Laboratório de Sistemas Digitais

Aula Teórico-Prática 6

Ano Letivo 2022/23

Resumo dos estilos de codificação adequados à síntese

Precauções fundamentais de projeto: sincronização de entradas, *bouncing* de contatos mecânicos, estrutura de um projeto típico

Universidade de Aveiro

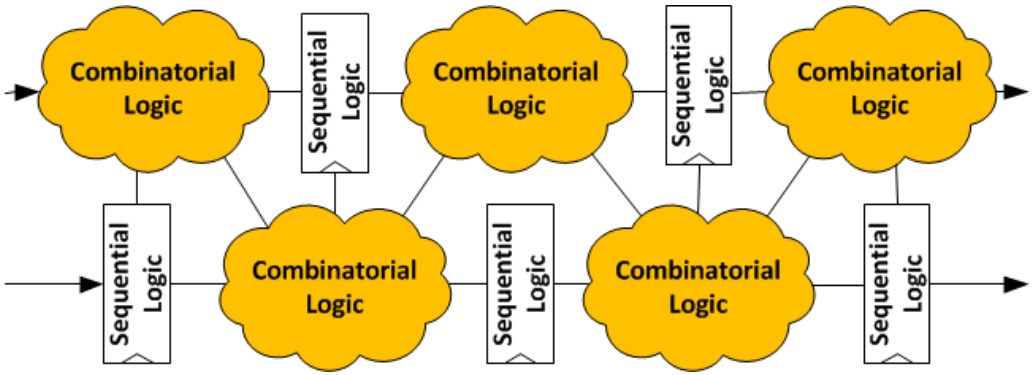
Conteúdo

- Sistematização da estrutura típica de processos relativos a circuitos combinatórios e sequenciais
  - Templates e estilos de codificação recomendados
  - Regras fundamentais e boas práticas
- Precauções fundamentais de projeto:
  - Sincronização de entradas
  - *Debouncing*
  - Estrutura de um projeto típico

Estrutura Típica de Processos Combinatórios

Estrutura geral:

```
process(<lista de sensibilidade
        com todas as entradas do
        processo>)
begin
    <atribuições a sinais/portos –
    saídas devem especificadas para
    todas as combinações dos vetores
    de entrada – mesmo que sejam
    don't care (para evitar latches)>
end process;
```



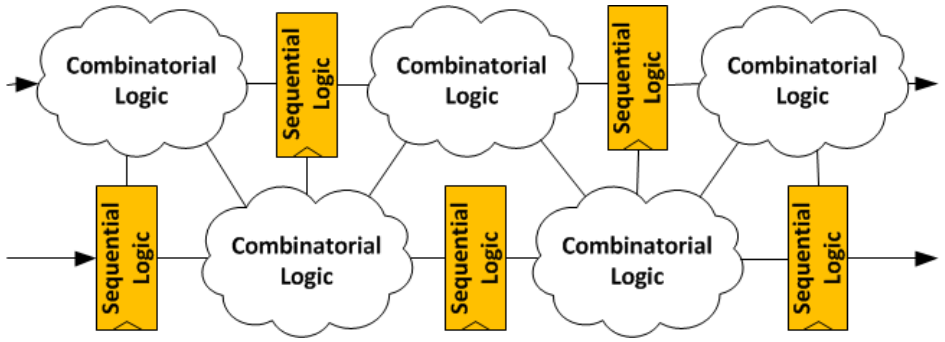
Exemplo (Codificador de prioridade 4→2):

```
process(decodIn)
begin
    if (decodIn(3) = '1') then
        validOut <= '1';
        encodOut <= "11";
    elsif (decodIn(2) = '1') then
        validOut <= '1';
        encodOut <= "10";
    elsif (decodIn(1) = '1') then
        validOut <= '1';
        encodOut <= "01";
    elsif (decodIn(0) = '1') then
        validOut <= '1';
        encodOut <= "00";
    else
        validOut <= '0';
        encodOut <= "--";
    end if;
end process;
```

Estrutura Típica de Processos Sequenciais

Estrutrura Geral:

```
process(<lista de sensibilidade
        com sinais de clock e
        set/reset assíncronos)
begin
   <teste de sinais assíncronos>
      <atribuições assíncronas>
   <teste do flanco ativo do clock>
      <teste de sinais síncronos >
         <atribuições síncronas>
end process;
```



