

# Laboratório de Sistemas Digitais

## Aula Teórico-Prática 11

Ano Letivo 2024/25

Recomendações e boas práticas no projeto  
de sistemas digitais

Resumo dos tipos de dados em VHDL

Macros/funções de conversão entre tipos

# Recomendações Gerais

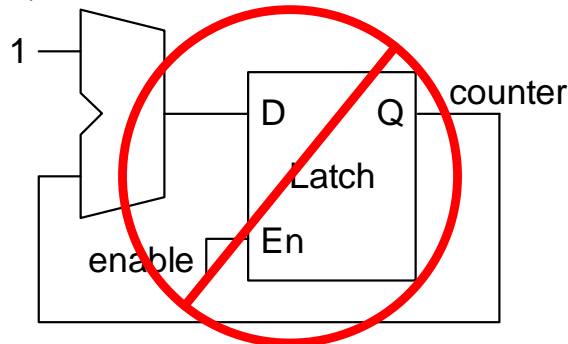
- Em projetos que usem componentes sequenciais recomenda-se que:
  - não sejam implementados ciclos combinatórios
  - sejam evitadas *latches*
    - *latches* são muito pouco usadas mas frequente e involuntariamente sintetizadas
  - passem todos os sinais de entrada por registos
  - usem apenas um sinal de relógio
  - tenham o devido cuidado com a inicialização (*reset*) do sistema
- Em todos os projetos recomenda-se que:
  - prestem atenção aos avisos (*warnings*) reportados pelo “Quartus Prime”
  - organizem o código de uma maneira visualmente bem estruturada
    - *indentação adequada do código*
  - comentem as partes menos óbvias do código

# Não Implementar Ciclos Combinatórios

## Ciclo Combinatório - **PROBLEMA!**

É sintetizada uma *latch* com *feedback* entre a saída e a entrada – comportamento imprevisível!

```
process(enable, counter)
begin
  if (enable = '1') then
    counter <= counter + 1;
  end if;
end process;
```



## Código **OK!**

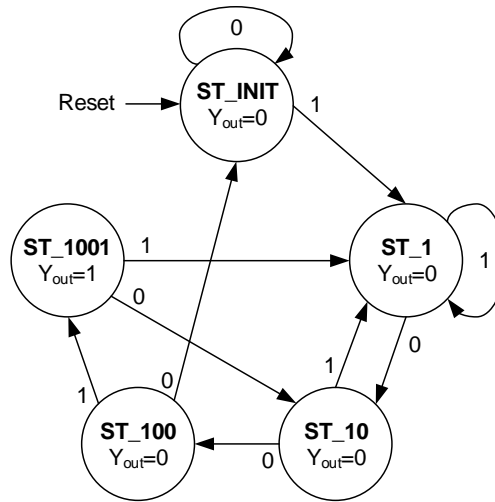
É sintetizado um contador *positive edge triggered*

```
process(sysClk)
begin
  if (rising_edge(sysClk)) then
    counter <= counter + 1;
  end if;
end process;
```

# Evitar a Inferência (involuntária) de *Latches*

- **Recomendação:** não escrever código VHDL que origine a inferência (síntese) de *latches*
- **Razão:** *latches* podem criar problemas temporais ou comportamentos inesperados
- **Causa frequente:** descrições combinatórias incompletas (involuntariamente, por não especificação de saídas para certas combinações das entradas e/ou encadeamentos incorretos de **if...then...elsif...else...**)
- **Como evitar:** num processo combinatório, garantir que é realizada a atribuição dos sinais/portos que dele dependem em todos os casos possíveis das entradas do processo
- Vamos ver um exemplo baseado na componente combinatória de uma FSM (parte do circuito que determina o estado seguinte e as saídas)...

# Detetor de Sequência "1001" (Modelo de Moore)



A detecção da sequência é realizada com ou sem sobreposição?

