USO DE FPGAs PARA O ENSINO DE ARQUITETURA DE MICROCONTROLADORES

Paulo Alexandre Martin – pauloalexandre@maua.br
Escola de Engenharia Mauá, Engenharia Elétrica
Praça Mauá, 01
09580-900 – São Caetano do Sul – SP

Resumo: *Este artigo apresenta o uso de FPGAs para o ensino de arquitetura de microcontroladores. Primeiro é apresentada uma introdução de FPGAs com um breve descritivo de suas estruturas internas. Depois é apresentado o projeto e a implementação de uma arquitetura semelhante a um microcontrolador (a arquitetura de um microprocessador com toda a interface de IO e periféricos) em uma FPGA. Todos os blocos para a implementação desta arquitetura, desde simples registradores até uma complexa unidade de controle contendo os micro-códigos são implementados utilizando as linguagens de descrição de hardware VHDL e AHDL. Metodologias para o projeto de sistemas digitais e o ensino de arquitetura de microcontroladores também são apresentadas.*

Palavras-chave: *FPGAs, Microcontroladores, Ensino de eletrônica digital.*

1 INTRODUÇÃO

Uma FPGA (*field-programmable gate array*) é um dispositivo lógico programável que consiste basicamente em diversas PLDs (*programmable logic devices*) integradas em um único chip (FLEX10K Datasheet). Como a FPGA possui diversas PLDs, podemos implementar circuitos digitais de grandes proporções desde que seja respeitada a capacidade da FPGA. A FPGA utilizada neste trabalho é a EPF10K70RC240-2 da família FLEX10K fabricada pela empresa ALTERA (*www.altera.com*) que produz circuitos integrados para lógica programável. Esta FPGA da família FLEX10K possui blocos específicos para a implementação de memórias ROM e memórias RAM (Embedded Array Block). Possui também blocos lógicos para a implementação de circuitos lógicos de propósito geral (Logic Array).

Os elementos lógicos que constituem uma parte da FPGA são compostos por flip-flops e hardwares dedicados que permitem a implementação de qualquer lógica booleana de 4 variáveis. Desta forma os elementos lógicos permitem a criação de máquinas de estados, registradores, contadores e outros circuitos digitais, pois possuem os elementos básicos para isto que são os flip-flops e os circuitos combinatórios. As memórias RAM ou ROM podem ser criadas através de seus blocos específicos e ser interligadas com os elementos lógicos

através das linhas e colunas de conexão. A Figura 1 apresenta a estrutura interna de uma FPGA da família FLEX10K.

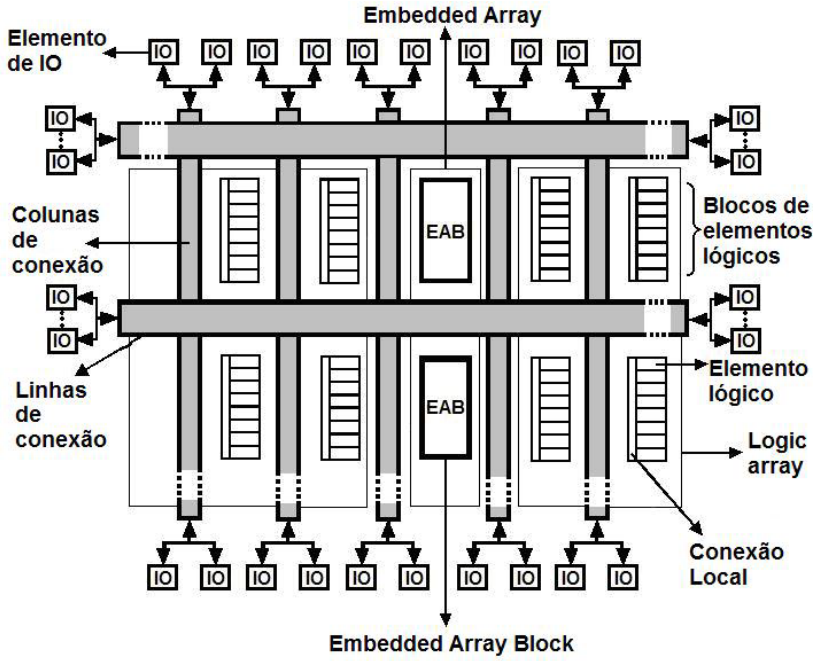

Figura 1 - Diagrama de blocos de uma FPGA da família FLEX10K.

A utilização de FPGAs na indústria e no ensino tem se tornado muito atrativa nos últimos anos, pois atualmente dispõem-se de FPGAs com alto grau de integração de componentes e capazes de operarem com freqüências da ordem de centenas de MHz. Devido a estas características, pode-se implementar sistemas digitais complexos operando com freqüências elevadas nos dando um alto poder de processamento. As FPGAs constituem uma ótima ferramenta de ensino de eletrônica digital pois através delas é possível implementar circuitos digitais complexos sendo que tal implementação seria muito difícil através de componentes discretos. Informações mais detalhadas a respeito das FPGAs da família FLEX10K podem ser obtidas em FLEX10K Data Sheet disponível em www.altera.com.

2 ARQUITETURA DO MICROCONTROLADOR

A arquitetura implementada na FPGA é uma arquitetura do tipo Von Neumann. Esta arquitetura utiliza um mesmo barramento para o tráfego das instruções e para o tráfego de dados, portanto o microcontrolador em questão não pode executar a instrução atual e carregar a próxima instrução simultaneamente como em uma arquitetura Harvard.

