

Laboratório 3 – Turma A – Tutorial PicoBlaze Quarta-feira, 19 de Outubro de 2016

O objetivo deste experimento é implementar um sistema com o PicoBlaze no kit Basys3 e testá-lo usando as chaves e leds. O projeto deve conter uma instância do PicoBlaze e uma da memória de instruções contendo o código Assembly da aplicação.

O primeiro passo é fazer o *download* dos arquivos compatíveis com a versão do kit a ser utilizado. Esses arquivos podem ser encontrados no site da **Xilinx** [1] e no **Moodle** da disciplina:

- **kcpsm6.vhd**: código-fonte do PicoBlaze
- **kcpsm6.exe**: ferramenta (Assembler) que gera o código VHDL da memória de instruções (ROM)
- **ROM_form.vhd**: arquivo utilizado pelo Assembler para gerar o código VHDL da memória de instruções
- **kcpsm6_design_template.vhd**: template a ser utilizado para instanciar o PicoBlaze
- **all_kcpsm6_syntax.psm**: arquivo contendo a definição e sintaxe de todas as instruções Assembly do KCPSM6.

O próximo passo é criar o arquivo com o código Assembly. Para isso, basta utilizar um editor de notas (por exemplo, o bloco de notas no Windows). A extensão do arquivo deve ser **.psm** (atenção: verificar se a extensão não ficou **.psm.txt**). Neste experimento, será implementado um programa simples que lê o conteúdo de 8 chaves e armazena em **s0**, **s1** e **s2**. Após manipular os bits em **s1** e **s2**, o conteúdo de **s1 + s2** é mostrado nos leds. A Fig. 1 mostra o código Assembly, com os comentários precedidos por **;**. Em caso de dúvidas sobre a sintaxe do Assembly, basta consultar o arquivo **all_kcpsm6_syntax.psm**.

```
; primeiro programa PB6

CONSTANT SWITCHES, 01      ; SWITCHES = 01h
CONSTANT LEDS, 02          ; LEDS = 02h

soma:                       ; label da rotina
    INPUT s0, SWITCHES      ; s0 recebe conteudo das chaves
    LOAD s1, s0              ; s1 = s0
    LOAD s2, s0              ; s2 = s0
    AND s1, 07               ; s1 = s1 AND 0000 0111
    AND s2, 38               ; s2 = s2 AND 0011 1000
    SR0 s2                   ; s2 = '0' & s2[7:1]
    SR0 s2                   ; s2 = '0' & s2[7:1]
    ADD s1, s2               ; s1 = s1 + s2
    OUTPUT s1, LEDS          ; leds recebem conteudo de s1
    JUMP soma                ; salta para rotina soma
```

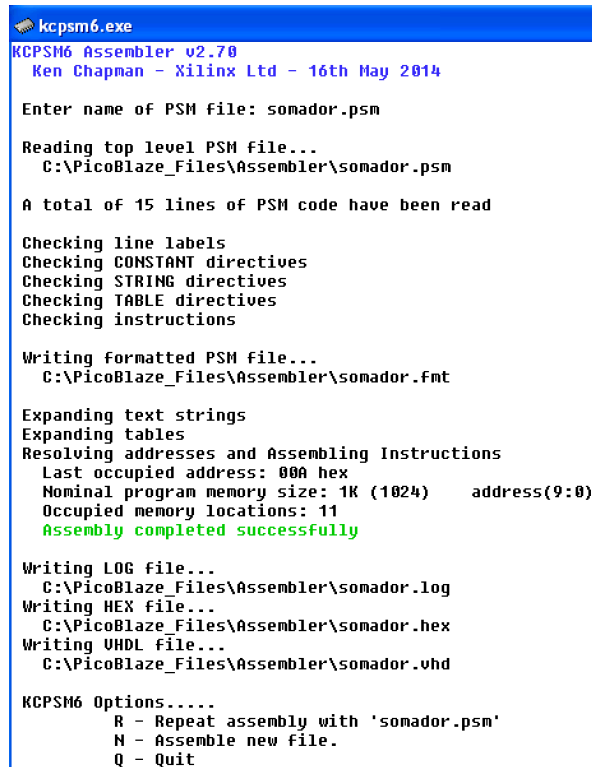
Fig. 1

Após digitar o código no editor de notas, o próximo passo é utilizar o Assembler para gerar o arquivo VHDL da memória de instruções. Para isso, basta copiar o arquivo **.psm** para a pasta Assembler, que contém os arquivos necessários. Em seguida, abrir um *prompt* de comando no diretório dessa pasta e executar o **kcpsm6.exe**, conforme mostra a Fig. 2.

```
C:\PicoBlaze_Files\Assembler>kcpsm6.exe somador.psm
```