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity SeqDetector1001 is
    port(reset : in std_logic;
          clk   : in std_logic;
          xIn   : in std_logic;
          yOut  : out std_logic);
end SeqDetector1001;

architecture MooreArch of SeqDetector1001 is
    type TState is (ST_INIT, ST_1, ST_10, ST_100, ST_1001);
    signal s_currentState, s_nextState : TState;

begin
    sync_proc : process(clk)
    begin
        if (rising_edge(clk)) then
            if (reset = '1') then
                s_currentState <= ST_INIT;
            else
                s_currentState <= s_nextState;
            end if;
        end if;
    end process;

```

```

comb_proc : process(s_currentState, xIn)
begin
    case (s_currentState) is
        when ST_INIT =>
            yOut <= '0';

            if (xIn = '1') then
                s_nextState <= ST_1;
            end if;

        when ST_1 =>
            yOut <= '0';

            if (xIn = '0') then
                s_nextState <= ST_10;
            elsif (xIn = '1') then
                s_nextState <= ST_1;
            end if;

        when ST_10 =>
            yOut <= '0';

            if (xIn = '1') then
                s_nextState <= ST_1;
            else
                s_nextState <= ST_100;
            end if;

        when ST_100 =>
            yOut <= '0';

            if (xIn = '1') then
                s_nextState <= ST_1001;
            else
                s_nextState <= ST_INIT;
            end if;

        when ST_1001 =>
            yOut <= '1';

            if (xIn = '1') then
                s_nextState <= ST_1;
            else
                s_nextState <= ST_10;
            end if;
    end case;
end process;
end MooreArch;

```

Qual o problema do código apresentado? Inferência indevida de latches!

# Avisos na Síntese de um Circuito com *Latches*

The screenshot displays the Quartus Prime Lite Edition interface. The main window shows the VHDL code for `SeqDetector1001_Wrong.vhd`. The code defines a combinational process `comb_proc` that updates `s_nextState` based on `xIn` and the current state. Two callouts highlight specific code snippets:

- Callout 1:** `s_nextState` não é atribuído quando `xIn = '0'` (highlighting the `when ST_INIT =>` block).
- Callout 2:** Encadeamento absurdo de `if ... then ... elsif` (highlighting the `when ST_10 =>` block).

The Messages window at the bottom shows the following messages:

- 12021 Found 2 design units, including 1 entities, in source file `regpulsegenerator.vhd`
- 12127 Elaborating entity "SeqDetector1001\_Wrong" for the top level hierarchy
- 10631 VHDL Process Statement warning at `SeqDetector1001_Wrong.vhd(28)`: inferring latch(es) for signal or variable "s\_nextState",
- 10041 Inferred latch for "s\_nextState.ST\_1001" at `SeqDetector1001_Wrong.vhd(28)`
- 10041 Inferred latch for "s\_nextState.ST\_100" at `SeqDetector1001_Wrong.vhd(28)`
- 10041 Inferred latch for "s\_nextState.ST\_10" at `SeqDetector1001_Wrong.vhd(28)`
- 10041 Inferred latch for "s\_nextState.ST\_1" at `SeqDetector1001_Wrong.vhd(28)`
- 10041 Inferred latch for "s\_nextState.ST\_INIT" at `SeqDetector1001_Wrong.vhd(28)`
- 13012 Latch `s_nextState.ST_1001_94` has unsafe behavior
- 13012 Latch `s_nextState.ST_100_106` has unsafe behavior
- 13012 Latch `s_nextState.ST_INIT_142` has unsafe behavior
- 13012 Latch `s_nextState.ST_10_118` has unsafe behavior
- 13012 Latch `s_nextState.ST_1_130` has unsafe behavior
- 286030 Timing-Driven Synthesis is running

At the bottom right, the status bar shows "100% 00:00:26".

```

Text Editor - C:/Users/asroliveira/CloudStation/LSDig2016/Projects/Miscellaneous - Miscellaneous - [SeqDetector1001_S...
File Edit View Project Processing Tools Window Help Search altera.com
comb_proc : process(s_currentState, xIn)
begin
  case (s_currentState) is
    when ST_INIT =>
      yOut <= '0';

      if (xIn = '1') then
        s_nextState <= ST_1;
      else
        s_nextState <= ST_INIT;
      end if;

    when ST_1 =>
      yOut <= '0';

      if (xIn = '0') then
        s_nextState <= ST_10;
      else
        s_nextState <= ST_1;
      end if;

    when ST_10 =>
      yOut <= '0';

      if (xIn = '1') then
        s_nextState <= ST_1;
      else
        s_nextState <= ST_100;
      end if;

    when ST_100 =>
      yOut <= '0';

      if (xIn = '1') then
        s_nextState <= ST_1001;
      else
        s_nextState <= ST_INIT;
      end if;

    when ST_1001 =>
      yOut <= '1';

      if (xIn = '1') then
        s_nextState <= ST_1;
      else
        s_nextState <= ST_10;
      end if;
  end case;
end process;
end MooreArch;

```

