Projetos de domínio de relógio único

Redefinição e inicialização

Modelando FSMs em VHDL

AULA 4

IOUL IIA SKL I AROVA

Projetos de domínio de relógio único

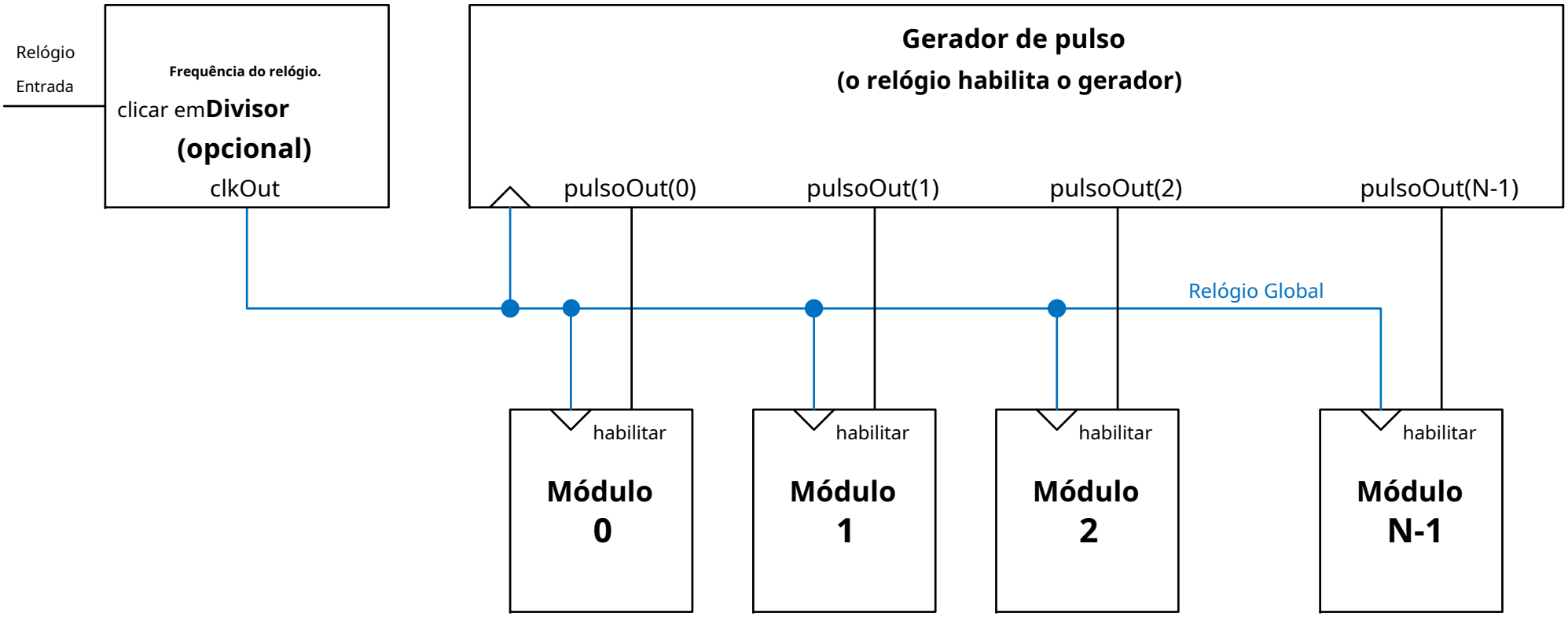
A **domínio do relógio** é o subconjunto dos componentes do sistema que são sincronizados por um único sinal de clock.

A utilização de dois ou mais domínios de relógio em um sistema é frequentemente necessária, mas pode levar a problemas complexos de temporização.

Recomendação: em todos os seus projetos você deve:

- Use apenas o sinal de clock “clk”, ou outro clock derivado dele (usando um divisor de frequência de clock ou um IP de clock).
- Use um único sinal de clock em conjunto com pulsos de habilitação para sincronizar/sequenciar operações mais lentas.
- Todos os componentes são sincronizados pelo mesmo sinal de clock e cada um possui sua(s) própria(s) habilitação(ões).

Domínio de relógio único com habilitações



Exemplo de gerador de pulso

```
biblioteca   IEEE;
usar IEEE.STD_LOGIC_1164.TODOS;
usar IEEE.NUMERIC_STD.TODOS;

entidadepulso_gené
        Porta(clique:emSTD_LOGIC;
                reiniciar :emSTD_LOGIC; pulso:
                foraSTD_LOGIC);
fimpulso_gen;

arquiteturaComportamentaldepulso_genéconstanteMÁX.: natural:= 100_000_000; sinals_cnt:
        naturalfaixa0paraMÁX-1; começar


processo(clk)
começar
        se   (rising_edge(clk))              então
                pulso <= '0';
                se(redefinir = '1')então
                        s_cnt <= 0;
                outro
                        s_cnt <= s_cnt + 1; se(s_cnt =
                        MAX-1)então
                                s_cnt    <=  0;
                                pulso    <=  '1';
                        fim   se;
                fim se;
        fim   se;
fim do processo;

fimComportamental;
```

Qual é a duração ativa da saída pulso?

Qual é a frequência de pulsosaída?