Fig. 2

Alternativamente, pode-se clicar duas vezes em **kcpsm6.exe** e digitar o nome do arquivo **.psm** na janela da aplicação, ou arrastá-lo para ela (Fig. 3).



```
kcpsm6.exe
KCPSM6 Assembler v2.70
Ken Chapman - Xilinx Ltd - 16th May 2014

Enter name of PSM file: somador.psm

Reading top level PSM file...
C:\PicoBlaze_Files\Assembler\somador.psm

A total of 15 lines of PSM code have been read

Checking line labels
Checking CONSTANT directives
Checking STRING directives
Checking TABLE directives
Checking instructions

Writing formatted PSM file...
C:\PicoBlaze_Files\Assembler\somador.fmt

Expanding text strings
Expanding tables
Resolving addresses and Assembling Instructions
Last occupied address: 00A hex
Nominal program memory size: 1K (1024) address(9:0)
Occupied memory locations: 11
Assembly completed successfully

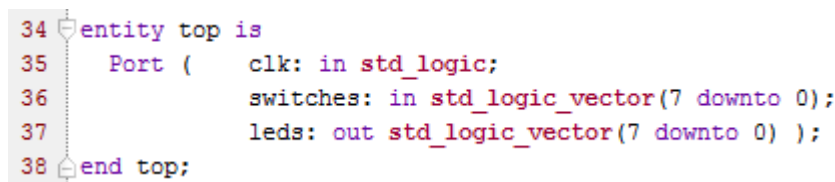
Writing LOG file...
C:\PicoBlaze_Files\Assembler\somador.log
Writing HEX file...
C:\PicoBlaze_Files\Assembler\somador.hex
Writing VHDL file...
C:\PicoBlaze_Files\Assembler\somador.vhd

KCPSM6 Options.....
R - Repeat assembly with 'somador.psm'
N - Assemble new file.
Q - Quit
```

Fig. 3

Caso não haja nenhum erro, o arquivo VHDL da memória de instruções será gerado na pasta com o nome do arquivo **.psm** (no caso da Fig. 3, **somador.vhd**).

A próxima etapa consiste em criar um projeto no Vivado, adicionar e instanciar os códigos-fonte do PicoBlaze (**kcpsm6.vhd**) e da memória de instruções (**somador.vhd**) em um *top level*, com as entradas **clk** e **switches** (8 bits), e com a saída **leds** (8 bits), conforme mostra a Fig. 4.



```
34 entity top is
35   Port (      clk: in std_logic;
36             switches: in std_logic_vector(7 downto 0);
37             leds: out std_logic_vector(7 downto 0) );
38 end top;
```

Fig. 4

Na arquitetura do *top level*, declarar ambos os componentes **kcpsm6** e **somador** (Fig. 5) e criar os sinais necessários para conectar ambos os blocos (Fig. 6). Observe que todo esse conteúdo pode ser copiado do arquivo **kcpsm6_design_template.vhd**.

```

42 component kcpsm6
43     generic(
44         hwbuild : std_logic_vector(7 downto 0) := X"00";
45         interrupt_vector : std_logic_vector(11 downto 0) := X"3FF";
46         scratch_pad_memory_size : integer := 64);
47     port (
48         address : out std_logic_vector(11 downto 0);
49         instruction : in std_logic_vector(17 downto 0);
50         bram_enable : out std_logic;
51         in_port : in std_logic_vector(7 downto 0);
52         out_port : out std_logic_vector(7 downto 0);
53         port_id : out std_logic_vector(7 downto 0);
54         write_strobe : out std_logic;
55         k_write_strobe : out std_logic;
56         read_strobe : out std_logic;
57         interrupt : in std_logic;
58         interrupt_ack : out std_logic;
59         sleep : in std_logic;
60         reset : in std_logic;
61         clk : in std_logic);
62 end component;
63
64 component somador
65     generic(
66         C_FAMILY : string := "S6";
67         C_RAM_SIZE_KWORDS : integer := 1;
68         C_JTAG_LOADER_ENABLE : integer := 0);
69     Port (
70         address : in std_logic_vector(11 downto 0);
71         instruction : out std_logic_vector(17 downto 0);
72         enable : in std_logic;
73         rdl : out std_logic;
74         clk : in std_logic);

