

BAB 6

FINITE STATE MACHINE

Pada bab ini, praktikan akan mempelajari Finite State Machine (FSM) dan bagaimana mengimplementasikannya dalam FPGA Nexys A7 menggunakan VHDL. FSM adalah model matematika yang digunakan untuk mendeskripsikan sistem berbasis keadaan (state), yang banyak digunakan dalam kontrol digital, komunikasi, dan otomasi.

Tujuan

| Tujuan | Penjelasan |
|---|--|
| Memahami konsep Finite State Machine (FSM) | Praktikan diharapkan memahami konsep dasar FSM, bagaimana cara kerja sistem berbasis keadaan, serta penerapannya dalam sistem digital. |
| Memahami komponen utama dalam FSM | Praktikan diharapkan memahami komponen utama dalam FSM seperti states, transitions, inputs, dan outputs. |
| Memahami tipe-tipe FSM (Moore & Mealy Machine) | Praktikan diharapkan memahami perbedaan antara Moore dan Mealy Machine, termasuk representasi dengan state diagram dan state table. |
| Memahami metode state encoding | Praktikan diharapkan memahami cara mengkodekan keadaan (state encoding) dalam FSM untuk efisiensi implementasi pada FPGA. |

Persyaratan

Disarankan praktikan menggunakan hardware dan software sesuai pada dokumentasi ini. Apabila terdapat versi hardware atau software yang cukup lama dari versi yang direkomendasikan maka sebaiknya bertanya kepada Asisten Mengajar Shift.

HARDWARE YANG DIBUTUHKAN PRAKTIKUM

PC / Laptop

FPGA Board Nexys A7 dan Nexys 4

Kabel Power USB

SOFTWARE YANG DIBUTUHKAN PRAKTIKUM

Vivado Design Suite



Diusahakan untuk memakai **Versi** dan **Aplikasi** yang sama agar tidak terjadinya kesalahan yang tidak diinginkan!

Teori

6.1 Pengertian

Dalam sistem digital, terdapat dua jenis rangkaian dasar. **Jenis pertama adalah rangkaian logika kombinasional.** Pada rangkaian logika kombinasional, keluaran hanya bergantung pada masukan. Contoh dari rangkaian logika kombinasional antara lain **adder, encoder, dan multiplexer.** Pada adder, misalnya, keluarannya hanyalah hasil penjumlahan dari masukan; **tidak peduli apa masukan atau keluaran sebelumnya.**

Jenis kedua dari rangkaian logika digital adalah rangkaian logika sekuensial. Pada rangkaian logika sekuensial, keluaran tidak hanya bergantung pada masukan, tetapi juga pada **keadaan (state)** sistem saat ini (yaitu nilai keluaran dan sinyal atau variabel internal). Rangkaian logika sekuensial dapat berupa rangkaian sederhana seperti counter yang berpindah dari satu state ke state lainnya dalam urutan dasar (misalnya 0,1,2,3... kemudian kembali ke 0,1,2,3...) hingga rangkaian berskala besar seperti mikroprosesor dengan jutaan state yang berbeda atau lebih. Sistem logika sekuensial merupakan Finite State Machine (FSM). Sebagai FSM, sistem ini terdiri atas sekumpulan state, sejumlah masukan, sejumlah keluaran, serta sekumpulan aturan untuk berpindah dari satu state ke state lainnya.

6.2 Komponen Finite State Machine

Sebuah Finite State Machine terdiri dari beberapa komponen utama yaitu sebagai berikut:

- **Finite States**

Finite states adalah mode atau kondisi yang berbeda dalam suatu sistem. Setiap

state merepresentasikan perilaku tertentu. Dalam representasi sistem digital, state biasanya digambarkan menggunakan simbol atau label.

- **State Transitions**

Dalam konteks FSM, transisi state didefinisikan sebagai perubahan dari satu state ke state lainnya. Perubahan ini terjadi berdasarkan input atau kondisi tertentu. Transisi state umumnya dipicu oleh suatu peristiwa (event) yang terkait dengan aturan atau kondisi, yang menentukan state berikutnya dari sistem.

- **State Diagram**

Transisi state dan perilaku dari FSM dapat direpresentasikan dalam bentuk grafis yang dikenal sebagai diagram state. Diagram ini membantu memvisualisasikan bagaimana sistem berpindah dari satu state ke state lain.

- **Inputs**

Masukan pada FSM adalah sinyal eksternal yang memicu transisi state dalam sistem. Masukan ini bisa berasal dari sensor, perangkat input pengguna seperti mikrofon, keyboard, dan lain-lain.

