Síntese e Implementação de Circuitos Digitais Reconfiguráveis Dinamicamente (Projecto 2)

Diogo Gomes, Nuno Carvalho

***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

O problema apresentado, deparava-nos com a necessidade de construir um circuito capaz de executar qualquer operação sobre um vector binário fornecido pelo computador e devolver ao mesmo o resultado dessa mesma operação. As operações podem ser as mais variadas pelo que o circuito deve ser reprogramável pelo computador de uma forma simples e dinâmica.

II. ESPECIFICAÇÃO DO PROJECTO

Para a resolução deste problema seguiu-se a abordagem descrita no artigo [1]. Com este projecto pretendia-se obter no final uma máquina capaz de realizar operações sobre um vector de tamanho variável. Este vector devia ser comunicado pelo PC a máquina, esta após realização de uma operação previamente programada deveria comunicar ao PC o resultado da mesma operação. A operação feita pela máquina pretende-se que se possa ser alterada. Assim construiu-se uma Máquina de Estados Finitos Reprogramável (MEFR) com três níveis e apartir de memórias *dual-port RAM*, que nos permitirão no futuro uma mais fácil reprogramação de todo sistema. Procedeu-se também à projecção e construção de uma unidade de controlo responsável pela interface com o PC e pela programação da MEFR (seu funcionamento será exposto mais a frente). Construiu-se ainda uma unidade de execução capaz de realizar o maior leque possível de operações sobre um vector booleano. Todos estes elementos básicos foram construídos e testados individualmente antes de serem interligados entre si. Tal permitiu uma mais fácil detecção de erros e um período de despistagem dos mesmos mais rápido.

III. ARQUITECTURA BÁSICA

A fig. 1 mostra a implementação do sistema em hardware de uma forma esquemática e por blocos funcionais. Assim sendo temos um bloco responsável pela interface com o PC (Unidade de Controlo) que comunica directamente com a memória (de forma a comunicar o vector) com a unidade de controlo (estabelecendo todo o sincronismo necessário e correcto funcionamento da máquina) e com a MEFR (afim de se proceder à reprogramação desta). O *DIP switch* permitirá especificar o tamanho do vector. A memória é *dual-port RAM* e permite não só o armazenamento do vector como também do resultado. A unidade de controlo é também uma máquina de estados finitos no entanto é construída com ROM's pela que não é reprogramável dinamicamente. A MEFR é composta por 3 níveis [1,2] e permite existência de 16 estados, o que nos dá uma margem bastante razoável na criação de algoritmos. Por fim existe uma pequena unidade responsável por segmentar os resultados de que são enviados devolta ao PC já que este apenas pode receber 4 bits por relógio por oposição a 8 bits no envio.

IV. SÍNTESE, MODELAÇÃO E IMPLEMENTAÇÃO DO SISTEMA

A. Interface

De forma a existir sincronismo com o PC definiu-se à partida que todas as instruções começam por 0x80. O bit mais significativo desencadeia a unidade de controlo no sentido de processar a instrução, na ausência deste bit a máquina reconhece estar na presença de um dado. De forma semelhante a sincronização com o PC é feita por meio do sinal de controlo 0x0F quando a máquina se encontra a espera de dados e ausência do mesmo quando já o recebeu (exemplos de instruções e sincronismo serão abordadas em IV.5.).

B. Unidade de execução

A unidade de execução é controlada através de um vector de 5 bits (yy) ao qual se junta um sinal de RESET. A estes sinais junta-se ainda o vector operando (V) o tamanho do mesmo (sz) e o relógio. Por sua vez, à saída temos um vector que retorna resultados (x) um sinal de

fim de operação (READY) e o resultado propriamente dito da operação (res). A fig. 2 apresenta-nos um extracto da UE utilizada.

