Biblioteca de Blocos de CI
$1^{\underline{a}}$ Versão

Paulo Flores, Horácio Neto

PRAXIS XXI - EuroLasic

Actividade 3.1 - Relatório D3.9
INESC
Instituto de Engenharia de Sistemas e Computadores

Índice

1 Introdução **2**

2 Biblioteca IC-Blocks **2**

2.1 BIN_SEGD 3

2.2 COUNTER 4

2.3 DATA_BUS 5

2.4 DEMUX 5

2.5 MUX 6

2.6 SR 6

3 Exemplo de Aplicação **7**

A Código VHDL da Biblioteca IC-Blocks **10**

1 Introdução

Este relatório descreve o trabalho realizado no INESC no desenvolvimento de uma biblioteca de macro-células parametrizáveis. O conjunto de células tecnologicamente independentes e parametrizáveis desenvolvidas até ao momento são baseadas em descrições VHDL de blocos frequentemente utilizados em circuitos digitais.

A biblioteca de células proposta nesta fase, *IC-Blocks*, é descrita na secção 2. Um exemplo de aplicação é apresentado na secção 3. Em anexo encontra-se a descrição completa em VHDL da biblioteca de células parametrizáveis - IC-Blocks.

2 Biblioteca IC-Blocks

A correcta utilização de uma linguagem de descrição de *hardware*, como o VHDL, requer por parte do projectista um profundo conhecimento quer da linguagem, para tirar o máximo partido de todas as suas potencialidades, quer do ambiente de síntese onde ela será usada, para utilizar o sub-conjunto de instruções sintetizáveis da linguagem e o estilo de descrição apropriado.

Portanto, o conjunto básico de células parametrizáveis independentes da tecnologia foram definidos e implementados tendo em atenção à metodologia de síntese, baseada em linguagens de programação de *hardware*, que foi adoptada no projecto. A utilização de parâmetros de genéricos (*generic*) em VHDL e uso de pinos de programação permite definir um conjunto de células/blocos parametrizáveis e programáveis. O mapeamento destes blocos num tecnologia especifica, e a sua adaptação e optimização ao circuito onde estão inseridos, é feita pela ferramenta de síntese. As células da biblioteca IC-Blocks podem ser usadas quer num ambiente de projecto puramente textual, quer no tradicional ambiente de projecto que recorre à edição de esquemáticos para descrever a funcionalidade do circuito. Portanto, cada célula da biblioteca necessita de ter duas vistas: uma descrição textual em VHDL e símbolo gráfico para o editor de esquemáticos.

Como o editor de esquemáticos seleccionado, o Composer da Cadence,

não possibilita a descrição de símbolos com um número variável de pinos, é necessário associar diferentes símbolos, com diferentes número de pinos, para um mesmo bloco da biblioteca. Quando for utilizado um editor de esquemáticos os valores dos pinos de programação são ligados a "1" (VDD) ou "0" (GND). Os valores do genéricos são especificados nas propriedades de cada uma das instâncias do esquemático.

Note-se, que a ferramenta de síntese só pode ler uma descrição com células da biblioteca IC-Blocks em VHDL, se todos os valores do genéricos forem definidos. Assim, foi desenvolvido um conversor de esquemático para VHDL usando a linguagem de extensão do Design Framework II, o *Skill*. O conversor lê a lista de portas e as suas propriedades directamente da base de dados do circuito e gera a correspondente descrição em VHDL, onde estão incluídos os blocos da biblioteca já parametrizados.

Na primeira versão da biblioteca IC-Blocks alguns dos blocos mais comuns e usados em circuitos digitais foram seleccionados. A tabela 1 sumariza os blocos actualmente disponíveis na biblioteca IC-BLOCKS.