A arquitetura implementada neste trabalho é uma arquitetura de microcontrolador com 8 bits e periféricos como PWM (*Pulse Width Modulation*), Timer (*Temporizador*) e comunicação serial. O periférico PWM permite a geração de um sinal PWM para o controle da potência de sistemas como por exemplo o controle de um motor DC ou o aquecimento de uma resistência. O periférico Timer é utilizado em operações que necessitam de temporização, sendo que isto pode ser feito através da interrupção por estouro do Timer. A comunicação serial neste caso é assíncrona e pode ser utilizada para comunicar o microcontrolador com dispositivos externos como, por exemplo, um computador pessoal. Além destes periféricos a arquitetura possui 2 registradores de IO (*dispositivo de entrada e saída – input/output*) de saída e 2 registradores de IO de entrada totalizando 16 bits de IO de

saída e 16 bits de IO de entrada (cada registrador de IO é de 8 bits). A ULA (*unidade de lógica e aritmética*) desta arquitetura apresenta 8 operações sendo que 4 operações são aritméticas (soma, subtração, incremento e decremento) e as demais operações são lógicas (AND, OR, EXOR e NOT).

Para acessar dispositivos externos esta arquitetura utiliza os registradores de IO PORTA, PORTB, PORTC e PORTD (*PORT é um termo técnico em inglês que significa porta de comunicação de dados*). Os registradores PORTA e PORTB são registradores de IO de entrada e podem ser utilizados, por exemplo, para a leitura de botões. Os registradores PORTC e PORTD são registradores de IO de saída e podem ser utilizados, por exemplo, para o acendimento de LEDs. A Figura 2 apresenta a arquitetura montada neste trabalho.

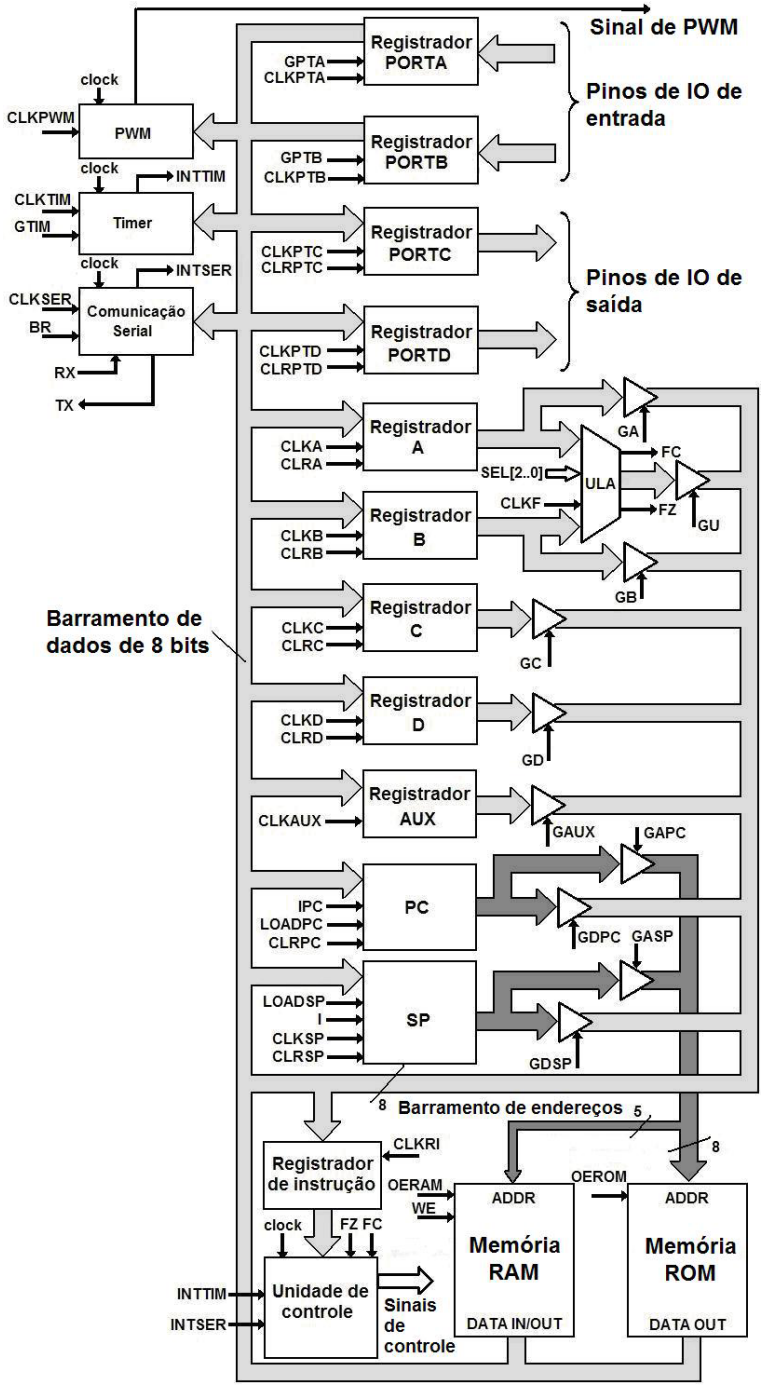

Figura 2 - Arquitetura de microcontrolador implementada.

Os registradores A, B, C e D são registradores utilizados para propósito geral e também para operações lógicas e aritméticas. De acordo com a arquitetura mostrada na Figura 2, somente os registradores A e B tem acesso a ULA, portanto para executar uma instrução que incrementa o registrador C, por exemplo, os micro-códigos (*a seqüência de sinais de controle*) deverão gerar sinais que transfiram o conteúdo do registrador A para o registrador AUX (salvando o conteúdo do registrador A) e depois transfiram o conteúdo do registrador C para o registrador A para a execução da operação de incremento. Após a operação de incremento o valor na saída da ULA é carregado no registrador C e depois o valor em AUX é carregado no registrador A, devolvendo o valor que o registrador A possuía antes da execução da instrução.

O conteúdo de cada registrador pode ser carregado no bus de dados através de um buffer tri-state conforme pode ser visualizado na Figura 2. Estes buffers são controlados pelas entradas G em cada um. O registrador AUX (*auxiliar*) é utilizado para guardar valores que não devem ser perdidos como no exemplo anterior onde ocorria o incremento do registrador C. Nesta arquitetura o usuário não tem acesso ao registrador AUX, ou seja, não serão criadas instruções que permitam o acesso direto ao registrador AUX como por exemplo uma instrução MOV AUX,k (*mova a constante k para AUX*).

