Revisão da linguagem VHDL (Vivado)

Bancadas de teste VHDL

Kit Digilent Nexys-4 – E/S básica

AULA 2

IOUL IIA SKL I AROVA

VHDL

V circuitos integrados de alta velocidade**H** hardware**D** descrição**eu** idioma (IEEE std 1076)

- Modelagem, simulação e síntese de sistemas digitais
- Permite descrever o comportamento e a estrutura do hardware digital

A síntese do Vivado suporta um subconjunto sintetizável de:

- VHDL: Padrão IEEE para linguagem VHDL (IEEE Std 1076-2002)
- VHDL 2008
- ...

Tipos de dados VHDL suportados

- Tipos enumerados predefinidos
  - bit (padrão)
  - booleano (padrão)
  - std_logic (std_logic_1164)
    - ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')
- Tipos enumerados definidos pelo usuário
  - **tipo**ESTADOS**é**(INICIAR, OCIOSO, EXECUTAR);
- Tipos de vetores de bits
  - std_logic_vector (std_logic_1164)
- Tipos sobrecarregados
  - não assinado (numeric_std)
  - assinado (numérico_std)
- Tipos inteiros (32 bits por padrão)
- Tipos de array multidimensionais (sem restrição, mas limite a 3)
- Tipos de registro

Estrutura do código VHDL

```
biblioteca IEEE;
usar IEEE.STD_LOGIC_1164.todos;
```

Inclusão de **bibliotecas**

```
entidade???é
    porta(sel      :em    std_logic;
          entrada0:em    std_logic;
          entrada1:em    std_logic;
          mOut:fora std_logic); fim???;
```

Entidade-definição de interface do módulo

O VHDL diferencia maiúsculas de minúsculas?

Os identificadores estão sujeitos a certas restrições

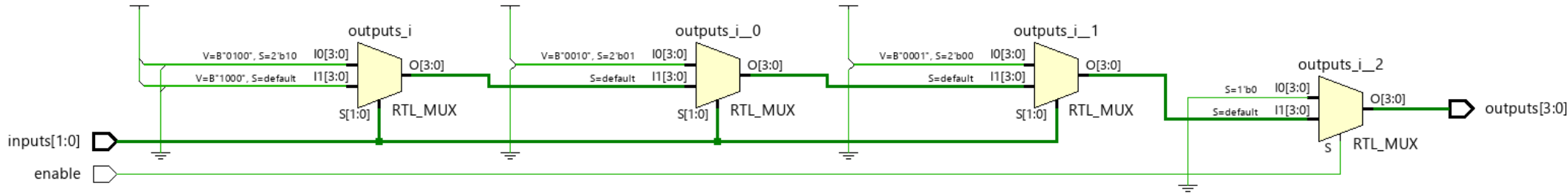
```
arquitetura Equações de???é

    sinal s_and0Out, s_and1Out: std_logic;

começar
    s_and0Out <= não sel e entrada0; s_and1Out
    <=                  sel e entrada1;
    mOut        <= s_and0Out ou s_and1Out;
fim Equações;
```

Sinal e constante declarações

Arquitetura-definição de implementação do módulo

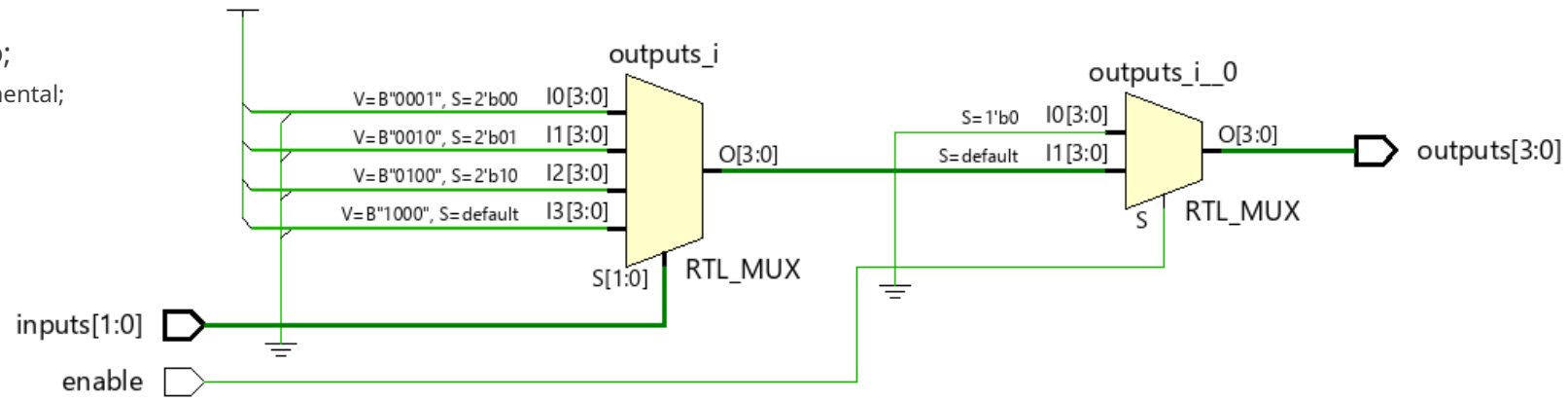
Módulo VHDL? Hardware inferido?

```
entidade???é
    porta(habilitar     :em     std_logic;
          entradas      :em     std_logic_vector (1até0);
          saídas:forastd_logic_vector (3até0)); fim???;


arquiteturaComportamentalde???é
começar
        processo(habilitar, entradas)
        começar
            se(ativar = '0')então
                saídas <= "0000";
            outro
                se(entradas = "00") Elif(entradas =  então      saídas <= "0001";
                "01")então Elif(entradas = "10")                saídas <= "0010";
                então mais                                      saídas <= "0100";
                                                                saídas <= "1000";
                fim se;
            fim  se;
        fim do processo;
fim     Comportamental;
```



Módulo VHDL? Hardware inferido?

