

Laboratório de Sistemas Digitais

Aula Teórico-Prática 10

Ano Letivo 2017/18

Recomendações e boas práticas no
projeto de sistemas digitais
(baseado em VHDL e FPGAs)

Arnaldo Oliveira, Augusto Silva, Guilherme Campos, Tomás Oliveira e Silva



Universidade
de Aveiro

Recomendações Gerais

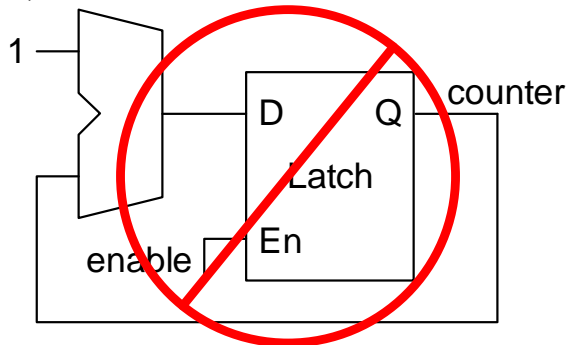
- Em projetos que usem componentes sequenciais recomenda-se que:
 - não sejam implementados ciclos combinatórios
 - sejam evitadas *latches*
 - *latches* são muito pouco usadas mas frequente e involuntariamente sintetizadas
 - passem todos os sinais de entrada por registos
 - usem apenas um sinal de relógio
 - tenham o devido cuidado com a inicialização (*reset*) do sistema
 - definam restrições temporais (e.g. frequência mínima de operação)
 - detetem e corrijam violações temporais
- Em todos os projetos recomenda-se que:
 - prestem atenção aos avisos (*warnings*) reportados pelo “Altera Quartus”
 - 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, s_counter)
begin
    if (enable = '1') then
        counter <= counter + 1;
    end if;
end process;
```



Código **OK!**

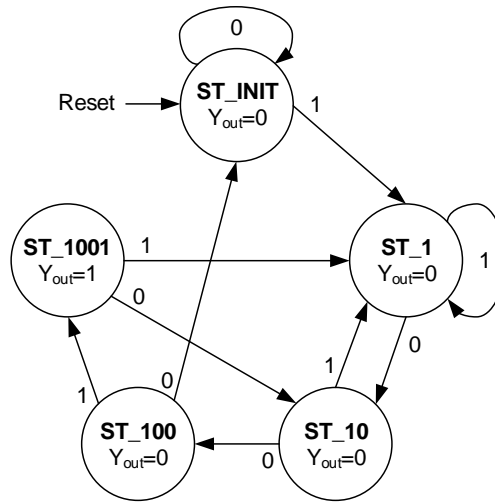
É sintetizado um contador *positive edge triggered*

```
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'` (s_nextState is not assigned when xIn='0'). This points to the `when ST_INIT =>` block where `yOut` is assigned but `s_nextState` is not.
- Callout 2:** `Encadeamento absurdo de if ... then ... elsif` (Absurd chaining of if ... then ... elsif). This points to the `when ST_10 =>` block where an `if` statement is nested within a `when` block.