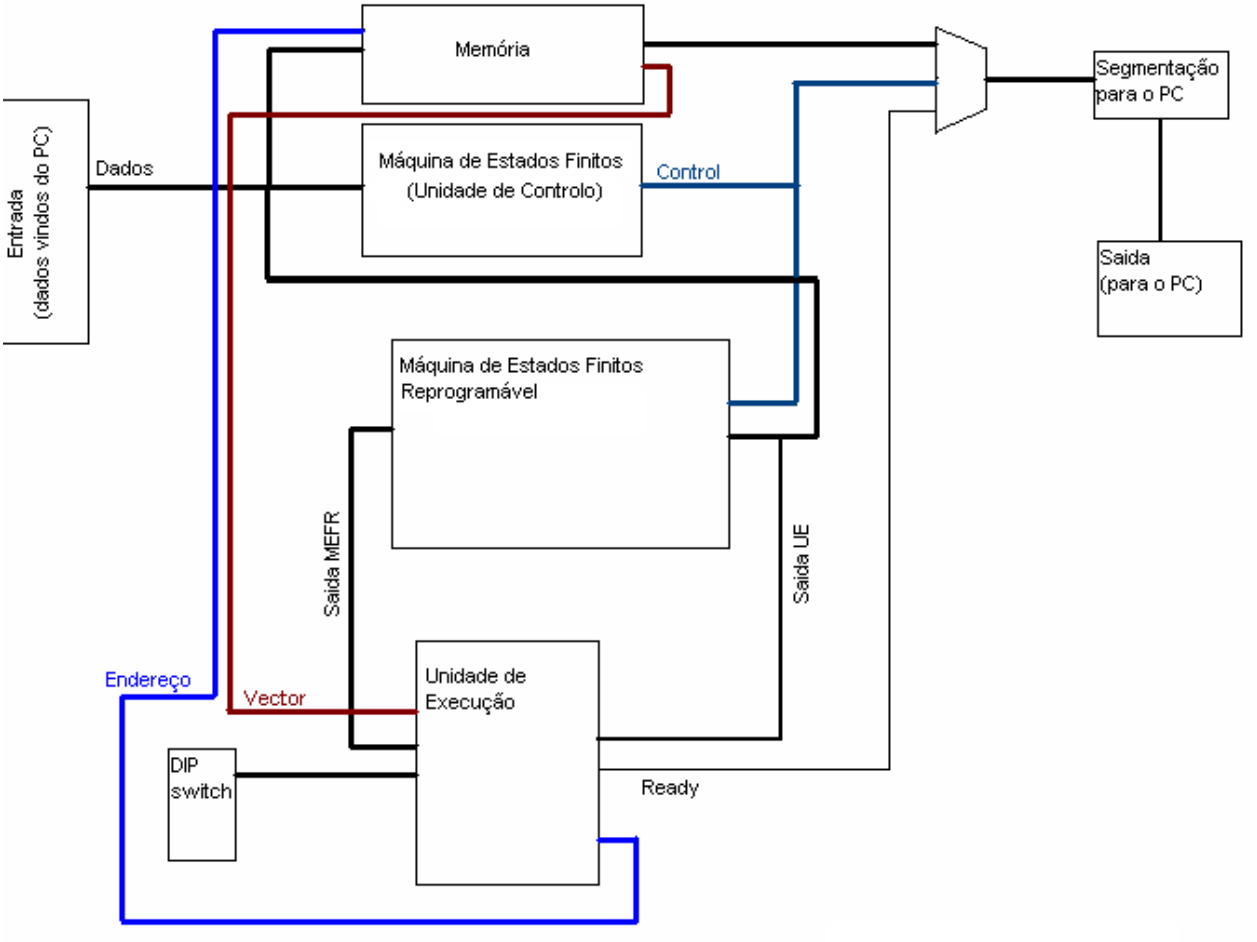

Fig. 1 - Arquitectura básica do sistema

C . Unidade de controlo

A unidade de controlo foi também ela implementada como uma máquina de estados finitos no entanto a sua implementação foi feita recorrendo a ROM's, uma vez que não existe o requisito de reprogramação dinâmica da unidade de controlo. A unidade de controlo controla não só o interface com o PC como também todo o processo de programação da MEFR. A fig. 3 apresenta-nos o diagrama de estados da unidade de controlo.

Existem pois 5 instruções base: Write Data (escreve dados num endereço de memória), Read *RAM* (lê dados apartir de um endereço de memória), Read result (lê o bit ready da UE), Write Address (escreve valor de endereço num registo temporário (estado A6)) e Write *RAM* select (escolhe em quem memória vamos proceder a escrita/leitura dos dados (estado A7)).