- **Outputs**

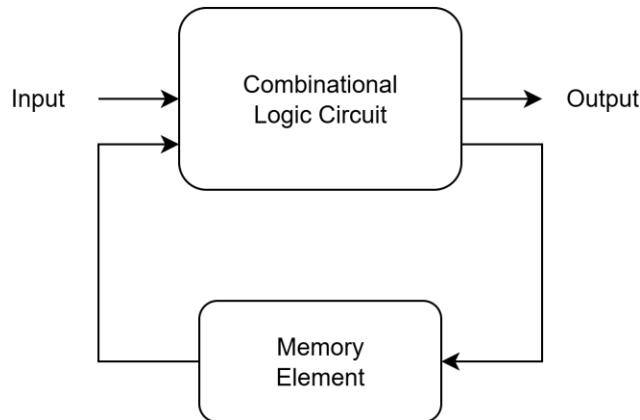
Keluaran adalah hasil yang dihasilkan oleh sistem berdasarkan masukan dan state saat ini. Keluaran ini dapat digunakan untuk memicu suatu peristiwa, mengendalikan aktuator, atau memberikan umpan balik ke lingkungan eksternal.

63. Tipe Finite State Machine

Finite State Machine terdiri dari dua tipe yaitu Mealy State Machine dan Moore State Machine yang memiliki cara kerja yang berbeda dari setiap tipe :

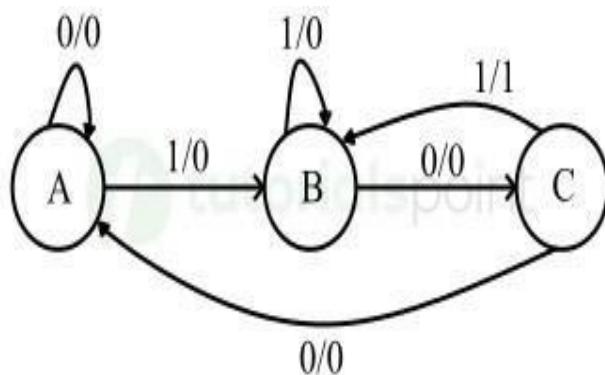
63.1 Mealy State Machine

Sebuah Finite State Machine disebut sebagai Mealy State Machine apabila keluarannya bergantung pada masukan saat ini dan juga state saat ini. Diagram blok dari Mealy State Machine ditunjukkan pada gambar berikut :



Gambar 6.1 Blok Diagram Mealy State Machine

Seperti yang terlihat pada gambar, terdapat dua bagian utama dalam Mealy State Machine, yaitu rangkaian logika kombinasional dan elemen memori. Elemen memori berfungsi untuk memberikan sebagian keluaran sebelumnya dan state saat ini sebagai masukan ke rangkaian logika kombinasional. Berdasarkan masukan saat ini dan state saat ini, Mealy State Machine menghasilkan keluaran. Oleh karena itu, keluaran hanya akan valid pada transisi positif atau negatif dari sinyal clock.