Nome do Bloco	**Descrição**
BIN_SEGD	Codificador de binário para *display* de 7-segmentos
COUNTER	Contador crescente/decrescente com *enable* e carreamento paralelo
DATA_BUS	Interface a *Buses*
DEMUX	*Demultiplexer*
MUX	*Multiplexer*
SR	Registo de deslocamento

Tabela 1: Blocos disponíveis na biblioteca IC-Blocks

Nas secções seguintes apresenta-se para cada célula genérica a declaração do componente em VHDL, juntamente com uma descrição sumária das suas potencialidades de parametrização.

2.1 BIN_SEGD

Este bloco descreve um codificador binário para um *display* de 7 segmentos. O genérico NOENABLE define o valor dos pinos de saída quando o

```
entity BIN_SEGD is
    generic (NOENABLE: INTEGER RANGE 0 to 3 := 0);
    port(DATA_IN: in STD_LOGIC_VECTOR(3 downto 0);
        ENABLE: in STD_LOGIC;
        DATA_OUT: out STD_LOGIC_VECTOR(6 downto 0));
end BIN_SEGD;
```

Figura 1: Declaração do componente codificador

descodificador não está activo (ENABLE = 0), de acordo com a tabela 2.

Valor de NOENABLE	**Valor na saída quando ENABLE=0**
0	0
1	1
2	Z
3	X

Tabela 2: Efeito do NOENABLE nas saídas

As saídas pode ser colocadas ao valor lógico "1", "0", de alta impedância ("Z") ou a um valor a determinar pela ferramenta de síntese ("X"). Por exemplo, com o valor de NOENABLE = 2, e se o pino da entrada inibidora estiver activo (ENABLE = 0), a saída apresenta-se toda como um *bus* em alta impedância ("Z").

2.2 COUNTER

```
entity COUNTER is
  generic(AINITIAL, SINITIAL : integer := 0;
          N : integer := 4;
          MIN : integer := 0;
          MAX : integer := 0;
          INC : integer := 1;
          DEC : integer := 1);
  port(AINIT, SINIT, CLK : STD_LOGIC;
       ENABLE, DIRECTION : STD_LOGIC;
       LOAD, PROG : STD_LOGIC;
       LOAD_REG, PROG_REG : STD_LOGIC_VECTOR(N-1 downto 0);
       COUNT : out STD_LOGIC_VECTOR(N-1 downto 0));
end COUNTER;
```

Figura 2: Declaração do componente contador

O bloco COUNTER descreve um contador muito genérico que pode ser

configurado por forma a abranger a enorme gama de contadores geralmente usados em circuitos digitais.

O genérico N define o número de bits do contador, MIN e MAX os valores de contagem mínimos e máximos do contador. O incremento de contagem e definido por INC, quando a contagem é crescente (DIRECTION = 1) e por DEC se a contagem é decrescente (DIRECTION = 0). Os valores de AINITIAL e SINITIAL definem os valores do contador quando de inicialização assíncrona (AINIT = 1) ou síncrona (SINIT = 1), respectivamente. O contador tem ainda um pino para inibir/desinibir a contagem (ENABLE) e um pino de carregamento paralelo (LOAD), que quando activo a "1" coloca o contador com o valor presente na entrada PROG_REG. O valor máximo de contagem pode ser redefinido para o valor de PROG_REG quando o pino PROG apresenta o valor "1".

2.3 DATA_BUS

```
entity DATA_BUS is
  generic(WIDTH: integer := 8);
  port(THE_BUS : inout STD_LOGIC_VECTOR(WIDTH-1 downto 0);
       DATA_IN: in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
       DATA_OUT: out STD_LOGIC_VECTOR(WIDTH-1 downto 0);
       READ, WRITE: in std_logic);
end DATA_BUS;
```

Figura 3: Declaração do componente de interface *buses*

O bloco de DATA_BUS faz a interface entre um bloco genérico e um *buses*. A largura do *bus* é dado pelo valor do genérico WIDTH. Este bloco permite as operações de leitura (READ = 1) e de escrita (WRITE = 1) no mesmo *bus*.

