Laboratório de Sistemas Digitais

Trabalho Prático nº 5

Parametrização de componentes

Objetivos

- Modelação em VHDL, simulação, implementação em FPGA e teste de circuitos combinatórios e sequenciais parametrizáveis.
- Análise de um projeto completo para correção de erros funcionais.

Sumário

Este trabalho prático está dividido em cinco partes. Na primeira parte é abordada a implementação de um comparador parametrizável de "N" bits. Na segunda parte é implementado um acumulador de "N" bits. A terceira parte é dedicada à parametrização de um componente sequencial, baseado num temporizador. Na quarta parte é fornecido um projeto de um relógio digital, estrutural e sintaticamente correto, mas com erros funcionais que devem ser corrigidos. Finalmente, na última parte, é implementada uma versão parametrizavel da ALU implementada no trabalho prático 3.

Parte I

1. Abra a aplicação *"Quartus Prime"* e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade *top-level* como "CmpN_Demo".

2. Analise o código VHDL apresentado na Figura 1 relativo a um comparador de 4 bits.

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

entity Cmp4 is
  port(input0     : in  std_logic_vector(3 downto 0);
       input1     : in  std_logic_vector(3 downto 0);
       equal      : out std_logic;
       notEqual   : out std_logic;
       ltSigned   : out std_logic;
       ltUnsigned : out std_logic);
end Cmp4;

