Síntese e Implementação de Circuitos Digitais Reconfiguráveis Dinamicamente (Projecto 4)

João Paulo Barraca, Nuno João Sénica

***Resumo* – Este artigo apresenta os resultados do projecto proposto aos autores (alunos do 4.º ano da LECT) e explica como o problema específico foi resolvido. A descrição do projecto e os requisitos básicos foram considerados anteriormente nos artigos [1, 2].**

***Abstract* – The paper presents the results of the project proposed to the authors (who are the 4th year students of LECT) and shows how the specified problem has been solved. The description of the project and the basic requirements have been considered in the papers [1,2].**

I. INTRODUÇÃO

Ao longo da disciplina de Computação Reconfigurável [3] leccionada pelo Prof, Valery Sklyarov, explorando as facilidades oferecidas pelos circuitos reconfiguráveis criámos assim um projecto a desenvolver.

O objectivo do projecto seria a obtenção de um circuito, implementado numa FPGA Xilinx, com as capacidades de comunicação com um computador e reconfiguração do algoritmo implementado. O computador enviaria um vector para o circuito, que, dependendo do algoritmo codificado, realizava uma determinada operação sobre o vector e devolvia o resultado. Em qualquer momento, o algoritmo codificado podia ser alterado pelo computador que deveria enviar uma sequência de dados representativos do novo algoritmo pretendido.

No computador, foi criado um pequeno software que serve de interface para comunicação com o hardware desenvolvido. Através desse software, é possível realizar todas as operações suportadas pelo hardware

Embora o software da Xilinx ofereça grandes facilidades para construção de circuitos através de blocos, utilizando um método visual, optamos por privilegiar a utilização da linguagem VHDL.

II. ESPECIFICAÇÃO DO PROJECTO

Quando nos debruçámos sobre o projecto, distinguimos rapidamente quatro áreas de actuação distintas:

1. A criação de uma Unidade de Execução utilizando uma Máquina de Estados Finitos (MEF), que realizasse uma determinada operação sobre um vector booleano.
2. Permitir que a MEF criada podesse ser reprogramada com um outro algoritmo.
3. A criação de uma interface de comunicação entre o computador e o hardware, para além de um protocolo de comunicação que implementasse alguma sincronização.
4. Criar uma Unidade de Controlo que possibilitasse o correcto funcionamento do sistema.

Depois de alguma discussão concordámos que estes seriam os pontos chave do nosso projecto e deveriam ser implementados pela ordem apresentada na figura 1.

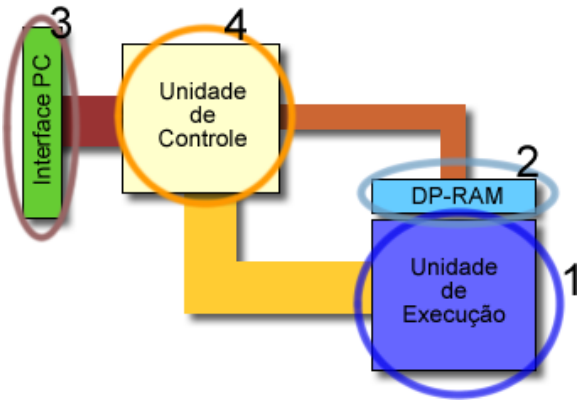

Fig 1: Principais blocos identificados do sistema

Durante o decorrer das aulas, foram apresentados diversos blocos funcionais, relativamente simples, que posteriormente alterámos e utilizámos para o nosso projecto. Grande parte da Unidade de Execução utiliza estes blocos funcionais, com as alterações necessárias, para ser possível a integração com o restante circuito. A Unidade de Controle, no entanto é completamente implementada utilizando a linguagem VHDL.

III. ARQUITECTURA BÁSICA

Segundo a arquitectura que criámos, o nosso sistema pode ser dividido em duas partes distintas: o software no PC e a FPGA. O software no PC comunica com a FPGA através de um interface paralelo seguindo um rígido

protocolo de comunicação. Dentro da FPGA estão implementados diversos blocos funcionais, que por sua vez, são constituídos por blocos mais simples. A arquitectura implementada possui uma estrutura que se encontra apresentada na figura 2.

