STATE MACHINE

Theoretical background

State machines are mathematical models used to describe the behavior of systems with a finite number of states, transitions, and inputs. They consist of **states** (representing conditions), **transitions** (defining state changes based on inputs or events), and **actions** that may occur during transitions or in specific states. State machines are widely used in fields like computer science, automata theory, and embedded systems to model sequential logic, control processes, and the behavior of software and hardware systems.

One example of a state machine is a vending machine. But, before you can get a refreshing beverage or a tasty snack, you must implement one!

Goal

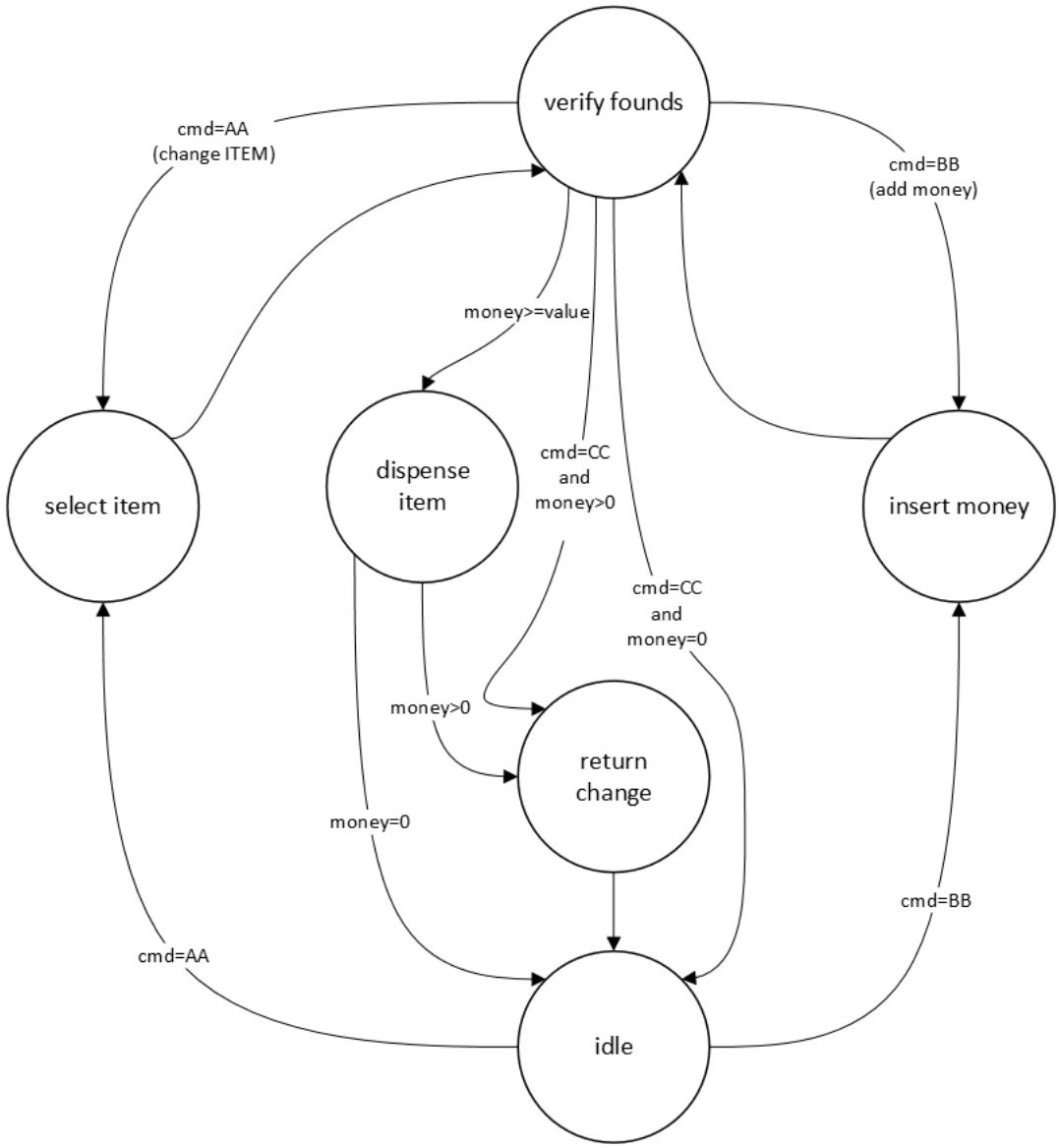
Your task is to implement a state machine provided its design in the form of a graph and interface that will be used.

You only get the outline of the **task_5** module. Inputs and outputs are declared as shown below in Table 1.

Table 1 Lists of inputs and outputs to the task_5 module.

Signal name	Signal type	Bit length
i_clk	Input	1 bit
i_rst	Input	1 bit
i_valid	Input	1 bit
i_first	Input	1 bit
i_last	Input	1 bit
i_data	Input	8 bit
o_valid	Output	1 bit
o_last	Output	1 bit
o_data	Output	16 bit

Machine graph



The initial state is **Idle** state.

Machine – initial state

Assume that item is always available.