architecture Behavioral of Cmp4 is
begin
 equal      <= '1' when (input0 = input1) else
               '0';
 notEqual   <= '1' when (input0 /= input1) else
               '0';
 ltSigned   <= '1' when (signed(input0) < signed(input1)) else
               '0';
 ltUnsigned <= '1' when (unsigned(input0) < unsigned(input1)) else
               '0';
end Behavioral;
```

Figura 1 – Código VHDL de um comparador de 4 bits.

3. Construa uma versão parametrizável do comparador da Figura 1, em que a dimensão dos operandos seja configurável, de modo a ser possível processar operandos de qualquer tamanho, definido com base numa constante genérica (ou parâmetro) "N" fixada aquando da instanciação do componente. Para tal será necessário declarar, na entidade relativa ao comparador, a constante genérica "N" com a construção **`generic`** em VHDL e expressar as dimensões dos portos "**`input0`**" e "**`input1`**" em função dessa constante. Não atribua, para já, nenhum valor por omissão à constante genérica "N". O nome da entidade deverá ser "CmpN". Grave o ficheiro com o nome "CmpN.vhd".

4. Selecione o ficheiro "CmpN.vhd" como o *top-level* do projeto e execute a opção "*Analysis & Synthesis*". Qual o erro reportado pelas ferramentas durante a análise do projeto? Como resolvê-lo?

5. Atribua agora o valor por omissão 8 à constante genérica "N".

6. Efetue a simulação do componente "CmpN", realizando para tal os seguintes passos:

- selecione o ficheiro "CmpN.vhd" como o *top-level* do projeto e execute a opção "*Analysis & Synthesis*".
- crie um ficheiro VWF de suporte à simulação, selecione os portos a usar e especifique os vetores de entrada de forma a validar adequadamente as várias saídas do comparador para diversos operandos de entrada, entre os quais: "01111111" e "11111111"; "10000000" e "00000000"; "01111111" e "10000000" ; "00000000" e "00000000".
- execute a simulação e analise os resultados, alterando a base e tipo de representação dos operandos entre binário, hexadecimal, decimal sem sinal e decimal com sinal (opção *"Radix"* no menu que surge quando se clica com o botão direito do rato em cada uma das entradas na janela com os resultados da simulação). De notar que a alteração da base e tipo de representação afeta apenas a forma como os valores são visualizados.

7. Após a simulação crie um símbolo para o comparador "CmpN" de forma a poder instanciá-lo em diagramas lógicos.

8. Crie em seguida um novo ficheiro, chamado "CmpN_Demo.bdf" de forma a instanciar graficamente o comparador. Após instanciar o comparador, atribua o valor 4 ao parâmetro "N" na respetiva caixa de configuração junto ao símbolo. Ligue cada uma das entradas "**`input0`**" e "**`input1`**" a 4 interruptores (SW[3..0] e SW[7..4], respetivamente) e as saídas a 4 LEDs verdes.

9. Importe as definições de pinos da FPGA do *kit* DE2-115 (ficheiro "master.qsf"), efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do comparador.

10. Adicione ao ficheiro "CmpN_Demo.bdf" uma nova instância do módulo "CmpN" atribuindo o valor 5 ao parâmetro "N". Ligue as entradas "**`input0`**" e "**`input1`**" da nova instância do comparador a interruptores (SW[17..13] e SW[12..8], respetivamente) e as saídas a 4 LEDs vermelhos do *kit* DE2-115.

11. Volte a importar as definições de pinos da FPGA do *kit* DE2-115 (ficheiro "master.qsf"). Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento dos dois comparadores.

Parte II

Nos próximos pontos será construído um acumulador parametrizável de "N" bits com base num somador e num registo, ambos parametrizáveis. Um acumulador adiciona em cada flanco ativo do sinal de relógio, o valor presente na entrada ao valor armazenado no registo.

1. Abra a aplicação "*Quartus Prime*" e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade *top-level* como "AccN_Demo".