The bottom panel shows the Messages window with 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

```

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;

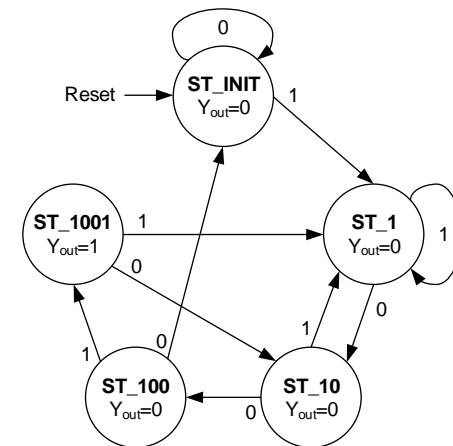
    when ST_1 =>
      if (xIn = '0') then
        s_nextState <= ST_10;
      end if;

    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;

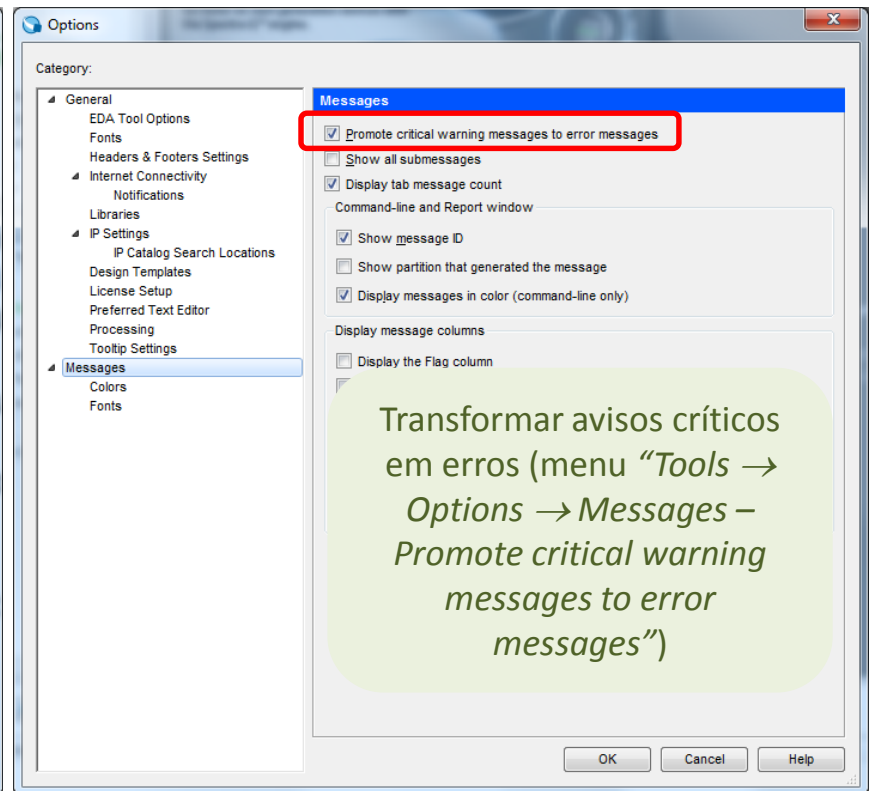
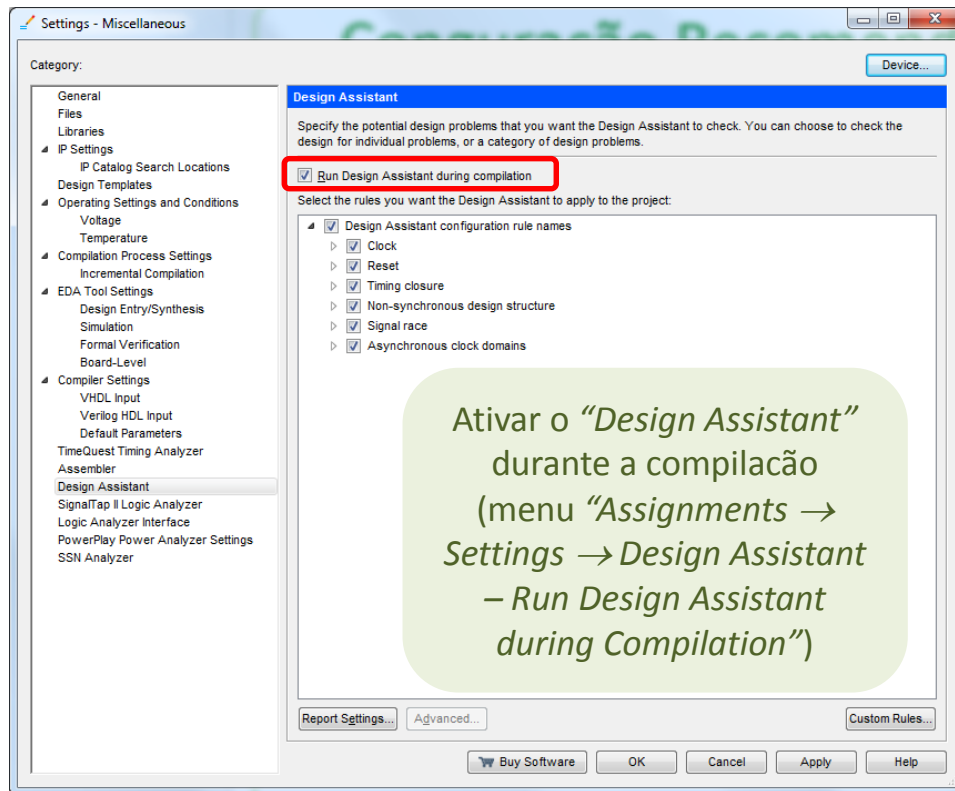
```



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)!