Exemplo de gerador de pulso

```
biblioteca IEEE;
usar IEEE.STD_LOGIC_1164.TODOS;
usar IEEE.NUMERIC_STD.TODOS;

entidadegeradoré
        genérico(NUMBER_STEPS: positivo:= 50_000_000); Porta(
                clique      :emSTD_LOGIC;
                reiniciar :emSTD_LOGIC;
                piscar:foraSTD_LOGIC);
fim  gerador;

arquiteturaComportamentaldegeradoré
        sinalcontador_s: naturalfaixa0paraNUMBER_STEPS-1; começar


contagem_proc:   processo(clk)
começar
        se  borda_crescente(clk)então
                se(redefinir = '1')ou(s_counter >= NUMBER_STEPS-1)então
                        s_counter <= 0;
                outro
                        s_counter <= s_counter + 1; fim se;

                piscar <= '1'quandos_counter >= (NUMBER_STEPS/2)outro'0'; -VHDL-2008! se;
        fim
fim do processo;

fimComportamental;
```

Qual é o ciclo de trabalho da saídapiscar?

Qual é a frequência de piscarsaída?

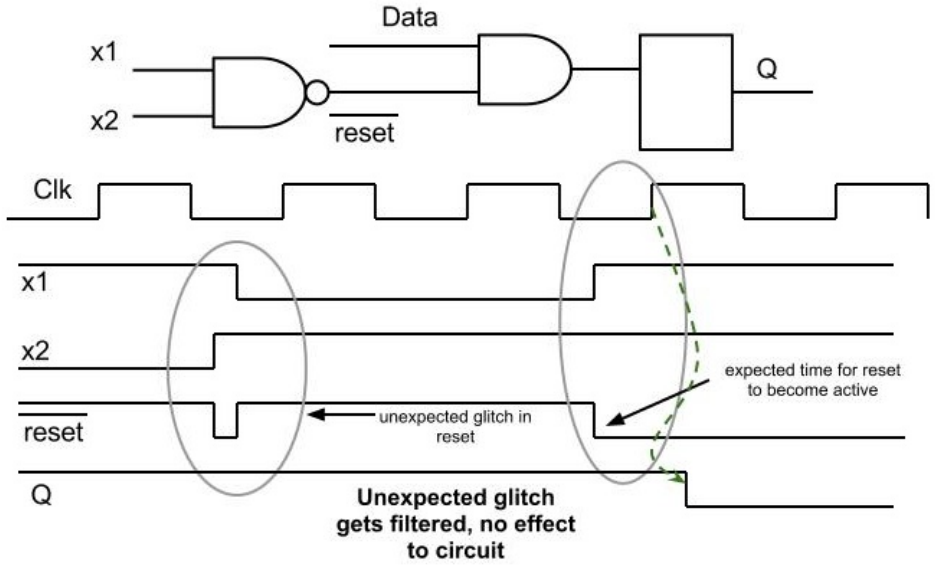
Inicialização e reinicialização

A maioria dos circuitos sequenciais requerem a inicialização de seus elementos de memória (por exemplo, registrador de estado FSM, contadores, acumuladores, etc.)