2.4 DEMUX

O bloco de DMUX descreve um *desmultiplexer* de largura variável. Sempre que o ENABLE = 1, o valor da entrada D é transferida para a saída seleccionada por SEL. O valor colocado nas outras saídas é controlado por NOSEL de acordo com a tabela 2. Com o ENABLE = 0 todas as saídas são inibidas. O valor do genérico N define o número de saídas e o WIDTH_SEL

```
entity DEMUX is
  generic(N : integer := 2;
          WIDTH_SEL : integer := 1;
          NOSEL : integer range 0 to 3 := 0);
  port(D : In STD_LOGIC;
       ENABLE : In STD_LOGIC;
       SEL : In STD_LOGIC_VECTOR(WIDTH_SEL-1 downto 0);
       Q : out STD_LOGIC_VECTOR(N-1 downto 0));
end DEMUX;
```

Figura 4: Declaração do componente *desmultiplexer*

a largura da entrada de selecção. A correcta parametrização deste bloco implica que a seguinte restrição seja verificada: $WITH\_SEL = \log_2 N$

2.5 MUX

```
entity MUX is
  generic(N : integer := 2;
          WIDTH_SEL : integer := 1);
  port(D : In STD_LOGIC_VECTOR(N-1 downto 0);
       SEL : In STD_LOGIC_VECTOR(WIDTH_SEL-1 downto 0);
       Q : out STD_LOGIC);
end MUX;
```

Figura 5: Declaração do componente *multiplexer*

O bloco MUX descreve um *multiplexer* de largura variável. Neste bloco o valor da entrada indicada por SEL é transferida para a saída. O valor do genérico N determina o número de entradas e o WIDTH_SEL a largura da entrada de selecção. Naturalmente que as restrição $WITH\_SEL = \log_2 N$ tem também que ser verificada.

2.6 SR

O bloco SR permite implementar diferente tipos de registos de deslocamento. O valor do genérico WIDTH determina o número de *bits* do registo. AINITIAL e SINITIAL definem os valores do registo no caso de uma inicialização assíncrona (AINIT = 1) ou síncrona (SINIT = 1), respectivamente. A direcção do deslocamento do registo é definida por ROTATE e valor do pino FILL definirá ou o bit mais significativo (MSB) do registo, se o sentido de rotação for para a direita (ROTATE = 1), ou o bit menos

```
entity SR is
  generic (WIDTH : INTEGER := 8;
           AINITIAL : INTEGER := 0;
           SINITIAL : INTEGER := 0);
  port(CLK : In STD_LOGIC;
       ROTATE : In STD_LOGIC;
       FILL : In STD_LOGIC;
       LOAD : STD_LOGIC;
       ENABLE : STD_LOGIC;
       AINIT : In STD_LOGIC;
       SINIT : In STD_LOGIC;
       DATA_IN : In STD_LOGIC_VECTOR (width-1 downto 0);
       DATA_OUT : out STD_LOGIC_VECTOR (width-1 downto 0));
end SR;
```

Figura 6: Declaração do componente registo de deslocamento

significativo (LSB) se o sentido de rotação for para a esquerda (ROTATE = 0). Este bloco tem ainda um pino para inibir/desinibir o deslocamento (ENABLE) e a possibilidade de realizar um carregamento paralelo do valor de DATA_IN quando o valor do pino LOAD = 1.

3 Exemplo de Aplicação

Nesta secção apresenta-se um exemplo da aplicação da biblioteca de células parametrizáveis - IC-Blocks. Pretende-se projectar um contador de 8 bits com inicialização assíncrona no valor de contagem 4.

A figura 7 mostra o esquemático do contador usando o símbolo genérico para contadores de 8 bits - COUNTER8. A funcionalidade desejada é especificada ligando os vários pinos de programação a "0" ou "1", e definindo o valor dos genéricos nas propriedades correspondentes à instância do contador..