Configuração Recomendada do “Altera Quartus”

Forçar o “Altera Quartus” a reportar um erro (em vez de um aviso - *warning*) quando for sintetizada uma *latch* ou um ciclo combinatório

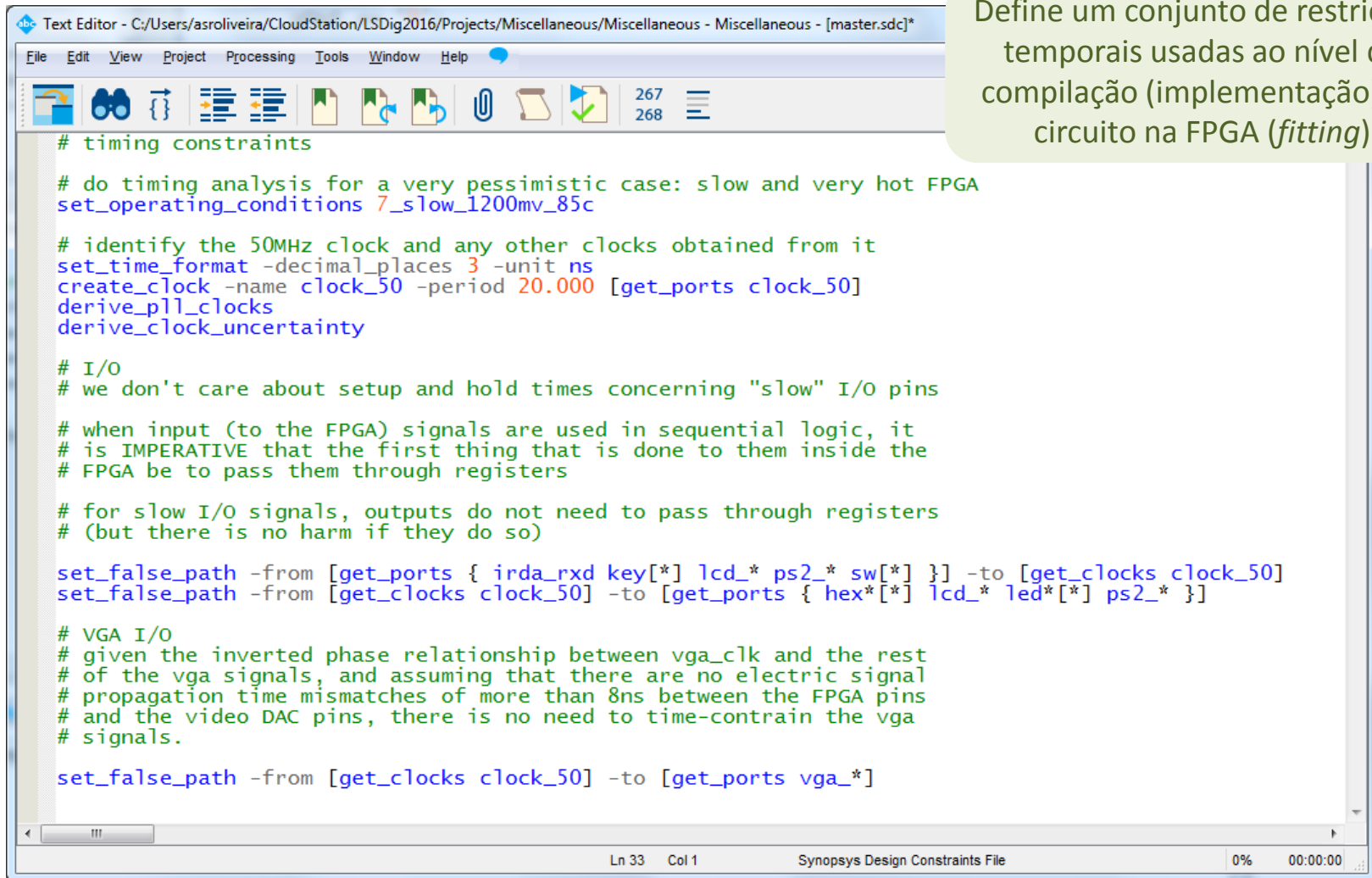


NOTA: É necessário acrescentar o ficheiro “master.sdc” ao projeto para garantir que, nestas condições, o projeto é compilado



Ficheiro “master.sdc”

