc5 = 0;

#2 fc5 = 1;

#1 fc5 = 0;

//Sale

#100 en = 1;

#2 en = 0;

#1 en = 1;

#40 en = 0;

#2 en = 1;

#1 en = 0;

#20 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#40 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

#20 fc50 = 1;

#2 fc50 = 0;

#1 fc50 = 1;

#40 fc50 = 0;

#2 fc50 = 1;

#1 fc50 = 0;

#20 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#40 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

#20 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#40 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

//Deliver: change should be 95 cents

//we should wait long enough to sale again

//Sale again

#500 od100 = 1;

#2 od100 = 0;

#1 od100 = 1;

#40 od100 = 0;

#2 od100 = 1;

#1 od100 = 0;

#20 fc50 = 1;

#2 fc50 = 0;

#1 fc50 = 1;

#40 fc50 = 0;

#2 fc50 = 1;

#1 fc50 = 0;

#20 tc10 = 1;

#2 tc10 = 0;

#1 tc10 = 1;

#40 tc10 = 0;

#2 tc10 = 1;

#1 tc10 = 0;

#20 fc5 = 1;

#2 fc5 = 0;

#1 fc5 = 1;

#40 fc5 = 0;

#2 fc5 = 1;

#1 fc5 = 0;

//this time change should be 0

//return to init

#100 rst = 1;

#2 rst = 0;

#1 rst = 1;

#40 rst = 0;

#2 rst = 1;

#1 rst = 0;

end

\*/