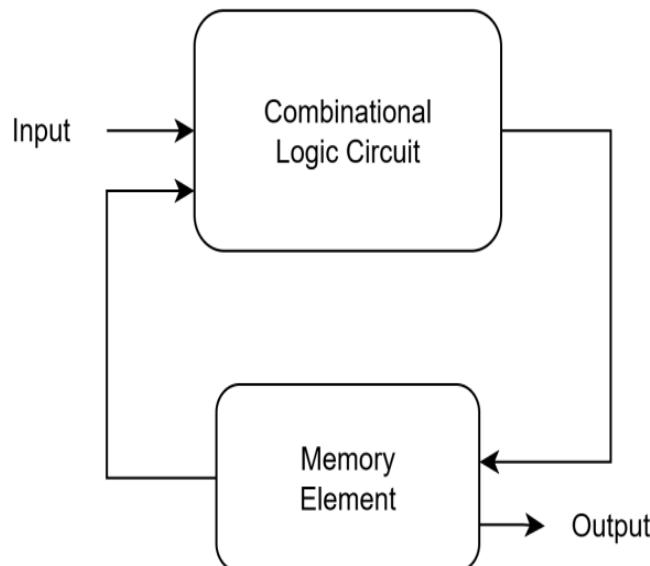


Gambar 6.2 State Diagram Mealy

Pada gambar di atas, terdapat tiga state, yaitu A, B, dan C. State-state ini diberi label di dalam lingkaran, dan setiap lingkaran merepresentasikan satu state. Transisi antar state ditunjukkan dengan garis berarah. Pada gambar, notasi 0 / 0, 1 / 0, dan 1 / 1 menunjukkan masukan / keluaran. Dari setiap state, terdapat dua kemungkinan transisi state berdasarkan nilai masukan. Secara umum, jumlah state yang dibutuhkan pada Mealy State Machine lebih sedikit atau sama dengan jumlah state yang dibutuhkan pada Moore State Machine. Untuk setiap Mealy State Machine, selalu ada Moore State Machine yang ekuivalen.

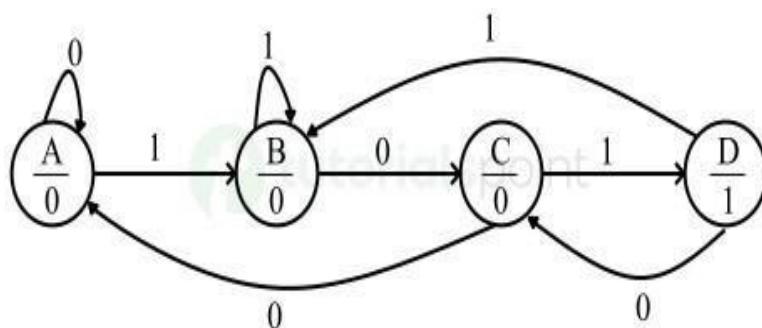
63.1. Moore State Machine

Sebuah Finite State Machine disebut sebagai Moore State Machine apabila keluarannya hanya bergantung pada state saat ini. Diagram blok dari Mealy State Machine ditunjukkan pada gambar berikut :



Gambar 6.3 Blok Diagram

Seperti yang ditunjukkan pada gambar, terdapat dua bagian utama dalam Moore State Machine, yaitu rangkaian logika kombinasional dan elemen memori. Dalam hal ini, masukan saat ini dan state saat ini menentukan state berikutnya. Berdasarkan state berikutnya inilah Moore State Machine menghasilkan keluaran. Oleh karena itu, keluaran pada Moore State Machine hanya akan valid setelah terjadi transisi state.



Gambar 6.4 State Diagram Moore

Pada gambar di atas, terdapat empat state, yaitu A, B, C, dan D. State- state ini beserta keluarannya diberi label di dalam lingkaran. Pada transisi antar state, hanya nilai masukan yang ditunjukkan. Dari setiap state, terdapat dua kemungkinan transisi

berdasarkan nilai masukan. Secara umum, jumlah state yang dibutuhkan pada Moore State Machine lebih banyak atau sama dengan jumlah state yang dibutuhkan pada Mealy State Machine. Untuk setiap Moore State Machine, selalu ada Mealy State Machine yang ekuivalen. Oleh karena itu, pemilihan apakah menggunakan Mealy atau Moore dapat ditentukan berdasarkan kebutuhan sistem.

64. State Encoding

Dalam VHDL, Finite State Machine (FSM) dapat dituliskan dengan berbagai cara. State Encoding pada sebuah FSM memengaruhi kinerjanya dalam hal kecepatan, penggunaan sumber daya (register, logika), dan juga konsumsi daya.

Algoritma pengkodean state meliputi:

- **Binary encoding** : setiap state diwakili dengan angka biner yang terenkode, misalnya: "000", "001", "010", "011", "100" ...
- **One-hot encoding** : setiap state direpresentasikan dengan pola bit yang hanya memiliki satu '1', misalnya: "000001", "000010", "000100", "001000", "010000".
- **Gray coding** : pengkodean di mana setiap transisi antar state hanya berbeda satu bit, misalnya: "000", "001", "011", "010", "110".

Pengkodean yang dipilih sangat bergantung pada sifat desain:

- **Binary encoding** meminimalkan panjang vektor state, sehingga cocok untuk desain berbasis CPLD (Complex Programmable Logic Device).
- **One-hot encoding** biasanya lebih cepat, menggunakan lebih banyak register tetapi lebih sedikit logika. Hal ini membuat one-hot encoding lebih cocok untuk desain berbasis FPGA, di mana register biasanya banyak digunakan.
- **Gray encoding** dapat mengurangi glitches pada FSM yang memiliki cabang terbatas atau bahkan tidak memiliki cabang.

```

type t_state is (IDLE, START, RUN, DONE);
signal state: t_state;
attribute fsm_state : string;
attribute fsm_state of state : signal is "ONE_HOT";
case state is
when IDLE =>
    state <= START;
when START =>
    state <= RUN;
when RUN =>
    state <= DONE;
when DONE =>
    state <= IDLE;