Define um conjunto de restrições temporais usadas ao nível da compilação (implementação) do circuito na FPGA (*fitting*)



```
# timing constraints

# do timing analysis for a very pessimistic case: slow and very hot FPGA
set_operating_conditions 7_slow_1200mv_85c

# identify the 50MHz clock and any other clocks obtained from it
set_time_format -decimal_places 3 -unit ns
create_clock -name clock_50 -period 20.000 [get_ports clock_50]
derive_pll_clocks
derive_clock_uncertainty

# I/O
# we don't care about setup and hold times concerning "slow" I/O pins

# when input (to the FPGA) signals are used in sequential logic, it
# is IMPERATIVE that the first thing that is done to them inside the
# FPGA be to pass them through registers

# for slow I/O signals, outputs do not need to pass through registers
# (but there is no harm if they do so)

set_false_path -from [get_ports { irda_rxd key[*] lcd_* ps2_* sw[*] }] -to [get_clocks clock_50]
set_false_path -from [get_clocks clock_50] -to [get_ports { hex*[*] lcd_* led*[*] ps2_* }]

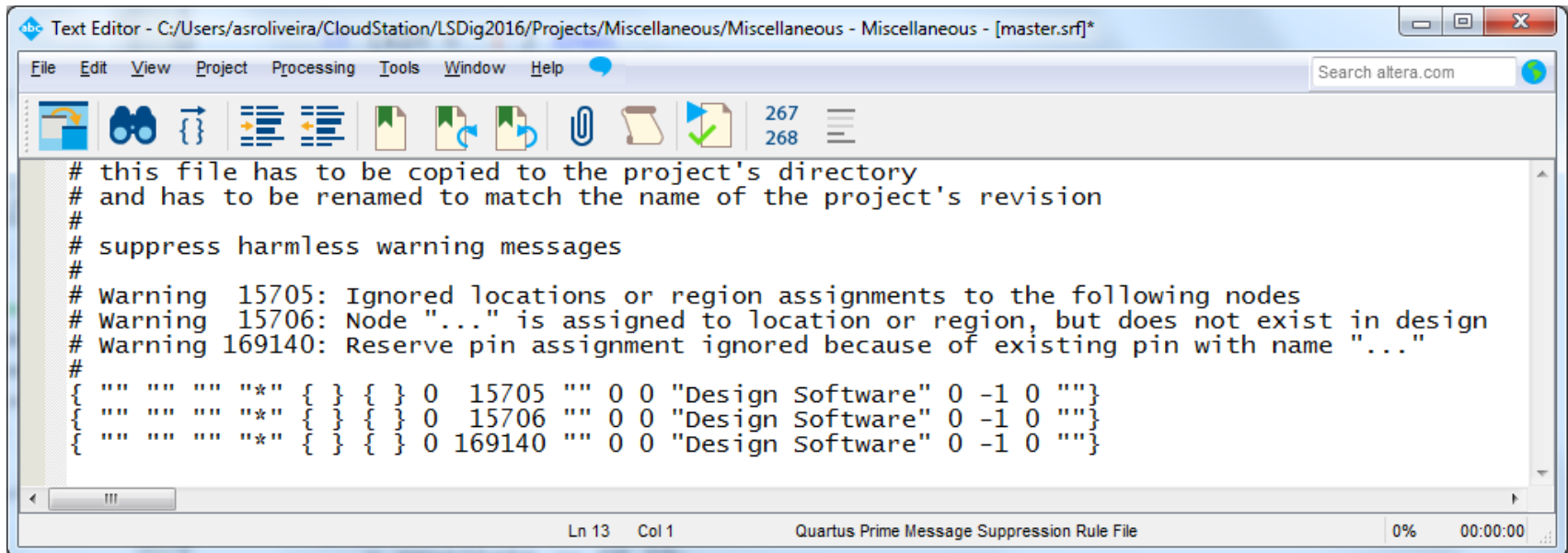
# VGA I/O
# given the inverted phase relationship between vga_clk and the rest
# of the vga signals, and assuming that there are no electric signal
# propagation time mismatches of more than 8ns between the FPGA pins
# and the video DAC pins, there is no need to time-constrain the vga
# signals.

set_false_path -from [get_clocks clock_50] -to [get_ports vga_*]
```

Disponível no site de LSDig (deverá ser renomeado – nome igual ao projeto – e adicionado a um projeto como ficheiro fonte)

Ficheiro “master.srf”

- Especifica os avisos (*warnings*) não críticos que não devem ser mostrados durante a compilação
 - Diminuição do número de avisos reportados para facilitar a visualização dos avisos críticos
 - Muitos deles devido a definições no ficheiro “DE2_115.qsf” que não são usadas no projeto
- Este ficheiro (como pode ler-se no seu cabeçalho) deve ter o seu nome alterado para ficar igual ao nome do projeto e deve residir na pasta do projeto



```
# this file has to be copied to the project's directory
# and has to be renamed to match the name of the project's revision
#
# suppress harmless warning messages
#
# Warning 15705: Ignored locations or region assignments to the following nodes
# Warning 15706: Node "... " is assigned to location or region, but does not exist in design
# Warning 169140: Reserve pin assignment ignored because of existing pin with name "... "
#
{ "" "" "" "*" { } { } 0 15705 "" 0 0 "Design Software" 0 -1 0 "" }
{ "" "" "" "*" { } { } 0 15706 "" 0 0 "Design Software" 0 -1 0 "" }
{ "" "" "" "*" { } { } 0 169140 "" 0 0 "Design Software" 0 -1 0 "" }
```

Ficheiro “master.srf” disponível no site de LSDig
(deverá ser colocado na pasta do projeto e renomeado – nome igual ao projeto)