A descrição VHDL do circuito é gerada a partir do esquemático utilizando o conversor que foi desenvolvido. Na figura 8 apresenta-se o código VHDL que descreve o contador que satisfaz as especificações desejadas. Esta descrição contem não só a informação sobre a conectividade de alguns pinos de programação, como também especifica os valores do parâmetros genéricos.

A descrição VHDL é lida pela ferramenta de síntese que gera uma descrição do circuito ao nível lógico. A figura 9 apresenta o esquemático gerado

Figura 7: Esquemático de um contador de 8 bits

pelo Design Compiler. Pode-se verificar que após a optimização, apenas a portas lógicas necessárias para a funcionalidade do circuito são usadas.

Figura 8: Descrição do circuito em VHDL

Figura 9: Esquemático do circuito sintetizado

A Código VHDL da Biblioteca IC-Blocks

```
--------------------------------------------------------------------------------
-- VERSION : 1.1
-- FILE : icb.vhd
--
-- Generic and Parameterisable VHDL blocks for use with
-- Synopsys Design Compiler
--
-- This file contains the description of the following cells :
--     COUNTER  : Up/Down counter with enable and parallel load
--     DATA_BUS : Bus interface
--     DEMUX    : Demultiplexer
--     MUX      : Multiplexer
--     SR       : Shift Register
--     BIN_SEGD : Binary to 7-segment decoder
--
--
-- CONTACT PERSON : Paulo Flores
-- ADDRESS : INESC
--           Instituto de Engenharia de Sistemas e Computadores
--           Rua Alves Redol, 9
--           1000 Lisboa
--           Portugal
-- TEL : + 351.1.3100000
-- FAX : + 351.1.525843
-- E-MAIL : pff@inesc.pt
--------------------------------------------------------------------------------
library IEEE;

use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;


entity COUNTER is
    generic(AINITIAL, SINITIAL : integer := 0;
            N : integer := 4;
            MIN : integer := 0;
            MAX : integer := 0;
            INC : integer := 1;
            DEC : integer := 1);
    port(AINIT, SINIT, CLK : STD_LOGIC;
         ENABLE, DIRECTION : STD_LOGIC;
         LOAD, PROG : STD_LOGIC;
         LOAD_REG, PROG_REG : STD_LOGIC_VECTOR(N-1 downto 0);
         COUNT : out STD_LOGIC_VECTOR(N-1 downto 0));
end COUNTER;


architecture BEHAVIORAL of COUNTER is

    signal NEXT_COUNT, COUNT_DUMMY: STD_LOGIC_VECTOR(N-1 downto 0);
```

```
    signal MAX_COUNT: integer range 0 to 2**N-1;

begin
    COUNT <= COUNT_DUMMY;

    process (AINIT, CLK)
    begin
        if AINIT = '1' then
            COUNT_DUMMY <= CONV_STD_LOGIC_VECTOR
                (CONV_UNSIGNED(AINITIAL, N),N);
        elsif clk\'{e}vent and clk = '1' then
            if SINIT = '1' then
                COUNT_DUMMY <= CONV_STD_LOGIC_VECTOR
                    (CONV_UNSIGNED(SINITIAL,N),N);
            elsif SINIT = '0' and LOAD = '1' then
                COUNT_DUMMY <= LOAD_REG;
            elsif SINIT = '0' and LOAD = '0' and ENABLE = '1' then
                COUNT_DUMMY <= NEXT_COUNT;
            else
                COUNT_DUMMY <= COUNT_DUMMY;
            end if;
        end if;
    end process;


    process (COUNT_DUMMY, DIRECTION, PROG, PROG_REG)

        variable temp_count, prog_reg_sig: unsigned(N-1 downto 0);

    begin
        for I in 0 to N-1 loop
            prog_reg_sig(I) := prog_reg(I);
            temp_count(I) := COUNT_DUMMY(I);
        end loop;
        if DIRECTION = '1' then             -- increment
            for I in 0 to INC-1 loop
                if temp_count = MAX_COUNT or