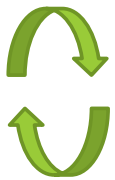
```
entidade???é
   porta(habilitar     :em      std_logic;
         entradas      :em      std_logic_vector (1até0);
         saídas:forastd_logic_vector (3até0)); fim???;


arquiteturaComportamentalde???é
começar
      processo(habilitar, entradas)
      começar
      se(ativar = '0')entãosaídas <= (outros=> '0'); outro

            caso(entradas)    é
                  quando"00"       => saídas <= x"1"; =>
                  quando"01"       saídas <= x"2"; => saídas
                  quando"10"       <= x"4";
                  quando outros=>saídas <= x"8";
            fim   caso;
      fim se;
fim   processo;
fim   Comportamental;
```



Módulo VHDL? Hardware inferido?

```
entidade???é
    porta (redefinir      :em     std_logic;
           clique         :em     std_logic;
           habilitar      :em     std_logic;
           dadosIn        :em     std_logic;
           saída de dados:forastd_logic); fim
???;
```

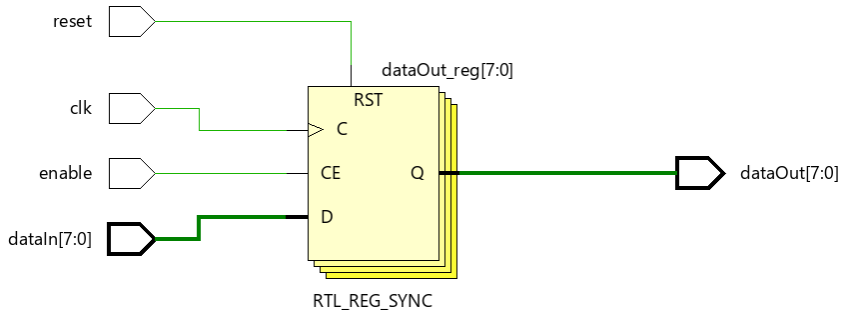
```
arquiteturaComportamental1de???é
começar
    processo(reiniciar,clk)
    começar
        se(redefinir = '1')então
            dataOut <= '0';
        Elif(borda_crescente(clk))então
            se(habilitar = '1')então
                dataOut <= dataIn;
            fim se;
        fim se;
    fim do processo;
fimComportamental1;
```



```
arquiteturaComportamental2de???é
começar
    processo(clk)
    começar
        se(borda_crescente(clk))então
            se(redefinir = '1')então
                dataOut <= '0';
            Elif(habilitar = '1')então
                dataOut <= dataIn;
            fim se;
        fim se;
    fim do processo;
fimComportamental2;
```

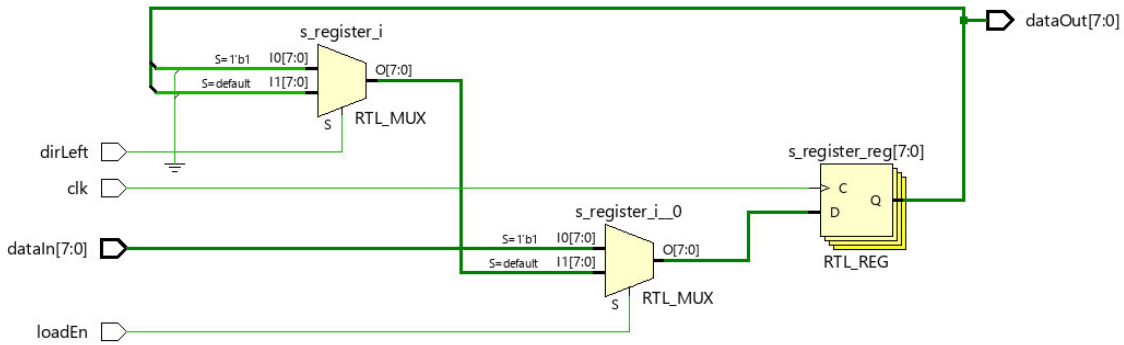
Módulo VHDL?

```
entidade???é
    genérico(N          : positivo := 8); :
    porta(reiniciar          em   std_logic;
              clique         :   em   std_logic;
              habilitar      :   em   std_logic;
              dadosIn        :   em   std_logic_vector((N-1)          até        0);
              saída de dados:forastd_logic_vector((N-1)até0)); fim???;


arquiteturaComportamentalde???é
começar
    processo(clk)
    começar
        se(borda_crescente(clk))então
            se(redefinir = '1')então
                dataOut <= (outros mais(  => '0');
            habilitar = '1')então
                dataOut <= dataIn;
            fim se;
        fim se;
    fim do processo;
fimComportamental;
```



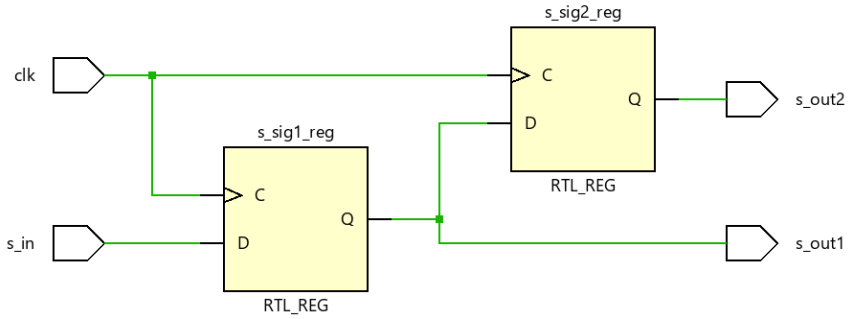
Módulo VHDL? Hardware inferido?

```
entidade???    é
   porta(clk              :   em   std_logic;
         carregarEn       :   em   std_logic;
         dadosIn          :   em   std_logic_vector(7        até       0);
         dirEsquerda      :   em   std_logic;
         saída de dados   :fora std_logic_vector(7até0));
fim???;

arquitetura Comportamental de ??? é
   sinal registro_s: std_logic_vector(7até0); começar

   processo(clk)
   começar
      se(borda_crescente(clk))então
         se(carregarEn = '1')então
            s_register <= dataIn;
         Elif(dirEsquerda = '1')então
            s_register <= s_register(6até0) & '0'; outro

            s_register <= '0' & s_register(7até1); fim se;

      fim se;
   fim do processo;

   dataOut <= s_register;

fim Comportamental;
```