```

Text Editor - C:/Users/asroliveira/CloudStation/LSDig2016/Projects/Miscellaneous/Miscellaneous - Miscellaneous - [SeqDetector1001_S...
File Edit View Project Processing Tools Window Help Search altera.com
comb_proc : process(s_currentState, xIn)
begin
  yOut <= '0'; -- Most frequent output value
  s_nextState <= s_currentState; -- Assume state is kept (no transition)
  -- override below (in case of transition)

  case (s_currentState) is
    when ST_INIT =>
      if (xIn = '1') then
        s_nextState <= ST_1;
      end if;

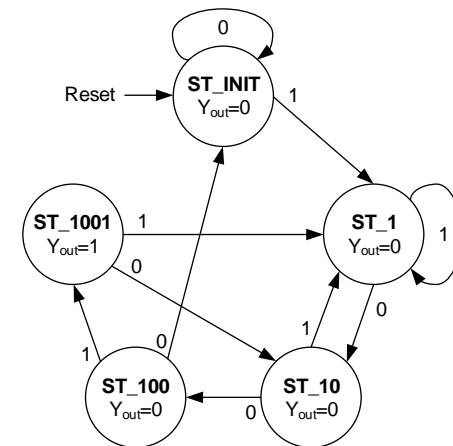
    when ST_1 =>
      if (xIn = '0') then
        s_nextState <= ST_10;
      end if;

    when ST_10 =>
      if (xIn = '1') then
        s_nextState <= ST_100;
      else
        s_nextState <= ST_INIT;
      end if;

    when ST_100 =>
      if (xIn = '1') then
        s_nextState <= ST_1001;
      else
        s_nextState <= ST_INIT;
      end if;

    when ST_1001 =>
      yOut <= '1'; -- Override the above output assignment
      if (xIn = '1') then
        s_nextState <= ST_1;
      else
        s_nextState <= ST_10;
      end if;
  end case;
end process;
end MooreArch;

```



Duas soluções possíveis. Em ambas garante-se que é realizada a atribuição do estado seguinte e da saída em todos os casos possíveis do estado atual e da entrada, resultando num circuito combinatório (como esperado)!

# Registo de Todos os Sinais de Entrada da FPGA

- **Recomendação:** passar os sinais de entrada da FPGA por registos sincronizado pelo *clock* do sistema
- **Razão:** se isto não for feito e se um sinal de entrada mudar de nível lógico muito perto de uma transição ativa de relógio, então o estado do sistema pode ficar inconsistente (saídas de blocos combinatórios rápidos do circuito podem “ver” o novo valor lógico, mas saídas de partes mais lentas podem ainda “ver” o valor antigo)
- **Solução:** com o uso de registos à entrada, problemas deste tipo desaparecem, porque todos os blocos do circuito vêem o mesmo nível lógico durante (quase) todo o período do sinal de relógio
- Exemplo (em VHDL e assumindo que o *clock* do sistema é o **CLOCK\_50**):

```
process (CLOCK_50)
begin
    if (rising_edge(CLOCK_50)) then
        s_key <= not KEY;
        s_sw  <= SW;
    end if;
end process;
```

(No resto do sistema devem ser usados os sinais **s\_key**, **s\_sw**, etc.)

- **Nota:** para evitar o esquecimento de algum sinal, é preferível que isto seja feito no *top-level* (em VHDL ou em diagrama lógico)



# Utilização de Apenas um Sinal de Relógio

- **Problema:** a utilização de dois ou mais domínios de relógio num sistema pode levar a problemas temporais complexos
  - O domínio de um relógio é o subconjunto de componentes do sistema que é sincronizado por esse sinal de relógio
  - A abordagem, análise e resolução destes problemas está fora do âmbito de LSD!
- **Solução:** em projetos de LSD que usem componentes sequenciais recomenda-se a utilização de apenas um sinal de relógio
  - A complexidade típica e as interfaces dos projetos de LSD não justificam a utilização de mais do que um sinal de relógio
  - Usar apenas um sinal de relógio (CLOCK\_50 ou outro derivado deste a partir de um divisor de frequência)
    - Nos casos em que 50 MHz é uma frequência de operação demasiado elevada
  - Usar em conjunto com o sinal de relógio, pulsos de ativação (*enables*) para sincronizar e sequenciar ações mais lentas
    - Usar um gerador de pulsos em vez de divisores de frequência
  - Todos os componentes são sincronizados pelo mesmo sinal de relógio e cada um possui o(s) seu(s) *enable(s)*

# Utilização de Apenas um Sinal de Relógio