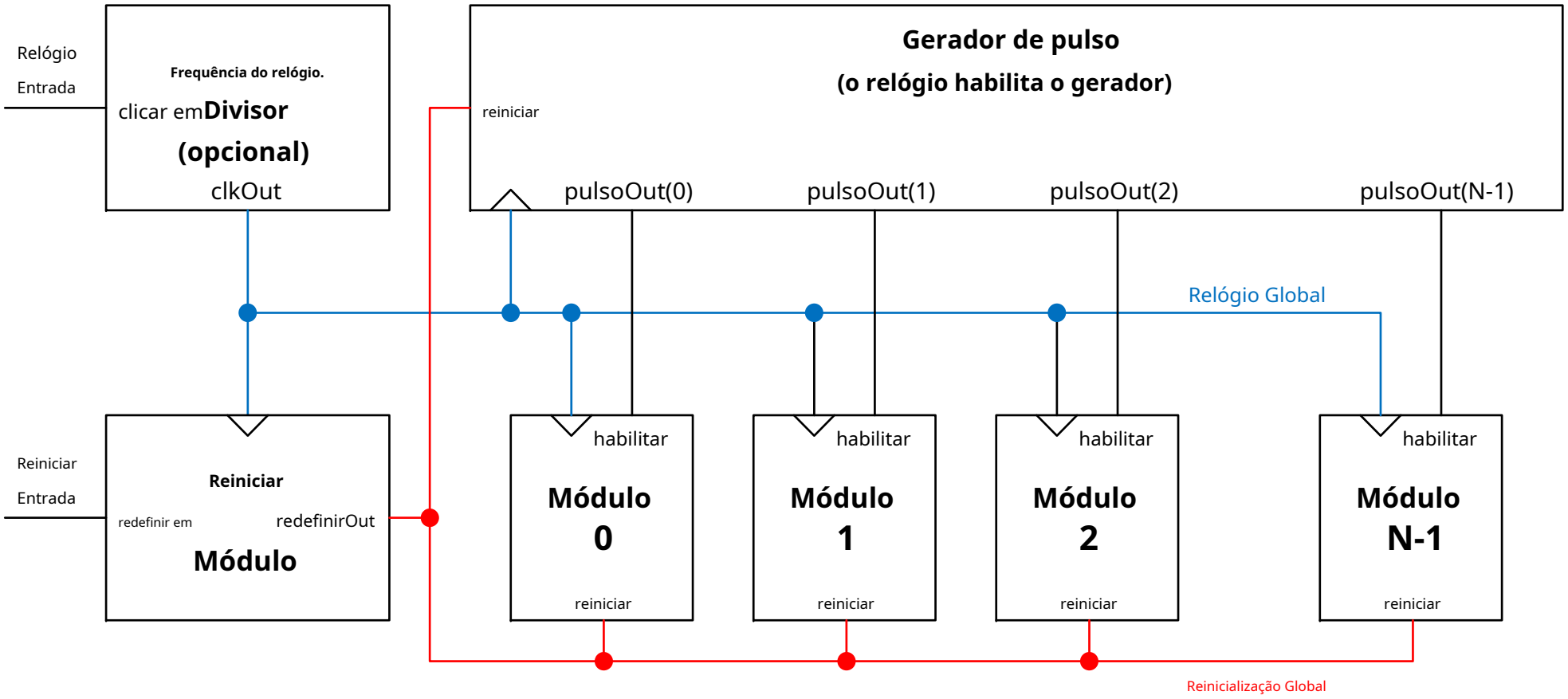
A inicialização deve ser realizada

- na inicialização do sistema / após a programação do FPGA
- sempre que necessário, através da ativação de sinais de reset globais ou locais

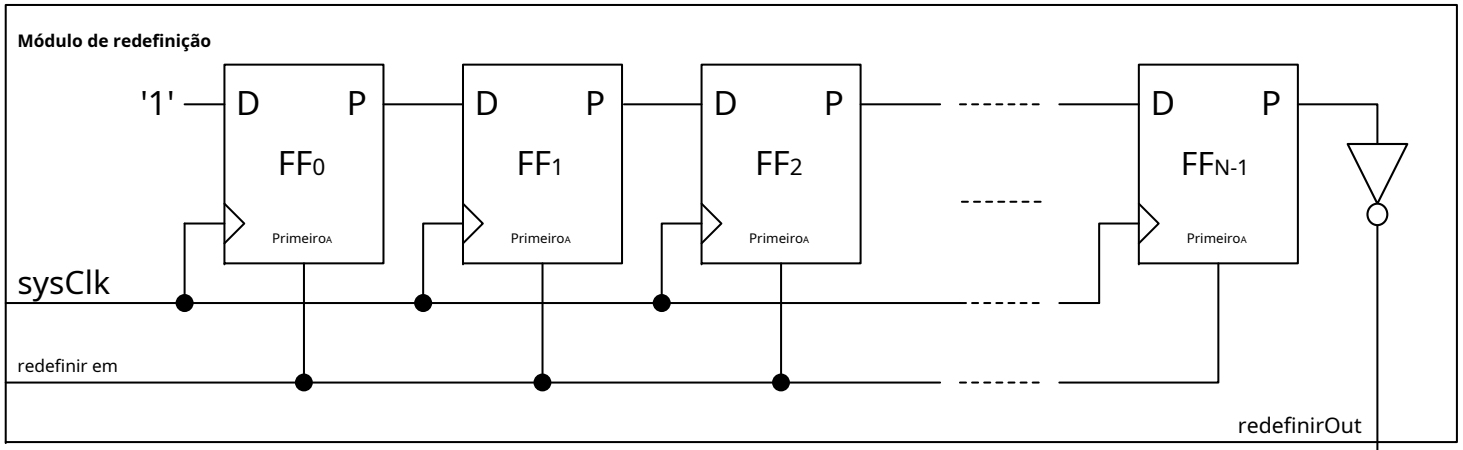
O uso de reinicialização assíncrona pode facilmente criar circuitos com falhas => componentes de reinicialização síncrona devem ser preferidos

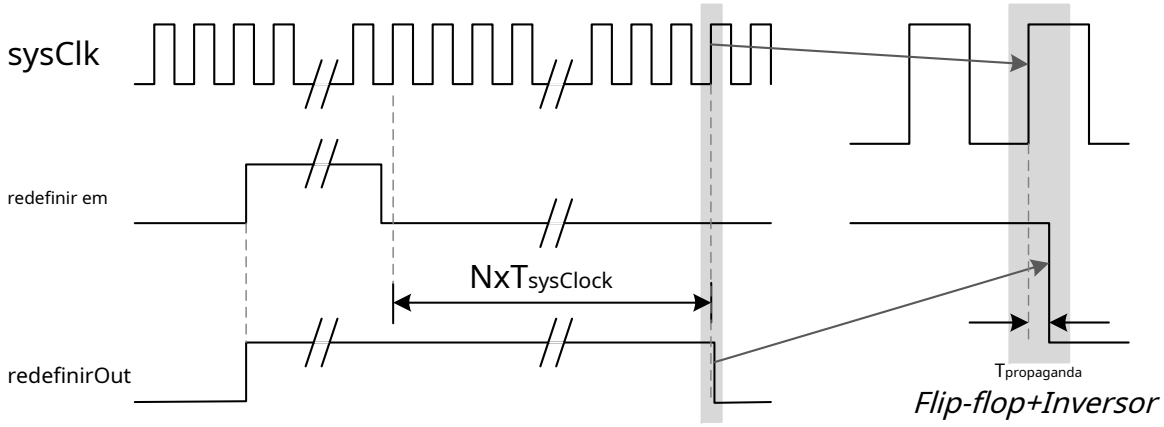


Domínio de relógio único com ativação e redefinição



Exemplo do Módulo de Redefinição





O circuito **(sincron. com "sysClk") que usa a redefinição sinal**

Depois do FPGA programação, todos os FF são carregado com 0 e o módulo ativa o redefinir saída

