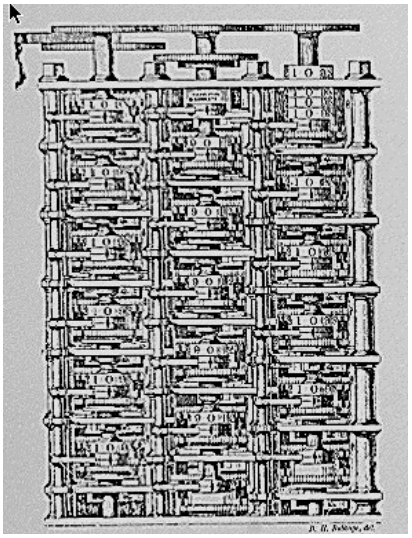
Volume

1

MINI-PROJECTOS V0.1

Disciplina de Integração Hardware-Software: 2005/2006



Universidade do Algarve

João M. P. Cardoso

JOÃO M. P. CARDOSO

Mini-Projectos

Faculty of Sciences and Technology • University of Algarve
Cell Phone +351 916629046 • Email: jmpc@acm.org
URL: http://w3.ualg.pt/~jmcardo
Portugal

Índice

Capítulo

1

Mini-projectos

Conjunto de projectos que envolvem desenvolvimento hardware/software.

Este capítulo apresenta um conjunto de mini-projectos que pretende ilustrar o projecto de sistemas digitais utilizando linguagens de descrição de hardware, ferramentas de síntese lógica, e implementação utilizando FPGAs. Alguns dos projectos contemplam verdadeiras soluções hardware/software. Os exemplos apresentados cobrem um leque variado de implementações, desde cores de microprocessadores ou de microcontroladores até à implementação de arquitecturas específicas para realizar determinadas tarefas.

Nos trabalhos mencionados nas subsecções seguintes pode ser necessário desenvolver ou adaptar pequenos programas de software de forma a, por exemplo, validar os resultados.

tinyMIPS

O tinyMIPS pretende ser um microprocessador com algumas facilidades de configuração. O tinyMIPS é destinado a sistemas embebidos com necessidade de um núcleo de processamento baseado na arquitectura ISA do MIPS (R2000/R3000), com a possibilidade de alguns aspectos do microprocessador poderem ser configurados. A ideia principal é desenvolver um core com possibilidade de ser conectado a hardware específico ou a outros cores de processamento no mesmo FPGA.

A ISA (*Instruction-Set Architecture*) do MIPS R3000 utiliza instruções de 32 bits, representadas pelos três formatos de instruções ilustrados na Tabela I. A Figura 1 apresenta o diagrama de blocos simplificado da arquitectura do MIPS.

Esta versão do tinyMIPS deve ser implementada com *datapath* multiciclo sem *pipelining*. O sistema também não terá níveis de cache, que pode ser uma vantagem em determinados sistemas de tempo-real, pois permite lidar com tempos de execução mais previsíveis (os hit/miss de caches produzem imprevisibilidade em termos de tempo de execução).

As Tabela II, Tabela III, Tabela IV, Tabela V, e Tabela VI apresentam as instruções que o tinyMIPS executa (as linhas das tabelas coloridas indicam instruções que não são consideradas nesta versão).

Tabela I. Tipos de instruções e campos de cada tipo.

Tipo	Código da instrução					
R	6 bits	rs 5 bits	rt 5 bits	rd 5 bits	Shamt 5 bits	Funct 6 bits
I	6 bits	rs 5 bits	rt 5 bits	Endereço imediato (16 bits)		
J	6 bits	Endereço (26 bits)				

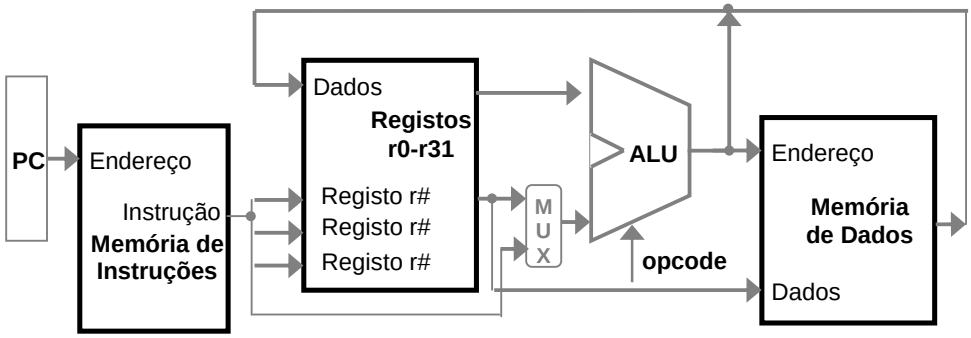


Figura 1. Diagrama de blocos simplificado da arquitectura do MIPS.

Tabela II. Instruções aritméticas

Instrução	**Exemplo**	**Significado**	**Comentário**	**Formato**
add	add $1,$2,$3	$1=$2+$3	3 operandos	R
subtract	sub $1,$2,$3	$1=$2-$3	3 operandos	R
add immediate	addi $1,$2,10	$1=$2+10	add constant	I
add unsigned	addu $1,$2,$3	$1=$2+$3	3 operandos	R
subtract unsigned	subu $1,$2,$3	$1=$2-$3	3 operandos	R
add immed. unsigned	addiu $1,$2,10	$1=$2+10	3 operandos	I

Tabela III. Instruções lógicas e de deslocamento.

