سوال ۱

تبدیل هر FSM دلخواهی به کد Verilog یا VHDL که قابل سنتز باشد.

جواب سوال ۱

نحوه‌ی کارکرد قفل دیجیتالی

ورودی‌ها:

- clk: سیگنال ساعت
- rst: سیگنال ریست
- inp: سیگنال ورودی (۰ یا ۱)

خروجی:

- unlocked: این خروجی زمانی که رمز صحیح وارد شود به '۱' تنظیم می‌شود.

وضعیت‌ها:

- IDLE: وضعیت اولیه یا زمانی که رمز نادرستی وارد شده باشد.
- FIRST_BIT: اولین بیت صحیح وارد شده است.
- SECOND_BIT: دو بیت اول صحیح وارد شده است.
- UNLOCKED: وضعیت قفل باز است.

توضیحات قفل دیجیتالی

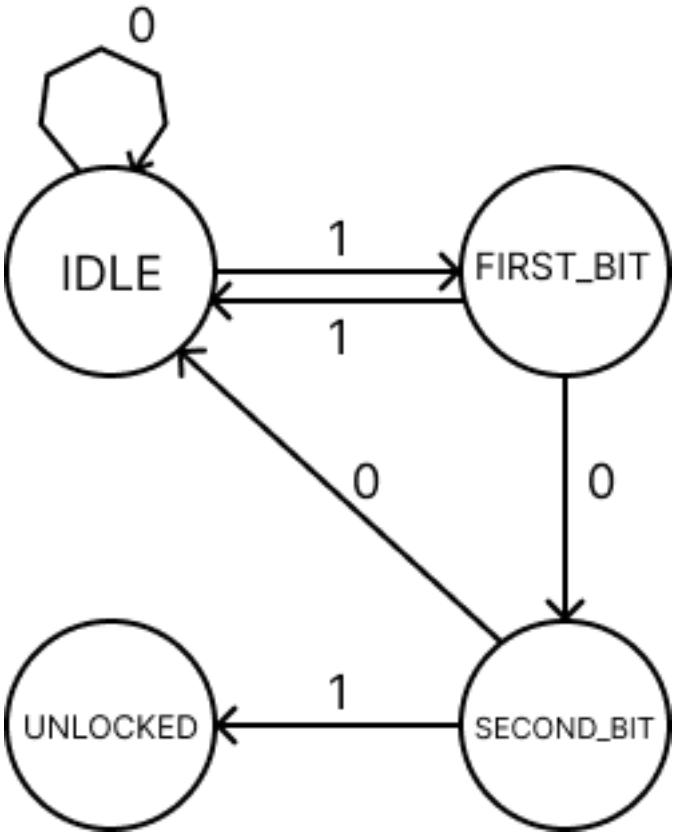
قفل دیجیتالی با استفاده از یک FSM طراحی شده است که رمز "۱۰۱" را برای باز کردن قفل می‌پذیرد. این قفل می‌تواند در موارد زیر مورد استفاده قرار گیرد:

- **واحدهای ذخیره‌سازی اطلاعات**: مانند یک فلش مموری یا یک هارد دیسک خارجی که نیاز به یک مکانیزم امنیتی ساده برای محافظت از داده‌ها دارند.
- **دستگاه‌های امنیتی**: مانند یک قفل درب الکترونیکی که با ورود یک رمز ساده باز و بسته می‌شود.
- **تجهیزات الکترونیکی**: مانند یک گجت یا یک دستگاه خاص که باید با ورود یک رمز خاص فعال یا غیرفعال شود.
- **آموزش و آزمون**: به عنوان یک مثال ساده برای آموزش مفاهیم مرتبط با هاFSM و طراحی مدارات دیجیتال.

به طور کلی، این قفل دیجیتال می‌تواند در هر جایی که نیاز به یک مکانیزم امنیتی ساده و الکترونیکی وجود داشته باشد، مورد استفاده قرار گیرد.

شکل FSM برای یک قفل دیجیتالی

در ابتدا، به دایرکتوری مورد نظر برای فعال‌سازی TrueTime در متلب مراجعه کردیم با این دستور:



کد JSON همان FSM

```json
{
    "states": ["IDLE", "FIRST_BIT", "SECOND_BIT", "UNLOCKED"],
    "inputs": ["clk", "rst", "inp"],
    "output": "unlocked",
    "transitions": {
        "IDLE": {
            "1": "FIRST_BIT",
            "0": "IDLE",
            "rst": "IDLE"
        },
        "FIRST_BIT": {
            "0": "SECOND_BIT",
            "1": "IDLE",
            "rst": "IDLE"
        },
        "SECOND_BIT": {
            "1": "UNLOCKED",
            "0": "IDLE",
            "rst": "IDLE"
        },
        "UNLOCKED": {
            "rst": "IDLE"
        }
    },
    "initial_state": "IDLE",
    "final_state": "UNLOCKED"
}
```

- در وضعیت FIRST_BIT اگر $inp = ۱$ باشد به IDLE بر می‌گردیم.
- از وضعیت UNLOCKED فقط با $rst = ۱$ به IDLE بر می‌گردیم.