```

Fig. 5

```

73 signal address : std_logic_vector(11 downto 0);
74 signal instruction : std_logic_vector(17 downto 0);
75 signal bram_enable : std_logic;
76 signal in_port : std_logic_vector(7 downto 0);
77 signal out_port : std_logic_vector(7 downto 0);
78 signal port_id : std_logic_vector(7 downto 0);
79 signal write_strobe : std_logic;
80 signal k_write_strobe : std_logic;
81 signal read_strobe : std_logic;
82 signal interrupt : std_logic;
83 signal interrupt_ack : std_logic;
84 signal kcpsm6_sleep : std_logic;
85 signal kcpsm6_reset : std_logic;

```

Fig. 6

Após o **begin** da arquitetura, instanciar ambos os componentes e conectá-los usando os sinais

criados (Fig. 7). Esse conteúdo também está disponível no arquivo `kcpsm6_design_template.vhd`.

```
89 processor: kcpsm6
90     generic map (
91         hwbuild => X"00",
92         interrupt_vector => X"3FF",
93         scratch_pad_memory_size => 64)
94     port map(
95         address => address,
96         instruction => instruction,
97         bram_enable => bram_enable,
98         port_id => port_id,
99         write_strobe => write_strobe,
100        k_write_strobe => k_write_strobe,
101        out_port => out_port,
102        read_strobe => read_strobe,
103        in_port => in_port,
104        interrupt => interrupt,
105        interrupt_ack => interrupt_ack,
106        sleep => kcpsm6_sleep,
107        reset => kcpsm6_reset,
108        clk => clk);
109
110 kcpsm6_sleep <= '0';
111 interrupt <= interrupt_ack;
112
113 program_rom: somador
114     generic map(
115         C_FAMILY => "7S",
116         C_RAM_SIZE_KWORDS => 2,
117         C_JTAG_LOADER_ENABLE => 1)
118     port map(
119         address => address,
120         instruction => instruction,
121         enable => bram_enable,
122         rdl => kcpsm6_reset,
123         clk => clk);
```

Fig. 7

Por fim, criar os processos para ler as chaves na porta de entrada (Fig. 8) e escrever o resultado na porta de saída (Fig. 9).

```
121 input_ports: process(clk)
122 begin
123     if clk'event and clk = '1' then
124         case port_id(1 downto 0) is
125             -- Read input_port_c at port address 01 hex
126             when "01" => in_port <= switches;
127             when others => in_port <= "XXXXXXXX";
128         end case;
129     end if;
130 end process input_ports;
```

Fig. 8

```
132 output_ports: process (clk)
133 begin
134     if clk'event and clk = '1' then
135         -- 'write_strobe' is used to qualify all writes to general output ports.
136         if write_strobe = '1' then
137             -- Write to output_port_v at port address 02 hex
138             if port_id(1) = '1' then
139                 leds <= out_port;
140             end if;
141         end if;
142     end if;
143 end process output_ports;
144
145 end Behavioral;
```

Fig. 9

Para testar na placa, basta agora adicionar o arquivo **.xdc**, editar as chaves e os leds que serão utilizados, e em seguida realizar a síntese, implementação e programação da placa.

Visto:

Mostrar o sistema funcionando na placa.

Tarefa:

Responder o questionário no Moodle.

Referências:

[1] http://www.xilinx.com/ipcenter/processor_central/picoblaze/member/

[2] Taylor, Adam P., ***Getting the Most Out of your PicoBlaze Microcontroller***, Xcell Journal, 2014

Bom trabalho!