```

Contoh FSM pada kode di atas bekerja dengan cara berpindah dari satu state ke state berikutnya secara berurutan dan terus berulang. Awalnya, sistem berada pada state IDLE, kemudian pada siklus berikutnya ia akan bertransisi ke state START. Setelah itu, FSM melanjutkan ke state RUN, lalu berpindah ke state DONE. Dari state DONE, FSM kembali lagi ke state IDLE sehingga membentuk suatu siklus yang berulang tanpa henti. Dengan demikian, FSM ini dapat dikatakan sebagai mesin keadaan yang berjalan dalam pola lingkaran, di mana setiap state memiliki transisi yang sudah pasti menuju state berikutnya sesuai urutan yang telah ditentukan.

65. Aplikasi Finite State Machine

Dalam bidang elektronika digital dan ilmu komputer, finite state machine banyak digunakan dalam berbagai aplikasi karena kemampuannya dalam memodelkan sistem logika sekuensial secara efektif. Berikut adalah beberapa contoh penerapannya :

- Finite state machine umum digunakan dalam perancangan dan implementasi berbagai jenis rangkaian logika sekuensial, seperti digital counters, timers, control units, dan lain sebagainya.
- Finite state machine digunakan dalam sistem kendali digital untuk mengontrol dan mengatur perilaku sistem otomatis yang kompleks, seperti robot, sistem kendali industri, serta sistem otomasi.
- Finite state machine digunakan dalam implementasi protokol komunikasi, misalnya protokol jaringan, serta sistem digital berbasis state seperti transmisi data dan konverter protokol.
- Finite state machine juga digunakan dalam bidang pengembangan perangkat lunak untuk memodelkan dan mendefinisikan perilaku sistem berbasis state pada

aplikasi, membangun user interfaces, mengimplementasikan mekanika permainan (game mechanics), serta mengembangkan sistem manajemen alur kerja (workflow management systems).

- Pada FPGA (Field Programmable Gate Array), finite state machine banyak digunakan untuk:
 - Mengendalikan alur data dalam sistem berbasis VHDL/Verilog, misalnya dalam komunikasi UART, SPI, atau I²C.
 - Membuat kontroler hardware, seperti kontroler memori, kontroler display (misalnya seven-segment atau LCD), dan modul kontrol sensor.
 - Implementasi algoritma sekuensial, misalnya pengkodean (encoding), penguraian data (decoding), serta protokol transmisi data.

Praktek

1. Penerapan Finite State Machine pada Seven Segment Untuk menampilkan urutan angka secara bergantian

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity sevenseg_fsm is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        an : out STD_LOGIC_VECTOR(7 downto 0);
        seg : out STD_LOGIC_VECTOR(6 downto 0));
end sevenseg_fsm; -- FSM: pindah ke digit berikutnya
process(clk, reset)
begin
    signal state : integer := 0;
    begin
        if reset = '1' then
            state <= 0;
        elsif rising_edge(clk) then
            if state = 7 then
                state <= 0;
            else
                state <= state + 1;
            end if;
        end if;
    end process;

    process(state)
    begin
        an <= (others => '1');
        an(state) <= '0';
    end process;
end pr;
process(state)
begin
    case state is
        when 0 => digitval <= 1;
        when 1 => digitval <= 2;
        when 2 => digitval <= 3;
        when 3 => digitval <= 4;
        when 4 => digitval <= 5;
        when 5 => digitval <= 6;
        when 6 => digitval <= 7;
        when 7 => digitval <= 8;
        when others => digitval <= 0;
    end case;
end process;
```