Exemplo (contador binário up/down com enable e reset síncrono):

```
process(clk)
begin
   if (rising_edge(clk)) then
      if (reset = '1') then
        s_cnt <= (others => '0');
      elsif (enable = '1') then
         if (up = '1') then
            s_cnt <= s_cnt + 1;
         else
            s_cnt <= s_cnt - 1;
         end if;
      end if;
   end if;
end process;
```

Saídas não Completamente Especificadas (Inferência de Latches)

- Quando o valor de um sinal/porto não é especificado para um ou mais conjuntos de entradas
  - A ferramenta de síntese infere que esse sinal/porto deve corresponder à saída de um elemento de memória (porquê?)
    - Flip-flop
    - Latch
- *Latches* são pouco usados, MAS são frequentemente inferidos devido a "descrições combinatórias incompletas NÃO pretendidas"

Se a linha **`encodOut<="--"`** for removida, o sinal **`encodOut`** não está especificado para **`decodIn="0000"`**, levando a ferramenta de síntese a inferir uma *latch* para este sinal!!!

Exemplo: Codificador de prioridade 4→2

```
process(decodIn)
begin
    if (decodIn(3) = '1') then
        validOut <= '1';
        encodOut <= "11";
    elsif (decodIn(2) = '1') then
        validOut <= '1';
        encodOut <= "10";
    elsif (decodIn(1) = '1') then
        validOut <= '1';
        encodOut <= "01";
    elsif (decodIn(0) = '1') then
        validOut <= '1';
        encodOut <= "00";
    else
        validOut <= '0';
        encodOut <= "--";
    end if;
end process;
```

Múltiplas Atribuições a Sinais/Portos no Caminho de Execução de um Processo

- Boa prática
  - Realizar apenas uma atribuição a um sinal/porto ao longo do caminho de execução de um processo
- No entanto, se forem realizadas múltiplas atribuições, segundo a semântica de VHDL, prevalece a última
  - Utilizar esta facilidade para tornar o código mais compacto e se não afetar a legibilidade

Exemplo com o codificador de prioridade 4→2:

```
process(decodIn)
begin
    validOut <= '1';

    if (decodIn(3) = '1') then
        encodOut <= "11";
    elsif (decodIn(2) = '1') then
        encodOut <= "10";
    elsif (decodIn(1) = '1') then
        encodOut <= "01";
    elsif (decodIn(0) = '1') then
        encodOut <= "00";
    else
        validOut <= '0';
        encodOut <= "--";
    end if;
end process;
```

Sobre as Atribuições a Sinais/Portos

- Um sinal pode corresponder à saída de um componente combinatório ou sequencial (tudo depende da forma como for feita a atribuição)

```
process(sel, dataIn0, dataIn1)
begin
  if (sel = '0') then
    dataOut <= dataIn0;
  else
    dataOut <= dataIn1;
  end if;
end process;
```

Combinatório (mux)

```
process(enable, dataIn)
begin
  if (enable = '1') then
    dataOut <= dataIn;
  end if;
end process;
```

Sequencial (latch)

```
process(clk)
begin
  if (clk'event and clk = '1') then
    dataOut <= dataIn;
  end if;
end process;
```

Sequencial (flip-flop)