Instrução	Exemplo	Significado	Comentário	Formato
and	and $1,$2,$3	$1=$2&$3	3 operandos	R
or	or $1,$2,$3	$1=$2|$3	3 operandos	R
and immediate	andi $1,$2,10	$1=$2&10	AND constante	I
or immediate	or $1,$2,10	$1=$2|10	OR constante	I
shift left logical	sll $1,$2,10	$1=$2<<10	Deslocamento à esquerda	R
shift right logical	srl $1,$2,10	$1=$2>>10	Deslocamento à direita	R
shift right arithmetic	sra $1,$2,10	$1=$2>>10, com extensão de sinal	Deslocamento à direita	R

Tabela IV. Instruções de transferência de dados.

Instrução	Exemplo	Significado	Comentário	Formato
Load word	lw $1,10($2)	$1=Memory[$2+10]	Carrega registo com a palavra lida da memória	I
store word	sw $1,10($2)	Memory[$2+10]=$1	Escreve registo na posição de memória	I
load upper immed.	lui $1,10	$1=10x2^16	Carrega constante nos 16 bits mais significativos	I

Tabela V. Saltos condicionais e comparações.

Instrução	Exemplo	Significado	Comentário	Formato
branch on equal	Beq $1,$2,10	if($1==$2)go to PC+4+10	Equal test	I
branch on not equal	Bne $1,$2,10	if($1!=$2)go to PC+4+10	Not equal test	I
set less then	Slt $1,$2,$3	if($2<$3)$1=1;else $1=0	Less than compare	R
Set less then immed.	Slti $1, $2, 100	if($2<100)$1=1;else $1=0	Less than compare with constant	I
set less then unsigned	Sltu $1,$2,$3	if($2<$3)$1=1;else $1=0	Less than unsigned compare	R

Set less then immed. unsigned	Sltiu $1, $2, 100	if($2<100)$1=1;else $1=0	Less than unsigned compare with constant	I

Tabela VI. Saltos incondicionais.

Instrução	Exemplo	Significado	Comentário	Formato
jump	j 1000	go to 1000	Jump to target address	J
jump register	jr $31	go to $31	For switch, procedure return	J
jump and link	jal 1000	$31=PC+4;go to 1000	For procedure call	R

A Figura 2 ilustra um possível diagrama de estados considerando um datapath multiciclo para o cMIPS.

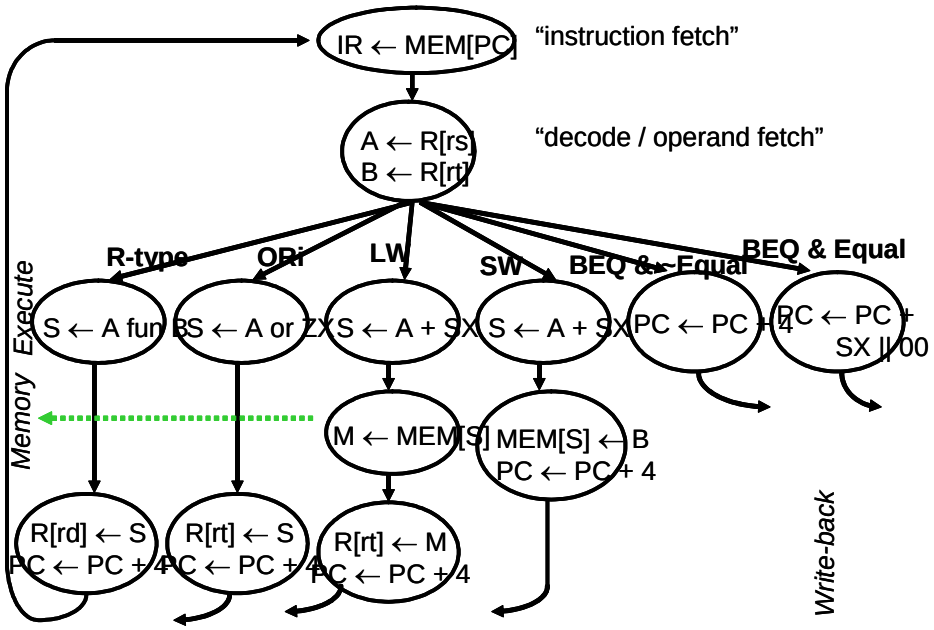


Figura 2. Diagrama de estados para o cMIPS datapath multiciclo.

Para além das instruções definidas anteriormente, o tinyMIPS deve ter as seguintes facilidades de configuração:

- Número de bits do datapath e dos registos do banco de registos;

- Tamanho da memória de programa;

- Tamanho da memória de dados e número de bits de cada posição de memória;

- Possibilidade de ter dois registos ($2 e $3) do banco de registos conectados a portos de entrada e dois registos ($4 e $5) conectados a portos de saída. A escrita/leitura destes registos deve ter funcionalidade similar aos registos de portos do PIC, com a diferença que neste caso não há portos de entrada/saída e por isso não há necessidade de controlo *tri-state*).