Este diagrama de estados permite que se escreva tanto na memória principal (onde se armazena o vector) como nas memórias da MEFR (que permite a reprogramação desta) de forma transparente. Notar ainda que todas estas instruções precisam de dois passos, um primeiro envio do PC do código da instrução (com o bit 0x80) seguido (logo após o PC receber o final 0x0F) por um novo vector seja ele dados, endereço memória, endereço da *RAM* ou vector de 0's (vector de zeros corresponde a uma operação nula em que o vector mantém o seu valor).

D. Reconfiguração

As operações realizadas são descritas pelo algoritmo programado na MEFR, pelo que a alteração deste implica a reprogramação da mesma. Afim de reprogramar-mos a MEFR é necessário instruir a Unidade de Controlo (UC) que desejamos alterar o conteúdo das *RAM*'s. Para alterar um só parâmetro numa *RAM* são necessárias 3 instruções: Write *RAM* select (escolha da *RAM* a reprogramar), Write Address (endereço da *RAM* a reprogramar) e por fim Write Data (escrita do novo conteúdo).

De forma a programar toda a máquina estas 3 instruções são repetidas para todos endereços que sejam necessários reprogramar.

E. A parte de software

Do lado do PC foi programada uma pequena aplicação que nos permite sistematizar todas as operações que a máquina é capaz de realizar (ver fig. 4) assim como fazer interface e sincronismo com a máquina.

```
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.2 – Extracto do código VHDL para a Unidade de Execução.

Aplicação foi desenvolvida em C++ em Ambiente Windows e recorre a uma biblioteca externa chamada «dlport». Existem duas funções básicas na interface com a FPGA implementadas de acordo com a fig. 5.

Uma das funcionalidade mais importantes da parte de software é a implementação correcta do mecanismo de sincronização. A fig. 6 expõe duas funções que verificam se o programa pode proceder ao envie de dados e se o mesmo foi concluído.

Várias outras funções foram consideradas e podem ser consultadas [3].

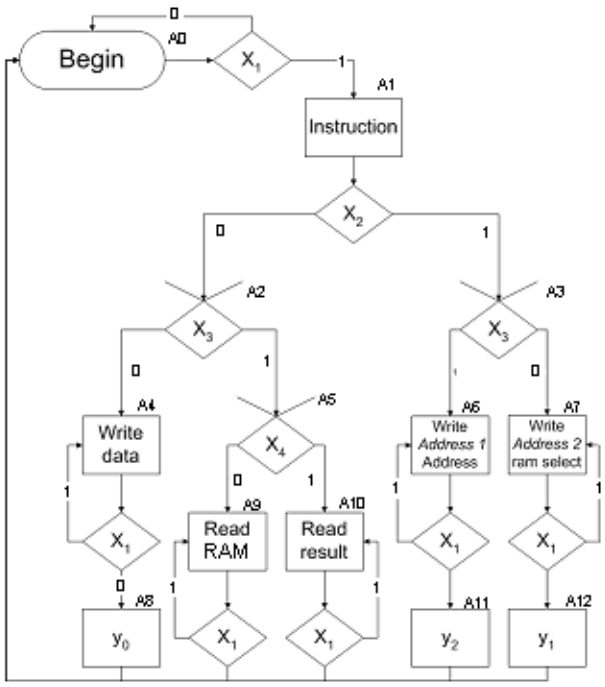

Fig. 3 – Diagrama de Estados da Unidade de Controlo

```
run - Run code in the FSM
w - write a 8 bit word to a specific address
r - read a 8 bit word from a specific address
p - program the FSM
prog - program a specific entry in the FSM
reset - clean the data memory
x - exit
```

Fig. 4 – Menu de Ajuda da aplicação desenvolvida ilustrativo das funções disponibilizadas

```
void sender(unsigned char value)
{
        ULONG port = 0x0378;
        /*DLPORT_API*/
        DlPortWritePortUchar(port,value);
}

unsigned char receiver()
{
        unsigned char value;
        ULONG port = 0x0379;
        value = DlPortReadPortUchar(port);

        value &= 0x78;
```

```
        value >>= 3;
        return value;
}
```

Fig. 5 – Funções que permitem comunicar com a FPGA

```
void wait_send_ready() {
        unsigned v = receiver();
        while((v=receiver())!=0x0f);
}
void wait_send_done() {
        unsigned v = receiver();
        while((v=receiver())!=0x00);
}
```

Fig. 6 – Funções de sincronização com FPGA

V. IMPLEMENTAÇÃO DE OPERAÇÕES SOBRE VECTORES BOOLEANOS

A operação demonstrada no nosso projecto foi muito simples e consistia apenas da contagem de 1's no vector.

Assim procedeu-se à programação das *RAM*'s da MEFR com os arrays descritos na fig. 7 (ver 4,5 para detalhes).