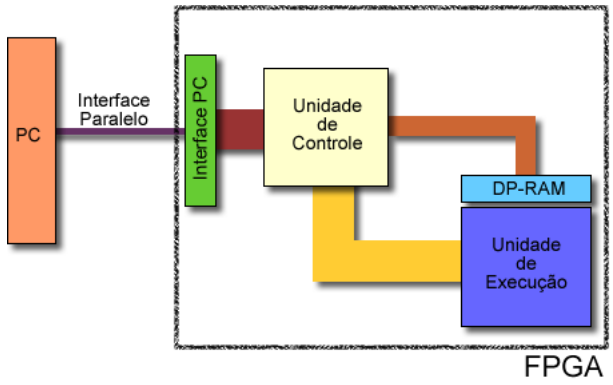

Fig. 2: Arquitectura implementada

Nota-se claramente a divisão entre software e hardware, assim como quais os principais blocos implementados na FPGA.

O Interface PC é um bloco elementar e deve ser entendido mais como o protocolo de comunicação entre FPGA e PC do que um bloco físico complexo. Apenas fornece os dados presentes no barramento paralelo à Unidade de Controle. Existe no entanto um protocolo de comunicação onde se tenta, de uma maneira algo rudimentar, manter uma sincronização constante entre a FPGA e o software.

A Unidade de Execução é uma MEF, modificada de modo a que seja possível reprogramar o seu algoritmo. Para isso foram substituídos os módulos de ROM que a compunham por outros de DualPort-RAM (DP-RAM). Estas DP-RAMs pertencem à Unidade de Execução, mas no entanto, estão ligadas directamente à Unidade de Controlo. É possível reprogramar qualquer byte das RAMs sem grande dificuldade e assim alterar instantaneamente o funcionamento da Unidade de Execução. Esta unidade foi implementada utilizando blocos funcionais como DP-RAMs e Multiplexadores apresentando já alguma complexidade. Com o devido algoritmo programado nas suas DP-RAMs, efectua rapidamente a operação desejada sobre o vector booleano presente na sua entrada.

A Unidade de Controle é o centro da complexidade e das dificuldades encontradas, ao longo deste projecto. Não é extraordinariamente complexa, mas é o bloco funcional que apresentou maior complexidade na construção. Quando decidimos criar um bloco como este, rapidamente chegámos à conclusão que o mais apropriado seria utilizar somente a linguagem VHDL. Assim, construímos uma máquina de estados finitos apenas utilizando a linguagem VHDL e sem sub-blocos constituintes. A escolha foi interessante, mas levou a que aparecessem inúmeros (muitos) problemas relacionados com glitches internos e que escapavam totalmente ao nosso controlo. O software da Xilinx também não ajudou na tarefa. Alguns comandos, que pretendíamos usar, não eram permitidos pelo compilador de VHDL o que nos obrigou a arranjar outros meios, mais de acordo com o suportado pelo compilador. Um problema com que igualmente deparámos foi o de que a linguagem VHDL permite muito mais do que aquilo que é fisicamente possível e nem sempre o compilador identifica isso.

A Unidade de Controle é assim constituída por uma MEF em VHDL com 15 estados distintos (ver figura 3).

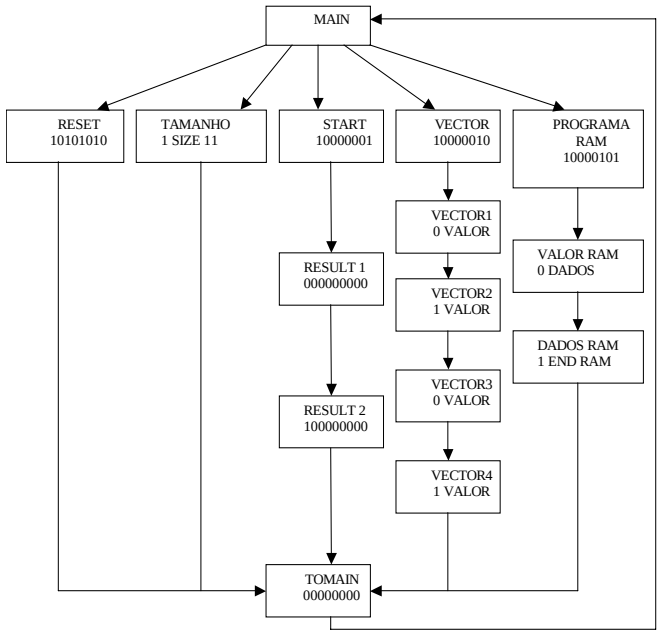

Fig. 3: Estados suportados pela Unidade de Controle.