Erro *"Multiple Drivers" / "Multisource"*

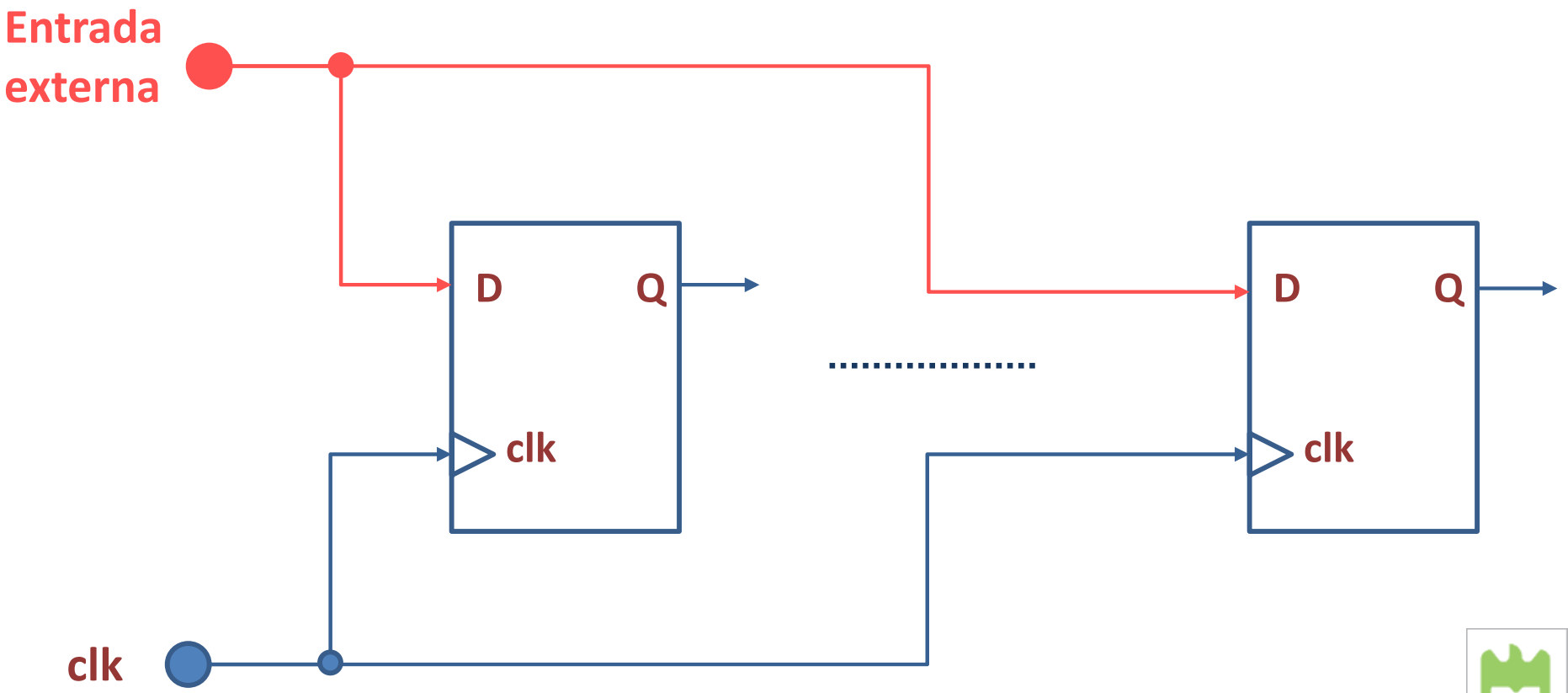
- Em geral, apenas um processo/módulo pode controlar (*to drive*) um sinal/porto de saída
  - Não fazer atribuições a um dado sinal/porto em mais do que um/uma processo/atribuição concorrente
  - Exceção: sinais com múltiplos *drivers* com capacidade *tri-state* (alta impedância) – a abordar mais tarde...

Exemplo de erros de "*multiple / conflict drivers / multisource*":