Todo o sistema os componentes devem use de preferência síncrono redefine

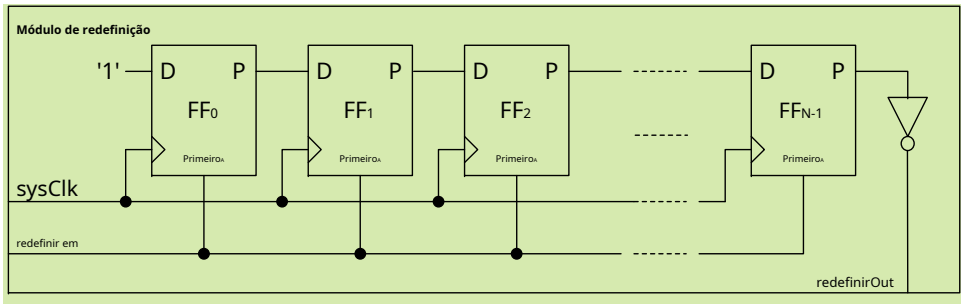
O período do relógio e o número de flip-flops garantem um reset mínimo tempo de ativação

Exemplo do Módulo de Redefinição

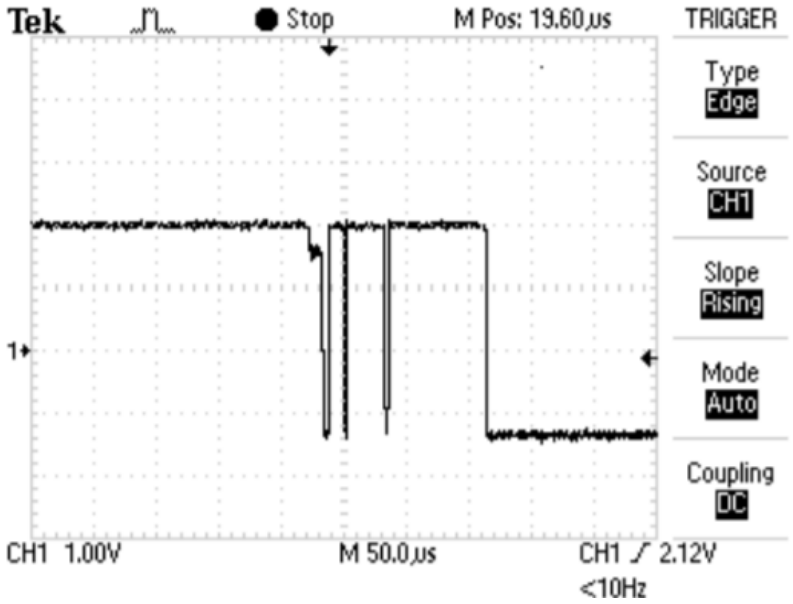
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ResetModule is
    generic(N      : positive := 4);
    port(sysClk    : in  std_logic;
         resetIn   : in  std_logic;
         resetOut  : out std_logic);
end ResetModule;

architecture Behavioral of ResetModule is

    signal s_shiftReg : std_logic_vector((N - 1) downto 0) := (others => '0');

begin
    assert(N >= 2);

    shift_proc : process(resetIn, sysClk)
    begin
        if (resetIn = '1') then
            s_shiftReg <= (others => '0');
        elsif (rising_edge(sysClk)) then
            s_shiftReg((N - 1) downto 1) <= s_shiftReg((N - 2) downto 0);
            s_shiftReg(0) <= '1';
        end if;
    end process;

    resetOut <= not s_shiftReg(N - 1);
end Behavioral;
```

Gera um pulso de reset, com duração ~Nx**sysClk**períodos

Inicialização dos **s_shiftReg** sinal durante a programação FPGA

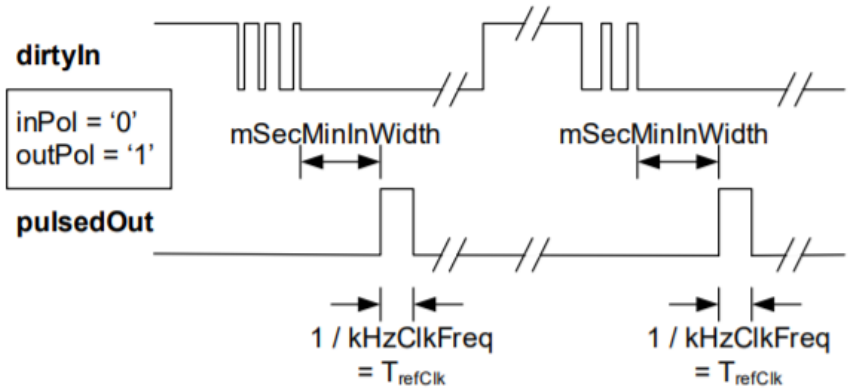


Debounce de entrada



E-learning:

```
entity DebounceUnit is
    generic(kHzClkFreq      : positive := 50000;
            mSecMinInWidth  : positive := 100;
            inPolarity      : std_logic := '0';
            outPolarity     : std_logic := '1');
    port(refClk     : in  std_logic;
         dirtyIn    : in  std_logic;
         pulsedOut  : out std_logic);
end DebounceUnit;
```



debounce_BTNC:

entidadetrabalho.DebounceUnit(Comportamental)

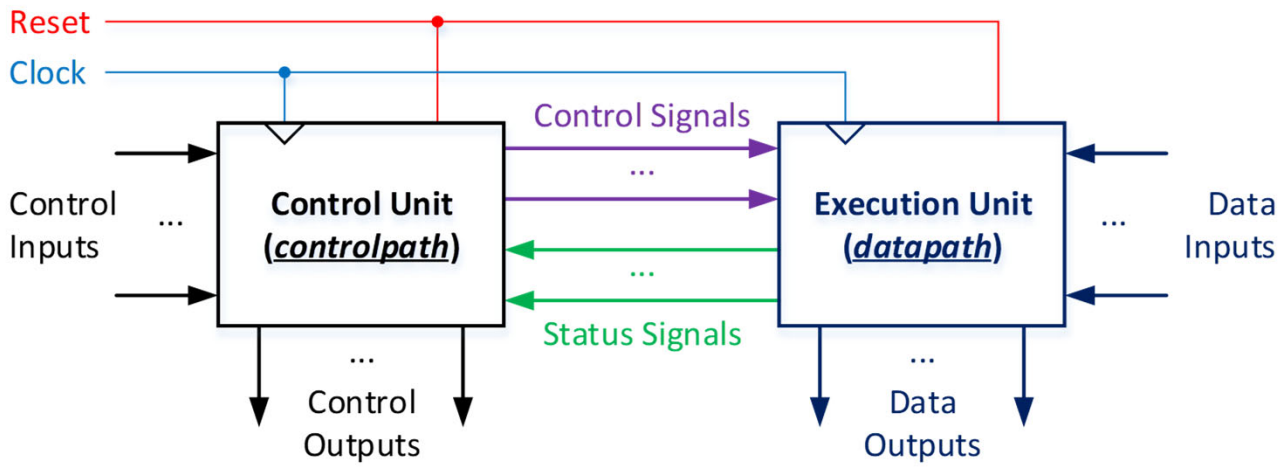
mapa genérico(kHzClkFreq => 100_000,
mSecMinInWidth => 100,
inPolaridade => '1',
outPolaridade => '1')

mapa do porto(refClk => clk,
sujoIn => btnC,
pulsedOut => s_startStop);

Sistema Computacional

Datapath (unidade de execução)

- Componentes
  - Funcional
  - Roteamento
  - Armazenar



Caminho de controle

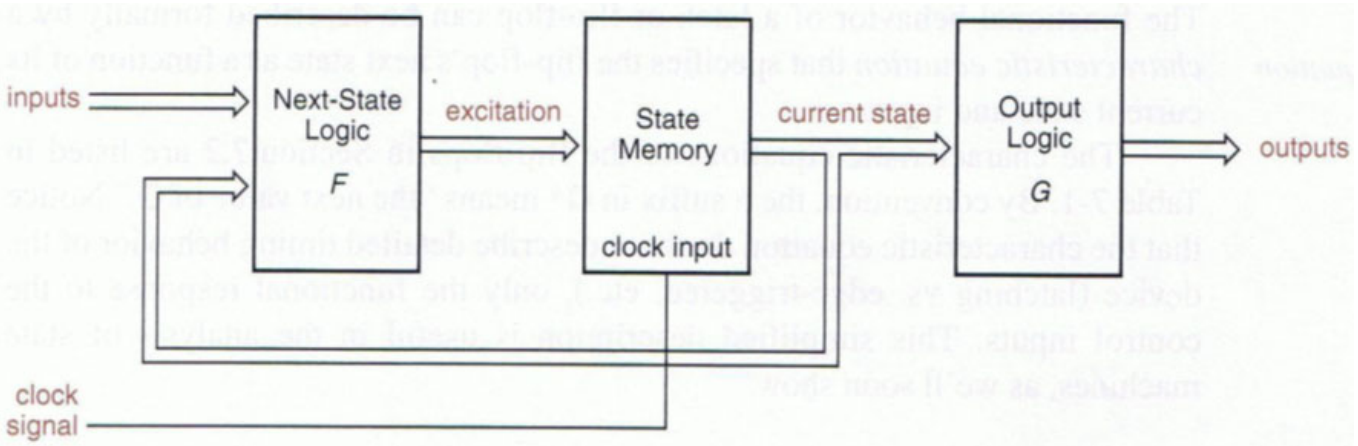
- Unidade de controle
  - FSM(ões)

Controlpath - interconexão de caminho de dados

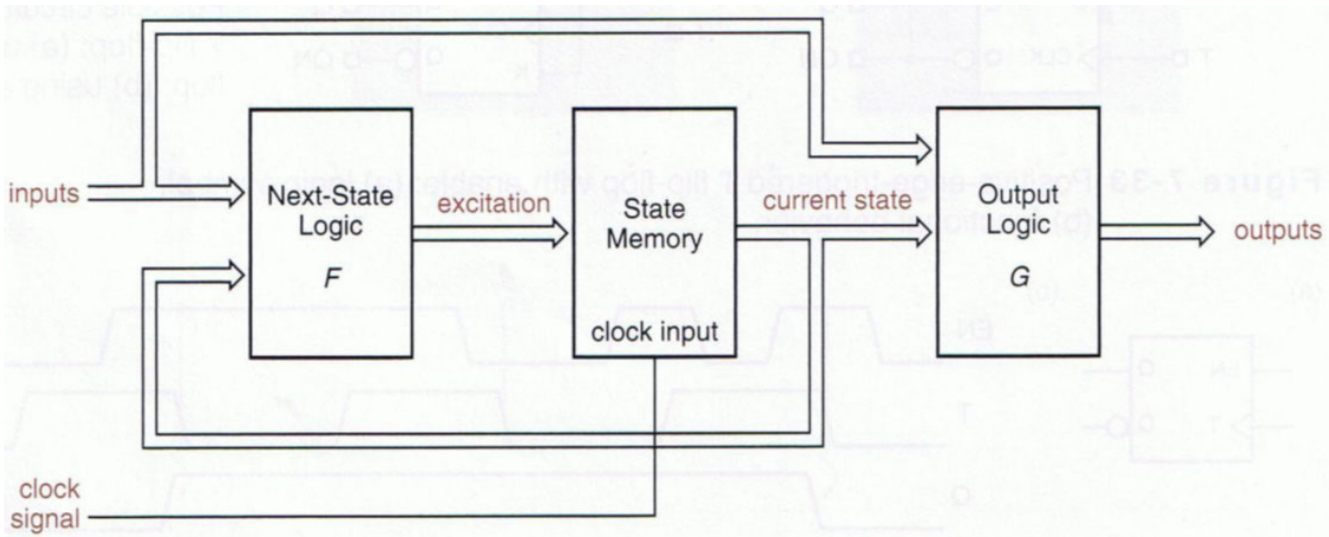
- Sinais de controle (caminho de controle-caminho de dados)