```
entity counter_mod16 is
  Port ( clk      : in STD_LOGIC;
        reset    : in STD_LOGIC;
        en_1Hz   : in STD_LOGIC;
        count    : out STD_LOGIC_VECTOR (3 downto 0));
end counter_mod16;

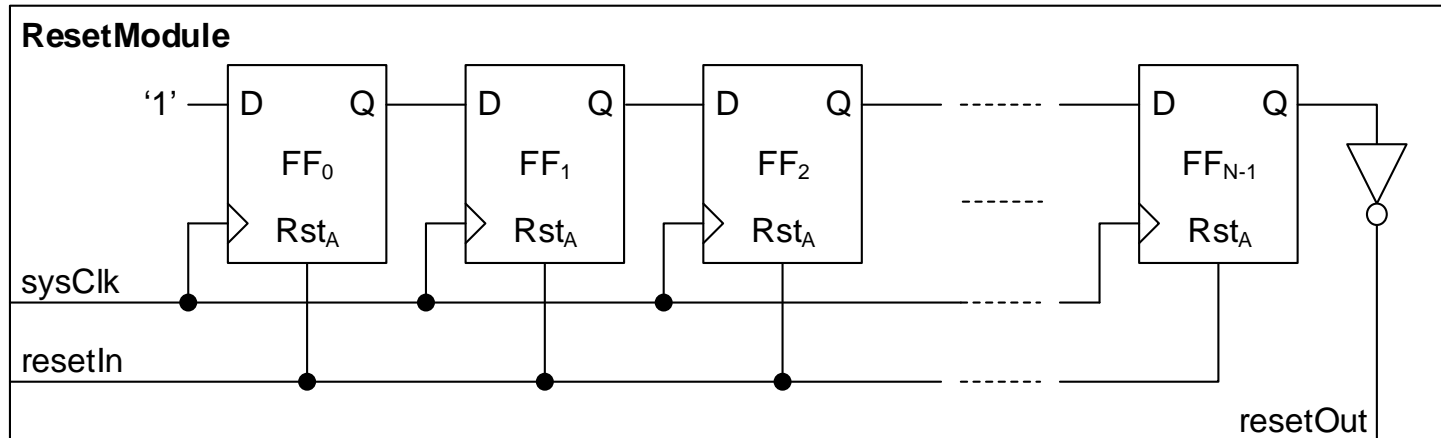
architecture Behavioral of counter_mod16 is
  signal s_count : unsigned(3 downto 0) ;
begin

  process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        s_count <= (others => '0');
      elsif (en_1Hz = '1') then
        s_count <= s_count + 1;
      end if;
    end if;
  end process;
  count <= std_logic_vector(s_count);
end Behavioral;
```

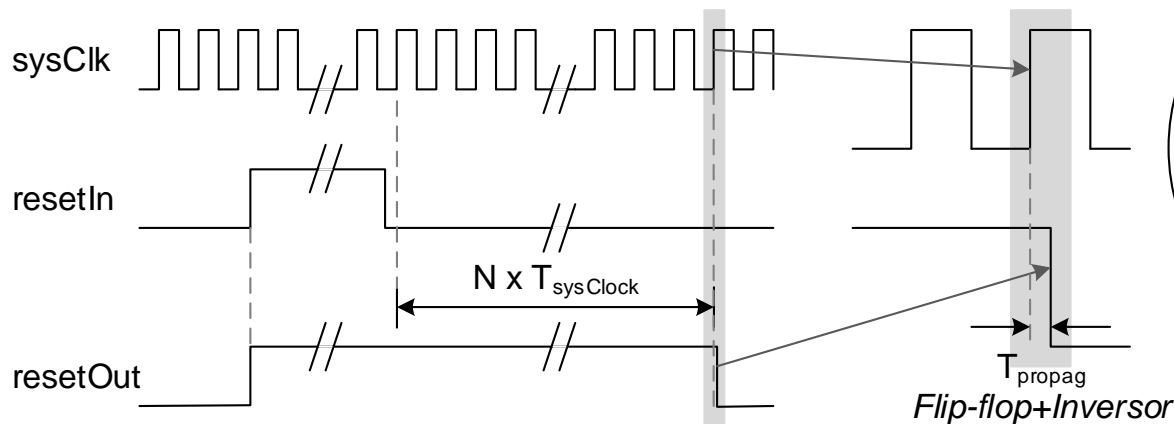
# Precauções de Inicialização (*Reset*)

- A maior parte dos sistemas com componentes sequenciais requerem a inicialização dos seus elementos de memória (e.g. registo de estado de uma FSMs, contadores, acumuladores, etc.)
- A inicialização deve ser realizada
  - No arranque do sistema / após programação da FPGA
  - Sempre que for ativado um sinal de inicialização global (tipicamente uma entrada acessível externamente)
- Devem ser preferidos componentes com *reset* síncrono

# Exemplo de um Módulo de *Reset*



Se após a programação da FPGA todos os FFs forem carregados com 0's, o módulo ativa inicialmente o *reset* de saída



Circuito (síncrono com “ $sysClk$ ”) que utiliza o sinal de *reset*

Os componentes usados no circuito devem (preferencialmente) usar *resets* síncronos

O período do sinal de relógio e o número de *flip-flops* asseguram um tempo mínimo durante o qual o sinal de *reset* está garantidamente ativo

# Exemplo de um Módulo de *Reset*