Para compilar para o tinyMIPS pode ser utilizado o ambiente gratuito fornecido pela MIPS [8] que requer a instalação do compilador gcc da GNU [9]. Contudo deve referir-se que o código assembly gerado (opção –s do gcc) pode conter instruções MIPS não suportadas pelo tinyMIPS. O assembly gerado necessita de um assemblador que por exemplo forneça a representação binária do programa de forma a ser carregada na memória de programa. O desenvolvimento do assemblador está fora do âmbito deste trabalho.

tinyPIC

O tinyPIC é um microprocessador baseado na arquitectura do PIC16C54 [5]. A versão, cujo diagrama de blocos simplificado é apresentado na Figura 3, não incorpora temporizadores, WDTs, nem modo *sleep*. Nesta versão existem vários parâmetros de configuração:

- Número de bits para representar os dados (afecta o banco de registos, o hardware para as entradas saídas do microcontrolador, a ALU, e o registo W);

- Possibilidade de inclusão do suporte a chamadas de rotinas (instruções RETLW e CALL);

Os formatos das instruções do tinyPIC são apresentados nas Figura 4. As instruções que o tinyPIC deve executar estão ilustradas nas Tabela VII, Tabela VIII, e Tabela IX. Nas tabelas as linhas coloridas a laranja representam instruções que não devem ser consideradas nesta versão do tinyPIC e as linhas coloridas a cinzento representam instruções cuja execução depende da configuração inicial do tinyPIC.

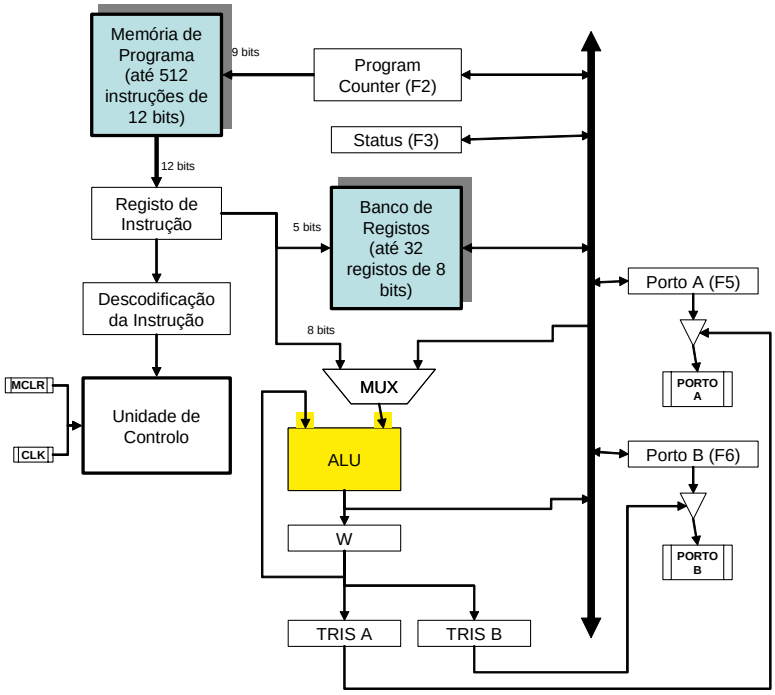


Figura 3. Diagrama de blocos simplificado da arquitectura interna do tinyPIC.

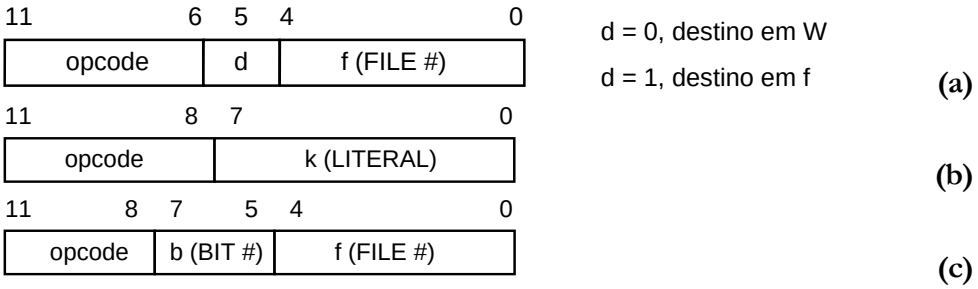


Figura 4. Formato das instruções: (a) de cada instrução sobre o ficheiro de registos orientada ao *byte*; (b) de instruções com imediatos e de instruções de controlo; (c) de cada instrução sobre o ficheiro de registos orientada ao *bit*.

