Módulo 7
Projecto

Objectivos

Neste módulo pretende-se que os alunos projectem, implementem e testem a parte programável do periférico PG04 versão 2 (v2). A descrição e simulação deve ser feita em VHDL. A implementação tem por alvo a CPLD disponível no periférico, tendo sido utilizada no módulo anterior. O programa executável que permite testar o projecto a partir do PC, via porta paralela, será fornecido.

1. Descrição do projecto

A funcionalidade a implementar na CPLD consiste da lógica (combinacional e sequencial) que permitirá aceder, a partir da porta paralela, ao recursos do PG04 v2. Os recursos a que se pretende aceder são: um conjunto de 8 interruptores, um conjunto de 8 LEDs, um visor de 7+1 segmentos, um conversor analógico-digital (ADC) e um conversor digital-analógico (DAC).

O diagrama de blocos do PG04 v2 é apresentado na figura 22 do anexo A e a descrição estrutural da lógica que se pretende implementar na CPLD encontra-se na figura 20. A descrição inclui:

- Um multiplexador para ler os interruptores (`I[7:0]`) ou a ADC (`AD[7:0]`);
- Três registos, implementados com *flip-flops* tipo D, para escrever e registar o valor escrito no LEDs (`O[7:0]`), no visor de 7+1 segmentos (`S[7:0]`) e na DAC (`DD[7:0]`);
- Um bloco de lógica combinacional (***controlBlock***) que gera: (i) o sinal de selecção do multiplexador (`sel`), (ii) o sinal de */OE* dos buffers tri-state que permitem escrever no sinal bi-direccional `D7..D0` da porta paralela (`/OEd`), (iii) os sinais de *enable* dos 3 registos (`CEdd`, `CEs` e `CEo`), (iv) o sinal `/ack` da porta paralela e (v) os sinais de controlo da ADC (`convert` e `/enable`).

A funcionalidade do bloco ***controlBlock*** é apresentada sob a forma das tabelas de verdade 2 e 3. Estas tabelas especificam o comportamento de cada saída do bloco para cada combinação das suas entradas.