```
Text Editor - C:/Users/asoliveira/CloudStation/LSDig2016/Projects/Miscellaneous/Miscellaneous - Miscellaneous - [ResetModule.vhd]
File Edit View Project Processing Tools Window Help Search altera.com

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;

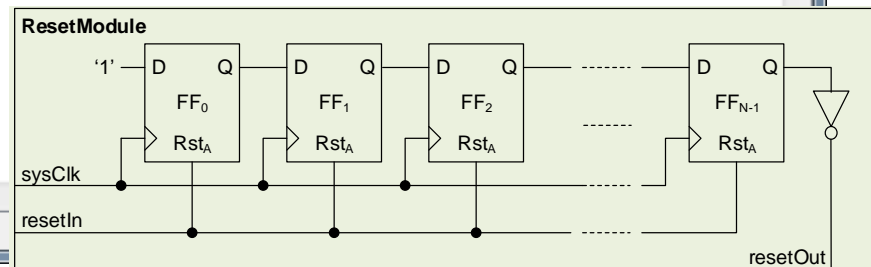
architecture Behavioral of ResetModule is
    signal s_shiftReg : std_logic_vector((N - 1) downto 0) := (others => '0');
begin
    assert(N >= 2);

    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;

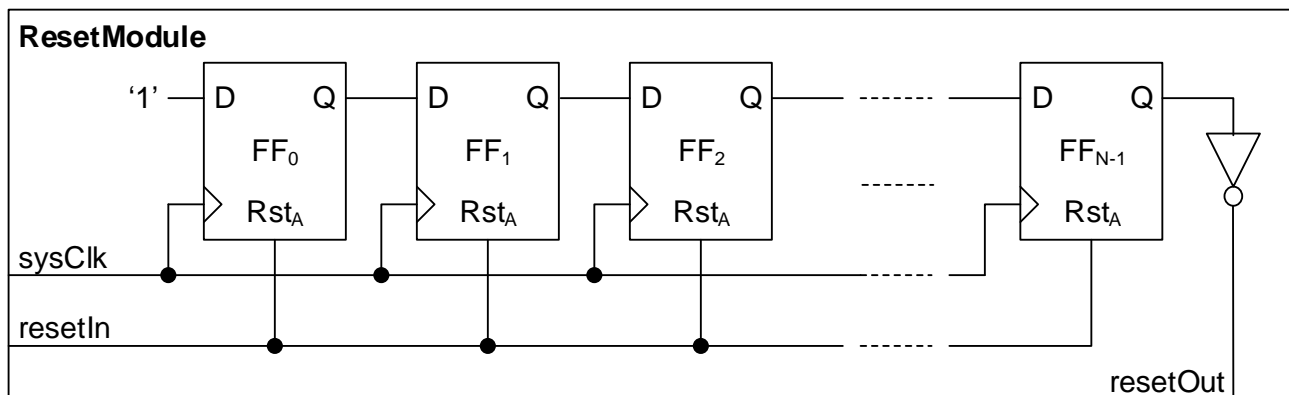
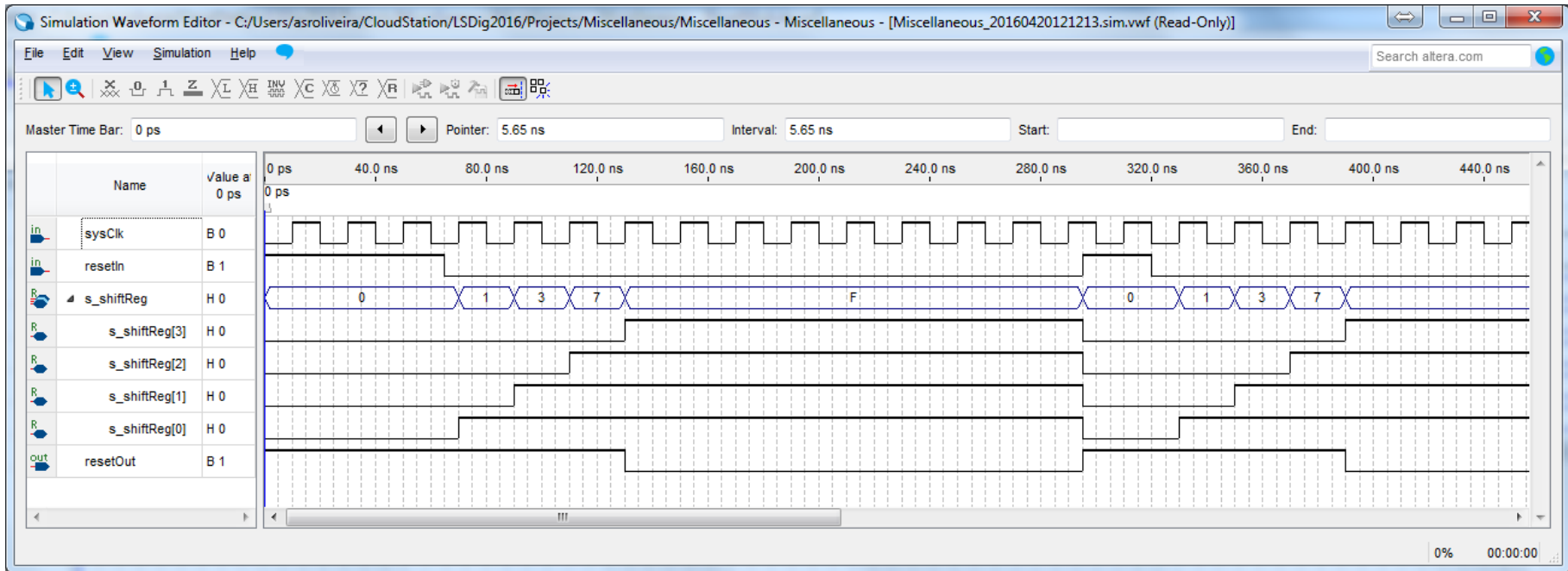
    resetOut <= not s_shiftReg(N - 1);
end Behavioral;
```

Gera impulsos de *reset* na saída, com a duração de “N” períodos de **sysClk** + tempo de ativação da entrada (aprox.)

Inicialização do sinal **s\_shiftReg** durante a programação da FPGA



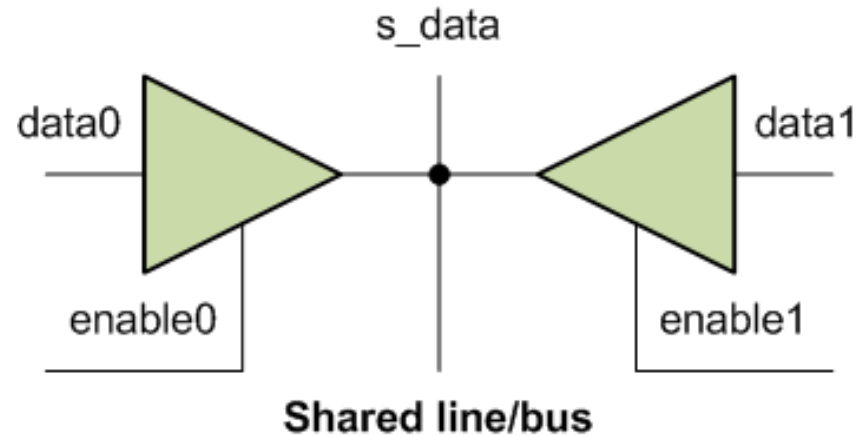
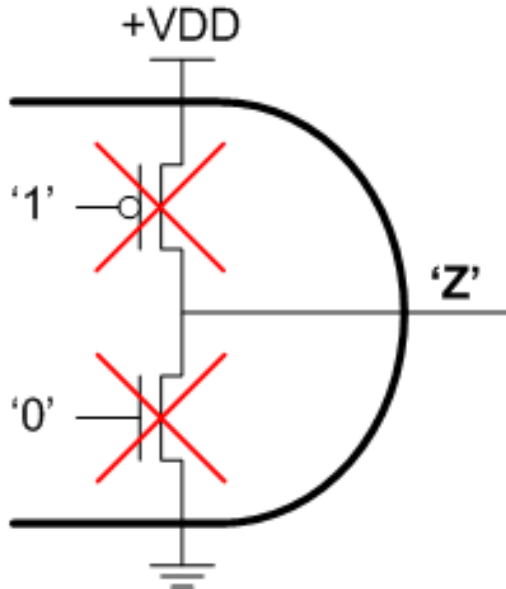
# Simulação do Módulo de *Reset*



# Outras Recomendações

- **Recomendação:** tomar em consideração os avisos emitidos pelo “*Quartus Prime*”
- **Razão:** alguns dos avisos (mensagens a azul ou violeta) assinalam problemas que devem ser corrigidos
- **Quando fazer:** deve-se “dar uma vista de olhos” pelas mensagens de aviso “de vez em quando”, e deve-se certamente fazê-lo mesmo antes de dar um projeto como concluído
- **Recomendação:** organizar o código de uma maneira visualmente bem estruturada
- **Razão:** o código deve ser fácil de entender por terceiros (e pelo próprio alguns meses ou anos depois)
- **Como fazer:** indentar o código de uma maneira adequada e consistente
- **Recomendação:** comentar as partes menos óbvias do código
- **Razão:** o código deve ser fácil de entender por terceiros (e pelo próprio alguns meses ou anos depois)
- **O que não fazer:** comentar o óbvio  
(e.g. `count <= count + 1; -- incrementa "count"`)

# std logic 1164 – Utilização de Tri-state



Modelação de linhas partilhadas (*shared*)

exemplo com atribuições condicionais