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 LSDig!
- **Solução:** em projetos de LSDig que usem componentes sequenciais recomenda-se a utilização de apenas um sinal de relógio
 - A complexidade típica e as interfaces dos projetos de LSDig 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)*
- Vamos analisar um exemplo de um gerador de pulsos...

```
Text Editor - C:/Users/asroliveira/CloudStation/LSDig2016/Projects/Miscellaneous/Miscellaneous - Miscellaneous - [PulseGenerator....]
File Edit View Project Processing Tools Window Help Search altera.com
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity PulseGenerator is
port(reset : in std_logic;
sysClk : in std_logic;
pulseOut : out std_logic_vector(7 downto 0));
end PulseGenerator;

architecture Behavioral of PulseGenerator is
constant NUMBER_STEPS : positive := 6;
subtype TCounter is natural range 0 to (NUMBER_STEPS - 1);
signal s_counter : TCounter;

begin
count_proc : process(sysClk)
begin
if (rising_edge(sysClk)) 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;

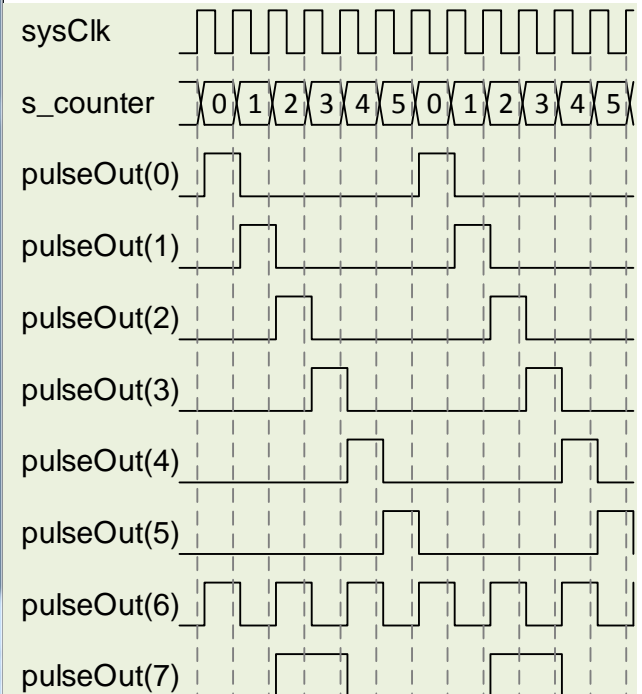
pulseOut(0) <= '1' when (s_counter = 0) else '0';
pulseOut(1) <= '1' when (s_counter = 1) else '0';
pulseOut(2) <= '1' when (s_counter = 2) else '0';
pulseOut(3) <= '1' when (s_counter = 3) else '0';
pulseOut(4) <= '1' when (s_counter = 4) else '0';
pulseOut(5) <= '1' when (s_counter = 5) else '0';
pulseOut(6) <= '1' when ((s_counter rem 2) = 0) else '0';
pulseOut(7) <= '1' when ((s_counter >= 2) and (s_counter <= 3)) else '0';

end Behavioral;
```

Cada uma das saídas pode ser usada como *enable* de um componente do circuito

Estrutura típica adaptável às necessidades de um sistema em concreto

Exemplo de um Gerador de Pulsos (*enables*) com Saídas Combinatórias



Em hardware real as saídas podem apresentar *glitches* (também observável numa simulação temporal) – não crítico em muitas situações (o importante é o pulso estar estável na vizinhança do flanco ativo do sinal de relógio (cumprimento dos tempos de *setup* e de *hold*))

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

entity RegPulseGenerator is
    port(reset      : in  std_logic;
          sysClk    : in  std_logic;
          pulseOut   : out std_logic_vector(7 downto 0));
end RegPulseGenerator;

architecture Behavioral of RegPulseGenerator is
    constant NUMBER_STEPS : positive := 6;
    subtype TCounter is natural range 0 to (NUMBER_STEPS - 1);
    signal s_counter : TCounter;

    signal s_pulseOut : std_logic_vector(7 downto 0);

begin
    count_proc : process(sysClk)
    begin
        if (rising_edge(sysClk)) 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;

    s_pulseOut(0) <= '1' when (s_counter = 0) else '0';
    s_pulseOut(1) <= '1' when (s_counter = 1) else '0';
    s_pulseOut(2) <= '1' when (s_counter = 2) else '0';
    s_pulseOut(3) <= '1' when (s_counter = 3) else '0';
    s_pulseOut(4) <= '1' when (s_counter = 4) else '0';
    s_pulseOut(5) <= '1' when (s_counter = 5) else '0';
    s_pulseOut(6) <= '1' when ((s_counter rem 2) = 0) else '0';
    s_pulseOut(7) <= '1' when ((s_counter >= 2) and (s_counter <= 3)) else '0';