O registrador PC é o contador de programa, este registrador é o responsável por endereçar a memória ROM que contém o programa a ser executado. O registrador SP é o stack pointer utilizado na instrução de chamada de sub-rotinas, interrupções e nas instruções de leitura e escrita de dados na pilha (instruções PUSH e POP). A memória RAM mostrada na arquitetura é utilizada para guardar o endereço de retorno nas interrupções e nas chamadas de sub-rotinas ou guardar dados da pilha, que é o caso das instruções PUSH e POP. A memória RAM neste caso é de 32 endereços, portanto seu endereçamento é feito com os 5 bits menos significativos de PC ou SP conforme pode ser visualizado na Figura 2. É possível utilizar a memória RAM como um conjunto de registradores de propósito geral semelhante aos registradores A, B, C e D.

O registrador de instrução guarda o código da operação da instrução que esta sendo executada no momento. Através do registrador de instrução, a unidade de controle da arquitetura lê a instrução e executa os micro-códigos para a execução da mesma. A unidade de controle é uma máquina de estados que gera todos os sinais para o controle de toda a arquitetura. Sinais como o clock e o clear dos registradores (CLKA e CLRA por exemplo), SEL[2..0] na ALU (responsável pela seleção da operação lógica ou aritmética) e muitos outros são controlados pela unidade de controle. As seções a seguir mostram como foram implementados alguns blocos desta arquitetura. Alguns blocos foram implementados utilizando a linguagem de descrição de hardware VHDL (*VHSIC Hardware Description Language*) e outros foram implementados utilizando a linguagem de descrição de hardware AHDL (*Altera Hardware Description Language*). Através destas linguagens de descrição de hardware (AMORE, 2005) é possível escrever um código que descreva o funcionamento do hardware (semelhante a um arquivo fonte de um software). O compilador ao compilar o código escrito gera o hardware que será implementado na FPGA.

3 PERIFÉRICOS

3.1 PWM

Os blocos periféricos desta arquitetura (PWM, Timer e comunicação serial), foram implementados utilizando a linguagem de descrição de hardware VHDL (NAVABI, 1998). A Figura 3 mostra o código em VHDL que implementa o bloco gerador do PWM.

```
ENTITY PWM IS
PORT (CLKPWM:  IN BIT;
        CLK:        IN BIT;
        DBUS:       IN INTEGER RANGE 0 TO 255;
        SAIDA:      OUT BIT);
END PWM;
ARCHITECTURE PWM_BEHAVIOR OF PWM IS
BEGIN
PROCESS(CLK,CLKPWM)
VARIABLE CONTADOR: INTEGER RANGE 0 TO 255;
VARIABLE NIVEL:  INTEGER RANGE 0 TO 255;
BEGIN
IF (CLK'EVENT) AND (CLK='1') THEN
   CONTADOR:=CONTADOR+1;
END IF;
IF (CLKPWM'EVENT) AND (CLKPWM='1') THEN
   NIVEL:=DBUS;
END IF;
IF (NIVEL>CONTADOR) THEN
    SAIDA<='1';
ELSE
    SAIDA<='0';
END IF;
END PROCESS;
END PWM_BEHAVIOR;
```

Figura 3 – Código em VHDL que implementa o bloco gerador do PWM.

No desenho da arquitetura mostrado na Figura 2, o bloco do PWM possui duas entradas para clock (CLK e CLKPWM), uma saída para o sinal PWM e uma entrada de 8 bits. A entrada CLK recebe o sinal de clock para incrementar o contador que gera o sinal PWM sendo que esta entrada pode se o próprio clock do microcontrolador. A entrada CLKPWM é responsável pela captura do valor de 8 bits no bus de dados que irá informar o duty cicle do PWM (no caso é um PWM de 8 bits, portanto o valor 255 corresponde ao máximo duty cicle e o valor 0 corresponde ao mínimo duty cicle). Quando um bordo de subida é aplicado em CLKPWM, o valor no bus de dados é carregado na variável interna NÍVEL, o valor da variável NÍVEL é sempre comparado com o valor da variável CONTADOR de modo que quando o valor em NIVEL aumente, o valor do duty cicle também aumente.

3.2 Timer e comunicação serial

Os blocos do Timer e da comunicação serial foram implementados através da linguagem VHDL (PERRY, 1999). O Timer é utilizado para operações com temporização onde é utilizada a interrupção por estouro do Timer (o flag INTTIM mostrado na Figura 2 é o flag de interrupção por estouro do Timer). A comunicação serial envia e recebe bytes no modo full-duplex (*full-duplex é um termo técnico em inglês que significa "envio e recebimento de dados simultaneamente"*) sendo que a mesma pode trabalhar com a interrupção por recepção da comunicação serial (o flag INTSER avisa a unidade de controle quando chegou um byte pela comunicação serial).

4 REGISTRADORES

Os registradores de IO de entrada possuem uma entrada de 8 bits para ler o conteúdo dos pinos de entrada da FPGA e uma saída de 8 bits para carregar o conteúdo lido no bus de

