Máquina de Estados em Systemverilog

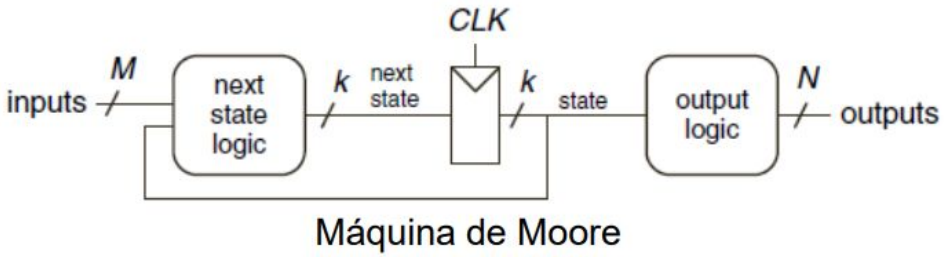
Por Victor Emanuel Farias da Costa Borges - victor.borges@ccc.ufcg.edu.br

O que é uma máquina de estados?

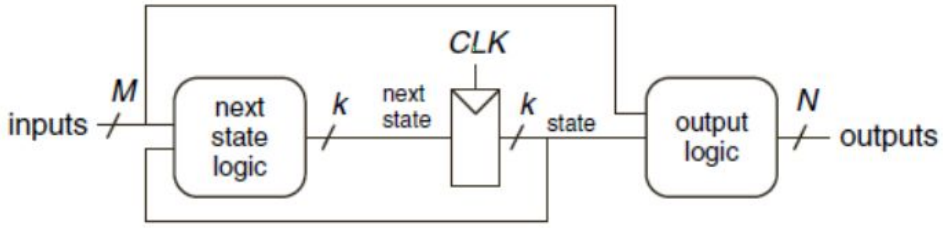
- Modelo matemático usado para representar programas de computadores ou **circuitos lógicos**.
- A máquina possui um conjunto de **estados**.
- Uma **transição** indica uma mudança de estado e é descrita por uma condição que precisa ser realizada para que a transição ocorra.
- A máquina está em apenas um estado por vez, este estado é chamado de **estado atual**.

FSM – Finite State Machine

- A máquina de estados pode ser implementada de duas maneiras equivalentes entre si:
  - Máquina de Moore
  - Máquina de Mealy



Máquina de Moore



Máquina de Mealy

Implementando FSM em Systemverilog

- O nível de abstração da FSM implementada em Systemverilog é mais alto que a teoria.

- Não diferenciamos Moore ou Mealy!

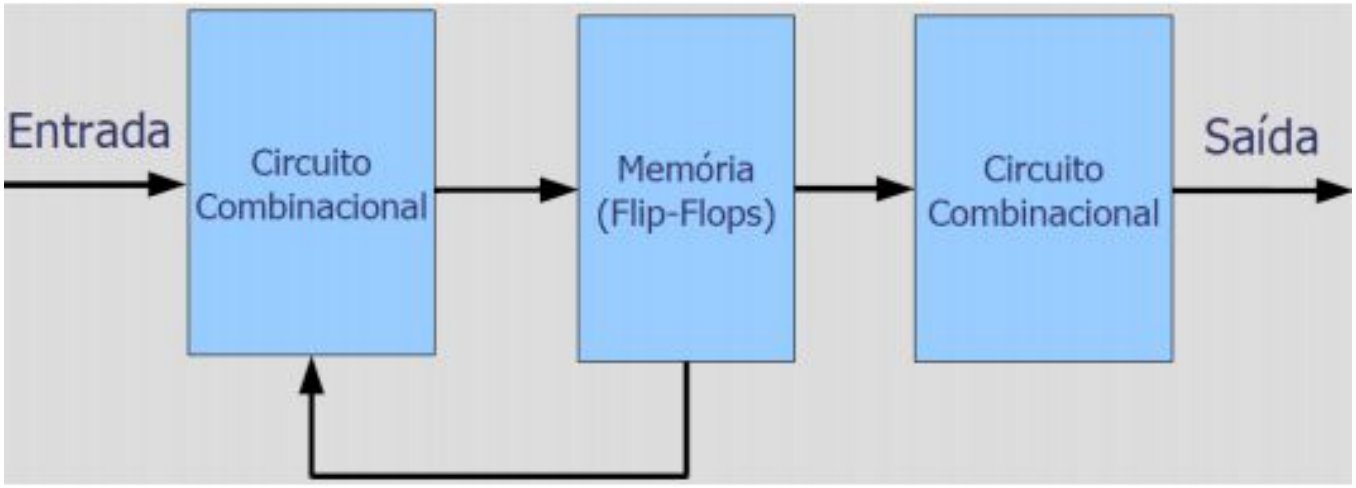
Implementando FSM em Systemverilog

- Como implementar uma máquina de estados em Systemverilog?

Implementando FSM em Systemverilog

- Como implementar uma máquina de estados em Systemverilog?

- Lembrar do “formato” da FSM



Implementando FSM em Systemverilog

- A **entrada** da FSM será composta de um **circuito combinacional** com todas as portas de entrada.

- Na prática: **always_comb** com **switches**

Implementando FSM em Systemverilog

- A **memória** da FSM será composta de um **circuito sequencial** em que as mudanças de estados serão implementadas.
- Na prática: **always_ff** com **clock atrasado (lento)**

- **Mas como serão feitos os estados?**

Implementando FSM em Systemverilog

- Solução I: **parameter**

```
parameter state_0 = 0, state1_ = 1, state_2 = 2, state_3 = 3

always_ff @(posedge clock) ...
```

- **Importante!** Criar uma variável que guarde o **estado atual**.