کد وریلاگ

```
module digital_lock(
    input clk, rst, inp,
    output reg unlocked
);

typedef enum {
    IDLE, FIRST_BIT, SECOND_BIT, UNLOCKED
} state_t;

reg [1:0] current_state, next_state;

always @(posedge clk or posedge rst) {
    if (rst) current_state <= IDLE;
    else current_state <= next_state;
}

always @(current_state, inp) begin
    case(current_state)
        IDLE: begin
            if (inp == 1'b1) next_state = FIRST_BIT;
            else next_state = IDLE;
            unlocked = 1'b0;
        end

        FIRST_BIT: begin
            if (inp == 1'b0) next_state = SECOND_BIT;
            else next_state = IDLE;
            unlocked = 1'b0;
        end

        SECOND_BIT: begin
            if (inp == 1'b1) next_state = UNLOCKED;
            else next_state = IDLE;
            unlocked = 1'b0;
        end

        UNLOCKED: begin
            next_state = UNLOCKED;
            unlocked = 1'b1;
        end
    endcase
end

endmodule
```

توضیح کد وریلاگ

ماژول:

digital_lock: این ماژول سه ورودی و یک خروجی دارد.

ورودی‌ها:

الف) **clk**: سیگنال ساعت

ب) **rst**: سیگنال ریست

ج) **inp**: سیگنال ورودی که می‌تواند ۰ یا ۱ باشد

خروجی:

الف) **unlocked**: وضعیت قفل. وقتی قفل باز است، این خروجی ۱ است.

تعریف‌ها:

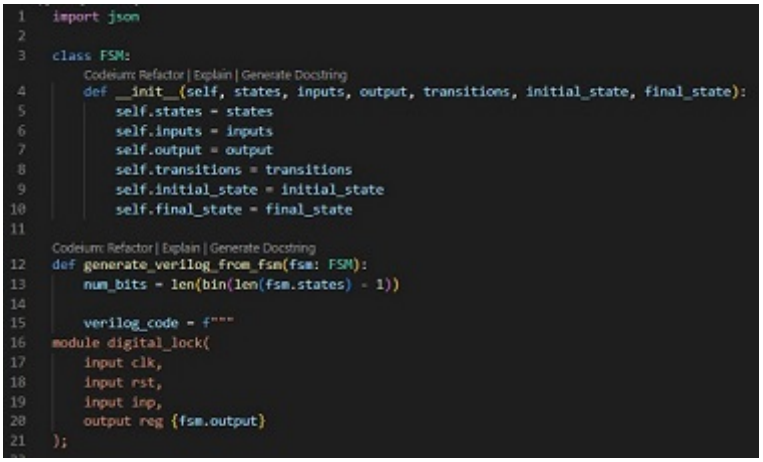
- **state_t**: این enum چهار وضعیت مختلف FSM را تعریف می‌کند:
  IDLE ، FIRST_BIT ، SECOND_BIT و UNLOCKED.
- **current_state و next_state**: این دو متغیر وضعیت فعلی و وضعیت بعدی FSM را نگه می‌دارند.

پیاده‌سازی FSM:

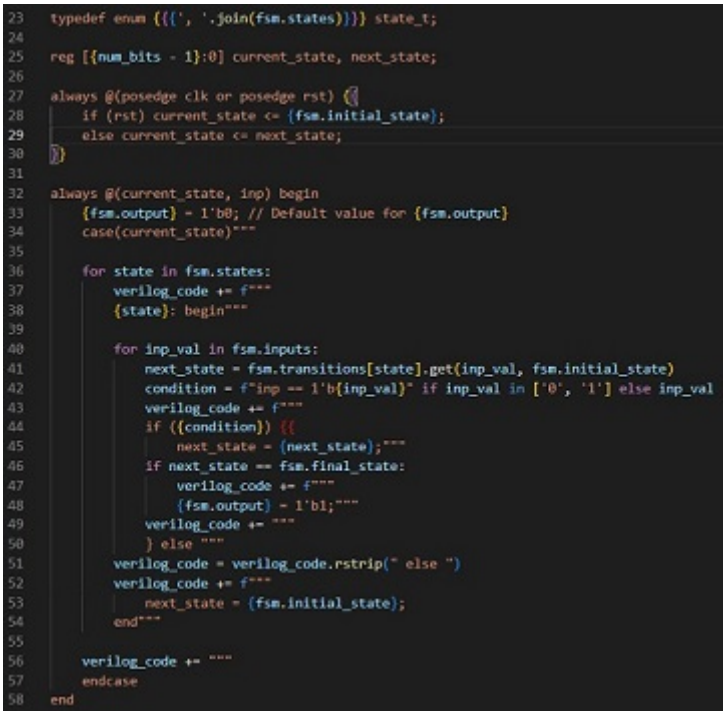
الف) در هر لبه مثبت clk ، `current_state` به `next_state` تغییر می‌کند، مگر زمانی که `rst` فعال باشد که `current_state` به IDLE باز می‌گردد.