dados. As entradas GPTA(B) e CLKPTA(B) são utilizadas no processo de leitura, sendo que um bordo de subida em CLKPTA(B) irá capturar a informação dos pinos de IO no latch (*latch é um termo técnico em inglês que significa registrador*) interno e GPTA(B) funciona como um controle do tri-state interno do registrador. Quando GPTA(B) esta em nível lógico baixo, as saídas dos registradores de entrada de IO ficam em alta impedância, quando GPTA(B) esta em nível lógico alto estas saídas carregam a informação lida pelos pinos no bus de dados. Portanto em um processo de leitura de IO primeiramente é aplicado um bordo de subida em CLKPTA(B) para capturar o valor dos pinos de IO de entrada no latch interno e após isto GPA(B) fica em nível lógico alto para que o valor lido seja carregado no bus de dados. A Figura 4 mostra o código em VHDL que implementa os registradores de IO de entrada PORTA e PORTB.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY PORTA IS
PORT(PIN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        POUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        GPTA: IN BIT;
        CLKPTA: IN BIT);
END PORTA;
ARCHITECTURE PORTA_BEHAVIOR OF PORTA IS
BEGIN
PROCESS(CLKPTA,GPTA)
VARIABLE PTA: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
IF (CLKPTA'EVENT) AND (CLKPTA='1') THEN
   PTA:=PIN;
END IF;
IF (GPTA='1') THEN
   POUT<=PTA;
ELSE
   POUT<="ZZZZZZZZ";
END IF;
END PROCESS;
END PORTA_BEHAVIOR;
```

Figura 4 – Código em VHDL que implementa os registradores de IO PORTA e PORTB.

Os registradores de IO de saída (PORTC e PORTD) possuem 8 bits de entrada que lêem o valor no bus de dados e carregam este valor no latch interno para que o mesmo seja exibido nos pinos de IO de saída. A entrada CLKPTC(D) captura o valor do bus de dados no latch interno do registrador e a entrada CLRPTC(D) funciona como um clear que zera o valor do registrador. Os registradores de propósito geral (A, B, C e D) e o registrador AUX podem ser implementados utilizando o código do registrador de IO de saída, pois seu funcionamento é semelhante. O registrador PC (*program counter*) possui uma entrada de 8 bits para o carregamento de valores do bus de dados. A entrada IPC é utilizada para o incremento de PC, sendo que quando um bordo de subida é aplicado em IPC o conteúdo de PC é incrementado. A entrada LOADPC é sensível a nível, quando LOADPC estiver em nível lógico alto, a saída do registrador PC irá copiar a entrada de 8 bits que lê o valor no bus de dados. A entrada CLRPC é utilizada para zerar o valor de PC (um comando de clear) sendo que a mesma será muito útil nos micro-códigos que tratam interrupções como veremos adiante. A Figura 5 mostra o código em VHDL que implementa o registrador PC.

```
ENTITY PC IS
PORT(PCIN: IN INTEGER RANGE 0 TO 255;
       PCOUT: OUT INTEGER RANGE 0 TO 255;
        IPC: IN BIT;
        LOADPC: IN BIT;
        CLRPC: IN BIT);
END PC;
ARCHITECTURE PC_BEHAVIOR OF PC IS
BEGIN
PROCESS(IPC,LOADPC,CLRPC)
VARIABLE PCAUX: INTEGER RANGE 0 TO 255;
BEGIN
IF (LOADPC='1') THEN
   PCAUX:=PCIN;
ELSIF (CLRPC='1') THEN
   PCAUX:=0;
ELSIF (IPC'EVENT) AND (IPC='1') THEN
   PCAUX:=PCAUX+1;
END IF;
PCOUT<=PCAUX;
END PROCESS;
END PC_BEHAVIOR;
```

Figura 5 – Código em VHDL que implementa o registrador PC.

O registrador SP (*stack pointer*) possui uma entrada de 8 bits para ler valores do bus de dados e uma saída de 8 bits para o endereçamento de memórias. A entrada CLKSP é utilizada para o incremento ou decremento do stack pointer, sendo que quando a entrada I estiver em nível lógico alto e um bordo de subida for aplicado em CLKSP o stack pointer é incrementado, quando I estiver em nível lógico baixo e for aplicado um bordo de subida em CLKSP, o stack pointer é decrementado. A entrada CLRSP é utilizada para zerar o valor do stack pointer. CLRSP funciona como um clear no registrador SP. A entrada LOADSP é utilizada para carregar valores do bus de dados para o SP similar ao funcionamento de LOADPC em PC.

5 ULA

A ULA (unidade de lógica e aritmética) utilizada possui 8 operações sendo que 4 operações são lógicas e 4 operações são aritméticas. A seleção da operação lógica ou aritmética é feita pela entrada SEL[2..0]. A entrada CLKF é usada para atualizar os flags da ULA, neste caso foram implementados os flags carry (FC) e zero (FZ). Um bordo de subida em CLKF atualiza os flags carry e zero. A saída da ULA não possui registrador, portanto para que o resultado fique na saída da ULA é necessário que os valores em suas entradas não mudem (entradas referentes ao registradores A, B e SEL[2..0]). Somente os flags são registrados, ou seja, se os valores de A e B forem modificados, o valor dos flags fica inalterado até que seja aplicado um bordo de subida em CLKF. A Tabela 1 mostra o conjunto de operações da ULA.

Tabela 1 - Operações da unidade de lógica e aritmética.

SEL[2..0]			Operação
0	0	0	A+B
0	0	1	A-B
0	1	0	A+1
0	1	1	A-1
1	0	0	A AND B
1	0	1	A OR B
1	1	0	A EXOR B
1	1	1	NOT A

Para a implementação da ALU foi utilizada a linguagem VHDL que permite uma fácil síntese de circuitos aritméticos. Nesta parte do projeto os alunos põem em prática os conhecimentos adquiridos relativos a circuitos lógicos e aritméticos (TOCCI & WIDMER, 2003). Os alunos também tem a opção de implementar os circuitos lógicos e aritméticos através do Mapa de Karnaugh (UYEMURA, 2002).

6 MEMÓRIAS

A arquitetura implementada utiliza duas memórias, uma memória ROM que armazena o programa e uma memória RAM que é utilizada como pilha (instruções PUSH, POP, CALL e interrupções) ou como memória de dados de propósito geral. A memória ROM pode se facilmente implementada utilizando a estrutura CASE WHEN da linguagem VHDL, onde é possível digitar o conteúdo de cada endereço da memória. A Figura 6 mostra o código em VHDL que implementa o conteúdo de alguns endereços da memória ROM.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ROM IS
PORT(ADDR: IN INTEGER RANGE 0 TO 255;
DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
OE: IN BIT);
END ROM;
ARCHITECTURE ROM_BEHAVIOR OF ROM IS
BEGIN
PROCESS(ADDR,OE)
BEGIN
IF (OE='0') THEN
      DATA_OUT<="ZZZZZZZZ";
ELSE
CASE ADDR IS
      WHEN 0 => DATA_OUT<="11110000";
      WHEN 1 => DATA_OUT<="11001100";
      WHEN 2 => DATA_OUT<="00000000";
      WHEN 3 => DATA_OUT<="00001100";
      WHEN 4 => DATA_OUT<="11111111";
      WHEN 5 => DATA_OUT<="10101010";
      WHEN 6 => DATA_OUT<="10101110";
      WHEN 7 => DATA_OUT<="10100001";
      WHEN 8 => DATA_OUT<="11011000";
      WHEN 9 => DATA_OUT<="10000000";
      WHEN 10 => DATA_OUT<="11110111";
      WHEN OTHERS => DATA_OUT<="00000000";
      END CASE;
END IF;
END PROCESS;
END ROM_BEHAVIOR;
```