Tabela VII. Instruções sobre o ficheiro de registos orientadas ao *byte*.

opcode (Bin)	Mnemónica		Operação	Registo de ***Status***	Descrição
0001 11df ffff	**ADDWF**	**f,d**	W + f → d|f	C, DC, Z	Soma o conteúdo de W com o registo f (um registo do ficheiro de registos).
0001 01df ffff	**ANDWF**	**f, d**	W & f → d|f	Z	"AND" de W com o registo f.
0000 011f ffff	**CLRF**	**F**	0 → f	Z	Coloca a zero o registo f.
0000 0100 0000	**CLRW**	**-**	0 → W	Z	Coloca a zero o registo W.
0010 01df ffff	**COMF**	**f, d**	~f → d|f	Z	Complementa o registo f.
0000 11df ffff	**DECF**	**f, d**	f - 1 → d|f	Z	Decrementa o registo f.
0010 11df ffff	**DECFSZ**	**f, d**	f - 1 → d|f	-	Decrementa o registo f. Se o resultado for zero salta a próxima instrução.
0001 00df ffff	**IORWF**	**f, d**	W | | f → d|f	Z	"OR" do registo W com o registo f.
0010 10df ffff	**INCF**	**f, d**	f + 1 → d|f	Z	Incrementa o registo f.
0011 11df ffff	**INCFSZ**	**f, d**	f + 1 → d|f	-	Incrementa o registo f. Se o resultado for zero salta a próxima instrução.
0010 00df ffff	**MOVF**	**f, d**	f → d|f	Z	O conteúdo do registo f é movido.
0000 001f ffff	**MOVWF**	**F**	W → f	-	Move o conteúdo de W para o registo f.
0000 0000 0000	**NOP**	**-**	-	-	Nenhuma operação.
0011 01df ffff	**RLF**	**f, d**	f(n) →[d|f](n+1), C → [d|f](0), f(7) → C	C	Rotação de um *bit* para a esquerda do conteúdo do registo f. O *bit* de transporte é envolvido na rotação.
0011 00df ffff	**RRF**	**f, d**	f(n) → [d|f](n-1), C → [d|f](7), f(0) → C	C	Rotação de um *bit* para a direita do conteúdo do registo f. O *bit* de transporte é envolvido na rotação.
0001 10df ffff	**XORWF**	**f, d**	W ⊕ f →d|f	Z	"XOR" de W com o registo f.
0000 10df ffff	**SUBWF**	**f, d**	f - W → d|f	C, DC, Z	Subtrai W ao registo f em complemento para dois.
0011 10df ffff	**SWAPF**	**f, d**	f(0-3)↔ f(4-7) → d|f	-	Troca no registo f os 4 bms com os 4 bMs.

Tabela VIII. Instruções com imediatos e instruções de controlo.

opcode (Bin)	Mnemónica		Operação	Registo de ***Status***	Descrição
1110 kkkk kkkk	**ANDLW**	**k**	k & W → W	Z	"AND" de W com o imediato de 8 *bits*, k. O resultado é colocado no registo W.
1001 kkkk kkkk	**CALL**	**k**	PC + 1 → Pilha (PC + 1 → Topo da pilha), k →PC<7:0>, '0' → PC<8>	-	Chamada a uma sub-rotina. Primeiro, o endereço de retorno (PC+1) é colocado na Pilha. O valor de 8 *bits*, k, é carregado no PC<7:0>. O *bit* 8 do PC é colocado a zero.

0000 0000 0100	**CLRWDT**	**-**	00h → WDT, 0 → WDT	TO, PD	*Reset* do WDT e também *reset* do pré-escalar do WDT se estiver atribuído. Os *bits* de *status*, TO e PD, são colocados a um.
101k kkkk kkkk	**GOTO**	**k**	k → PC<8:0>	-	Coloca no PC o conteúdo de k.
1101 kkkk kkkk	**IORLW**	**k**	k | | W → W	Z	"OR" de W com os 8 *bits* do imediato k. O resultado é colocado no registo W.
1100 kkkk kkkk	**MOVLW**	**k**	k → W	-	Os 8 *bits* do imediato k são colocados no registo W.
0000 0000 0010	**OPTION**	**-**	k → OPTION	-	Os 6 bms do registo W são carregados no registo OPTION.
1000 kkkk kkkk	**RETLW**	**k**	k → W, Pilha → PC (TOS → PC)	-	O registo W é carregado com os 8 *bits* do imediato k. O PC é carregado do topo da Pilha (o endereço de retorno). Esta instrução demora 2 ciclos.
0000 0000 0011	**SLEEP**	**-**	0 → PD, 1 → TO; 00h → WDT, 0 → WDT	TO, PD	O *bit* de *status* PD (*power down*) é colocado a zero. O *bit* de *status* TO (*time-out*) é colocado a um. O WDT e o pré-escalar se lhe estiver atribuído são colocados a zero. Desliga os relógios internos (modo de adormecimento).
0000 0000 0fff	**TRIS**	**f**	W → TRIS do porto f	-	O registo TRIS de f (f = 5, 6, ou 7) é carregado com o conteúdo do registo W.
1111 kkkk kkkk	**XORLW**	**k**	k ⊕ W → W	Z	XOR de W com os 8 *bits* do imediato k. O resultado é colocado no registo W.

Tabela IX. Instruções sobre o ficheiro de registos orientadas ao *bit*.

opcode (Bin)	Mnemónica		Operação	Registo de *Status*	Descrição
0100 bbbf ffff	**BCF**	**f, b**	0 → f(b)	-	O *bit* b do registo f é colocado a zero.
0101 bbbf ffff	**BSF**	**f, b**	1 →f(b)	-	O *bit* b do registo f é colocado a um.
0110 bbbf ffff	**BTFSC**	**f, b**	Testa *bit* b do ficheiro f: Salta se zero.	-	Se o *bit* b do registo f é igual a zero, então salta a próxima instrução.
0111 bbbf ffff	**BTFSS**	**f, b**	Testa *bit* b do ficheiro f: Salta se um.	-	Se o *bit* b do registo f é igual a um, então salta a próxima instrução.