Hardware inferido?

```
entidade???é
        Porta(clique        :emSTD_LOGIC;
                pecado      :emSTD_LOGIC;
                s_out1:foraSTD_LOGIC; s_out2:
                foraSTD_LOGIC);
fim???;

arquiteturaComportamentalde???é
        sinals_sig1, s_sig2: std_logic; começar
```




```
processo(clk)
começar
        se(borda_crescente(clk))então
                s_sig1 <= s_in;
                s_sig2 <= s_sig1;
        fim se;
fim do processo;

        s_out1 <= s_sig1;
        s_out2 <= s_sig2;

fimComportamental;
```

```
processo(clk)
começar
        se(borda_crescente(clk))então
                s_sig2 <= s_sig1;
                s_sig1 <= s_in;
        fim se;
fim do processo;
```

Módulo VHDL? Hardware inferido?

```
entidade???é
    genérico(K        : positivo := 4); :em
    porta(reiniciar                  std_logic;
              clicar em     :em      std_logic;
              clkOut:forastd_logic); fim???;


arquiteturaComportamentalde???é
    sinalcontador_s: naturais; começar

    processo(clicar em)
    começar
        seborda_crescente(clicar em)então
            se((redefinir = '1')ou(s_contador =K-1))então
                clkOut <= '0';
                s_counter <= 0;
            outro
                se(s_contador =K/2 - 1)então
                    clkOut <= '1';
                fim se;
                s_counter <= s_counter + 1; fim se;


        fim se;
    fim do processo;
fimComportamental;
```

Divisor de relógio

Contador de corrida livre Módulo K

clkOut <= '1' no "meio" da contagem

clkOut <= '0' no final da contagem

Supondo

$f_{clicar\ em}$ = 50MHz

K=10

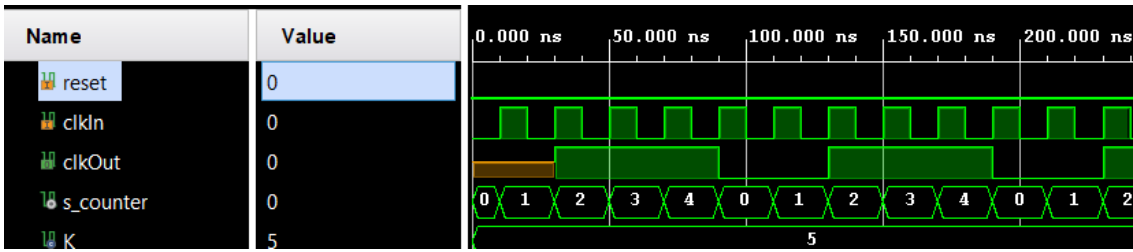
f_{clkOut}?

Ciclo de trabalho?

K=5

f_{clkOut}?

Ciclo de trabalho?



Declarações Simultâneas

- Instruções simultâneas definem uma lógica que é inerentemente paralela.
- As declarações simultâneas são avaliadas independentemente da ordem em que aparecem.
- Os sinais passam valores entre instruções simultâneas, da mesma forma que os fios conectam componentes em um esquema.
- As declarações simultâneas incluem:
  - Atribuições de sinais (simples, selecionados e condicionais)
  - Declarações de processo
  - Instanciações de componentes
  - Gerar declarações
  - Chamadas de procedimentos e funções

Atribuições de Sinais

- Atribuição de sinal simples:

```
uma <= bec;
```

- Atribuição de sinal condicional:

```
fora <= entrada1quandos = '0'outroem 2;
```

- Atribuição de sinal selecionada:

```
comentradasselecionesaídas <=
    x"1"quando"00",x"2"
    quando"01",x"4"
    quando"10",x"8"
    quando outros;
```

Declarações de Processo

Um processo inclui**declarações sequenciais**, assim chamados porque são executados em sequência.

A declaração do processo inclui uma**lista de sensibilidade**-uma lista de sinais aos quais o processo é sensível. Quando qualquer um desses sinais muda de valor, o processo retoma e executa as instruções sequenciais.

Depois de executar a última instrução, o processo é suspenso novamente. Os valores

dos sinais em um processo são atualizados quando o processo é suspenso.

Declarações sequenciais:

- declarações if
- atribuições condicionais (VHDL-2008)
- declarações de caso
- tarefas selecionadas (VHDL-2008)
- ...

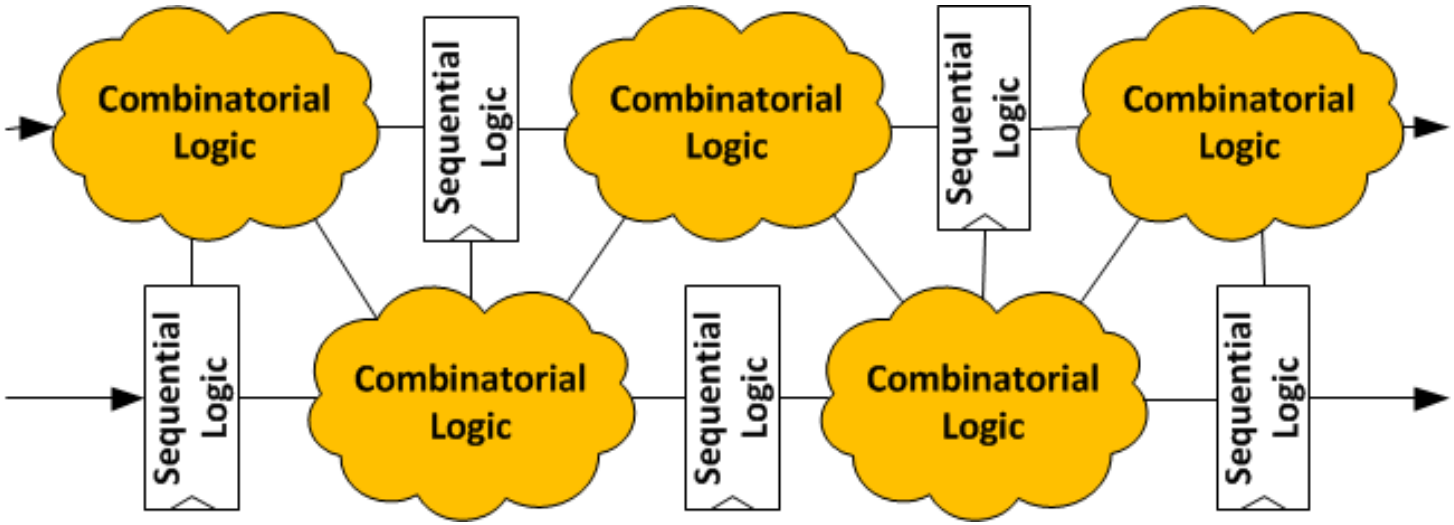
Modelo de processo VHDL típico para um componente combinacional