Figura 6 – Código em VHDL que implementa alguns endereços da ROM.

7 UNIDADE DE CONTROLE

A unidade de controle é a parte mais complexa deste projeto. Trata-se de uma máquina de estados Moore que gera todos os sinais para o controle da arquitetura (BARTEE et al., 1962). O diagrama de estados simplificado da unidade de controle é apresentado na Figura 7.

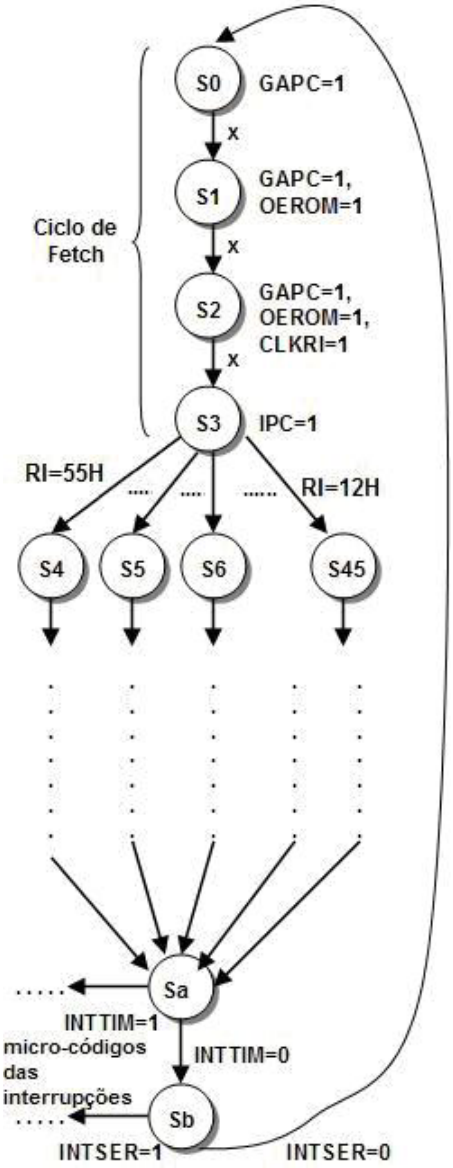

Figura 7 - Diagrama de estados simplificado da unidade de controle.

Para sua implementação, foi utilizada a linguagem AHDL pois na linguagem AHDL podemos criar máquinas de estados simplesmente digitando a tabela de estados. Através da tabela de estados digitada, o compilador gera o hardware que implementa a unidade de controle. A unidade de controle possui 4 estados iniciais que compõem o ciclo de Fetch (*Fetch é um termo técnico em inglês que significa busca por instrução*). O ciclo de Fetch é a busca por instrução, ou seja, a geração dos sinais de controle para que o conteúdo do endereço atual da memória ROM seja carregado no registrador de instrução (CHU, 1962). Após o carregamento no registrador de instrução (RI), a unidade de controle sabe qual instrução deve executar e então gera os sinais para a execução da instrução conforme pode ser visto na Figura 7. Como a unidade de controle possui muitas saídas, foram mostradas somente as saídas que apresentam nível lógico alto no estado corrente. Portanto se somente GAPC esta em nível lógico alto no estado S0, então todas as demais saídas estão em nível lógico baixo.

Na transição de estados somente aparecem as entradas relevantes à transição como, por exemplo, no estado Sa onde somente INTTIM é relevante. Nos casos onde aparece a letra “x”

na transição, indica que a transição irá ocorrer independentemente de qualquer valor nas entradas na unidade de controle.

Toda a unidade de controle é uma máquina de estados Moore, nesta parte do projeto os alunos entendem as diferenças entre uma máquina de estados Mealy e uma máquina de estados Moore (TAUB, 1984). O projeto desta arquitetura foi desenvolvido utilizando o software MAX+PLUX II versão 10.2. Este software apresenta bons exemplos de códigos com a linguagem AHDL.

Nos 4 primeiros estados ocorre a busca por instrução (estados S0, S1, S2 e S3), a unidade de controle coloca GAPC em nível lógico alto para que o registrador PC enderece a memória ROM. Após isto, OEROM (*habilitação da saída da ROM*) fica em nível lógico alto (saída da memória ROM carrega o bus de dados) e depois no estado S2, CLKRI fica em nível lógico alto. Como CLKRI estava em nível lógico baixo nos estados S0 e S1, então no estado S2 ocorre um bordo de subida em CLKRI já que neste estado CLKRI fica em nível lógico alto. Com este bordo de subida o registrador de instrução captura o valor da instrução e no estado S3 o PC é incrementado para que da próxima vez que o ciclo de Fetch seja executado, a unidade de controle faça a leitura da próxima instrução. Desta forma a unidade de controle executa instrução após instrução seqüencialmente. No estado S3 é tomada uma decisão com base no valor do registrador de instrução (RI). Repare que neste estado existem ramificações que seguem para outros estados. Cada ramificação é composta por estados que compõem os micro-códigos de uma instrução, onde a decisão de qual ramificação será seguida depende do valor de RI que contém o código de operação da instrução. O código de operação 55h conforme pode ser visualizado na Figura 7 é o código correspondente a instrução MOV B,A (instrução que move o conteúdo do registrador A para o registrador B). Portanto os estados a partir do estado S4 geram os sinais para que o conteúdo do registrador A seja copiado para o registrador B conforme é mostrado na Figura 8.