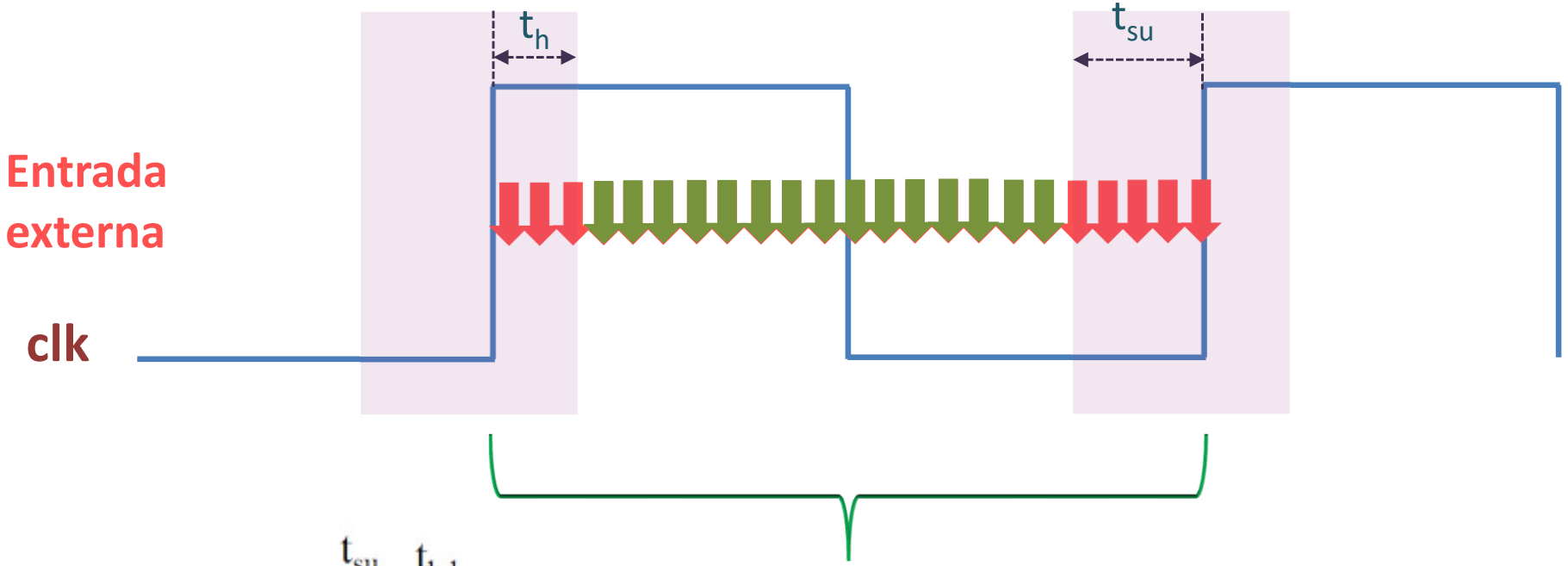
```
p1 : process(…)
begin
  ...
  dataOut <= ...;
  ...
end process;
  ...
p2 : process(…)
begin
  ...
  dataOut <= ...;
  ...
end process;
  ...
dataOut <= ...;
  ...
dataOut <= ...;
  ...
        port map(... => dataOut);

  ...
        port map(... => dataOut);
```

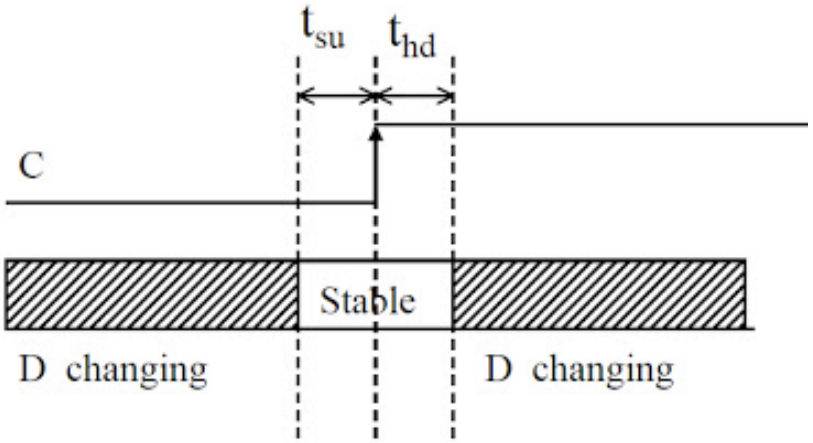
Entradas Assíncronas

- Circuitos controlados por um sinal de *clock* são circuitos **síncronos**
- Circuitos síncronos podem ter entradas **assíncronas**