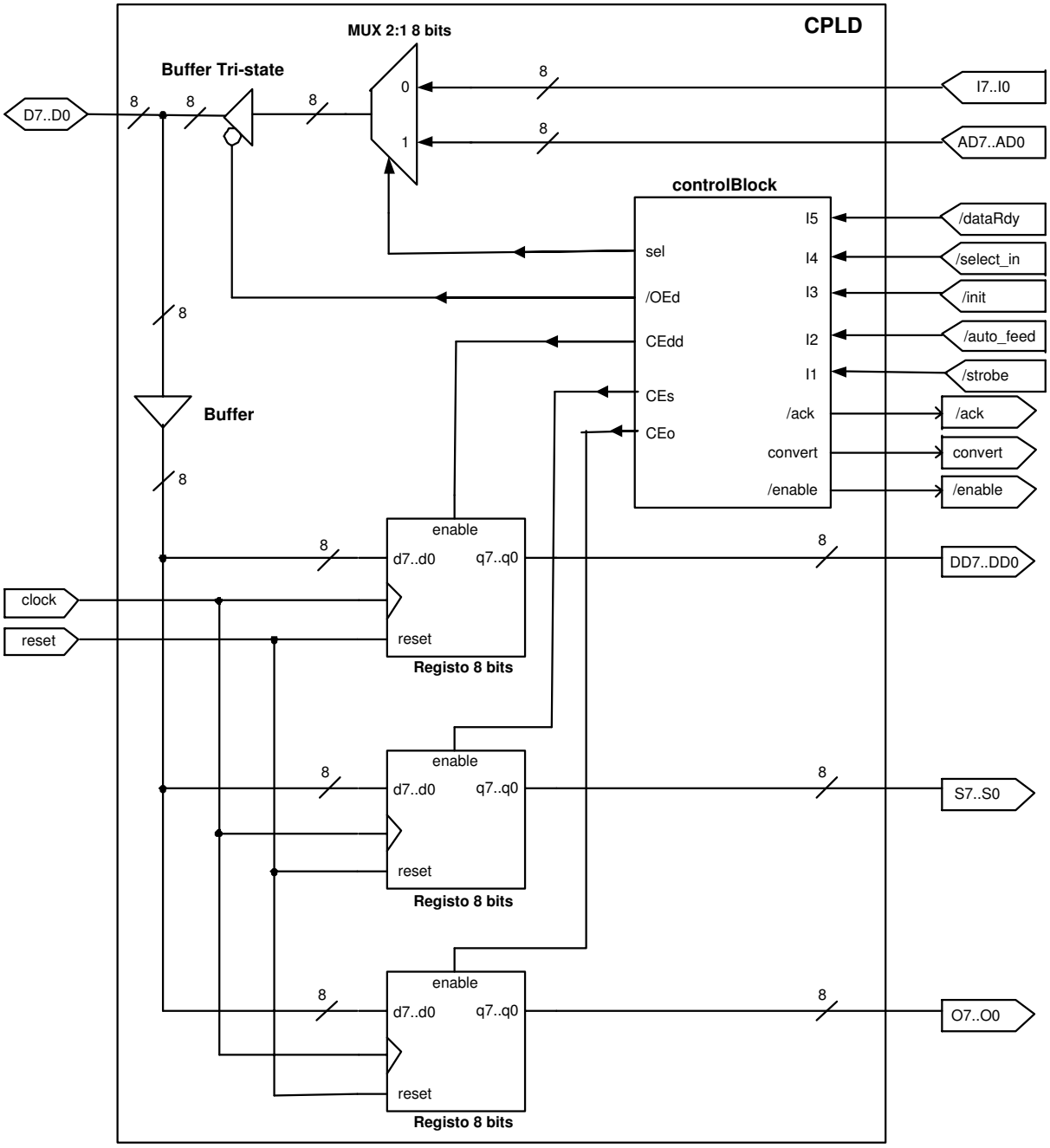

Figura 20: Estrutura da parte programável do PG04 v2 a implementar na CPLD.

Função	***Entradas***					***Saídas***						
	I5	***I4***	***I3***	***I2***	***I1***	***/enable***	***convert***	***/OEd***	***CEdd***	***CEs***	***CEo***	***sel***
Escrever nos LEDs	X	1	1	0	0	1	0	1	0	0	1	0
Ler os interruptores	X	1	0	1	0	1	0	0	0	0	0	0
Iniciar conversão A/D	X	1	0	0	1	1	1	1	0	0	0	0
Ler ADC	X	1	0	0	0	0	0	0	0	0	0	1
Escrever na DAC	X	1	1	1	1	1	0	1	1	0	0	0
Escrever no visor 7 segmentos	X	1	1	1	0	1	0	1	0	1	0	0
Fim de conversão A/D	X	X	X	X	X	1	0	1	0	0	0	0
————	X	Combinações restantes				1	0	1	0	0	0	0

I5 ≡ ***/dataRdy***
I4 ≡ ***/select_in***
I3 ≡ ***/init***
I2 ≡ ***/auto_feed***
I1 ≡ ***/strobe***

Tabela 2: Tabela de verdade das saídas do bloco ***controlBlock.***

Função	***Entradas***					***Saída***
	/dataRdy I5	***/select_in I4***	***/init I3***	***/auto_feed I2***	***/strobe I1***	***/ack***
Escrever LEDs, Ler interruptores, Iniciar conversão A/D, Ler ADC, Escrever DAC, Escrever visor 7 segmentos	1	X	X	X	X	1
Fim de conversão A/D	0	X	X	X	X	0
————	1	X	X	X	X	1

Tabela 3: Tabela de verdade da saída ***/ack*** do bloco ***controlBlock.***

2. Sintetizar e descrever em VHDL o bloco *controlBlock*

Represente a tabela de verdade de cada saída do bloco *controlBlock* (tabelas 2 e 3) num mapa de Karnaugh e obtenha a respectiva expressão minimizada.

Com base nas expressões minimizadas obtidas anteriormente, descrever em VHDL (estilo fluxo de dados) a arquitectura e a entidade do bloco *controlBlock*. Na interface, os nomes dos portos devem coincidir com os nomes das entradas ou saídas do bloco *controlBlock* incluído na figura 20. Substitua apenas a `/` por `not`, como por exemplo: `/ack` passa a `notAck` em VHDL.

3. Descrever em VHDL a parte programável do PG04 v2

Descrever em VHDL a arquitectura e a entidade do circuito em projecto, ou seja, a parte programável do PG04 v2.

Na interface, os nomes dos portos devem coincidir com os nomes das entradas/saídas do bloco *CPLD* na figura 20. Substitua apenas a `/` por `not`.

Sempre que possível, descreva a arquitectura de forma comportamental, o mesmo é dizer, os três registos e o multiplexador devem ser descritos comportamentalmente. O bloco *controlBlock* deve ser instanciado pois foi definido à parte com uma arquitectura própria.

4. Alterar a descrição VHDL para incluir especificidades da CPLD

Embora não seja obrigatório, por vezes é recomendável que ao descrever um sistema a implementar em CPLD (ou FPGA) se tenha em consideração algumas especificidades da arquitectura dessa CPLD. No caso presente, vamos instanciar os componentes correspondentes aos *buffers* que a CPLD possui junto dos pinos.