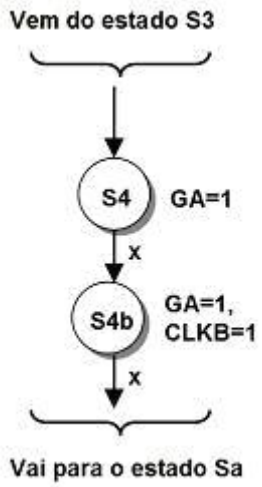

Figura 8 - Micro-códigos para a execução da instrução MOV B,A.

No estado S4, GA fica em nível lógico alto, colocando o valor do registrador A no bus de dados. Depois mantendo GA em nivel lógico alto é aplicado nivel lógico alto em CLKB de modo que ocorra um bordo de subida e o registrador B copie o valor corrente no bus de dados. Ao final de cada instrução, existe a transição para o estado Sa. Neste estado é verificado se ocorreu a interrupção por estouro do Timer. Se esta interrupção ocorreu (INTTIM=1), então ocorre uma transição para os estados que executam os micro-códigos relativos a execução do processo de interrupção do Timer. Se não ocorreu a interrupção por estouro do Timer

(INTTIM=0), então ocorre a transição para o estado Sb onde é testado se ocorreu a interrupção por recepção da comunicação serial. Se esta interrupção por recepção da comunicação serial ocorreu (INTSER=1), então o diagrama de estados desvia para os estados que executam os micro-códigos relativos a execução do processo de interrupção por recepção da comunicação serial. Se não ocorreu esta interrupção, o diagrama desvia para o ciclo de Fetch para executar a próxima instrução. Nesta arquitetura os endereços para a execução das interrupções são os endereços 02h (interrupção por estouro do Timer) e 04h (interrupção por recepção da comunicação serial). A Figura 9 mostra os estados que geram os micro-códigos para a interrupção por estouro do Timer.

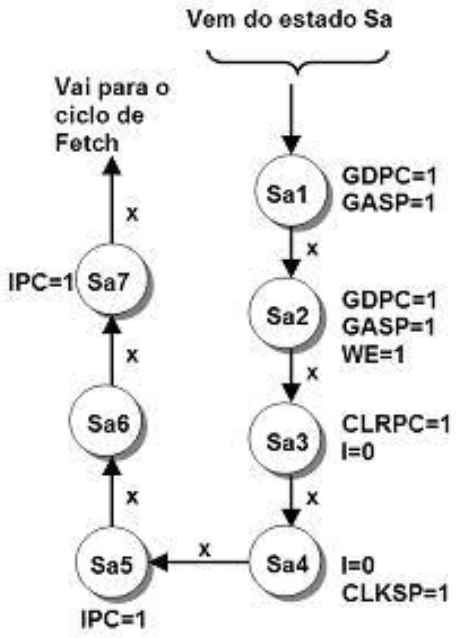

Figura 9 - Micro-códigos para a execução da interrupção por estouro do Timer.

Inicialmente no estado Sa1 é iniciado o procedimento para que o valor de PC seja salvo na pilha. Para isto a memória RAM é endereçada pelo stack pointer (GASP=1) e o valor de PC é carregado no bus de dados (GDPC=1). Mantendo os sinais do estado Sa1 (GASP=1 e GDPC=1), no próximo estado (Sa2), WE fica em nivel lógico alto para que o valor em PC seja salvo na pilha. No estado Sa3 PC é zerado e I fica em nivel lógico baixo. No estado Sa4, mantendo I em nível lógico baixo, aplica-se um nível lógico alto em CLKSP, portanto o valor do stack pointer é decrementado e preparado para um próximo carregamento na pilha. Como PC foi zerado em Sa3, o mesmo contém o valor zero. Veja que em Sa5 e Sa7 são aplicados níveis lógicos altos em IPC, portanto o valor de PC será 02h. Desta forma ocorre o pulo para o endereço onde existe a execução da rotina de interrupção por estouro do Timer. Nesta parte do projeto, os alunos conseguem entender como funciona o processo de interrupção em um microcontrolador (MANO, 1993).

Por se tratar de uma arquitetura onde a memória de programa (memória ROM) é de 256 endereços de 8 bits cada, algumas instruções utilizam 2 bytes como por exemplo a instrução MOV A,k (*instrução que move a constante k para o registrador A*). O primeiro byte desta instrução contém o código da operação e o segundo byte da instrução contém a constante de 8 bits a ser carregada no registrador A. A Figura 10 mostra o diagrama de estados que executa os micro-códigos desta instrução.

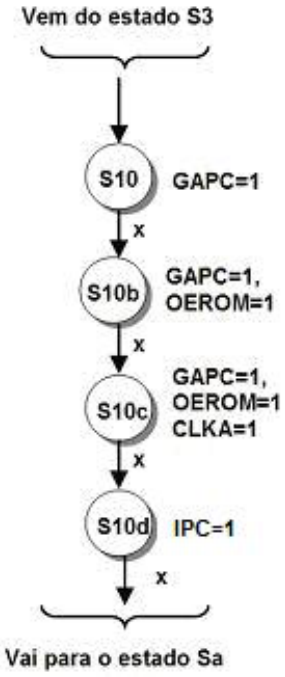

Figura 10 - Micro-códigos para a execução da instrução MOV A,k.