processo(<todas as entradas>)

começar

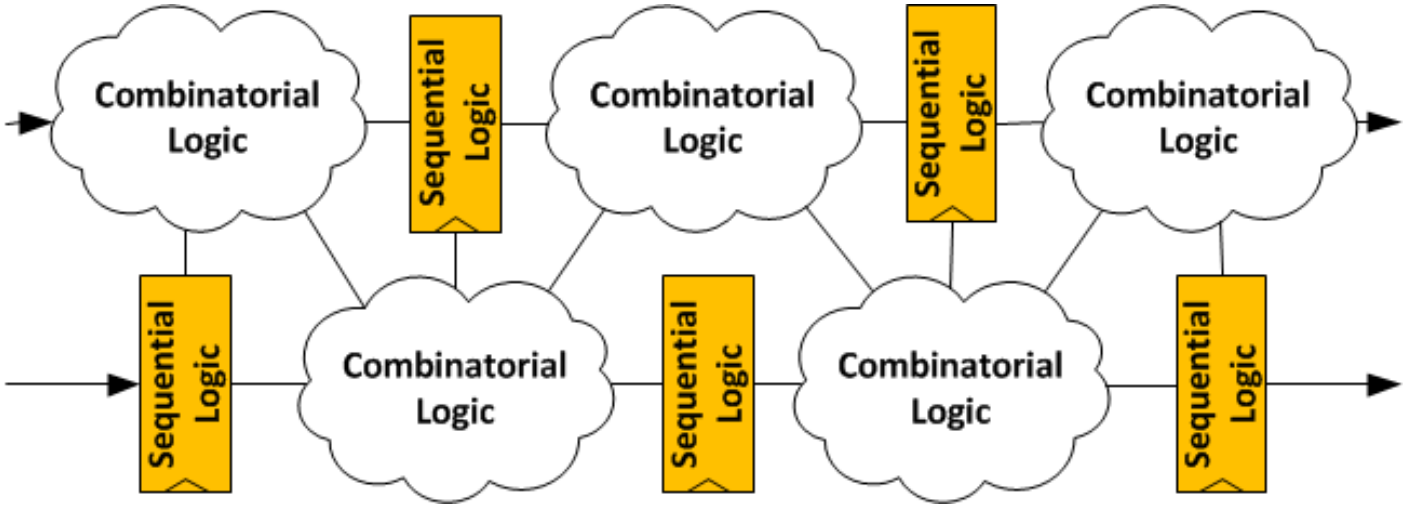
<atribuições a sinais/portas – as saídas devem ser especificadas para todas as combinações de sinais de entrada – mesmo que não se importe (para evite travas)>

fim processo;



Modelo de processo VHDL típico para um componente sequencial

processo(<configurações/redefinições de relógio e assíncronas>)
começar
<teste de sinais assíncronos>
<atribuições assíncronas> <teste de clock ativo>
<teste de sinais síncronos>
<tarefas síncronas> **fim do processo**;



Componentes

Você sabe escrever declarações de entidades e corpos de arquitetura que descrevem a estrutura de um sistema.

Dentro de um corpo de arquitetura, podemos escrever instruções de instanciação de entidade que descrevem instâncias de uma entidade e conectam sinais às portas das instâncias.

bit0:**entidade**trabalho.d_ff(básico)
mapa do porto(d0, int_clk, q0);

Esta abordagem simples para construir um design hierárquico funciona bem se conhecermos antecipadamente todos os detalhes das entidades que queremos utilizar.

No entanto, nem sempre é esse o caso, especialmente num grande projeto de design.

Os componentes são uma forma alternativa de descrever a estrutura hierárquica de um projeto que proporciona significativamente mais flexibilidade ao custo de um pouco mais de esforço no gerenciamento do projeto.

Componentes

```
16  Entity tutorial_tb Is
17  end tutorial_tb;
18
19  Architecture behavior of tutorial_tb Is
20      Component tutorial
21      port (
22          sw  : in STD_LOGIC_VECTOR(7 downto 0);
23          led : out STD_LOGIC_VECTOR(7 downto 0)
24          );
25      End Component;
26
27      Signal switch : STD_LOGIC_VECTOR(7 downto 0) := X"00";
28      Signal led_out : STD_LOGIC_VECTOR(7 downto 0) := X"00";
29      Signal led_exp_out : STD_LOGIC_VECTOR(7 downto 0) := X"00";
30
31      Signal count_int_2 : STD_LOGIC_VECTOR(7 downto 0) := X"00";
32
33      procedure expected_led (...
49
50  begin
51      uut:  tutorial PORT MAP (
52              sw  => switch,
53              led => led_out
54           );
```

Declaração de componente: especifica a interface externa ao componente em termos de constantes genéricas e portas

Instanciação de componentes: especifica um uso do módulo em um design

Gerar declarações

Gerar declaraçãoé uma declaração concorrente contendo outras declarações concorrentes que devem serreplicadodurante a elaboração de um projeto.

```
entidade???é
        porta(código_in         :emSTD_LOGIC_VECTOR (3até0); :em
                    pt              STD_LOGIC;
                    código_out:foraSTD_LOGIC_VECTOR (15até0));
fim???;

arquiteturaComportamentalde???é
começar
ger_out:paraeuemcódigo_out'geração de intervalo
    code_out(i) <= ptquando(i = to_integer(unsigned(code_in)))
                                outro'0';
final gerar;

fimComportamental;
```

Erros típicos de especificação

Quando o valor de um sinal/porta não é especificado para uma ou mais combinações de entrada, a ferramenta de síntese infere um elemento de memória para esse sinal/porta (por quê?):

- Chinelo de dedo
- Robusto

Esta situação pode ser absolutamente aceitável (para circuitos sequenciais) ou indesejável (para circuitos combinacionais).