Sinais Assíncronos



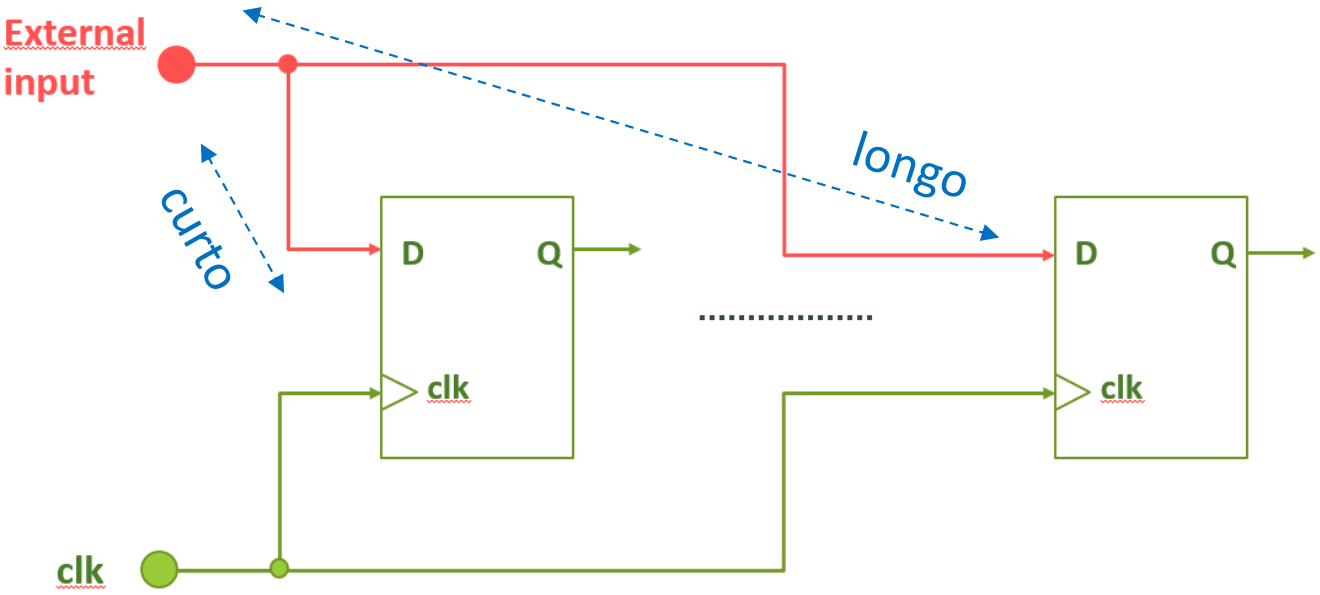


Problema de metaestabilidade:

- Ocorre quando entradas comutam perto do flanco ativo de *clock*
- Nem todas as transições que violam t_{su} e t_h resultam em metaestabilidade

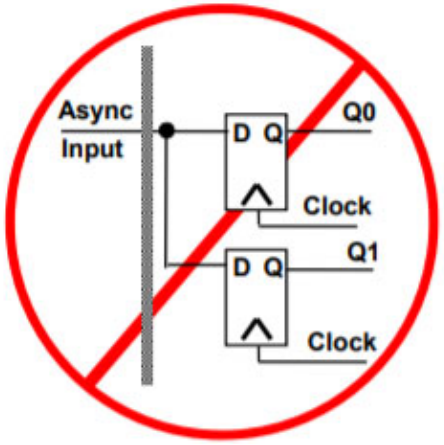
Entradas Assíncronas

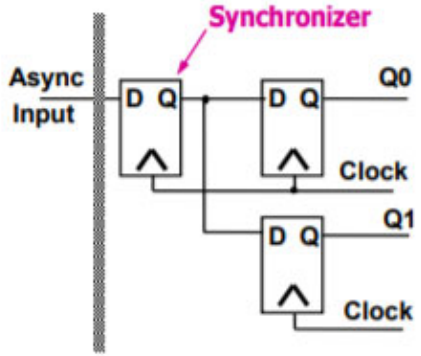
- Pequenas diferenças de atraso significam que os registros podem discordar no valor de entrada
- Problema de “valor inconsistente”:
  - Dois caminhos da entrada para dois registos diferentes:



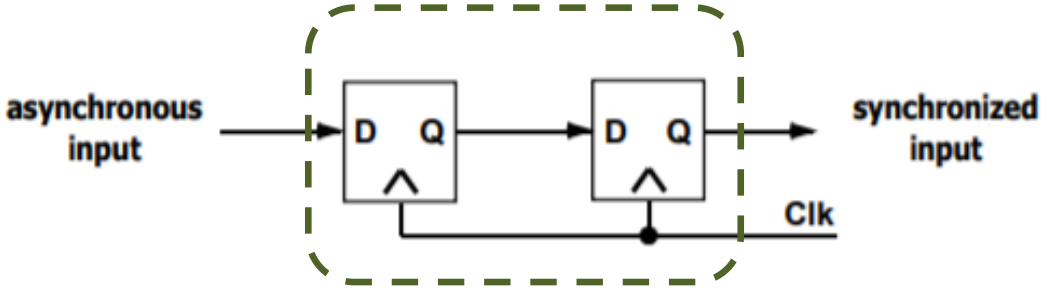
Solução do Problema

- Nunca usar diretamente entradas assíncronas => **sincronizá-las**
- A probabilidade de falha nunca pode ser nula!
- Para minimizar falhas, utilizar uma cascata de 2 ou mais **registos-sincronizadores**