No estado S10 o valor de PC (já apontando para a constante, pois o registrador PC foi incrementado no estado S3) endereça a memória ROM (GAPC=1). Mantendo o endereçamento de PC (GAPC=1) e colocando OEROM em nível lógico alto, o valor da constante é carregado no bus de dados. No estado S10c é dado um bordo de subida em CLKA e a constante é carregada no registrador A. Repare que o registrador PC é incrementado no estado S10d para que a constante (que esta sendo apontada por PC até o estado S10c) são seja interpretada como instrução no próximo ciclo de Fetch.

Muitas outras instruções foram implementadas nesta arquitetura como, por exemplo, instruções lógicas e aritméticas entre registradores, instruções de movimentação de dados entre os registradores e a memória RAM, instruções de pulo, chamada e retorno de sub-rotinas (instruções JMP, CALL e RETURN). A idéia desta arquitetura foi a implementação de um microcontrolador completo para um melhor entendimento por parte do aluno em relação ao funcionamento de microprocessadores e microcontroladores.

8 METODOLOGIAS PARA O ENSINO

As FPGAs constituem uma poderosa ferramenta de ensino de eletrônica digital. Uma FPGA de alta capacidade integrada a um ótimo ambiente de desenvolvimento permitem que o curso de eletrônica digital vá mais longe. Neste trabalho foi passado para o aluno uma metodologia de projeto de sistemas digitais. Esta metodologia é constituída de vários passos cujo objetivo é construir um sistema digital (como a arquitetura mostrada neste trabalho) que não apresente falhas em seu funcionamento. O diagrama da Figura 11 mostra cada passo desta metodologia.

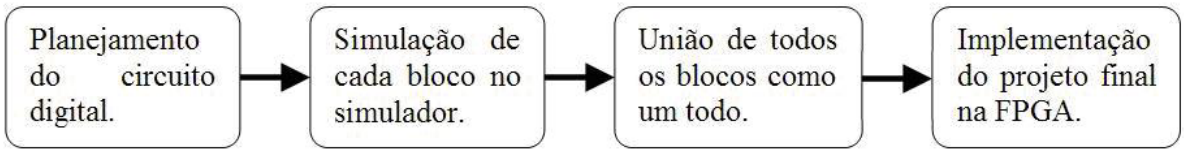

Figura 11- Metodologia de projeto de um sistema digital.

O primeiro passo mostrado na Figura 11 é o projeto e o planejamento do circuito a ser implementado. É neste passo que o aluno coloca em prática todo o conhecimento visto em teoria. O projeto de máquinas de estado, contadores, registradores, circuitos aritméticos e outros são feitos no papel nesta fase inicial do projeto. Também são estudadas partes importantes como por exemplo a minimização de um circuito lógico do sistema de modo que tenha relevância econômica no projeto. Completado o planejamento em papel, o aluno passa para a próxima fase do projeto que é a implementação de cada bloco do sistema digital no ambiente de desenvolvimento da FPGA com o objetivo de simular cada bloco. Quando um projeto digital não funciona corretamente, isto é quase sempre devido a um problema lógico em um de seus blocos constituintes. Desta forma o aluno é ensinado a dividir seu projeto digital em blocos, cada bloco é simulado individualmente para verificar seu correto funcionamento. Quando todos os blocos estiverem funcionando corretamente, os mesmos são unidos para formar o projeto como um todo (penúltima fase), sendo que o projeto como um todo também é simulado. Aplicando esta metodologia de "dividir para conquistar", minimizamos as probabilidades de erros no hardware do nosso projeto. Na última fase o aluno grava seu projeto digital na FPGA e verifica seu funcionamento na prática. O projeto da arquitetura pode ser dividido em duas partes, onde a primeira parte é o projeto e a implementação da arquitetura utilizando a metodologia da Figura 11 e a segunda parte é o estudo de melhorias da arquitetura conforme é mostrado na Figura 12.