Arquitectura para implementar controladores PID

O controlador PID (Proporcional, Integral, Derivativo) é um componente muito importante em uma grande variedade de sistemas de controlo (veja-se o exemplo de uma implementação do controlador PID pela NASA [7]). Alguns sistemas, na robótica por exemplo, podem ter vários controladores PID que funcionam concorrentemente.

A Figura 5 apresenta um diagrama de blocos de um sistema de controlo típico utilizando um controlador PID.

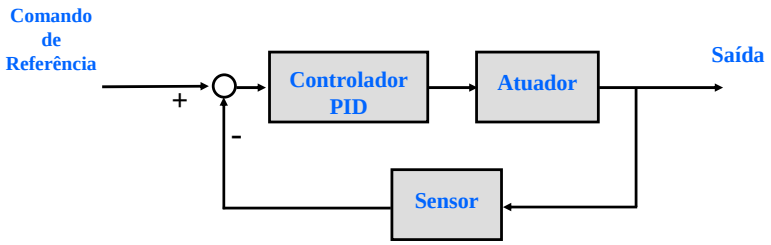


Figura 5. Diagrama de blocos de um sistema de controlo baseado no controlador PID [6].

Neste trabalho pretende-se a implementação em FPGA de uma arquitectura que implemente o PID. A arquitectura deverá permitir a parametrização do número de bits para representar os dados de entrada e de saída e para representar os dados internos. Deverá ser utilizada a virgula fixa para representar os dados.

Uma possível arquitectura para o PID encontra-se representada na Figura 6. O diagrama corresponde a uma implementação da equação do PID discreta:

$$U(k) = U(k-1) + q0 \cdot e(k) + q1 \cdot e(k-1) + q2 \cdot e(k-2)$$

No diagrama, R?s representam registos, e q0, q1, e q2 representam os coeficientes obtidos com os parâmetros Kc, Td, e Ti de sintonia. A arquitectura utiliza 3 somadores, 1 subtrator, 3 multiplicadores, e 3 limitadores que são utilizados para que caso os resultados à saída dos somadores ultrapassem a gama de valores representável utilizando o formato de vírgula fixa especificado, são saturados com o mínimo ou com o máximo valor representável. Os valores dos coeficientes q? podem ser sintonizados automaticamente com a utilização de uma unidade de processamento capaz de implementar um método de sintonia existente (note, contudo, que a implementação de um destes métodos está fora do âmbito deste trabalho).

Os valores para q0, q1, e q2 são obtidos pelo cálculo das equações:

$$q_0 = K_C\left(1+\frac{T_d}{T_S}\right) \qquad q_1 = -K_C\left(1+2\frac{T_d}{T_S}-\frac{T_S}{T_i}\right) \qquad q_2 = K_C\frac{T_d}{T_S}$$

Em que Td, Ti, e Kc representam os parâmetros de entrada do controlador e Ts representa o período de amostragem em segundos.

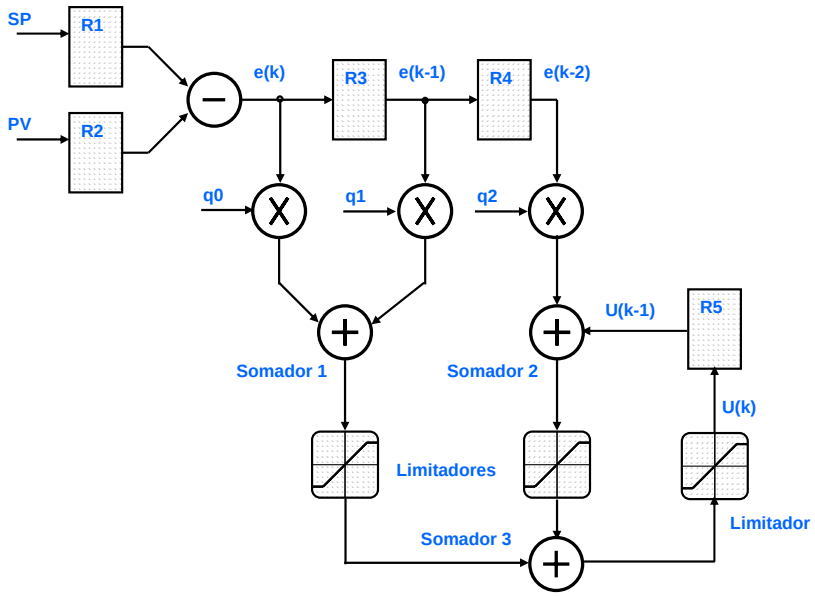


Figura 6. Possível arquitectura para o PID [6].

A saída do sistema é convertida em PWM (*Pulse-Width Modulation*). Na Figura 8 é apresentado um diagrama de blocos que indica as saídas do sistema após conversão para PWM. No exemplo são utilizados 16 bits para representar a saída do módulo PID. A Figura 9 ilustra o funcionamento da conversão para PWM para o exemplo anterior utilizando um contador.