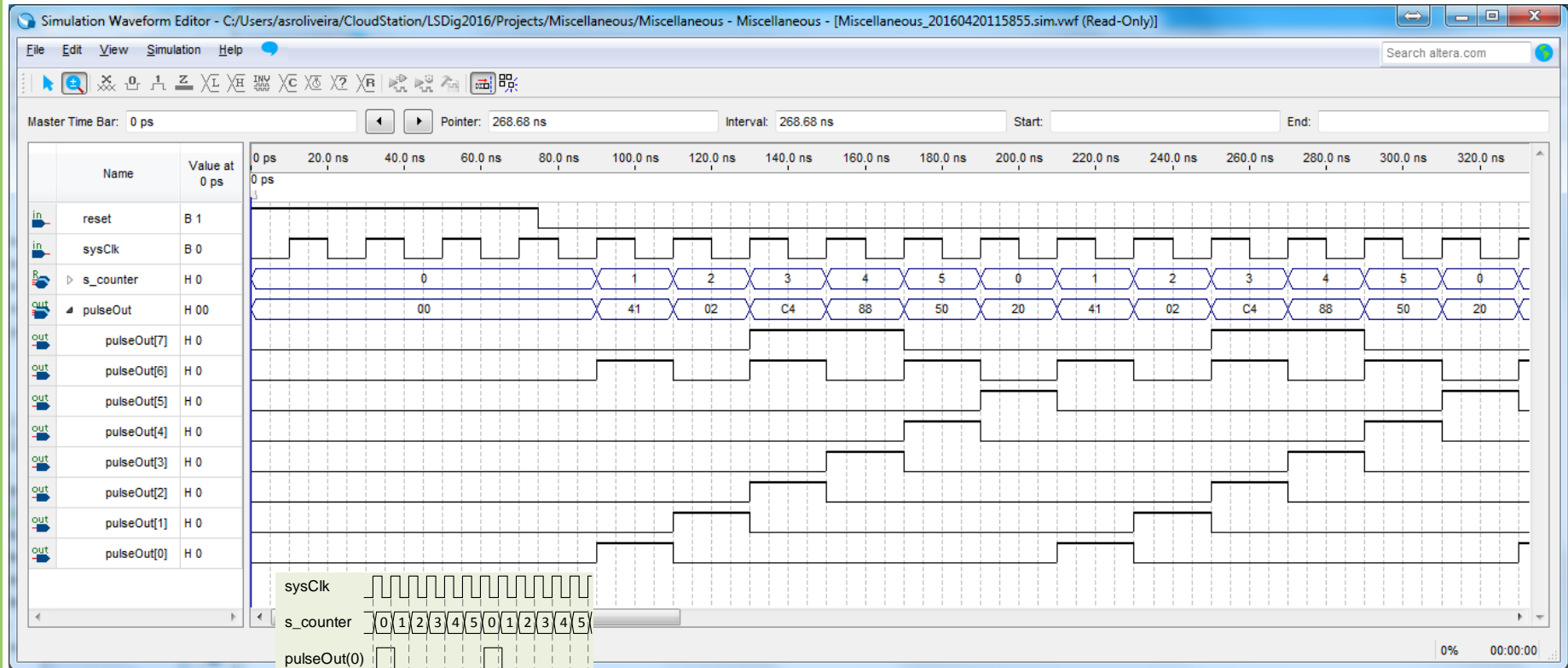
    out_reg_proc : process(sysClk)
    begin
        if (rising_edge(sysClk)) then
            if (reset = '1') then
                pulseOut <= (others => '0');
            else
                pulseOut <= s_pulseOut;
            end if;
        end if;
    end process;
end Behavioral;
```

Inclusão de um registo nas saídas do gerador de pulsos

Gerador de Pulsos com Saídas Registadas (elimina os *glitches*)

- A colocação de um registo nas saídas do gerador de pulsos:
 - Garante que as saídas só comutam num instante bem definido, evitando *glitches*
 - Atrasa as saídas um ciclo de relógio, mas sem alterar a ordem relativa de ativação