```
s_data <= data0 when (enable0 = '1') else  
    'Z' ;
```

```
s_data <= data1 when (enable1 = '1') else  
    'Z' ;
```

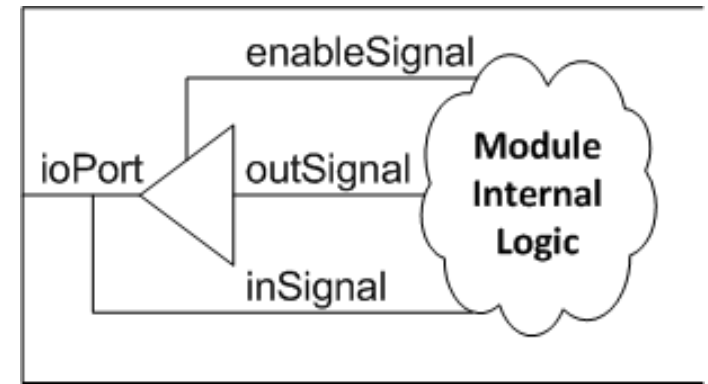
Pode também ser usado com barramentos do tipo `std_logic_vector`



# Portos Bidirecionais e Lógica Tri-state

```
entity EntityName
  port(...
    ioPort : inout std_logic;
    ...);
end EntityName;
```

```
architecture Behavioral of EntityName
  signal inSignal, outSignal, enableSignal : std_logic;
begin
  ...
  ioPort <= outSignal when (enableSignal = '1') else
    'Z';
  inSignal <= ioPort;
  ...
end Behavioral;
```



Tipicamente usados com pinos externos da FPGA (portos da entidade *top-level*).  
Podem também ser usados com barramentos do tipo `std_logic_vector`

# Resumo dos Tipos de Dados em VHDL (mais frequentes)

- integer

- Definição (na *package* STANDARD)