Erros típicos de especificação

```
processo(habilitar, dataIn)
começar
    se(habilitar = '1')então
        dataOut <= dataIn;
    fim se;
fim do processo;
```

Sequencial (trava)

```
processo(clk)
começar
    se(clk'evento eclk = '1')então
        dataOut <= dataIn;
    fim se;
fim do processo;
```

Sequencial (flip-flop)

```
processo(decodificação)
começar
        se(decodificaçãoIn(1) = '1')então
                validOut <= '1';
                encodOut <= "1";
        Elif(decodificação(0) = '1')então
                validOut <= '1';
                encodOut <= "0";
        outro
                validOut <= '0';
                encodOut <= "-";
        fim se;
fim do processo;
```

Combinacional (codificador de prioridade 2:1) Se a linha**encodOut<="-"**é removido, o sinal**encodOut**não está especificado para **decodificaçãoIn="00"**, levando as ferramentas a inferir uma trava para este sinal!

Múltiplas atribuições a um sinal

Se múltiplas atribuições forem feitas a um sinalem um processo , segundo a semântica do VHDL, prevalece a última.

Esta facilidade permite tornar o código mais compacto.

```
processo(decodificação)
começar
        validOut <= '1';
        se(decodificaçãoIn(1) = '1')então
              encodOut <= "1";
        Elif(decodificação(0) = '1')então
              encodOut <= "0";
        outro
                validOut <= '0';
                encodOut <= "-";
        fim se;
fim do processo;
```

No entanto, apenas uma instrução simultânea pode controlar um sinal (exceção: sinais multi-driver de três estados).

Simulação em VHDL - Testbenches

MÓDULOS COMBINACIONAIS

Entidade sem portas

Arquitetura:

- Declaração da UUT (**Unidade em teste**) na parte declarativa da arquitetura
- Declaração de sinais a serem conectados às portas UUT na parte declarativa da arquitetura
- Instanciação de UUT no corpo da arquitetura
- Definindo um processo gerando os vetores de simulação ao longo do tempo
  - Em sistemas mais complexos, mais de um processo pode ser utilizado para esse fim

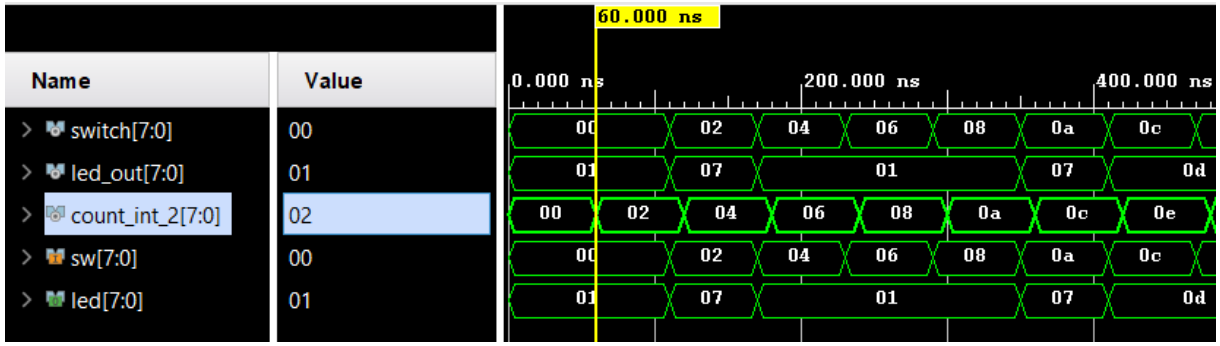
MÓDULOS SEQUENCIAIS

Entidade sem portas

Arquitetura:

- Declaração da UUT (**Unidade em teste**) na parte declarativa da arquitetura
- Declaração de sinais a serem conectados às portas UUT na parte declarativa da arquitetura
- Instanciação de UUT no corpo da arquitetura
- Definindo um processo para gerar o sinal de clock
- Definindo um processo para aplicar os vetores de simulação ao longo do tempo
  - Em sistemas mais complexos, mais de um processo pode ser utilizado para esse fim

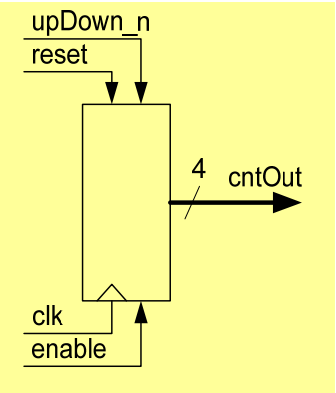
Exemplo de banco de testes VHDL (CS)




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

entidadetutorial_tbé
fimtutorial_tb;

arquiteturacomportamentodetutorial_tbé
        componentetutorial
        porta(      sw:emSTD_LOGIC_VECTOR(7até0); liderado :fora
                    STD_LOGIC_VECTOR(7até0));
        componente final;

        sinalinterruptor: STD_LOGIC_VECTOR(7até0) :=X"00"; sinalled_out :
        STD_LOGIC_VECTOR(7até0) :=X"00"; sinalcontagem_int_2: não assinado (7
        até0) :=X"00";
começar
        fora: tutorialMAPA DO PORTO(sw      => mudar,
                                    liderado => led_out);