- Sinais de status (caminho de controle-caminho de dados

Estrutura FSM

Moura:



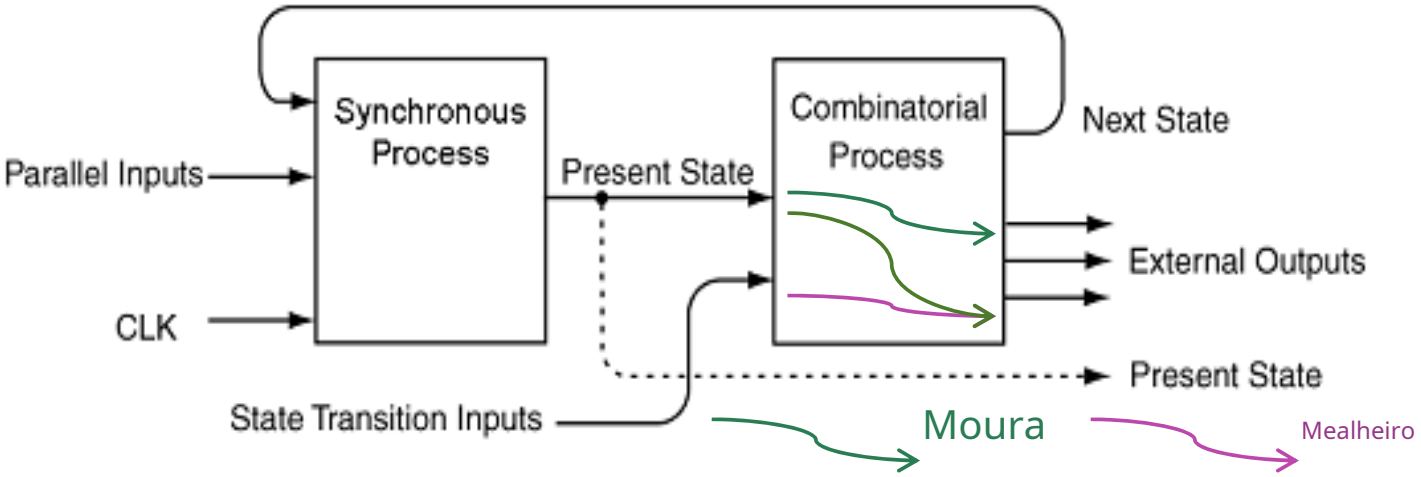
Farinhento:



Modelando FSMs em VHDL

Dois processos VHDL:

- Memória de estado (processo síncrono)
- Circuito combinacional (lógica do próximo estado + lógica de saída)
  - Dependendo da forma como as saídas são atribuídas
    - *Moura* (as saídas dependem apenas do estado atual)
    - *Mealheiro* (as saídas dependem do estado atual e das entradas do FSM)
  - Certifique-se de que um valor seja sempre atribuído ao próximo estado e saídas
    - Deve ser um circuito combinacional – sem travas!!!



Codificação VHDL

Codificação VHDL:

- Existem muitos estilos diferentes.
- O estilo aqui explicado considera dois processos: um para as transições de estado e outro para as saídas.

Etapas necessárias:

- Tenha seu diagrama de transição de estado pronto.
- A codificação simplesmente segue o diagrama de estado.