2. Construa em VHDL um somador, cuja interface é apresentada na Figura 2. O número de bits dos operandos e do resultado são parametrizáveis com base na constante genérica "N" definida na respetiva entidade ("AdderN") com a construção **`generic`** em VHDL. Grave o ficheiro com o nome "AdderN.vhd".

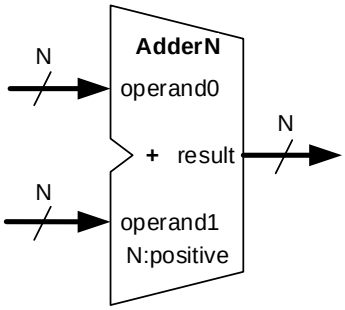

Figura 2 – Interface do módulo "AdderN".

3. Construa em VHDL um registo, cuja interface é apresentada na Figura 3. O número de bits de entrada e de saída são parametrizáveis com base na constante genérica "N" definida na respetiva entidade ("RegN"). Grave o ficheiro com o nome "RegN.vhd".

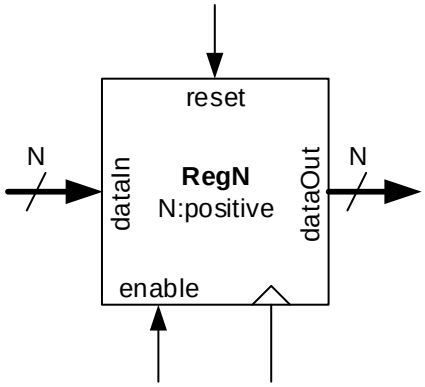

Figura 3 – Interface do módulo "RegN".

4. Crie em VHDL um ficheiro, chamado "AccN.vhd" de forma a construir um acumulador parametrizável de N bits, cuja interface está representada na Figura 4. A interligação do somador "AdderN" e do registo "RegN" de forma a construir o acumulador é apresentada na Figura 5. O parâmetro "N" declarado na entidade "AccN" deve ser propagado para o somador e para o registo nas respetivas instanciações em VHDL com a construção **`generic map`**.

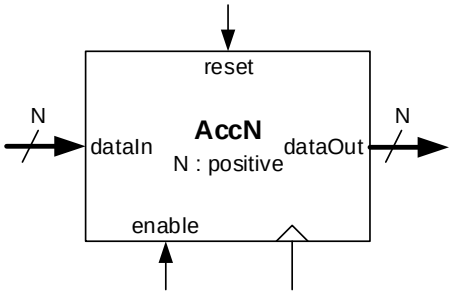

Figura 4 – Interface do módulo "AccN".

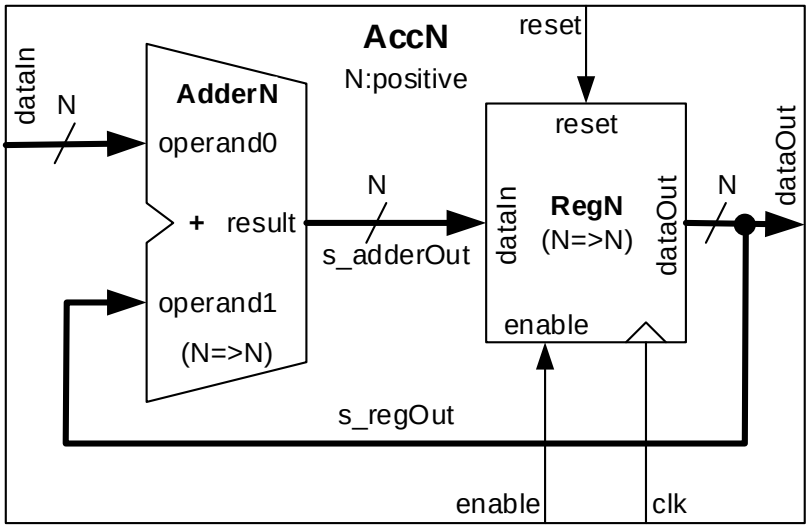

Figura 5 – Instanciação e interligação dos módulos “AdderN” e “RegN” para construção do “AccN”.

5. Crie um símbolo para o acumulador “AccN” de forma a poder instanciá-lo em diagramas lógicos.

6. Crie em seguida um novo ficheiro, chamado “AccN_Demo.bdf” de modo a instanciar graficamente o acumulador “AccN”. Após instanciar o comparador, atribua o valor 8 ao parâmetro “N” na respetiva caixa de configuração junto ao símbolo. Ligue as entradas "**`dataIn`**" a 8 interruptores (SW[7..0]) e as saídas "**`dataOut`**" a LEDR[7..0]. As entradas "**`reset`**" e "**`enable`**" podem ser ligadas a SW[8] e SW[9], respetivamente. A entrada de relógio, "**`clk`**", pode ser controlada de forma manual pelo botão KEY[0].

7. Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do acumulador.

[TPC1] Adicione ao ficheiro “AccN_Demo.bdf” uma nova instância do acumulador, com N igual a 4. Ligue as entradas "**`dataIn`**" a 4 interruptores (SW[13..10]) e as saídas "**`dataOut`**" a LEDR[13..10]. As entradas "**`reset`**" e "**`enable`**" podem ser ligadas a SW[14] e SW[15], respetivamente. A entrada de relógio, "**`clk`**", pode ser controlada de forma manual pelo botão KEY[1]. Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento de ambos os acumuladores.

[TPC2] Crie, compile e teste uma nova arquitetura para o módulo “AccN”, na qual descreva comportamentalmente o funcionamento do acumulador com base num único processo.

[TPC3] Acrescente ao acumulador um sinal que permita detetar a ocorrência de situações de *overflow*, assumindo que a entrada de dados e a saída possuem quantidades sem sinal.

Parte III

1. Abra a aplicação "*Quartus Prime*" e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade *top-level* como "TimerN_Demo".

2. Construa em VHDL um temporizador em que a saída é ativada após o disparo do temporizador e desativada após ter decorrido o intervalo de tempo N×T, com N programável estaticamente e sendo T o período do sinal de relógio aplicado ao temporizador. A interface do temporizador é apresentada na Figura 6. O registo de contagem deve ser de 32 bits (baseado no tipo de dados **`integer`** em VHDL).

Para ilustrar o funcionamento do temporizador, a Figura 7 mostra o resultado de uma simulação do mesmo (para N=6). Por conveniência é também apresentada a evolução do registo interno de contagem (representado pelo sinal "s_count"). A entrada "reset" quando ativa inicializa o temporizador. A entrada "enable" quando inativa congela o estado do temporizador. A entrada "start", quando ativa no flanco ascendente do sinal de relógio, dispara o temporizador. A saída do temporizador é o sinal "timerOut". Grave o seu código no ficheiro "TimerN.vhd".