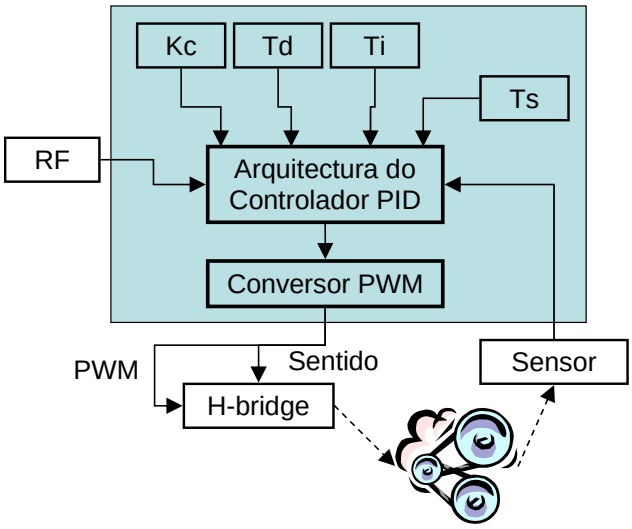


Figura 7. Diagrama de blocos do sistema: a região a sombreado representa os componentes implementados no FPGA.

A arquitectura para o PID e o bloco de conversão para PWM deverão adicionar a possibilidade de configurar:

- Número de bits da amplitude e da parte fraccionária para cada entrada/saída da arquitectura do PID;
- Número de bits de representação para a entrada do conversor PWM;
- A definição do período de amostragem com introdução deste em ms (parâmetro do sistema);
- A definição das constantes Kc, Td, e Ti (parâmetros do sistema).

A implementação final de teste no FPGA deverá possibilitar:

- A entrada do valor referência utilizando os botões da placa disponibilizada;
- A ligação da saída PWM a um interface externo ao FPGA que conecta este a um motor DC;
- A ligação da entrada a um medidor das rotações do motor.

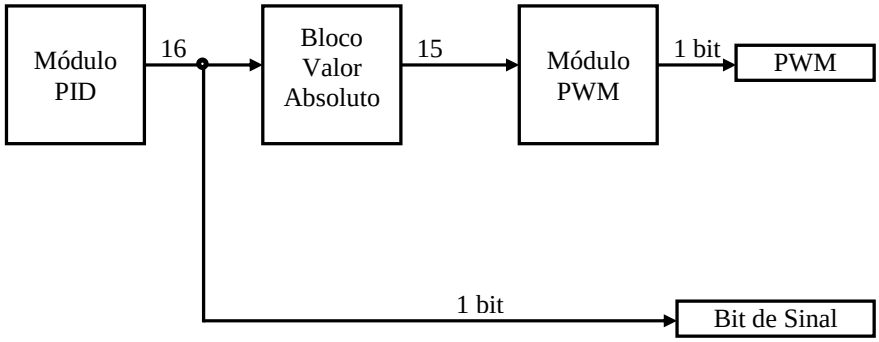


Figura 8. Saída do PID ligada a um conversor para PWM de forma a poder actuar num motor DC, por exemplo [6].

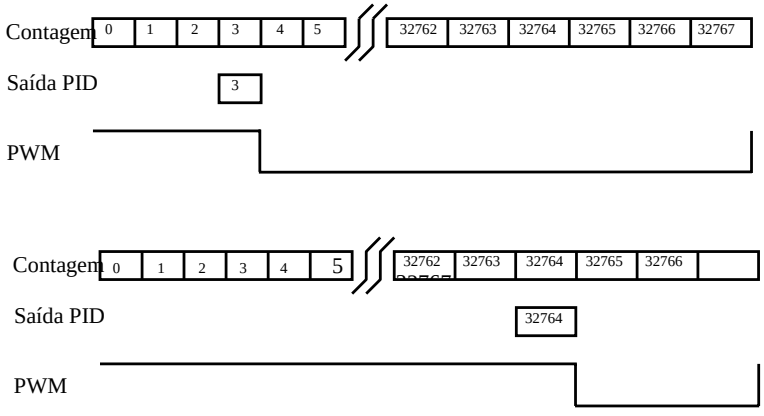


Figura 9. Conversão para PWM utilizando um contador [6].

Processamento de Sinal ou de Imagens

Os trabalhos seguintes representam algoritmos utilizados em sistemas com necessidades de processamento de imagens ou vídeo. Para armazenar os arrays utilizados pelos algoritmos, as implementações devem utilizar "block RAMs" dos FPGAs utilizados. Os tamanhos das imagens (Xsize e Ysize)[1] ou dos arrays utilizados pelos algoritmos devem ser definidos por utilização de parâmetros pré-definidos estaticamente (implementação de módulos com facilidades de parametrização).

Os algoritmos que necessitem de utilização de floats ou doubles devem ser implementados com virgula fixa. Neste caso devem ser parâmetros da implementação o número de bits à esquerda e à direita da virgula.

Filtro Passa-Baixo para Imagens

Em processamento de imagens é muitas das vezes necessário reduzir/eliminar o ruído existente em imagens. O algoritmo apresentado na Figura 10 é um exemplo de um algoritmo para redução de ruído utilizando um filtro passa baixo.

Na implementação deste trabalho a imagem de saída deve ser representada no ecrã utilizando o módulo de interface a um monitor VGA disponibilizado.