- Precisamos definir um tipo de dados de usuário personalizado (por exemplo, “estado”) para representar os estados:

tipoestado**é**(PARADO, INICIADO, OCUPADO); **sinal**y:

estado;

Dois processos devem ser construídos:

- Sync_proc: é onde são descritas as transições de estado (que ocorrem na transição do clock).
- Comb_proc: este é um circuito combinacional onde o próximo estado e as saídas são definidos com base no estado atual (e nos sinais de entrada).

Exemplo: Controlador de sequência de LED

Seqüência:00000011,00001111,00111111,11111100quando sw(0) = '0', ou11 000000,11110000,11111100,00111111quando sw(0) = '1'

O FSM inclui uma habilitação que permite transições de estado com frequência de 2Hz.

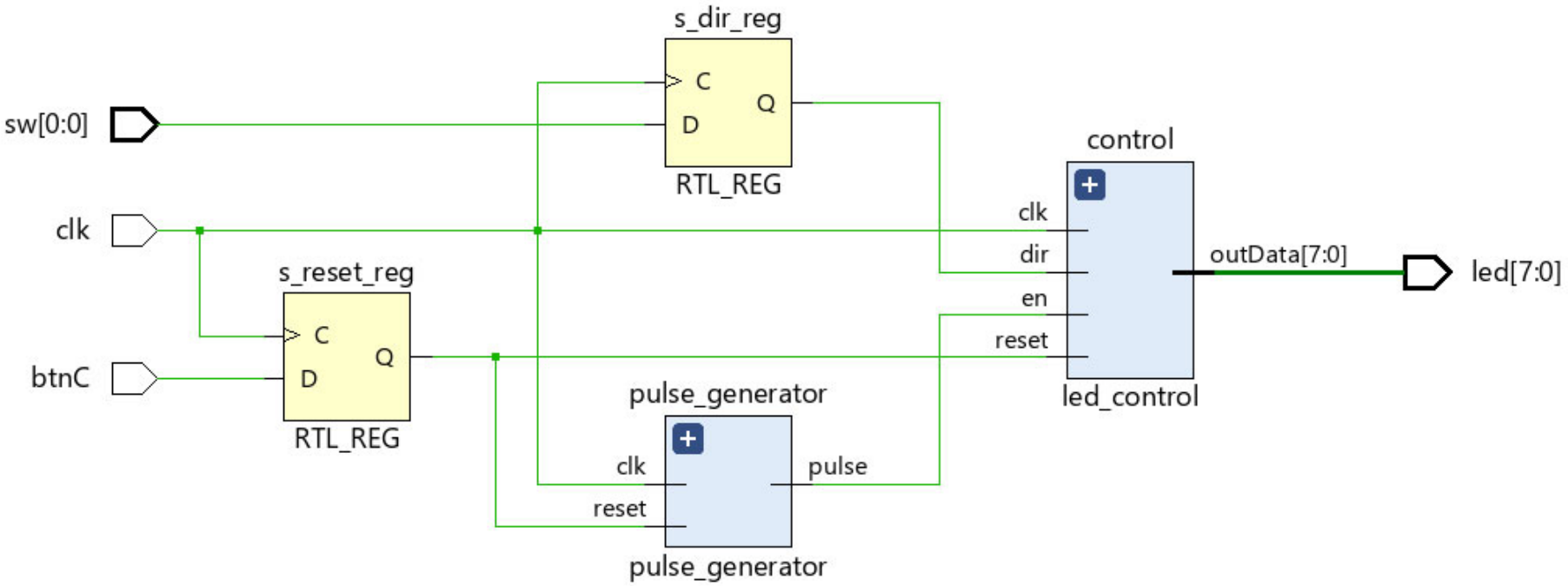
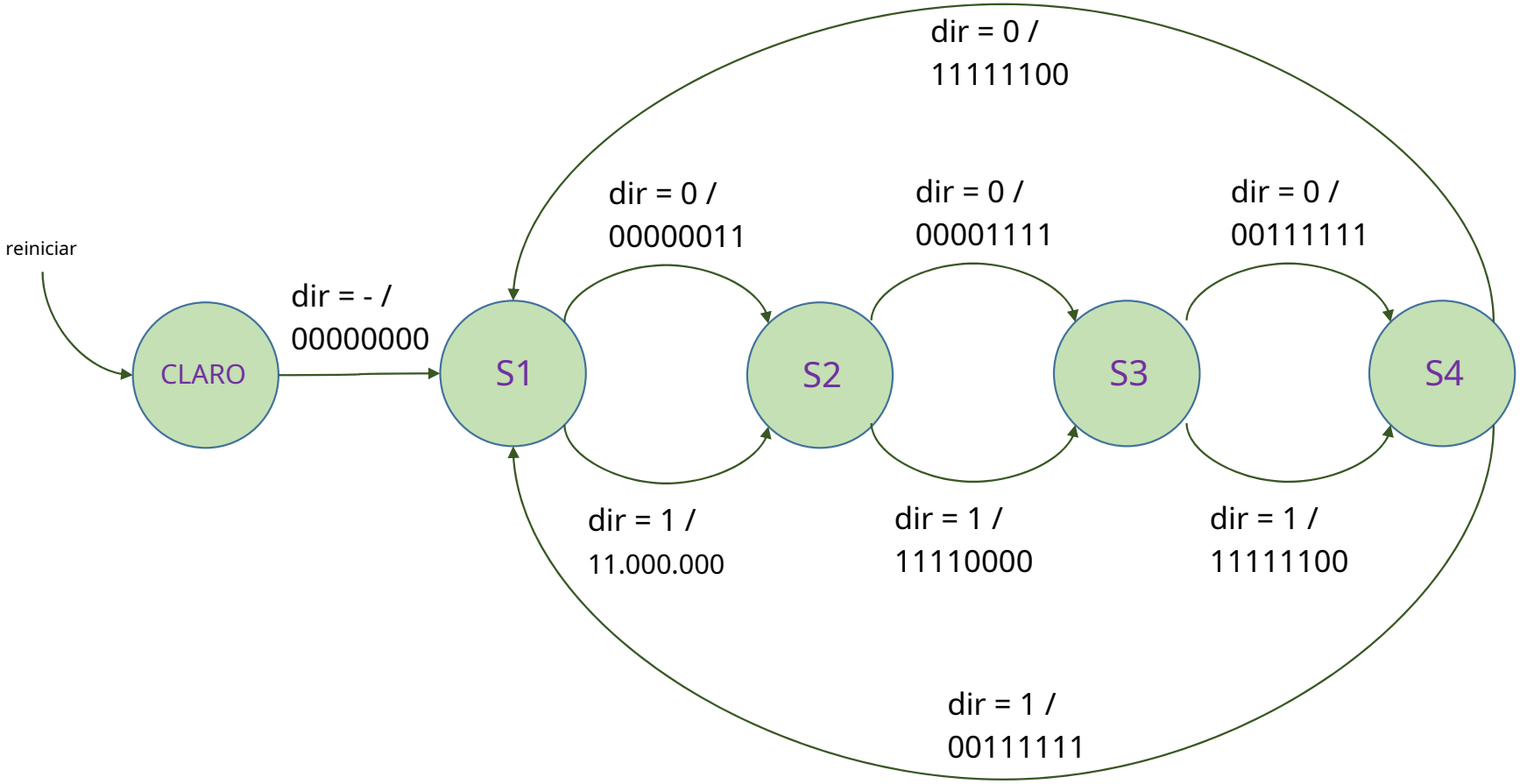


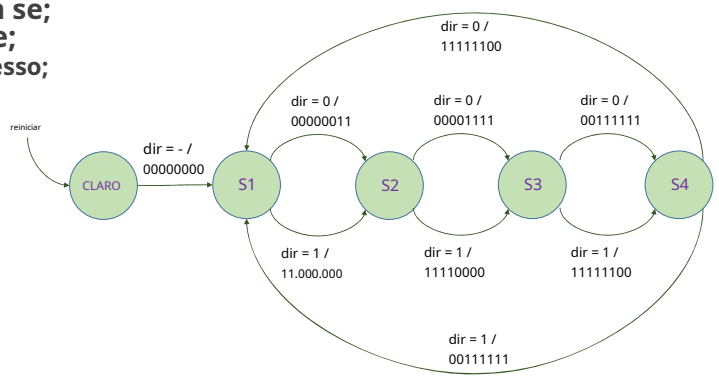
Diagrama de Estado

Seqüência:00000011,00001111,00111111,11111100quando sw(0) = '0', ou11000000,11110000,11111100,00111111quando sw(0) = '1'



Especificação em VHDL