ب) بر اساس وضعیت فعلی و ورودی، وضعیت بعدی و خروجی تعیین می‌شوند.

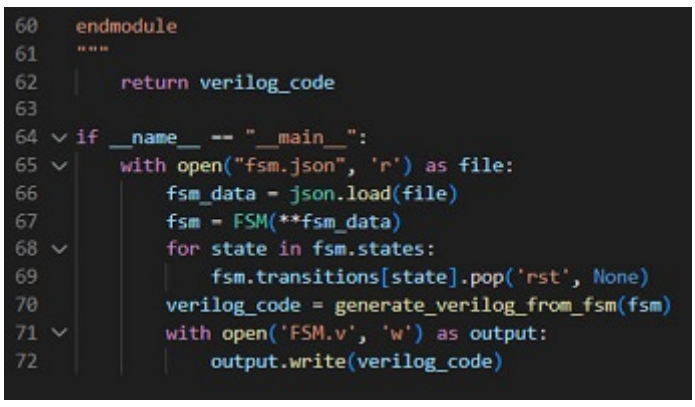
کد پایتون Generator کد وریلاگ از هر کد json یک FSM



```python
import json

class FSM:
    Codeium: Refactor | Explain | Generate Docstring
    def __init__(self, states, inputs, output, transitions, initial_state, final_state):
        self.states = states
        self.inputs = inputs
        self.output = output
        self.transitions = transitions
        self.initial_state = initial_state
        self.final_state = final_state

Codeium: Refactor | Explain | Generate Docstring
def generate_verilog_from_fsm(fsm: FSM):
    num_bits = len(bin(len(fsm.states) - 1))

    verilog_code = f"""
module digital_lock(
    input clk,
    input rst,
    input inp,
    output reg {fsm.output}
);
```



```
    typedef enum {{{', '.join(fsm.states)}}} state_t;

    reg [{num_bits - 1}:0] current_state, next_state;

    always @(posedge clk or posedge rst) {{
        if (rst) current_state <= {fsm.initial_state};
        else current_state <= next_state;
    }}

    always @(current_state, inp) begin
        {fsm.output} = 1'b0; // Default value for {fsm.output}
        case(current_state)"""

        for state in fsm.states:
            verilog_code += f"""
            {state}: begin"""

            for inp_val in fsm.inputs:
                next_state = fsm.transitions[state].get(inp_val, fsm.initial_state)
                condition = f"inp == 1'b{inp_val}" if inp_val in ['0', '1'] else inp_val
                verilog_code += f"""
                if ({condition}) {{
                    next_state = {next_state};"""
                if next_state == fsm.final_state:
                    verilog_code += f"""
                    {fsm.output} = 1'b1;"""
                verilog_code += """
                } else """
            verilog_code = verilog_code.rstrip(" else ")
            verilog_code += f"""
                next_state = {fsm.initial_state};
            end"""

        verilog_code += """
        endcase
    end
```



```
    endmodule
    """
        return verilog_code

if __name__ == "__main__":
    with open("fsm.json", 'r') as file:
        fsm_data = json.load(file)
        fsm = FSM(**fsm_data)
        for state in fsm.states:
            fsm.transitions[state].pop('rst', None)
        verilog_code = generate_verilog_from_fsm(fsm)
        with open('FSM.v', 'w') as output:
            output.write(verilog_code)
```

کد پایتون Generator توضیح کد وریلاگ از هر کد json یک FSM

کد معرفی شده، یک ماشین حالت محدود Machine State (Finite ـ (FSM را از یک فایل *JSON* می‌خواند و بر اساس آن یک ماژول *Verilog* ایجاد می‌کند.

کلاس FSM

این کلاس یک ماشین حالت محدود را نمایان می‌کند. ویژگی‌های اصلی آن شامل : حالت‌ها (states) ، ورودی‌ها (inputs) ، خروجی (output) ، گذارها (transitions) ، حالت اولیه (initial_state) و حالت نهایی (final_state) است.

تابع generate_verilog_from_fsm

این تابع بر اساس یک شئ از نوع FSM یک ماژول Verilog ایجاد می‌کند.

قسمت اجرایی

در این بخش، فایل JSON مربوط به FSM خوانده می‌شود، و با استفاده از تابع بالا یک ماژول Verilog ایجاد و در یک فایل با نام *FSM.v* ذخیره می‌شود.

استفاده از این کد پایتون

آن فایل json اولیه که مربوط به FSM قفل الکترونیک بود را به عنوان ورودی به این کد می‌دهیم و خروجی، کد وریلاگی‌ست که می‌خواهیم.