        processo_comb:processo
        começar
                Esperar por50ns;
                mudar <= std_logic_vector(count_int_2); Esperar por
                10ns;
                contagem_int_2 <= contagem_int_2 + x"02";
        fim do processo;
fim comportamento;
```

Exemplo de banco de testes VHDL (SS)

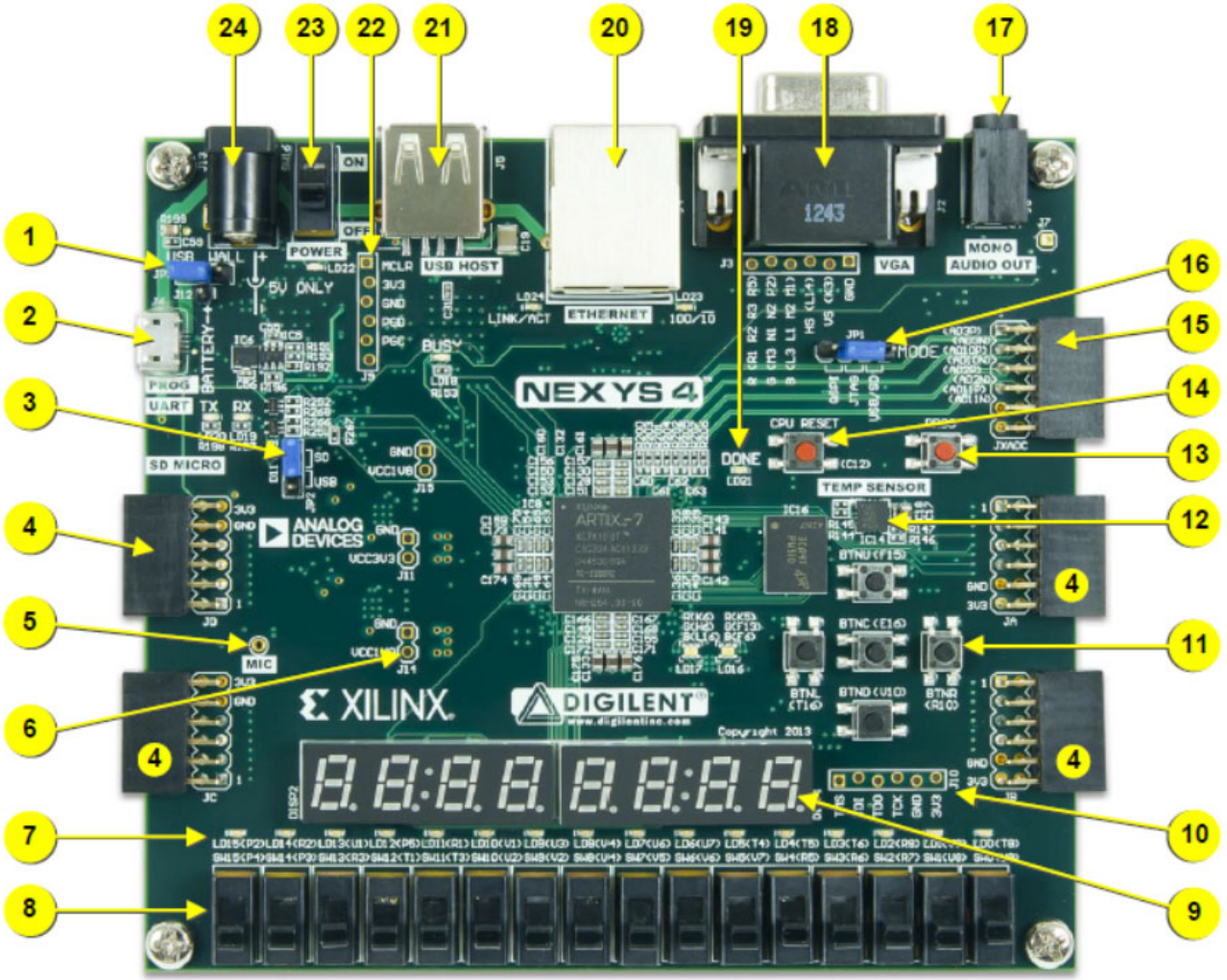
```
entidadeBinUDCntEnRst8Tbeu
envioBinUDCntEnRst8Tb;

arquiteturaEstímulodeBinUDCntEnRst8Tbé
    sinals_reset, s_clk                    : std_logic;
    sinals_enable, s_upDown_n: std_logic; sinals_cntOut:
    std_logic_vector(3até0);

    <Declaração do componente BinUDCntEnRst4>

começar
    uut: BinUDCntEnRst4
            mapa do porto(reiniciar     => s_reset,
                          clique        => s_clk,
                          habilitar     => s_enable,
                          upDown_n => s_upDown_n,
                          cntOut => s_cntOut);

    relógio_proc:processo
    começar
        s_clk <= '0';Esperar por100ns; s_clk <=
        '1';Esperar por100ns; fim do processo;
```




```
estímulo_proc:processo
    começar
        s_reset         <= '1';
        s_enable        <= '0';

        s_upDown_n <= '1';
        Esperar por325ns;

        s_reset         <= '0';
        Esperar por25ns;

        s_enable        <= '1';
        Esperar por925ns;
        s_enable <= '0';
        Esperar por375ns;

        s_upDown_n <= '0';
        s_enable <= '1';
        Esperar por975ns;