Para ilustrar este procedimento, considere-se o código VHDL necessário para instanciar esses *buffers*:

```
architecture archA of A is

    -- IBUF : buffer a colocar nos pinos de entrada

    component IBUF
        port(I: in std_logic; O: out std_logic);
    end component;

    -- OBUF : buffer a colocar nos pinos de saída

    component OBUF
        port(I: in std_logic; O: out std_logic);
    end component;

    -- OBUFT : buffer a colocar nos pinos de saída em tri-state

    component OBUFT
        port(I: in std_logic; T: in std_logic; O: out std_logic);
    end component;

    ...

    -- ADC é uma entrada na CPLD, usada internamente com o nome ADC_I
    signal ADC_I    :   std_logic ;

    -- LED é uma saída da CPLD, usada internamente com o nome LED_O
    signal LED_O    :   std_logic ;
```

```
   -- DATA é uma entrada/saída na CPLD, usada internamente com o nome
   -- DATA_I (entrada) e DATA_O (saída)

   signal DATA_I, DATA_O   :   std_logic ;

   ...

begin

   -- Instanciar um buffer OBUF no pino de saída LED

   PIN_LED:  OBUF port map (I => LED_O , O => LED ) ;

   -- Instanciar um buffer IBUF no pino de entrada ADC

   PIN_ADC:  IBUF port map (I => ADC , O => ADC_I ) ;

   -- Instanciar buffers de entrada e de saída em tri-state no
   -- pino bi-direccional DATA

   PIN_D_I: IBUF  port map (I => DATA(0) , O => DATA_I(0) ) ;
   PIN_D_O: OBUFT port map (I => DATA_O(0), T=> dataOE, O => DATA(0) );

   ...

   -- Na descrição da arquitectura usa-se LED_O, ADC_I, DATA_O e DATA_I
   -- (sinais internos) em vez de LED, ADC e DATA (sinais equivalentes
   -- aos pinos)

   ...

end archA ;
```

5. Simular em VHDL a parte programável do PG04 v2

Fazendo uso do ambiente de simulação em VHDL proporcionado pela ferramenta Active-HDL:

- Obter um *testbench* em VHDL para testar o circuito projectado que tenha em consideração as seguintes ajudas:

  (i) Para que o simulador conheça os componentes específicos do tipo de CPLD usado (`IBUF`, `OBUF` e `OBUFT`) é necessário incluir no ficheiro em que eles aparecem a declaração da biblioteca onde estão definidos. A biblioteca da Xilinx onde se define `IBUF`, `OBUF` e `OBUFT` é `UNISIM`. Para ter acesso ao conteúdo da biblioteca `UNISIM` deve adicionar duas linhas de código VHDL:

  ```
  library UNISIM;
  use UNISIM.VCOMPONENTS.all;
  ```

(ii) A simulação deve incluir cenários para cada uma das seguintes tarefas: escrever um valor nos LEDs (0x11 e depois 0x22), escrever um valor na DAC (0x33 e depois 0x44), escrever um valor no visor de 7 segmentos (0x55 e depois 0x66), ler um valor dos interruptores (0x77 e depois 0x88) e ler um valor da ADC (0x99 e depois 0xAA). Para se poder ler dos interruptores e da ADC para o sinal de entrada/saída D[7..0] deve colocar-se D[7..0] a "ZZZZZZZZ"(alta impedância). Ao ilustrar as formas de onda que deverão ser obtidas com a simulação, a figura 21 pode ser considerada no processo de escrita dos cenários de simulação.

- Efectuar a simulação de modo a verificar se o circuito se comporta como é suposto, obtendo formas de onda semelhantes às da figura 21.