Como se pode verificar, o interface desta MEF utiliza o paradigma de comandos. Sempre que ela se encontra no estado MAIN, está livre para receber um novo comando e um comando não pode ser interrompido. É uma solução com algumas limitações, especialmente quando em situações de erro mas pareceu-nos ser suficiente para os objectivos iniciais.

Os estados são os seguintes:

MAIN: UC pronta a receber um novo comando.

RESET: É realizado um RESET a todos os barramentos e variáveis internas. Apenas é preservado o conteúdo das DP-RAMs. Activo quando a UC está no estado MAIN e recebe a sequência 10101010.

TAMANHO: Comando que indica o tamanho de bits que a próxima operação vai utilizar. Existem 5 bits "livres" neste comando pelo que o tamanho máximo é 32 bits. Activo quando a UC se encontra no estado MAIN e recebe 1XXXXX11. XXXXX representa o tamanho em BCD e pode ser qualquer valor entre 0 e 32.

START: Quando recebe este comando, a UC activa a unidade de execução para que esta execute o algoritmo programado. Quando o resultado está pronto, a UC notifica o PC do evento. Como a comunicação entre a UC

e o PC apenas utiliza 4 bits, é necessário que o resultado seja enviado em duas partes, para isso o PC necessita de notificar a UC que se encontra disponível para receber os dados e utiliza os estados RESULTADO1 e RESULTADO2.

VECTOR: O PC envia o comando VECTOR para enviar o vector a ser calculado. Com um tamanho máximo de 24 bits, este vector tem de ser enviado em 4 partes, 6 bits de cada vez utilizando os estados VECTOR1,VECTOR2,VECTOR3 e VECTOR4.

PROGRAMA_RAM: Este comando serve para iniciar a programação de um byte do algoritmo que a Unidade de Execução implementa. Depois de activo, o PC deve enviar os novos dados e o endereço. O valor está limitado a 7 bits o que já permite implementar um grande número de operações.

TOMAIN: Este estado é um estado utilizado para reiniciar algumas variáveis internas e preparar a UC para a execução de um novo comando. Força igualmente a que o PC coloque o canal de dados a 00000000 antes de enviar qualquer comando.

IV. SÍNTESE, MODELAÇÃO E IMPLEMENTAÇÃO DO SISTEMA

A. Interface

Sempre que o PC deseja enviar um novo comando ou completar um comando, basta por os dados disponíveis no barramento e inverter o bit mais significativo. A Unidade de Controle "responde" enviando de volta a negação dos quatro bits mais significativos que recebeu (ver figura 4). Por exemplo, quando o software envia o comando RESET 10101010, a FPGA deve responder com 0101 assim que o estado actual atinja o final, neste caso que o RESET tenha sido executado. Ao receber este valor, o PC, deve enviar 00000000 para indicar que o tomou conhecimento que o comando terminou. A partir deste momento a Unidade de Controle fica pronta a receber um novo comando.