                    (temp_count = PROG_REG_SIG and PROG = '1') then
                        temp_count := CONV_UNSIGNED(MIN,N);
                else
                        temp_count := temp_count+1;
                end if;
            end loop;
        else                                -- decrement
            for I in 0 to DEC-1 loop
                if temp_count = MIN then
                    if PROG = '1' then
                        temp_count := PROG_REG_SIG;
                    else
                        temp_count := CONV_UNSIGNED(MAX_COUNT, N);
                    end if;
                else
                    temp_count := temp_count -1;
```

```
                end if;
            end loop;
        end if;
        for I in 0 to N-1 loop
            NEXT_COUNT(I) <= temp_count(I);
        end loop;
    end process;

    MAX_COUNT  <= 2**N-1 when (MAX = 0) else MAX;

end BEHAVIORAL;

--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity DATA_BUS is
    generic(WIDTH: integer := 8);
     port(THE_BUS : inout STD_LOGIC_VECTOR(WIDTH-1 downto 0);
          DATA_IN: in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
          DATA_OUT: out STD_LOGIC_VECTOR(WIDTH-1 downto 0);
          READ, WRITE: in std_logic);
end DATA_BUS;

architecture behavioral of DATA_BUS is
begin

    process(THE_BUS, READ)
    begin
        if READ = '1' then
            DATA_OUT <= THE_BUS;
        else
            DATA_OUT <= (others => 'Z');
        end if;
    end process;

    process(DATA_IN, WRITE)
    begin
        if WRITE = '1' then
            THE_BUS <= DATA_IN;
        else
            THE_BUS <= (others => 'Z');
        end if;
    end process;

end behavioral;

--------------------------------------------------------------------------------
library IEEE;

use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
entity DEMUX is
    generic( N : integer := 2;          -- number of outputs
            WIDTH_SEL : integer := 1;
            NOSEL : integer range 0 to 3 := 0);
    port(  D : In STD_LOGIC;
         ENABLE : In STD_LOGIC;
         SEL : In STD_LOGIC_VECTOR(WIDTH_SEL-1 downto 0);
           Q : out STD_LOGIC_VECTOR(N-1 downto 0));
end DEMUX;

architecture BEHAVIORAL of DEMUX is

    signal SEL_INT: integer range 0 to 2**WIDTH_SEL-1;

begin
    process (SEL)
        variable SEL_SIG: unsigned(WIDTH_SEL-1 downto 0);
    begin
        for I in 0 to WIDTH_SEL-1 loop
            SEL_SIG(I) := SEL(I);
        end loop;
        SEL_INT <= conv_integer(SEL_SIG);
    end process;


    process (D, SEL, ENABLE)
    begin
        case NOSEL is
            when 0 =>
                Q <= (others => '0');
            when 1 =>
                Q <= (others => '1');
            when 2 =>
                Q <= (others => 'Z');
            when 3 =>
                Q <= (others => '-');
        end case;
        if ENABLE = '1' then
            Q(SEL_INT) <= D;
        end if;
    end process;

end BEHAVIORAL;

--------------------------------------------------------------------------------
library IEEE;

use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity MUX is
    generic(        N : integer := 2;       -- number of inputs
            WIDTH_SEL : integer := 1);
```

```
    port(  D : In STD_LOGIC_VECTOR(N-1 downto 0);
         SEL : In STD_LOGIC_VECTOR(WIDTH_SEL-1 downto 0);
           Q : out STD_LOGIC);
end MUX;

architecture BEHAVIORAL of MUX is

    signal SEL_INT: integer range 0 to 2**WIDTH_SEL-1;

begin

    process (SEL)
        variable SEL_SIG: unsigned(WIDTH_SEL-1 downto 0);