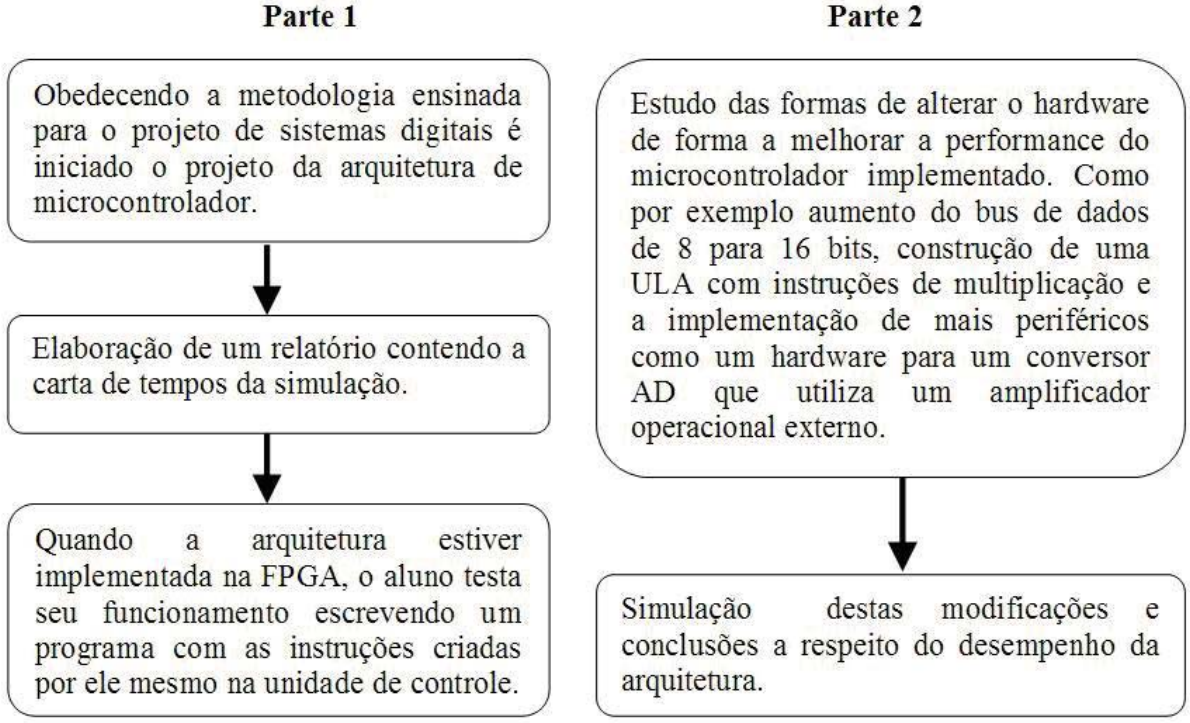

Figura 12 – Atividades didáticas constituintes do projeto da arquitetura.

No final do projeto, os alunos fazem alterações no hardware de modo a melhorar a eficiência da arquitetura como por exemplo aumentar de 8 para 16 bits o bus de dados. Podemos dividir o projeto da arquitetura em duas grandes partes, na primeira parte o aluno projeta e implementa a arquitetura do microcontrolador e na segunda parte (quando a arquitetura já esta pronta) o aluno desenvolve seu raciocínio pesquisando alterações de modo que a arquitetura seja mais eficiente em termos de processamento.

Através destas atividades o aluno aproveita ao máximo os recursos disponíveis e leva para os anos seguintes uma forte base teórica relativa a arquitetura de microcontroladores. Esta base teórica será muito importante nos anos seguintes, pois nestes anos o aluno irá desenvolver programação em Assembly e linguagem C em microcontroladores. Conhecendo a arquitetura básica de um microcontrolador, o aluno consegue compreender melhor o desempenho de um microcontrolador existente no mercado e elabora softwares que extraem ao máximo a capacidade de processamento do microcontrolador.

9 CONSIDERAÇÕES FINAIS

A utilização de FPGAs para o ensino de arquitetura de microcontroladores fornece ao aluno uma ótima visão de como funcionam todos os blocos que compõem um microcontrolador ou um microprocessador. Percebe-se que os alunos que participam deste tipo de atividade conseguem ter um melhor desempenho na programação em Assembly onde é necessário conhecer bem a arquitetura que eles estão programando. Erros comuns na programação em Assembly como, por exemplo, o uso da instrução CALL para a chamada de uma sub-rotina e o retorno ao programa principal utilizando a instrução JMP o que causa estouro no stack pointer são comuns em alunos que iniciam um curso de programação em Assembly. Alunos que realizam a atividade de implementar uma arquitetura como esta em uma FPGA dificilmente escrevem um programa que contém um erro que causa um estouro no stack pointer. A programação em Assembly atualmente ainda é muito usada devido ao fato que a mesma permite a elaboração de programas mais rápidos, menores e com uma melhor performance. As FPGAs tem sido muito utilizadas para a construção de arquiteturas dedicadas em certas funções especificas como, por exemplo, processamento de vídeo em tempo real. Trabalhando com FPGAs, o aluno desenvolve uma ótima capacidade de projeto de arquiteturas dedicadas. Outras grandes contribuições deste trabalho foram a introdução da linguagem de descrição de hardware VHDL muito presente nos dias de hoje e o aprendizado do tráfego de dados em um bus comandado por uma unidade de controle.

No final os alunos implementaram um programa que é formado pelas instruções criadas por eles mesmos (instruções criadas através dos micro-códigos escritos na tabela de estados da unidade de controle). Verificou-se perfeitamente a execução do software e a execução dos micro-códigos escritos na tabela de estados.

Agradecimentos

Especiais agradecimentos à Escola de Engenharia Mauá pelos kits didáticos para FPGAs e softwares de desenvolvimento.

10 REFERÊNCIAS BIBLIOGRÁFICAS

AMORE, R. **VHDL: Descrição e Síntese de Circuitos Digitais**. Rio de Janeiro: LTC, 2005.

BARTEE, T. C.; LEBOW, I. L. e REED, I. S. **Theory and design of digital machines.** New York: Mcgraw-Hill, 1962.

CHU, Y. **Digital computer design fundamentals**. New York: McGraw-Hill, 1962.
FLEX10K Data Sheet. Disponível em: <http://www.altera.com/products/devices/dev-index.jsp>. Acesso em 12 nov. 2006.

MANO, M. M**. Computer System Architecture.** New Jersey: Prentice-Hall, 1993.

NAVABI, Z**. VHDL: Analysis and Modelin of Digital Systems.** New York: McGraw-Hill, 1998.

PERRY, D. L. **VHDL.** New York: McGraw-Hill, 1999.

TAUB, H. **Circuitos Digitais e Microprocessadores**. São Paulo: McGraw-Hill, 1984.

TOCCI, R.J e WIDMER, N.S. **Sistemas Digitais: Princípios e Aplicações**. São Paulo: Prentice Hall, 2003. p. 224-262.

UYEMURA, J. P. **Sistemas Digitais: Uma Abordagem Integrada**. São Paulo: Pioneira Thomson Learning, 2002. p. 51-83.

USE OF FPGAs FOR MICROCONTROLLER ARCHITECTURE TEACHING

Abstract: *This paper presents the use of FPGAs for the microcontrollers architecture teaching. First it is presented a FPGAs introduction with a brief description of their internal structures. After it is presented the project and the implementation of an architecture similar to a microcontroller (a microprocessor architecture with all the IO interface and peripherals) in a FPGA. All the blocks to the implementation of this architecture, from simple registers to a complex control unit containing the micro-codes are implemented using the hardware description languages VHDL and AHDL. Methodologies to the design of digital systems and microcontroller architecture teaching are also presented.*

Key-words: *FPGAs, microcontrollers, digital electronic teaching.*