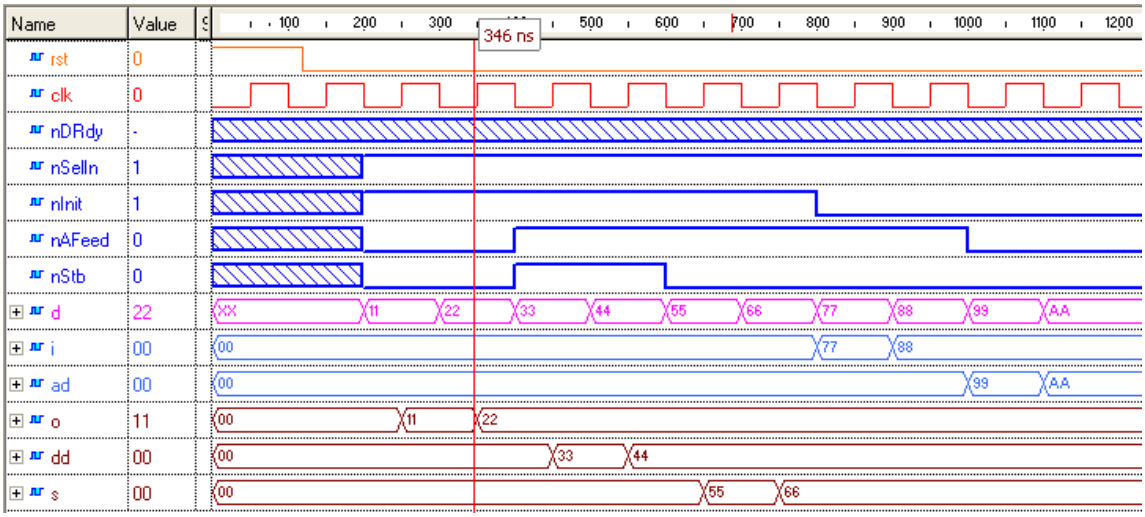

Figura 21: Formas de onda a usar como modelo para os cenários de simulação.

6. Obter a implementação a partir de VHDL

Utilizando a ferramenta de desenvolvimento da Xilinx, deve sintetizar o circuito usando a descrição do circuito obtida nos pontos 2 a 4 e posteriormente implementá-la com a CPLD do PG04 v2. Para sintetizar a descrição do circuito e programar a CPLD, recomendam-se os seguintes passos:

i. No ambiente de desenvolvimento da Xilinx, criar um novo projecto do tipo HDL usando as opções:

- `Device family:` XC9500 CPLDs
- `Device:` XC95108
- `Package:` PC84
- `Speed grade:` -15
- `Top level module type:` HDL
- `Synthesis tool:` XST (VHDL)

ii. Criar o projecto vazio e adicionar *a posteriori* os ficheiros VHDL obtidos nos pontos 2 a 4;

iii. Editar a localização desejada para os sinais de entrada e de saída do circuito quando implementado na CPLD. Para isso utiliza-se o utilitário `PACE`, que pode ser executado através da opção `Assign Package Pins`. Ao executar o `PACE` opte por responder `YES` à primeira pergunta colocada e atribua graficamente a localização dos sinais do controlador aos pinos da CPLD, usando o mapeamento da tabela 4 do anexo A.

A atribuição processa-se arrastando cada sinal da lista `I/O pins` para o pino adequado no esquema do encapsulamento da CPLD, o qual se encontra na janela `Package pins for XC95108-PC84-15`. Guarde o ficheiro no final da atribuição e feche o utilitário `PACE`.

iv. Gerar o ficheiro de configuração da CPLD através da opção `Generate Programming File`, mas para isso deve ter-se seleccionado/marcado a entidade (ficheiro) VHDL que descreve o circuito.

v. Inspeccionar os resultados da síntese, sobre a forma de esquemático, através da opção `View RTL Schematic`. O resultado é o esperado, ou seja, a estrutura gerada pela síntese segue a da figura 20?

vi. Configurar a CPLD do PG04 v2 usando os seguintes passos:

- Colocar os *jumpers* JP6 e JP7 do PG04 v2 no estado ON, para se poder programar a CPLD pela porta e cabo paralelos, e confirmar que o PG04 v2 está ligado ao PC através dum cabo paralelo;
- Executar o utilitário `Xilinx JTAG Programmer`;
- O `JTAG Programmer` deve detectar que existe na cadeia de JTAG, entre os sinais TDI e TDO, uma CPLD XC95108;
- Clicar duas vezes com o rato sobre o símbolo da CPLD presente na cadeia JTAG detectada e especificar o ficheiro de configuração JEDEC a enviar para a CPLD. Escolha o ficheiro JEDEC gerado no ponto (iv);
- Clicar uma vez com o rato sobre o símbolo da CPLD presente na cadeia JTAG, de modo a que este símbolo fique seleccionado, e em seguida seleccionar o comando `Program` do menu `Operations`. Com a opção `Erase before programming` seleccionada, mandar programar a CPLD. Se não ocorrerem erros, a CPLD está apta a funcionar, implementando o circuito projectado.

Pode finalmente verificar o correcto funcionamento do circuito. Para isso, utilize o programa executável fornecido, o qual permite escrever nos LEDS/Visor/DAC e ler os interruptores/ADC (byte-a-byte), através da porta paralela. Neste caso, como se vai usar a porta paralela com um objectivo que não é programar a CPLD, deve retirar-se os *jumpers* JP6 e JP7 (que ficam no estado OFF).