Simulação do Gerador de Pulsos com Saídas Registadas



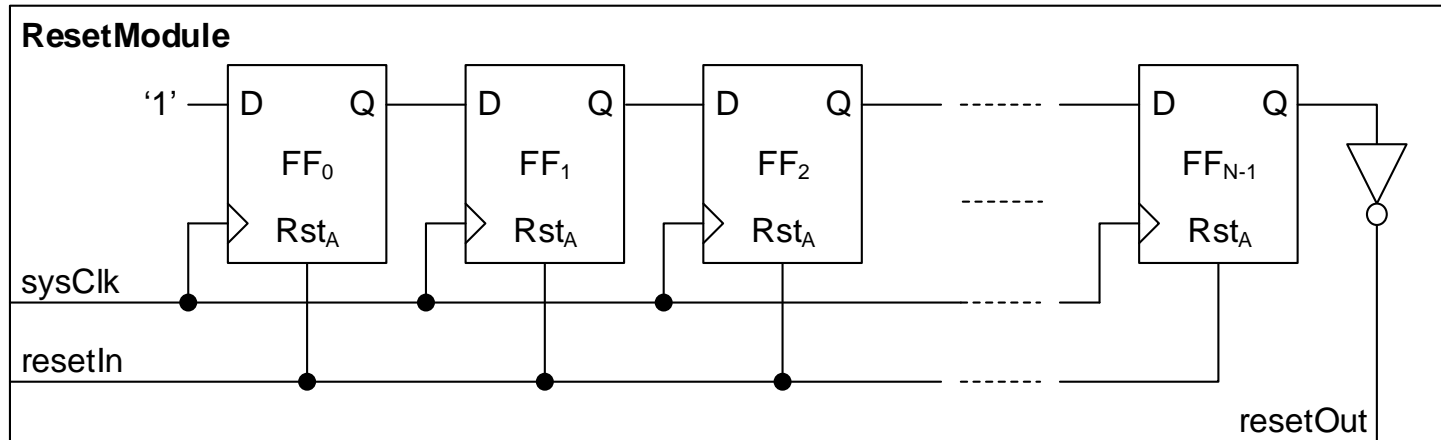
Comportamento
sem registo na
saída

Saídas atrasadas um ciclo de relógio, mas
sem alterar a ordem relativa de ativação

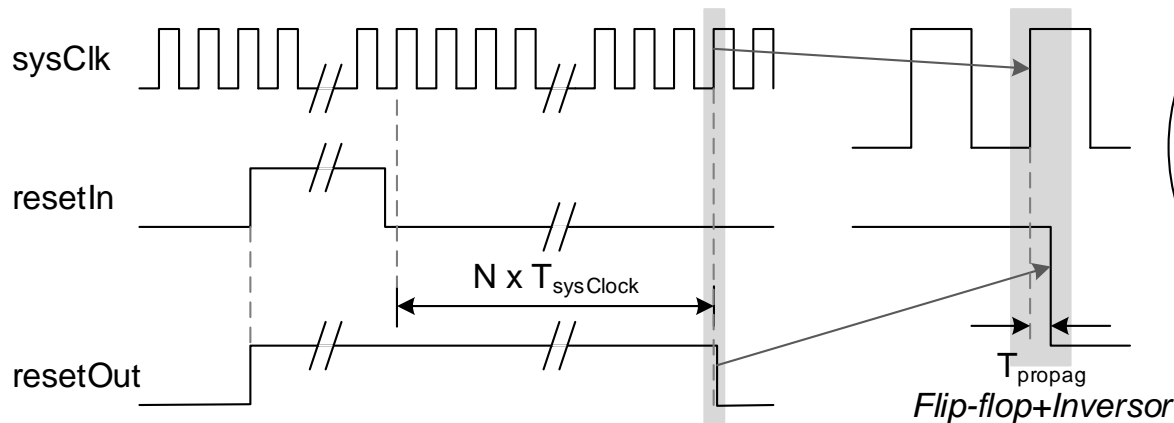
Precauções de Inicialização (*Reset*)

- 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
- Vamos ver um exemplo de um módulo que gera um sinal de *reset* nestas circunstâncias...

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;

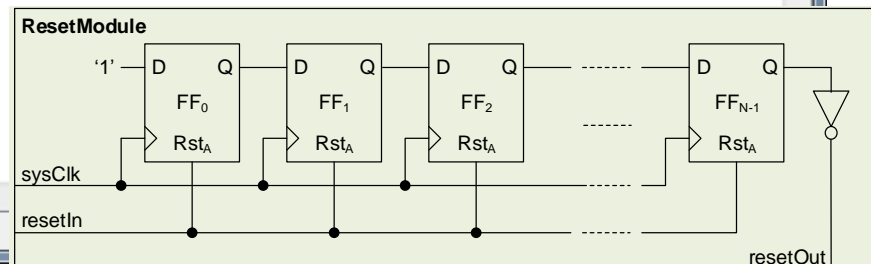
architecture Behavioral of ResetModule is
    signal s_shiftReg : std_logic_vector((N - 1) downto 0) := (others => '0');
begin
    assert(N >= 2);