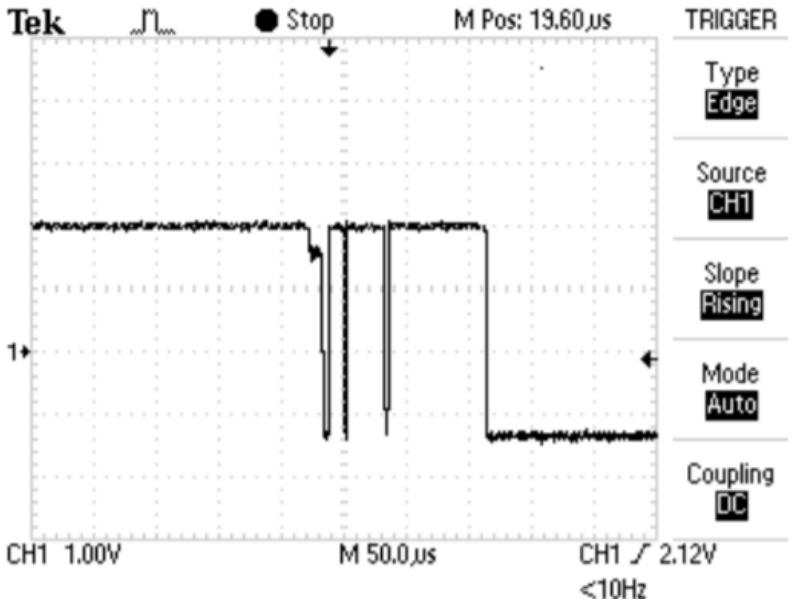


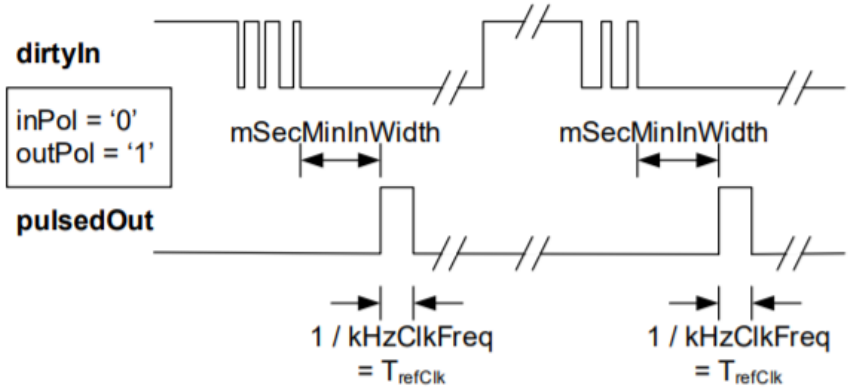


Synchronization chain



Bouncing de Contatos Mecânicos



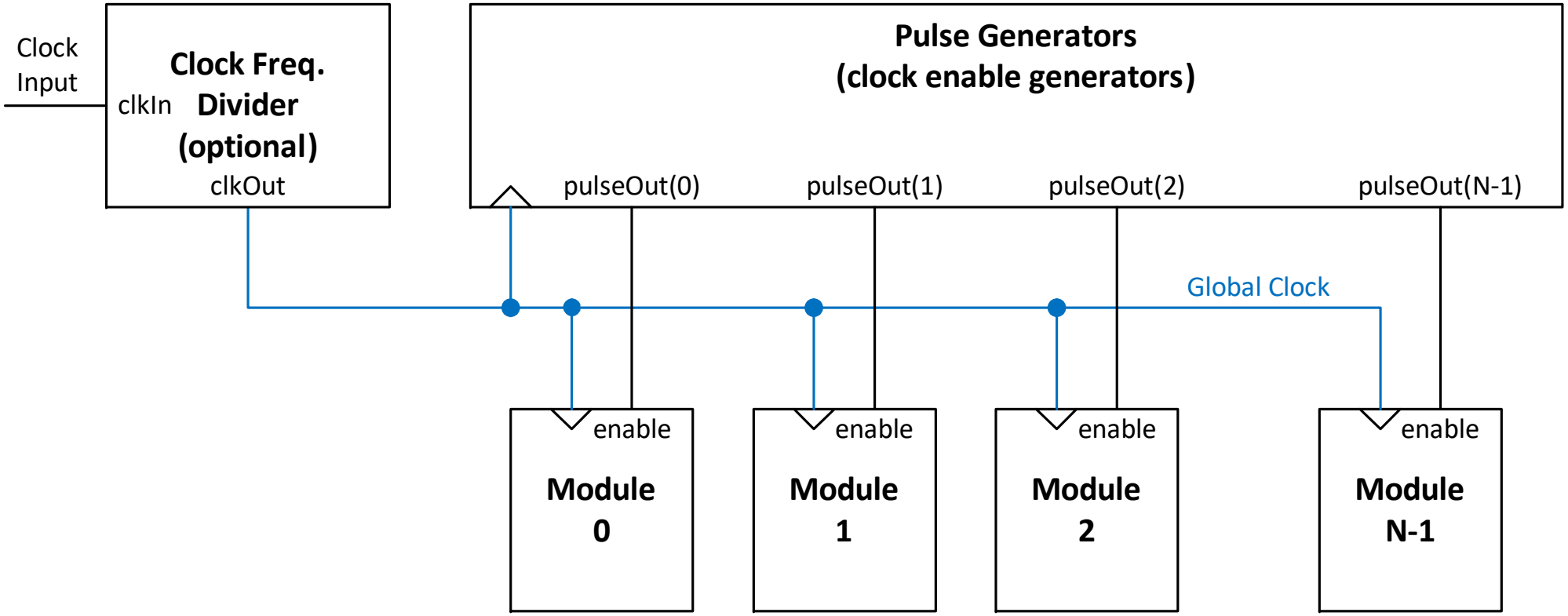


E-learning:

```
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_KEY0:
   entity work.DebounceUnit(Behavioral)
     generic map (kHzClkFreq => 50_000,
                  mSecMinInWidth => 100,
                  inPolarity => '0',
                  outPolarity => '1')
     port map(refClk => CLOCK_50,
         dirtyIn => KEY(0),
         pulsedOut => s_startStop);
```

Estrutura Típica de um Projeto



Exemplo de um Gerador de Pulsos Curtos (de sinais *enable*)

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

entity pulse_gen is
   generic (MAX : positive := 50_000_000);
   port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          pulse : out STD_LOGIC);
end pulse_gen;

architecture Behavioral of pulse_gen is
   signal s_cnt : natural range 0 to MAX-1;
begin

process(clk)
begin
    if (rising_edge(clk)) then
        pulse <= '0';
        if (reset = '1') then
            s_cnt <= 0;
        else