```
unsigned saidas[16] =       {0,8,4,16,
                            0,1,0,2,
                            0,0,0,0,
                            0,0,0,0};

unsigned level1[32] =       {0, 1, 4, 4, 3, 3, 4, 6,
                            3, 2, 1, 1, 6, 6, 7, 7,
                            8, 8, 9, 9, 10, 10, 11, 11,
                            12, 12, 13, 13, 14, 14, 15, 15};

unsigned level2[32] =       {0, 0, 1, 1,2, 2, 3, 3,
                            4, 4, 5, 5, 5, 7, 7, 7,
                            8, 8, 9, 9, 10, 10, 11, 11,
                            12, 12, 13, 13, 14, 14, 15, 15};

unsigned level3[32] =       {0, 0, 1, 1, 2, 2, 3, 3,
                            4, 4, 5, 5, 6, 6, 7, 7,
                            8, 8, 9, 9, 10, 10, 11, 11,
                            12, 12, 13, 13, 14, 14, 15, 15};

unsigned mux_ram[16] = {4,3,3,2,
                            0,3,1,3,
                            7,7,7,7,
                            7,7,7,7};
```

Fig. 7 – Array's usados para programar as RAM's da MEFR.

Trata-se de uma aplicação muito simples, no entanto outras operações podem ser realizadas tais como bit stuffing de um vector, negação do mesmo, etc. Necessitando para tal apenas de proceder alterações nestes array's no nosso programa em C++ que se encontra no PC.

Os materiais adicionais sobre o problema considerado podem ser encontrados em [6-8]. O projecto que inclui todas as partes consideradas a cima está apresentado na WebCT [8].

VI. CONCLUSÕES

Com esta cadeira tivemos pela primeira vez contacto físico com circuitos electrónicos, assim como com o paradigma da computação reconfigurável. A principio sentiamo-nos algo inseguros com as nossas capacidades de realizar este projecto dada a quase ausência de conhecimentos na área da electrónica. No entanto cedo nos apercebemos que para a realização deste projecto os conhecimentos adquiridos em Sistemas Digitais, Arquitectura de Computadores e Paradigmas de Programação 1 seriam mais que suficientes. A facilidade com que construímos e testamos o nosso primeiro bloco funcional usando as ferramentas da Xilinx e a orientação do prof. Valery Sklyarov permitiu-nos sermos mais ambiciosos e com alguma facilidade cedo atingimos os nossos objectivos. No entanto, pelo caminho algumas dificuldades de depuração mostraram-se desafios exigentes, mas também elas superadas graças ao apoio sempre pronto por parte do professor.

Esta técnica mostrou-se pois bastante indicada para desenvolvimento de circuitos reconfiguráveis por parte de alunos de uma licenciatura em eng. Informática que com os poucos conhecimentos de electrónica foram capazes de projectar, construir e testar um circuito desta complexidade.

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 (Projecto 1). Electrónica e Telecomunicações, Vol. 3, Nº 8, Jan. 2003 (nesta revista).

[3] Página http://sweet.ua.pt/~etdgomes/cr.zip

[4] V. Sklyarov, Modelação em C++, Síntese e Implementação de Circuitos Digitais com base em FPGA. Electrónica e Telecomunicações, Jan., Vol.. 3, Nº5, 2002, pp. 409-420.

[5] V. Sklyarov, Reconfigurable models of finite state machines and their implementation in FPGAs. Journal of System Architeture, 47, 2002, pp. 1043-1064.

[6] Página *http://webct.ua.pt,* "1º Semestre" a disciplina "Sistemas Digitais Avançados".

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

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