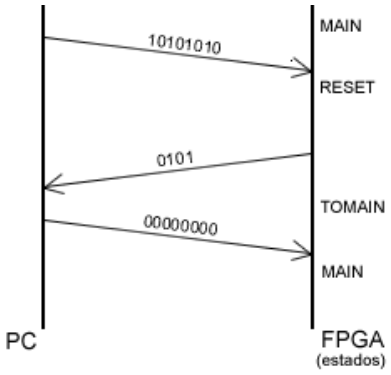

Fig. 4: Exemplo do processo de Reset

B. Unidade de execução

Esta unidade é controlada por um vector de 5 bits (yy) e por um sinal de RESET (1 bit). O vector (V) sobre o qual vão ser feitas as operações programadas, o tamanho do vector (sz) e o sinal de relógio pertencem também à entrada a esta unidade (ver figura 5). No lado da saída temos o vector X que retorna os resultados calculados, e um sinal (READY) que indica que o resultado já foi calculado, resultado este que se encontra no vector res.

```
architecture UE_arch of UE is
begin
  st: process(clock,reset)
  variable Index : integer range 0 to 7;
  variable flag : STD_LOGIC;
  begin
        if reset='1' then
                Index:=0;
                res<=0;
                address_vector<=0;
                ready<='0';
        elsif clock='0' and clock'event then
                if yy0='1' then
                     address_vector<=address_vector+1;
                end if;
                if yy1='1' then ready<='1';
                end if;
                if yy2='1' then res<=res+1;
                end if;
                if yy3='1' then Index:=0;
                end if;
                if yy4='1' then Index:=Index+1;
                end if;

                x0 <= V(Index);
                if Index = 7 then x2 <='1';
                        else x2 <= '0';
                end if;
                if sz = address_vector then x1 <='1';
                        else x1 <= '0';
                end if;
                x3 <= '0';
        end if;

end process st;
end UE_arch;
```

Fig. 5: Código VHDL da Unidade de Execução.

C. Unidade de controlo

O uso da linguagem VHDL permitiu que este bloco possui-se uma maior funcionalidade, em especial no processo de aquisição dos dados do PC, sincronismo e envio de resultados pelo barramento paralelo. Como foi implementada, a nossa unidade de controlo é um bloco encarregado de todo o mecanismo de comunicação e

sincronização, programação das DP-RAMs, e controlo da Unidade de Execução. Ao contrario de outras soluções que apresentavam diversos módulos adicionais, que realizavam sub-tarefas, no controlo do sistema, o nosso sistema apenas possui dois grandes blocos constituintes: Unidade de Controlo e Unidade de Execução. Graças a esta maior concentração de funcionalidade, aliada ao uso da linguagem VHDL, é possível simplificar todo o sistema e aumentar em grande escala o seu potencial. Claro está que estamos a sacrificar a modularidade, no entanto, não considerámos que fosse um factor muito importante para a realização deste projecto.

D. Reconfiguração

As operações efectuadas sobre vector estão descritas por um algoritmo programado na MEFR. Toda a vez que seja necessário a alteração da operação a efectuar é necessário estruturar o algoritmo de acordo com a MEFR e reprogramar esta. Para que seja possível reprogramar a MEFR torna-se necessário que a Unidade de Controlo receba o comando PROGRAMA_RAM. De seguida o PC envia o comando VALOR_RAM que já inclui, tal como todos os outros comandos, o bit de sincronismo, e qual o valor a ser inserido no endereço enviado, finalmente o comando DADOS_RAM indica qual o endereço a programar na RAM e qual a RAM seleccionada. Desta forma se processa a reprogramação de um endereço de uma RAM, o que terá de ser repetido para todos os endereços a programar. Alternar entre algoritmos não implica uma total reescrita das DP-RAMs mas apenas dos endereços necessários. O processo de reprogramação de um byte pode ser melhor visualizado, na figura 6.