Implementando FSM em Systemverilog

- Solução II: **enum logic**

```
enum logic [2:0] { reset_state,
    read_1_zero, read_1_one,
    read_2_zero, read_2_one }
 state;
```

Implementando FSM em Systemverilog

- **E como serão feitas as transições dos estados?**

- Solução: **case** e **condicionais**

```
case (estado_atual)
    state_0: if ... estado_atual <= state_1
    state_1: ...
     ...
```

Implementando FSM em Systemverilog

- A **saída** da FSM será composta de um **circuito combinacional** com todas as variáveis de saída.

- Na prática: **always_comb** com **LEDs**

Implementando FSM em Systemverilog

- Exemplo:

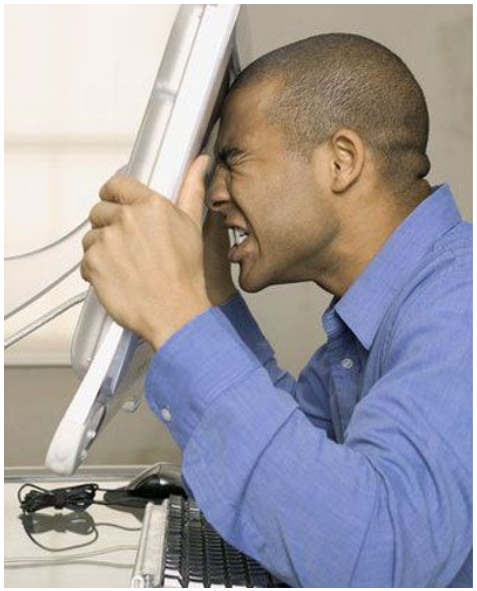
```
logic [1:0] state;

always_comb ...

parameter q0 = 0, q1 = 1, q2 = 2
always_ff @(posedge clk)
    if (reset) begin
        state <= q0;
        . . .
    end
    else
        unique case (state)
          q0: if (…) state <= q1;
          q1: ...
          . . .
        endcase

  always_comb ...
```

É importante **interpretar** antes de **implementar**!



Exercícios

1. Implemente o sistema para a cancela de um estacionamento onde o motorista pressiona um botão para a cancela abrir e depois solta o botão para a cancela fechar. O número máximo de carros no estacionamento é de 10 carros que deve ser mostrado nos LEDs. Considere que existe duas cancelas, uma de entrada e outra de saída.

- Entradas: cancela 1 – SWI[0] , cancela 2 - SWI[7]
- Saídas: número de carros – LED[6:3], cancela aberta 1 – LED[0], cancela aberta 2 – LED[1], clock - LED[7]

Exercícios

2. Implemente o circuito de um caixa eletrônico. No reset, as saídas ficam em 0. Depois do usuário inserir o cartão no caixa, ele precisa colocar o código de acesso, formado pelos valores 1, 3 e 7. O usuário pode demorar quanto tempo quiser para iniciar a colocação do código e pode demorar o quanto tempo quiser de um valor para o próximo, mas precisa pelo menos demorar 1 ciclo de clock para cada valor. Depois do código estiver recebido corretamente, o dinheiro sai. A saída do dinheiro demora 1 ciclo de clock. Após três tentativas fracassadas de colocar o código correto, o cartão é destruído. Isso demora 1 ciclo de clock.

- Entradas: clock – 1 Hz, aparecendo em LED[7], reset – assíncrono em SWI[0], cartão – o usuário inseriu um cartão no caixa em SWI[1], código de acesso – valores de 0 à 7 em SWI[6:4]
- Saídas: dinheiro – saída de dinheiro em LED[0], destrói – destruição do cartão em LED[1]

Exercícios

3. Implemente o circuito de controle da bomba de filtro de uma piscina. A bomba pode ser alimentada por painéis solares ou pela rede elétrica. A bomba deve ser ligada para prover filtragem da piscina em média 1 segundo a cada 2 segundos.

- Entradas: clock – 1 Hz, aparecendo em LED[7], reset – assíncrono em SWI[0], sol – incidência solar suficiente em SWI[1]
- Saídas: painel – liga bomba aos paineis solares em LED[0], rede – liga bomba na rede elétrica em LED[1]

No reset, as saídas ficam em 0. Quando houver incidência solar suficiente, a bomba deve ser ligada aos painéis solares durante 1 segundo a cada 2 segundos (1 segundo ligado, 1 segundo desligado). Quando não houver incidência solar suficiente, a bomba pode ficar desligada. Se após 2 segundos desligado a incidência voltar a ficar suficiente, a bomba deve ser ligada durante 2 segundos. Se após 3 segundos desligado a incidência solar voltar a ficar suficiente, a bomba deve ser ligada durante 3 segundos. Se a bomba não pode ser ligada nos painéis solares durante mais do que 3 segundos, ligue-a na rede. Quando a incidência solar voltar a ser suficiente, volte a operar pelos painéis solares.

Referências

- Slides da professora Joseana Fechine - http://www.dsc.ufcg.edu.br/~joseana/OAC120172.html

- Slides do professor Elmar Melcher - http://labarc.ufcg.edu.br/oac/index.php?n=OAC.Loac