    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;

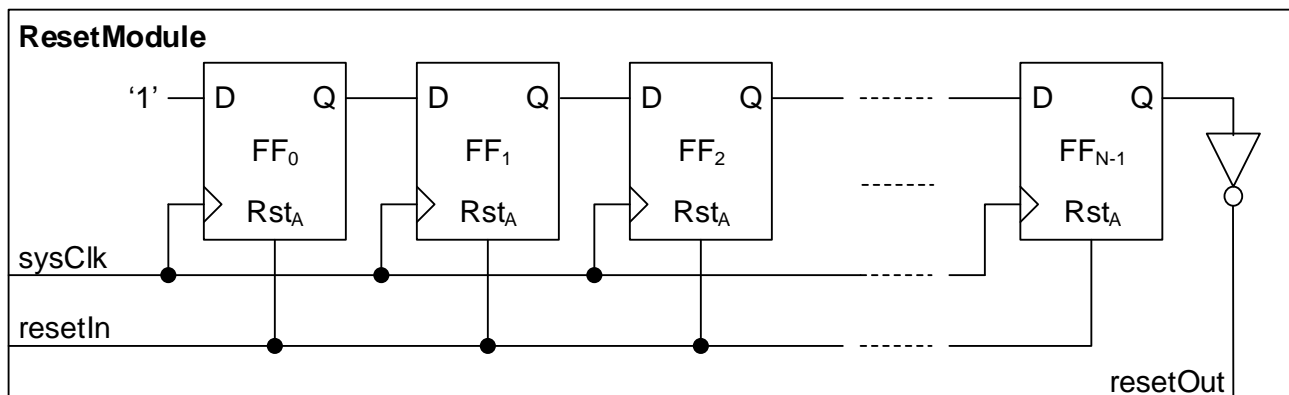
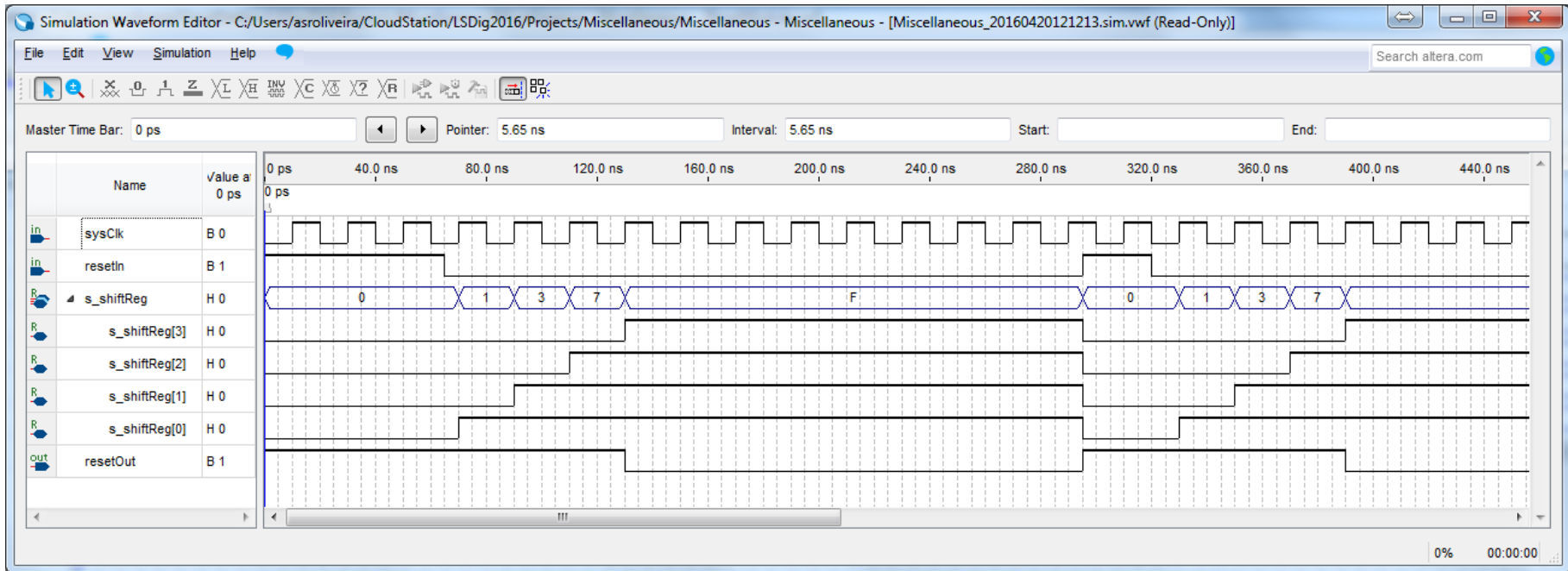
    resetOut <= not s_shiftReg(N - 1);
end Behavioral;
```

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 “*Altera Quartus*”
- **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"`)

Comentários Finais

- No final desta aula deverá ser capaz de aplicar as recomendações e boas práticas de projeto apresentadas e discutidas em LSDig
 - Fundamentais (sempre), incluindo o desenvolvimento e avaliação do projeto final!
 - Avaliada a sua aplicação no projeto final
- Ficheiros “master.sdc” e “master.srf” disponíveis no site da UC