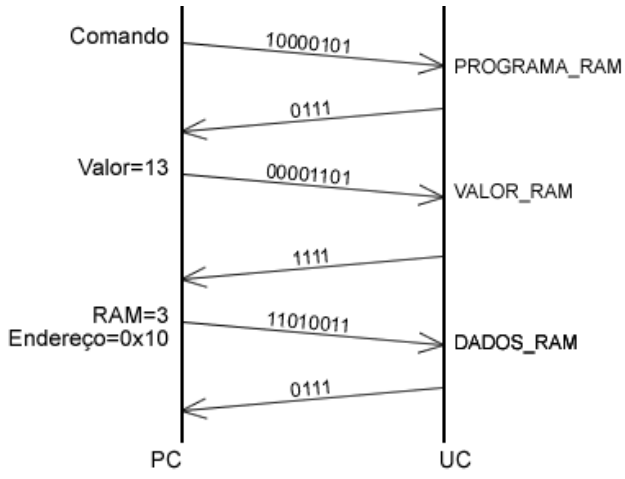

Fig. 6: Exemplo do endereço 0x10 da DP-RAM 3 com o valor 13

Ao ser enviado o comando PROGRAMA_RAM (10000101) o U.C. retorna a negação dos 4 bits mais significativos (0111) indicando que recebeu o comando e está pronta para continuar. De seguida o PC envia o comando VALOR_RAM (00001101) que informa que pretende programar a RAM com o valor 13 (0001101), a UC responde novamente da mesma forma (1111), informando o PC que se encontra preparada para o próximo passo. De salientar que a UC e o PC mantêm o sincronismo com o bit mais significativo, pois este é sempre negado o que permite uma certeza no sincronismo. Finalmente, o PC envia o ultimo comando DADOS_RAM (11010011), que inclui o endereço da RAM a programar (1010) e a RAM a programar (011), confirmando a UC a boa recepção do comando (0010), ficando assim confirmado que toda a operação foi conseguida sem problemas.

E. A parte de software

Foi desenhado um software que permite enviar todos os comandos para a FPGA. Este software foi desenvolvido em C++, tendo as seguintes funcionalidades:

- Enviar Vector
- Programar MEFR
- Iniciar algoritmo
- Reset

O programa é bastante simples, mas permite compreender o funcionamento do circuito que desenvolvemos. Deve ser encarado como um exemplo simples, de como se efectua a comunicação com o circuito, e não como um software completo. É de grande utilidade para efeitos de depuração de erros ou para experimentação do hardware.

V. IMPLEMENTAÇÃO DE OPERAÇÕES SOBRE VECTORES BOOLEANOS

Este projecto torna-se bastante útil quando se pretende construir, de uma maneira simples e económica, um sistema dedicado a executar um algoritmo específico, e assim libertar o processador de um PC comum. Muitos são os algoritmos possíveis de ser implementados, e a principal limitação é o numero de estados possíveis da MEFR. Exemplos “clássicos” são: contar 1s ou 0s de um vector, calcular paridade de vectores, executar operações lógicas mais complexas e pequenos sistemas de encriptação de dados. Para que um algoritmo seja implementado, é simplesmente necessário criar uma representação da maquina de estado que o implementa e proceder ás devidas alterações nas DP-RAMs.

VI. CONCLUSÕES

O âmbito desta cadeira era inicialmente um pouco desconhecido, pois não tínhamos muito a noção com o que nos iríamos deparar concretamente. Deparamo-nos então com um trabalho onde teríamos como suporte o software Xilinx e uma placa com uma FPGA, sendo-nos gradualmente explicado o que iríamos fazer. O facto de termos acesso a circuitos electrónicos deixou-nos um

pouco inseguros pois não temos grandes conhecimentos na área da electrónica, mas logo nos apercebemos que não seria com electrónica que iríamos trabalhar. A nossa ferramenta de trabalho revelou ser o software Xilinx, que nos proporcionou uma base de construção, de teste e de implementação dos circuitos realizados. As bases aprendidas em Sistemas Digitais, Arquitectura de Computadores, Circuitos Eléctricos e Electrónicos e Paradigmas da Programação I [4] mostraram-se muito úteis na elaboração e análise deste projecto.

No entanto, tivemos alguns problemas principalmente ao nível do teste em placa FPGA pois esta comportava-se de maneira diferente do que o simulador do Xilinx, o que nos levou a perder algum tempo a tentar compreender o porquê dos problemas e a sua resolução (quando possível).

O paradigma da computação reconfigurável abordado revelou-se bastante útil e de fácil compreensão para alunos de um curso onde as bases de electrónica são muito escassas.

As potencialidades do software Xilinx e da linguagem VHDL foram bem verificadas, bem como a possibilidade de manter um sistema funcional e interoperável implementado em hardware e software.

REFERÊNCIAS

[1] V.Sklyarov, Síntese e Implementação de Circuitos Digitais Reconfiguráveis Dinamicamente, Electrónica e Telecomunicações, Vol. 3, Nº 8, Jan. 2003 (nesta revista).

[2] Johnny Santos e Nuno Duarte, Síntese e Implementação de Circuitos Digitais Reconfiguráveis Dinamicamente, Electrónica e Telecomunicações, Vol. 3, Nº 8, Jan. 2003 (nesta revista).

[3] Página *http://webct.ua.pt*, "2º Semestre", a disciplina "Computação Reconfigurável".

[4] Página *http://webct.ua.pt*, "2º Semestre", a disciplina "Paradigmas de Programação *I*".