    begin
        for I in 0 to WIDTH_SEL-1 loop
            SEL_SIG(I) := SEL(I);
        end loop;
        SEL_INT <= conv_integer(SEL_SIG);
    end process;

   Q <= D(SEL_INT);

end BEHAVIORAL;

--------------------------------------------------------------------------------
library IEEE;

use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity SR is
    generic (WIDTH : INTEGER := 8;
             AINITIAL : INTEGER := 0;
             SINITIAL : INTEGER := 0);

    --
    -- ROTATE = 1 => right rotation;
    -- ROTATE = 0 => left rotation;
    --

    port(CLK : In STD_LOGIC;
         ROTATE : In STD_LOGIC;
         FILL : In STD_LOGIC;
         LOAD : STD_LOGIC;
         ENABLE : STD_LOGIC;
         AINIT : In STD_LOGIC;
         SINIT : In STD_LOGIC;
         DATA_IN : In STD_LOGIC_VECTOR (width-1 downto 0);
         DATA_OUT : out STD_LOGIC_VECTOR (width-1 downto 0));
end SR;

architecture BEHAVIORAL of SR is
```

```
    signal DATA_OUT_DUMMY: STD_LOGIC_VECTOR(width-1 downto 0);

begin
    DATA_OUT <= DATA_OUT_DUMMY;

    process(AINIT\-\c{c}lk)
    begin
        if AINIT = '1' then
            DATA_OUT_DUMMY <= conv_STD_LOGIC_VECTOR
                (CONV_UNSIGNED(AINITIAL, WIDTH),WIDTH);
        elsif clk\'{e}vent and clk = '1' then
            if SINIT = '1' then
                DATA_OUT_DUMMY <= conv_STD_LOGIC_VECTOR
                    (CONV_UNSIGNED(SINITIAL,WIDTH),WIDTH);
            elsif SINIT = '0' and LOAD = '1' then
                DATA_OUT_DUMMY <= DATA_IN;
            elsif SINIT = '0' and LOAD = '0' and ENABLE = '1' then
                case ROTATE is
                    when '1' =>
                        DATA_OUT_DUMMY <=
                            FILL & DATA_OUT_DUMMY(WIDTH-1 downto 1);
                    when others =>
                        DATA_OUT_DUMMY <=
                            DATA_OUT_DUMMY(WIDTH-2 downto 0) & FILL;
                end case;
            else
                DATA_OUT_DUMMY <= DATA_OUT_DUMMY;
            end if;
        end if;
    end process;

end BEHAVIORAL;

--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity BIN_SEGD is
    generic (NOENABLE: INTEGER RANGE 0 to 3 := 0);
    port(DATA_IN: in STD_LOGIC_VECTOR(3 downto 0);
         ENABLE: in STD_LOGIC;
         DATA_OUT: out STD_LOGIC_VECTOR(6 downto 0));
end BIN_SEGD;

architecture behavioral of BIN_SEGD is
begin
    process(DATA_IN, ENABLE)
    begin
        if ENABLE = '1' then
            case DATA_IN is               --abcdefg
                when "0000" => DATA_OUT <= "1111110";
                when "0001" => DATA_OUT <= "1100000";
```

```
                when "0010" => DATA_OUT <= "1011011";
                when "0011" => DATA_OUT <= "1110011";
                when "0100" => DATA_OUT <= "1100101";
                when "0101" => DATA_OUT <= "0110111";
                when "0110" => DATA_OUT <= "0111111";
                when "0111" => DATA_OUT <= "1100010";
                when "1000" => DATA_OUT <= "1111111";
                when "1001" => DATA_OUT <= "1110111";
                when others => DATA_OUT <= "-------";
            end case;
        else
            case noENABLE is
                when 0 =>
                    DATA_OUT <= (others => '0');
                when 1 =>
                    DATA_OUT <= (others => '1');
                when 2 =>
                    DATA_OUT <= (others => 'Z');
                when 3 =>
                    DATA_OUT <= (others => '-');
            end case;
        end if;
    end process;
end behavioral;
```