A imagem de entrada e a imagem de saída são imagens com 256 níveis de cinzento (8 bits de representação por cada píxel).

A implementação deverá incluir:

- Definição do tamanho da imagem (parâmetros Xsize e Ysize);
- Definição dos valores do array K (9 parâmetros);
- Definição do parâmetro scale.

Uma implementação melhorada é obtida se desenrolarem os dois loops internos do algoritmo apresentado (ver Figura 10). A primeira versão realizada deve implementar o algoritmo inicial e uma segunda versão poderá entrar em linha de conta com os desenrolamentos referidos. No final deve ser avaliada a melhoria de desempenho obtida.

[1] Xsize e Ysize referem o número de píxeis da imagem por cada linha e coluna, respectivamente.

```
// input/outputs
// unsigned short Xsize = 64;
// unsigned short Ysize = 64;
// unsigned byte [] IN, size = Xsize*Ysize;
// unsigned byte[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1}; // 6 bits são suficientes para representar os K
// unsigned byte [] OUT, size = Xsize*Ysize;
// unigned byte scale = 16;

for (int row=0; row < Ysize-3+1; row++) {
  for (int col = 0; col< Xsize-3+1; col++) {
        int sumval = 0;

        for (int wrow=0; wrow < 3; wrow++) {
          for (int wcol = 0; wcol<3; wcol++) {
              sumval += IN[(row +wrow) * Xsize + (col+wcol)]
                        * K[wrow * 3 + wcol];
          }
        }

        sumval = (sumval * scale) >> 8;
        OUT[row *Xsize + col] =  (unsigned byte) sumval;
  }
}
```

Figura 10. Algoritmo do filtro passa-baixo para processamento de imagens.

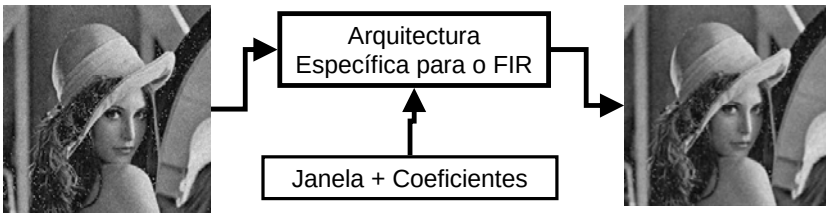


Figura 11. Diagrama de blocos do sistema para o filtro FIR.

Template Matching (correspondência de padrões)

A técnica de *template matching* é muito utilizada no processamento de imagem e de vídeo. A técnica permite calcular a semelhança entre imagens. As imagens podem ter o mesmo tamanho ou serem de diferente tamanho. No último caso, a imagem de entrada, de maior tamanho, é percorrida pela imagem de referência de forma a que em cada posição seja calculado o grau de semelhança. Depois de a imagem de referência ter sido deslocada horizontal e verticalmente por toda a imagem de entrada o algoritmo devolve o valor mínimo da função de semelhança utilizada e a posição na imagem original correspondente a esse valor mínimo.

O sistema a implementar é uma versão simplificada de um sistema de identificação de faces. O diagrama de blocos do sistema é apresentado na Figura 12. A Figura 13 apresenta uma provável implementação do algoritmo na qual é utilizada apenas uma imagem de referência.

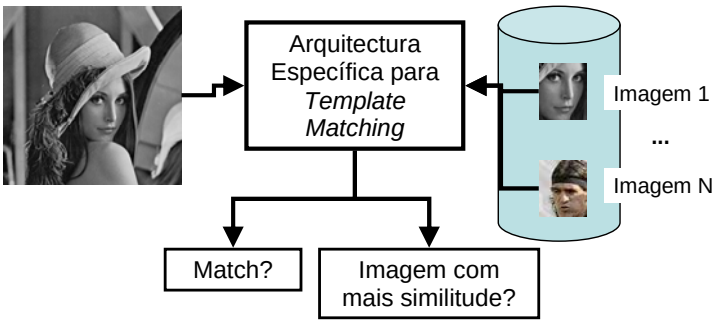


Figura 12. Diagrama de blocos do sistema de *template matching*.

Pretende-se com este trabalho implementar uma versão do algoritmo apresentado tendo em atenção o seguinte:

- O tamanho (Xsize e Ysize) da imagem de entrada deve ser pré-definido como parâmetro;
- Cada píxel da imagem de entrada e das imagens de referencia é representado por 8 bits (256 níveis de cinzento);
- O tamanho (templateXSize e templateYSize) das imagens de referência deve ser pré-definido como parâmetro (todas as imagens de referência devem ser do mesmo tamanho);
- O algoritmo deve calcular concorrentemente a semelhança entre a imagem de entrada e 3 imagens referência;
- Cada imagem considerada deve estar armazenada numa memória interna ao FPGA distinta;
- O algoritmo deve indicar qual foi a imagem referência com grau de semelhança com uma região da imagem de entrada maior e com base num valor de *threshold* (definido como parâmetro) indicar se o grau de semelhança permite concluir que as duas imagens são efectivamente semelhantes;

```
// input/outputs
// unsigned short Xsize = 64;
// unsigned short Ysize = 64;
// unsigned short templateXsize = 16;
// unsigned short templateYsize = 16;
// unsigned byte [] InputImage, size = Xsize*Ysize;
// unsigned byte [] TemplateImage, size = Xsize*Ysize;

unsigned int MinimumSimilitude = Integer.MAX_VALUE;
unsigned short Xpos=0, Ypos=0; // start position of the best matching
unsigned int Threshold = 400000;

for (int row=0; row < Ysize-templateYsize+1; row++) {  // traverse input image
   for (int col = 0; col< Xsize-templateXsize+1; col++) { // traverse input image
       int Similitude = 0;
       for (int wrow=0; wrow < templateYsize; wrow++) {
          for (int wcol = 0; wcol<templateXsize; wcol++) {
              byte grayInputImage = InputImage[col+wcol][ row + wrow];
              byte grayTemplateImage = TemplateImage[ wcol][wrow];
              int aux = (grayInputImage - grayTemplateImage);
              Similitude += aux*aux;
           }
        }
        if(Similitude < MinimumSimilitude) {
          MinimumSimilitude = Similitude;
          Xpos = col;
          Ypos = row;
        }
   }
}
if(MinimumSimilitude <= Threshold)
  System.out.println("match: "+MinimumSimilitude+" x: "+Xpos+" y: "+Ypos);
else
  System.out.println("not match: "+MinimumSimilitude+" x: "+Xpos+" y: "+Ypos);
```

Figura 13. Segmento de código para o *pattern matching* considerando apenas uma imagem na base de dados.

Melhoria de imagens utilizando equalização baseada em histograma

Pretende-se implementar um algoritmo de melhoria de uma imagem representada em 256 níveis de cinzento utilizando equalização baseada num histograma global. O resultado do algoritmo é expresso na Figura 14. A Figura 15 apresenta o código do algoritmo baseado em [10].

A implementação deverá possibilitar que o tamanho (Xsize e Ysize) da imagem de entrada possa ser pré-definido como parâmetro.

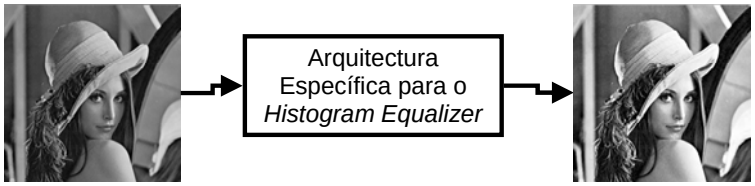


Figura 14. Diagrama de blocos do sistema *histogram equalizer*.

```
// #define        L      256 // number of gray values
// unsigned short Xsize = 64;
// unsigned short Ysize = 64;
// unsigned byte image[Xsize][Ysize]; (input image)
// unsigned byte histogram[L]; (histogram)
// int gray_level_mapping[L]; (gray mapping)
// unsigned byte out_image[Xsize][Ysize]; (output image)

 for (i = 0; i < L; i++)
   histogram[i] = 0;

  // Compute the image's histogram
  for (i = 0; i < N; i++) {
     for (j = 0; j < N; ++j) {
       histogram[image[i][j]] += 1;
     }
  }

  // Compute the mapping from the old to the new gray levels
  Float cdf = 0.0;
  Float pixels = (float) (Xsize*Ysize);
  for (int i = 0; i < L; i++) {
    cdf += ((float)(histogram[i])) / pixels;
    gray_level_mapping[i] = (int) (255.0 * cdf);
  }

// generate the new image
  for (int i = 0; i < Xsize; i++) {
      for (int j = 0; j < Ysize; j++) {
           out_image[i][j] = gray_level_mapping[image[i][j]];
       }
   }
```

Figura 15. Equalizador baseado em histograma.

Bibliografia

1. Dominic Sweetman, *See MIPS Run*, Morgan Kaufmann, San Francisco, CA, 1999, ISBN 1-55860-410-3.

2. John Hennessy, and David Patterson, *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kaufman, 3rd edition, August 2004.

3. Downcast Systems, *MIPSter - MIPS Editor for Windows*, http://www.downcastsystems.com/mipster/

4. SPIM: A MIPS R2000/R3000 Software, http://www.cs.wisc.edu/~larus/spim.html

5. *PIC16C5X Data Sheet EPROM/ROM-Based 8-bit CMOS Microcontroller Series*, © 2003 Microchip Technology Inc., http://www.microchip.com

6. A. L. Giorgini, "Implementação de um controlador PID digital para robótica baseado em computação reconfigurável", Dissertação de Mestrado, ICMC, Universidade de São Paulo, Brasil. 2001.

7. David Gwaltney, Ken King, and Keary Smith, "Implementation of Adaptive Digital Controllers on Programmable Logic Devices," in *5th Annual Military and Aerospace Programmable Logic Devices (MAPLD) International Conference*, 10-12 Sep. 2002, Laurel, MD, USA. Apresentação: http://klabs.org/richcontent/MAPLDCon02/presentations/session_e/e3a_gwaltney_s.ppt

8. MIPS Technologies Inc., MIPS® SDE Lite, http://www.mips.com/content/Products/SoftwareTools/SDE_Lite/content_html

9. http://www.cygwin.com/

10. P. M. Embree, and B. Kimble, *C Language Algorithms for Digital Signal Processing*, Prentice-Hall, ISBN: 0-13-133406-9, 1991.