            s_cnt <= s_cnt + 1;
            if (s_cnt = MAX-1) then
                s_cnt <= 0;
                pulse <= '1';
            end if;
        end if;
    end if;
end process;

end Behavioral;
```

Qual é largura do pulso positivo na saída `pulse`?

Qual é frequência de saída `pulse`?

Exemplo de um Gerador de Pulsos Largos (para *blinking*)

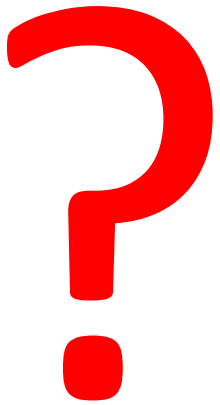
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity blink_gen is
    generic(NUMBER_STEPS : positive := 25_000_000);
    port ( clk   : in STD_LOGIC;
           reset : in STD_LOGIC;
           blink: out STD_LOGIC);
end blink_gen;

architecture Behavioral of blink_gen is
    signal s_counter : natural range 0 to NUMBER_STEPS-1;
begin

count_proc: process(clk)
begin
    if rising_edge(clk) then
        if (reset = '1') or (s_counter >= NUMBER_STEPS-1) then
            s_counter <= 0;
        else
            s_counter <= s_counter + 1;
        end if;
end if;
end process;

blink <= '1' when s_counter >= (NUMBER_STEPS/2) else '0';

end Behavioral;
```

Qual é largura do pulso na saída `blink`?