```

process(digitval)
begin
    case digitval is
        when 0 => seg <= "1000000"; -- 0
        when 1 => seg <= "1111001"; -- 1
        when 2 => seg <= "0100100"; -- 2
        when 3 => seg <= "0110000"; -- 3
        when 4 => seg <= "0011001"; -- 4
        when 5 => seg <= "0010010"; -- 5
        when 6 => seg <= "0000010"; -- 6
        when 7 => seg <= "1111000"; -- 7
        when 8 => seg <= "0000000"; -- 8
        when 9 => seg <= "0010000"; -- 9
        when others => seg <= "1111111"; -- blank
    end case;
end process;

end Behavioral;

```

Pinout Nexys A7

| All ports (17) | | | | | | | | | | | | | | | |
|------------------|---------|-----|-----|---|---|------------|---------|---|-------|----|---|---|------|---|-----------|
| | | | | | | | | | | | | | | | |
| ▼ | an (8) | OUT | | | ✓ | (Multiple) | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | an[7] | OUT | U13 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | an[6] | OUT | K2 | ▼ | ✓ | 35 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | an[5] | OUT | T14 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | an[4] | OUT | P14 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | an[3] | OUT | J14 | ▼ | ✓ | 15 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | an[2] | OUT | T9 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | an[1] | OUT | J18 | ▼ | ✓ | 15 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | an[0] | OUT | J17 | ▼ | ✓ | 15 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| ▼ | seg (7) | OUT | | | ✓ | (Multiple) | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | seg[6] | OUT | L18 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | seg[5] | OUT | T11 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | seg[4] | OUT | P15 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | seg[3] | OUT | K13 | ▼ | ✓ | 15 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | seg[2] | OUT | K16 | ▼ | ✓ | 15 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | seg[1] | OUT | R10 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| | seg[0] | OUT | T10 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | 12 | ▼ | ✓ | NONE | ▼ | FP_VTT_50 |
| Scalar ports (2) | | | | | | | | | | | | | | | |
| | clk | IN | E3 | ▼ | ✓ | 35 | LVCMS18 | ▼ | 1.800 | | | | NONE | ▼ | NONE |
| | reset | IN | N17 | ▼ | ✓ | 14 | LVCMS18 | ▼ | 1.800 | | | | NONE | ▼ | NONE |

2. Percobaan Finite State Machine menggunakan Modul Traffic Light

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TrafficLightFSM is
    Port (
        clk      : in STD_LOGIC;
        reset    : in STD_LOGIC;
        green    : out STD_LOGIC;
        yellow   : out STD_LOGIC;
        red      : out STD_LOGIC
    );
end TrafficLightFSM;

architecture Behavioral of TrafficLightFSM is
    type state_type is (S_RED, S_YELLOW, S_GREEN);
    signal state, next_state : state_type;
    signal counter : INTEGER range 0 to 100000000 := 0;

begin
    process (clk, reset)
    begin
        if reset = '1' then
            state <= S_RED;
            counter <= 0;
        elsif rising_edge(clk) then
            if counter = 50000000 then
                state <= next_state;
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    process (state, counter)
    begin
        case state is
            when S_RED =>
                red <= '1';
                green <= '0';
                yellow <= '0';
                if counter = 5000000 then -- 5 detik
                    next_state <= S_YELLOW;
                else
                    next_state <= S_RED;
                end if;

            when S_YELLOW =>
                red <= '0';
                green <= '0';
                yellow <= '1';
                if counter = 5000000 then -- 5 detik
                    next_state <= S_GREEN;
                else
                    next_state <= S_YELLOW;
                end if;

            when S_GREEN =>
                red <= '0';
                green <= '1';
                yellow <= '0';
                if counter = 5000000 then -- 5 detik
                    next_state <= S_RED;
                else
                    next_state <= S_GREEN;
                end if;
        end case;
    end process;
end Behavioral;
```

Pinout Nexys A7

▼ All ports (5)

 ▼ Scalar ports (5)

| | | | | | | | | | | | | | | | | |
|--------|-----|-----|---|---|----|------------|---|-------|--|----|---|------|------|------|-----------|---|
| clk | IN | E3 | ▼ | ✓ | 35 | LVC MOS33* | ▼ | 3.300 | | | | NONE | ▼ | NONE | ▼ | |
| green | OUT | D14 | ▼ | ✓ | 15 | LVC MOS33* | ▼ | 3.300 | | 12 | ▼ | ▼ | NONE | ▼ | FP_VTT_50 | ▼ |
| red | OUT | G16 | ▼ | ✓ | 15 | LVC MOS33* | ▼ | 3.300 | | 12 | ▼ | ▼ | NONE | ▼ | FP_VTT_50 | ▼ |
| reset | IN | N17 | ▼ | ✓ | 14 | LVC MOS33* | ▼ | 3.300 | | | | NONE | ▼ | NONE | ▼ | |
| yellow | OUT | F16 | ▼ | ✓ | 15 | LVC MOS33* | ▼ | 3.300 | | 12 | ▼ | ▼ | NONE | ▼ | FP_VTT_50 | ▼ |