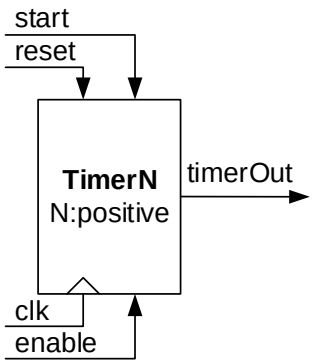

Figura 6 – Interface do módulo ""TimerN".

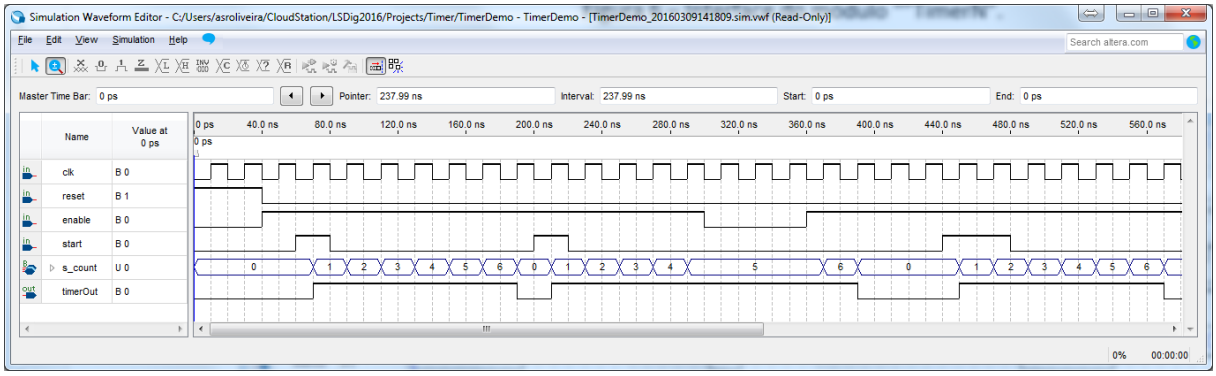

Figura 7 – Resultado da simulação do módulo "TimerN" para N=6.

3. Efetue a simulação do componente "TimerN" para diversos valores de "N" (e.g. 2 e 6), realizando a sequência de passos habitual.

4. Crie um símbolo para o temporizador "TimerN" de forma a poder instanciá-lo num diagrama lógico.

5. Crie em seguida um novo ficheiro, chamado "TimerN_Demo.bdf" de modo a instanciar graficamente o temporizador. Ligue as entradas "**reset**", "**start**" e "**enable**" a 3 interruptores (SW[2..0]) e a saída "**timerOut**" a LEDR[0]. A entrada "**clk**" deve ligar ao pino "CLOCK_50", correspondente ao relógio de 50 MHz do *kit* DE2-115. Atribua ao parâmetro "N" um valor que corresponda a um tempo de ativação da saída de 2 segundos, sendo T=1/50 MHz.

6. Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do temporizador.

Parte IV

Juntamente com este guião está disponível no *site* de LSDig um ficheiro ZIP ("Projeto para análise e correção de erros funcionais (relógio digital") que possui código fonte em VHDL relativo a um relógio digital (HH:MM:SS) implementável no *kit* DE2-115 e cujo o objetivo é analisarem e corrigirem alguns erros funcionais. O código disponibilizado está sintaticamente correto, pelo que pode ser compilado com sucesso e o ficheiro resultante descarregado na FPGA. No entanto, possui alguns erros funcionais que impedem a correta operação do relógio. A sua análise e correção consiste na sequência de tarefas elencadas nos seguintes pontos.

1. Abra a aplicação "*Quartus Prime"* e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade top-level como "BasicWatch".

2. Importe os ficheiros VHD fornecidos (com código fonte) para o projeto criado no ponto anterior.

3. Selecione o ficheiro "BasicWatch.vhd" como o *top-level* do projeto.

4. Compile o projeto e teste-o no *kit* DE2-115 (não se esqueça de importar o ficheiro "master.qsf").

5. Verifique o funcionamento do relógio. Para acertar o relógio deverá usar a tecla KEY[1] juntamente com:

- a KEY[0], para acerto dos minutos.
- a KEY[2], para acerto das horas.

6. Constate os seguintes problemas funcionais do relógio disponibilizado:

- A frequência de incremento do relógio não está correta.
- O relógio comporta-se bem em todas as transições, exceto quando alcança as 23:59:59. Neste caso, em vez de transitar para 00:00:00, prossegue até 29:59:59 e só depois transita para 00:00:00.
- Apesar de não ser propriamente um problema, o dígito mais esquerda do campo das horas está sempre acesso (mesmo que seja 0), o que normalmente não acontece nos relógios comerciais.

7. Elabore no seu *log book* um diagrama de blocos do módulo "BasicWatch" com todos os portos, sinais e interligações dos componentes que o constituem (semelhante ao que obteria se o módulo "BasicWatch" tivesse sido construído na forma de um diagrama lógico num ficheiro ".bdf"). Observe para que efeito é usada a parametrização nos módulos "ClkDividerN" e "Counter4Bits".

8. Analise com atenção o código fornecido (incluindo os comentários incluídos na declaração dos sinais no *top-level*) e implemente soluções que permitam resolver os problemas identificados. Tenha em atenção as seguintes opções que foram tomadas durante o desenvolvimento do projeto fornecido e que devem ser preservadas:

- Todos os blocos sequenciais do sistema são sincronizados por único sinal de relógio "**`s_clk4Hz`**" com a frequência de 4 Hz obtida diretamente do sinal "**`CLOCK_50`**" (50 MHz).