The items that can be purchased from the vending machine, price and ID:

- Chips
    - Price: 15
    - ID: 0xA1
- Chocolate Bar
    - Price: 20
    - ID: 0xB2
- Gum
    - Price: 2
    - ID: 0xC3
- Soda
    - Price: 75
    - ID: 0xD4
- Cookies
    - Price: 18
    - ID: 0xE5

Assume that machine will always return the change even if it is zero.

Machine – how it works

Judge will not send invalid command. The sum of money is always less that 255. The state machine needs to be ready to accept the first chunk of input data, process it and send output. And after that a second chunk of input data can appear.

Below are information's about different machine states. Refer to machine graph. In bold are state numbers value. **You must use the same numbers.**

1. Idle **0x0**
    a. Machine is waiting for user interaction.
    b. Transition occurs when Action push button (0xAA) or insert money (0xBB) is taken.
2. Select Item **0x1**
    a. Users select what item they wish to purchase.
    b. Transition to Verify founds.
3. Insert Money **0x3**
    a. The user inserts money into machine.

    b. Transition to Verify Funds.
4. Verify Funds **0x4**
    a. The machine checks if the sum of inserted money is equal to or greater than the item's price.
    b. If the money is insufficient, after action insert money, it gets back to Insert Money state.
    c. If the money inserted is correct, transition to Dispense item state.
    d. If Action push button 0xAA appear then it changes state to selest item, to change declared item.
    e. If Action cancel code is applied and the money is zero then it changes state to idle.
    f. If Action cancel code is applied and money is more than zero, then it changes state to return change.
5. Dispense Item **0x5**
    a. The machine dispenses the item to the user. Item Id need to be added to output data in bits [7:0]
    b. If the money inserted equals the item's price, transition to "Idle" occurs.
    c. If the inserted money exceeds the item's price, transition to "Return Change" occurs.
6. Return Change **0x6**
    a. The machine returns the change to the user. Change need to be added to output data in bits [7:0]
    b. After returning the change, transition to "Idle" occurs.

There is always only one item purchased.

Machine – user actions

Before action comes code 0xAA and before inserting money 0xBB.

Action push button 0xAA

Action insert money 0xBB

Action cancel code 0xCC

As a user you can take these actions and are coded like this:

Example input:

{0xAA, 0xB2, 0xBB, 0x0A, 0xBB, 0x0F}

Example output:

(0x0000, 0x0100, 0x0400, 0x0300, 0x0400, 0x0300, 0x0400, 0x05B2, 0x0605)

In input the first product was selected then money was inserted.

In output state went through: idle, select item, verify founds, insert money, verify funds, dispense item and return change. The change was added to output in return change.

Interface frame – input

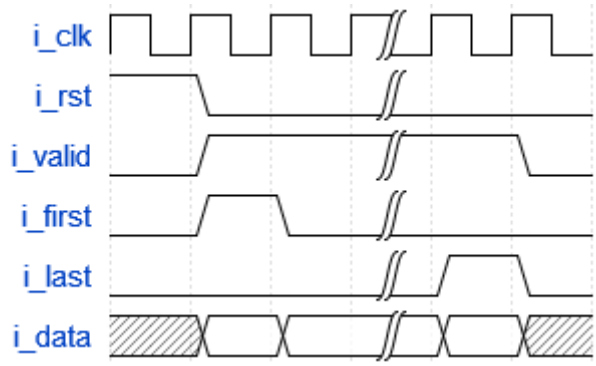


Figure 1 Input waveforms.

Interface frame – output

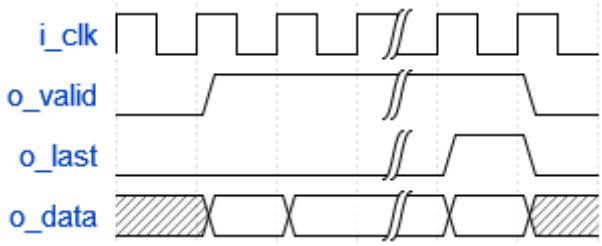


Figure 2 Output waveforms.

Output o_data is 16 bits. O_data[7:0] id change and o_data[15:8] is state. Change is added to o_data at the same time as return change state. I.e. 0x601 is return change stage number 0x6 and 0x1 of change.

Provided module

You can edit only the *task_5.sv* file. If you wish, you may add other files for supportive modules.

Evaluating the task

Module need to evaluate state machine and collect all states that there visit. After finishing, it need to send collected data and value of the change to o_data.

In Development mode, the task will be tested using a single set of data that is randomized on each run. In Evaluation mode, the task will be tested using 2 fixed sets of data, which remain the same across all runs and are identical for every team (for more information, check chapter 2.3.1 **Testing modes** in the F**PGA_Hackathon_2025_Project_Guide** document).

You earn 4 points for correctly processing each test vector. In Development mode, this value simply indicates that the task passed the test. Evaluation mode, your total base score is calculated by multiplying 4 by the number of test vectors, so you can earn up to 8 points.

Additional points may be awarded for **resource utilization** (for more information, check chapter 2.3.2 **Bonus Points** in the **FPGA_Hackathon_2025_Project_Guide** document).