```
biblioteca IEEE;
usar IEEE.STD_LOGIC_1164.TODOS;

entidade led_control é
    porta(clk          : em std_logic;
          pt           : em std_logic;
          reiniciar    :em  std_logic;
          diretório    :em  std_logic;
          dados externos :fora std_logic_vector(7até0));
fim led_control;

arquitetura Comportamental de led_control é
    tipo Estado é (LIMPAR, S1, S2, S3, S4); sinal pState,
    nState: TState;

    sinal s_data: std_logic_vector(outData'faixa) :=
                                        (outros => '0');

começar

sincronizar_proc :processo(clk)
    começar
        se (rising_edge(clk)) então se(
        redefinir = '1') então
            pEstado <= LIMPAR;
        Elif pt = '1' então
            pEstado <= nEstado;
        fim se;
    fim se;
    fim do processo;
```




```
comb_proc:processo(pEstado, dir)
começar
    caso pEstado é
        quando LIMPAR =>
            s_dados <= (outros => '0'); nEstado
            <= S1;
        quando S1 =>
            se dir = '0' então --esquerda
                s_dados <= "00000011";
            outro
                s_dados <= “11000000”;
            fim se;
            nEstado <= S2;
        quando S2 =>
            se dir = '0' então
                s_data <= "00001111";
            outro
                dados_s <= "11110000";
            fim se;
            nEstado <=S3;
        quando S3 =>
            se dir = '0' então
                s_dados <= "00111111";
            outro
                s_dados <= "11111100";
            fim se;
            nEstado <= S4;
        quando S4 =>
            se dir = '0' então
                s_data <= "11111100";
            outro
                s_dados <= "00111111";
            fim se;
            nEstado <= S1;
        quando outros => -- "Condição "pegar tudo"
            nEstado <= LIMPAR;
            s_dados <= (outros => '0'); caso
        final;
    fim processo;

    dados externos <= s_dados;

fim Comportamental;
```

Considerações finais

Ao final desta palestra você deverá ser capaz de:

- Use um único sinal de clock para todos os componentes do projeto
- Garanta a inicialização adequada do sistema (reset)
- Prefira reinicialização síncrona em vez de assíncrona
- Debounce entradas, se necessário
- Descrever FSMs em VHDL
- Decompor sistemas complexos em datapath e controlpath

Pendência:

- Teste o projeto fornecido no kit Nexys-4
- Faça laboratório. 4 parte 1