        s_enable        <= '0';
        Esperar por125ns;
    fim do processo;
fimEstímulo;
```

Placa de Desenvolvimento Nexys-4

- 16 interruptores de usuário
- 16 LEDs de usuário
- 2 LEDs tricolores
- 6 botões
- Oscilador de 100 MHz
- ...



FPGA: xc7a100Tcsg324-1

E/S básica Nexys-4

- 2 LEDs tricolores
- 16 interruptores deslizantes
- 6 botões
- 16 LEDs individuais

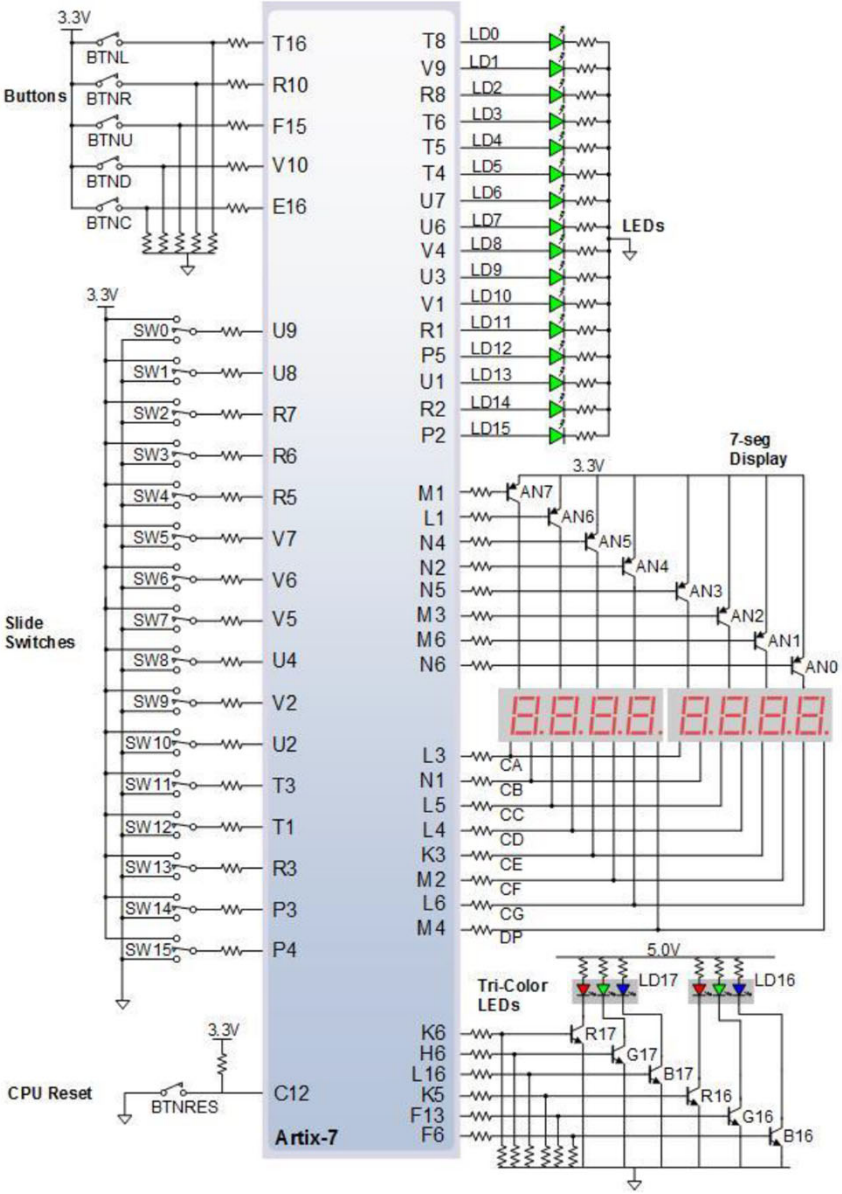
Os cinco botões geram uma saída baixa quando estão em repouso e uma saída alta somente quando são pressionados.

O botão vermelho denominado “CPU RESET” gera uma saída alta quando em repouso e uma saída baixa quando pressionado.

O botão CPU RESET destina-se a ser usado em designs Vitis para reiniciar o processador, mas você também pode usá-lo como um botão de uso geral.

Os interruptores deslizantes geram entradas constantes altas ou baixas dependendo de sua posição.

Os dezesseis LEDs individuais são conectados por ânodo ao FPGA por meio de resistores de 330 ohms, portanto, eles acenderão quando uma alta tensão lógica for aplicada ao seu respectivo pino de E/S.



LEDs tricolores

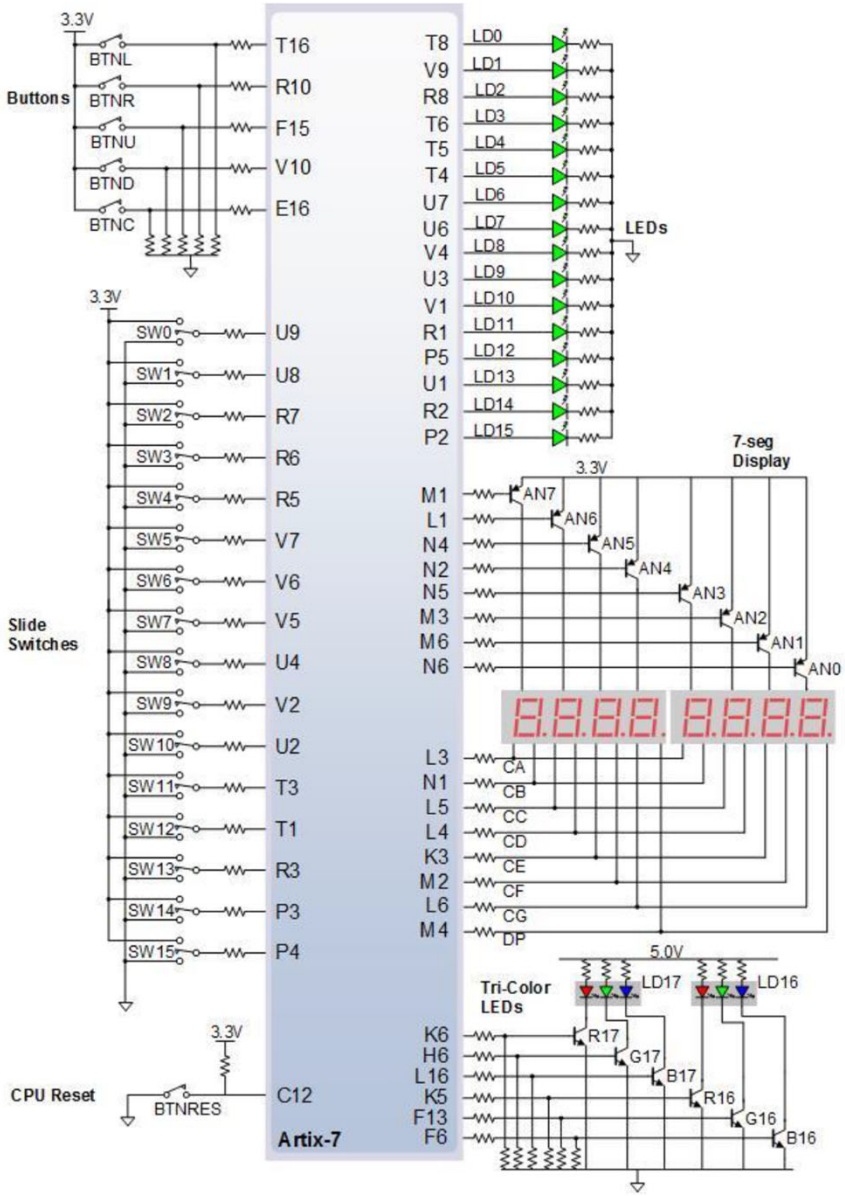
Cada LED tricolor possui três sinais de entrada que acionam os cátodos de três LEDs internos menores: um vermelho, um azul e um verde.

Dirigir o sinal correspondente a uma dessas cores para alto iluminará o LED interno.

Os sinais de entrada são acionados pelo FPGA através de um transistor, que inverte os sinais. Portanto, para acender o LED tricolor, os sinais correspondentes precisam ser elevados.

O LED tricolor emitirá uma cor dependente da combinação de LEDs internos que estão sendo iluminados no momento.

Nota: Conduzir qualquer uma das entradas para uma lógica estável '1' resultará na iluminação do LED em um nível desconfortavelmente brilhante. Você pode evitar isso garantindo que nenhum dos sinais tricolores seja acionado com um ciclo de trabalho superior a 50%.



Oscilador de relógio

A placa Nexys-4 inclui um único **100 MHz** oscilador de cristal.

CDX:

```
# Sinal de relógio
# Banco = 35, Nome do pino = IO_L12P_T1_MRCC_35,
            Nome Sch = CLK100MHZ
set_property PACKAGE_PIN E3 [get_ports clk]
            set_property IOSTANDARD LVCMOS33 [get_ports clk] create_clock -add -name
            sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

O clock de entrada pode acionar MMCMs ou PLLs para gerar clocks de várias frequências e com relações de fase conhecidas.

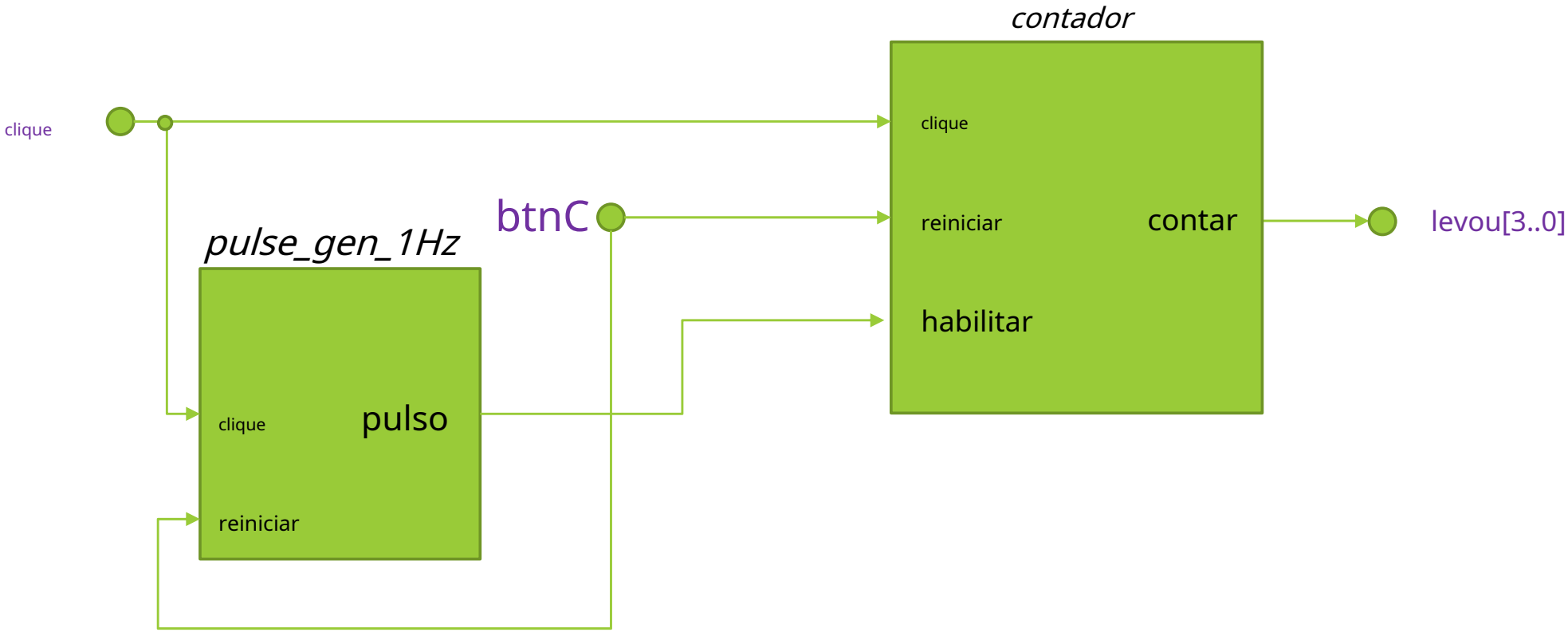
Xilinx oferece o núcleo Clocking Wizard IP para ajudar os usuários a gerar os diferentes relógios necessários para um projeto específico.

Artix-7 **blocos de gerenciamento de relógio (CMT)** fornecem funcionalidade de síntese de frequência de clock, enquadramento e filtragem de jitter. CMTs, cada um contendo um **gerenciador de relógio de modo misto (MMCM)** e um **loop de fase bloqueada (PLL)**, residem na coluna CMT próxima à coluna de E/S.

O FPGA xc7a100Tcsg324-1 inclui 6 CMTs.

Exemplo de projeto

Contador binário up de 4 bits, atualizado com frequência de 1Hz, com exibição do valor da contagem nos LEDs da placa.



Considerações finais

Ao final desta palestra você deverá ser capaz de:

- Registre as construções VHDL sintetizáveis conhecidas
- Escolha o estilo de codificação correto para descrever componentes combinacionais e sequenciais simples
- Identifique instruções VHDL simultâneas
- Declarar e instanciar componentes
- Projetar e usar bancadas de teste VHDL
- Evite erros típicos de codificação VHDL
- Crie, sintetize, implemente, analise e teste projetos VHDL simples para o kit Nexys-4 no Vivado

Pendência:

- Complete os exercícios do laboratório 2 e teste-os no kit Nexys-4