- Todos os contadores possuem um sinal de *"enable"*, ativo 1 em cada 4 ciclos de relógio do sinal "**`s_clk4Hz`**" (em funcionamento normal), ou ativo permanentemente, em caso de modo de acerto (para incremento rápido do valor do contador).
- Todos os contadores possuem um segundo sinal de *"enable"* que controla o incremento correto (transporte) entre os vários campos do relógio.

De notar que as soluções esperadas para os problemas não requerem grandes alterações estruturais do código fornecido. Assim, analise o código (incluindo o papel e o *timing* de cada sinal de *clock*, *reset* e *enable*) e reflita adequadamente sobre as soluções a implementar antes de efetuar alterações aos ficheiros fornecidos.

9. Compile o projeto com as correções introduzidas e teste-as adequadamente no *kit*.

[TPC] Adicione ao projeto um ficheiro chamado "BasicWatch.bdf" onde implemente na forma de um diagrama lógico, a mesma estrutura, interligações e interface do ficheiro "BasicWatch.vhd" (de notar que deve criar símbolos para todos os componentes usados no *top-level*). Remova o ficheiro "BasicWatch.vhd" do projeto, selecione o "BasicWatch.bdf" como novo *top-level* do projeto, importe o ficheiro "master.qsf", compile e teste no *kit*.

Parte V

O objetivo desta parte do guião é substituir a implementação da ALU da parte II do guião 3 ("Modelação em VHDL e implementação de circuitos aritméticos"), a qual só suporta operandos de 4 bits, por uma implementação parametrizável, em que o número de bits dos operandos (e consequentemente do resultado) pode ser especificado durante a instanciação.

1. Abra na aplicação "*Quartus Prime"* o projeto relativo à parte II do guião 3 ("Modelação em VHDL e implementação de circuitos aritméticos").

2. Crie um novo ficheiro chamado "ALUN.vhd", baseado no "ALU4.vhd", em que dimensão dos operandos e resultado da ALU são parametrizáveis, de modo a que seja possível processar operandos de qualquer tamanho, definido com uma constante genérica "N" fixada aquando da instanciação da ALU. Para tal será necessário declarar a constante genérica "N" com a construção **`generic`**, inicializá-la com um valor por omissão (8 no código seguinte) e expressar as dimensões dos portos "**`a`**", "**`b`**", "**`r`**" e "**`m`**", e dos sinais "**`s_a`**", "**`s_b`**" e "**`s_m`**" em termos desta constante. O excerto de código da Figura 8 ilustra algumas das modificações que devem ser realizadas.

3. Crie um símbolo para poder usar o módulo "ALUN.vhd" em diagramas lógicos.

4. Substitua no ficheiro "ALUDemo.bdf", a instanciação do módulo "ALU4" pelo "ALUN" configurada para processar operandos de 4 bits, i.e. deve atribuir, aquando da instanciação,

o valor 4 ao parâmetro “N”. Ligue as entradas e as saídas do módulo “ALUN” aos mesmos pinos usados no teste do módulo “ALU4”.

5. Efetue a síntese e implementação do projeto através do comando “*Compile Design*”. No final do processo de compilação, programe a FPGA e teste a ALU no *kit* de desenvolvimento.

```
...

entity ALUN
    generic(N : positive := 8);
    port(a, b : in std_logic_vector(N - 1 downto 0);
    ...
end ALUN;

architecture Behavioral of ALUN is

    signal s_a, s_b : unsigned(N - 1 downto 0);
    signal s_m      : unsigned((2 * N) - 1 downto 0);

    ...
```

Figura 8 – Indicação de algumas das alterações necessárias para construção do módulo “ALUN” a partir do “ALU4”.

6. Realize as alterações do projeto para que a ALU seja instanciada para operandos de 6 bits e sejam usados os interruptores e os LEDs do *kit* DE2-115 necessários para a testar.

7. Efetue a síntese, implementação e teste da nova instanciação da ALU.

PDF criado em 22/01/2025 às 14:23:28