- ```
type integer is range -2147483647 to 2147483647;
```

- Utilização típica

- Indexação de arrays e como segundo operando de deslocamentos (*shifts*) e rotações (*rotates*) – número de posições a deslocar

- natural

- Definição (na *package* STANDARD)

- ```
subtype natural is integer range 0 to integer'high;
```

- Utilização típica

- Semelhante ao tipo integer, mas para valores naturais

- positive

- Definição (na *package* STANDARD)

- ```
subtype positive is integer range 1 to integer'high;
```

- Utilização típica

- Semelhante ao tipo integer, mas para valores positivos

# Resumo dos Tipos de Dados em VHDL (mais frequentes)

- **unsigned**

- Definição (na *package* NUMERIC\_STD)

- ```
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
```

- Utilização típica

- Operações aritméticas e lógicas em quantidades inteiras **sem** sinal

- **signed**

- Definição (na *package* NUMERIC\_STD)

- ```
type SIGNED is array (NATURAL range <>) of STD_LOGIC;
```

- Utilização típica

- Operações aritméticas e lógicas em quantidades inteiras **com** sinal

- **tipos enumerados**

- Definição

- Pelo utilizador num módulo ou package

- Utilização típica

- Definição dos estados simbólicos de uma FSM

# Outros Tipos de Dados

- **boolean**

- Definição (na *package* STANDARD)

- `type boolean is (false, true);`

- Utilização típica

- Resultado de condições e expressões booleanas

- **character / string**

- Definição (na *package* STANDARD)

- Utilização típica

- Manipulação de caracteres e arrays de caracteres

- **real**

- Definição (na *package* STANDARD)

- `type real is range -1.0E308 to 1.0E308;`

- Utilização típica

- Operações aritméticas em quantidades reais – **apenas em simulação ou na síntese, mas quando os valores são estáticos**

# Outros Tipos de Dados

- **time**

- Definição (na package STANDARD)

```
type time is range -2147483648 to 2147483647
    units
        fs;
        ps  = 1000 fs;
        ns  = 1000 ps;
        us  = 1000 ns;
        ms  = 1000 us;
        sec = 1000 ms;
        min = 60    sec;
        hr  = 60 min;
    end units;
```

- Utilização típica

- Simulação e construção de testbenches

- **bit e bit\_vector**

- Definição (na package STANDARD)

```
type bit is ('0', '1');
```

```
Type bit_vector is array (natural range <>) of bit;
```

- Pouco usado devido à existência do tipo `std_logic(_vector)`

# Macros/Funções de Conversão entre Tipos

- Para simplificar a interface entre módulos deve-se utilizar sempre portos do tipo **std\_logic** ou **std\_logic\_vector**
- Se necessário, as conversões são efetuadas dentro dos módulos para os tipos requeridos pelas operações a realizar
- Macros de conversão **(un)signed <-> std\_logic\_vector**
  - **unsigned**(parâmetro do tipo **std\_logic\_vector**)
    - macro/operador de conversão de **std\_logic\_vector** para **unsigned**
  - **signed**(parâmetro do tipo **std\_logic\_vector**)
    - macro/operador de conversão de **std\_logic\_vector** para **signed**
  - **std\_logic\_vector**(parâmetro do tipo **signed/unsigned**)
    - macro/operador de conversão de **signed** ou **unsigned** para **std\_logic\_vector**

# Macros/Funções de Conversão entre Tipos

- Funções de conversão `integer <-> (un)signed`
  - `to_integer(arg : unsigned) return natural;`
  - `to_integer(arg : signed) return integer;`
  - `to_unsigned(arg : natural, size: natural)`  
`return unsigned;`
  - `to_signed(arg : integer, size: natural)`  
`return signed;`
- Consoante a conversão pretendida, pode ser necessário uma ou duas conversões em cascata
  - e.g. `std_logic_vector->integer`
    - `to_integer(unsigned(Vetor_de_Bits_a_Conv))`

# Comentários Finais

- No final desta aula deverá ser capaz de aplicar as recomendações e boas práticas de projeto apresentadas e discutidas em LSD
  - Fundamentais (sempre), incluindo o desenvolvimento e avaliação do projeto final!
  - Avaliada a sua aplicação no projeto final
- Também deverá ser capaz de:
  - Usar adequadamente os principais tipos de dados suportados pelo VHDL
  - Modelar sinais e portos *tri-state*
  - Utilizar corretamente as macros e funções de conversão entre tipos