Qual é frequência de saída `blink`?

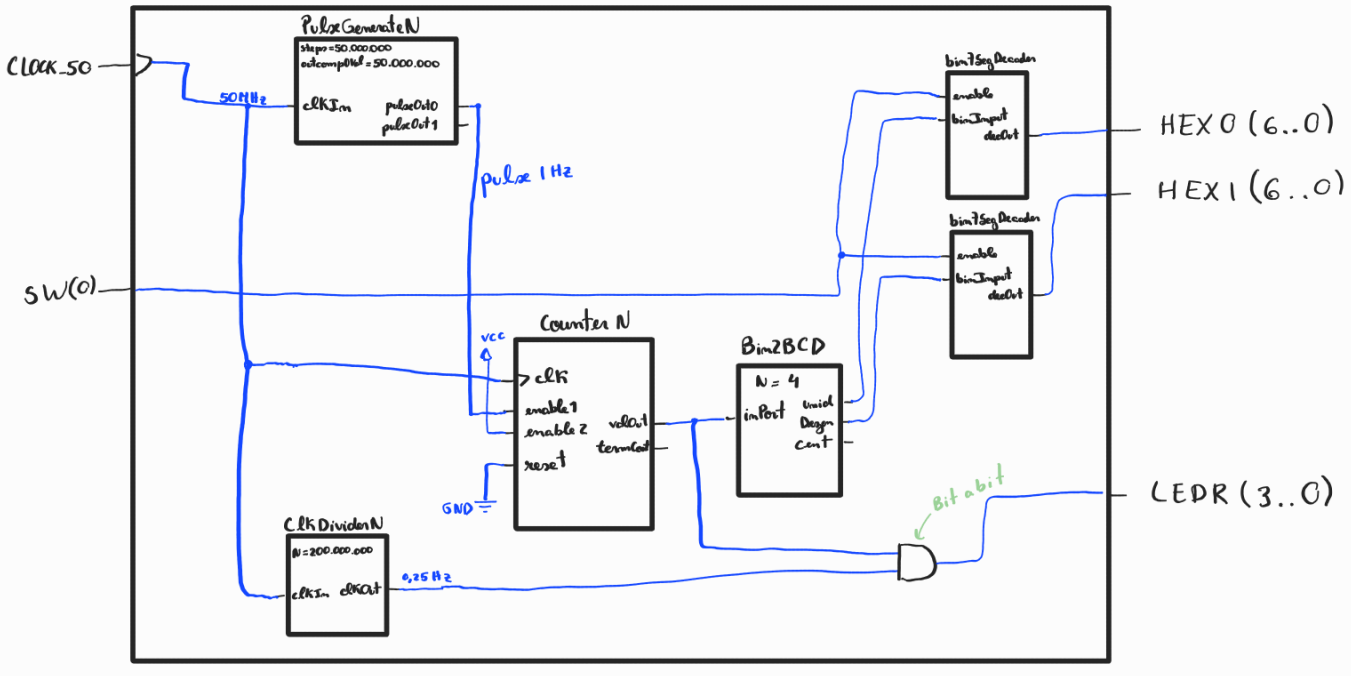
Discussão de Estrutura de um Projeto

- Pretende-se construir um contador crescente módulo 12, cujo valor é visualizado nos 4 LEDs e em *displays* de 7 segmentos (em decimal)
- O contador deve contar uma vez por segundo
- O valor em displays é mostrado de modo contínuo e pode ser desligado por um SW
- O valor nos LEDs deve sempre piscar 4 vezes por segundo
- Esboce um diagrama de blocos



0, 1, 2, 3, ..., 10, 11, 0, 1, 2, ...

- Contador 1 Hz
- SW desliga os Displays
- Leds devem piscar 4 vezes por segundo



Comentários Finais

- No final desta aula e do trabalho prático 6 de LSD, deverá ser capaz de:
  - Usar (ainda melhor) um subconjunto das construções de VHDL juntamente com estilos de codificação adequados para simulação e implementação
  - Sincronizar todas as entradas assíncronas
  - Resolver o problema de *bouncing* de contatos mecânicos
  - Construir corretamente projetos de um único domínio de relógio