Síntese e Implementação de Circuitos Digitais Reconfiguráveis Dinamicamente (Projecto 1)

Johnny Santos, Nuno Duarte

Resumo – O artigo apresenta os resultados do projecto proposto aos autores (que são os estudantes do 4º ano de LECT) e mostra como o problema especificado foi resolvido. A descrição do projecto e dos requisitos básicos foi considerada no artigo [1]. De acordo com a especificação do problema era necessário projectar um sistema simples implementado em hardware e software. O sistema tem que realizar operações várias sobre um vector binário com tamanho de 24 bits. Os tipos de operações que se podem efectuar sobre um vector binário podem ser arbitrários no entanto o número das operações no sistema é fixo. A personalização das operações deve ser conseguida através da modificação da funcionalidade do sistema. A parte de hardware foi realizada numa FPGA XC4010XL da Xilinx ligada ao computador PC através da porta paralela. Esta parte inclui os seguintes componentes básicos: interface com computador PC; unidade de execução que nos permite realizar operações especificadas sobre vectores binários; unidade de controlo, em que a sua funcionalidade é modificável dinamicamente, o que permite definir operações sobre vectores binários; controlador de reconfiguração, que permite mudar a funcionalidade da unidade de controlo apartir do computador. A parte do software foi implementada em C++ e corre no computador PC. Este fornece os dados necessários para as operações especificadas e permite modificar o conjunto de operações apartir do computador através da porta paralela. Este último é conseguido modificando a funcionalidade da unidade de controlo.

***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 paper [1]. According to the specification of the problem it was necessary to design a simple system implemented in hardware and in software. The system has to perform various operations over Boolean vectors with the size 24 bits. The number of feasible operations can be infinite and the number of particular operations implemented in the system has been fixed. Customizing of the operations has to be achieved through a modifiability of the system functionality. Hardware part has been realized in FPGA XC4010XL of Xilinx linked to the host PC computer through a parallel port. This part includes the following basic components: interface with PC computer; execution unit that enables us to carry out the specified operations over Boolean vectors; control unit with dynamically modifiable functionality that permits to customize operations over Boolean vectors; reconfiguration handler, which allows to change the functionality of the control unit from the host computer. Software part has been implemented in a C++ program running on the host computer. It provides the required data for the specified operations and permits to modify the set of operations from the host computer through the parallel port. The latter is achieved by modifying the control unit functionality.**

I. INTRODUÇÃO

O que se deveria mostrar no final do projecto, seria um circuito, implementado numa FPGA, e deveria estar implementado em hardware de modo a poder ser reconfigurado por software. O projecto deveria receber determinados dados, mais propriamente, um vector e o respectivo tamanho. Estes dados seriam utilizados pela máquina de estados finitos reprogramável (MEFR) que efectuaria determinadas operações para a qual tinha sido programada, por exemplo, contar o número de 1's do vector, verificar se o número de 1's do vector é par ou impar, etc, após isso o circuito deveria retornar o resultado da operação efectuada sobre o vector.

Como a comunicação com o hardware (FPGA) deveria ser feita por software recorreu-se ao uso da linguagem de programação C++ para poder escrever um programa que pudesse ler o vector, o respectivo tamanho, executar o respectivo algoritmo na máquina de estados finitos (MEF) e reprogramar a MEFR. Utilizámos também as ferramentas do Xilinx Foundation Software [2] e a linguagem VHDL para construir o circuito.

II. ESPECIFICAÇÃO DO PROJECTO

Talvez se possa dizer que para a resolução do projecto proposto grande parte dos componentes que são necessários já tinham sido apresentados nas aulas [3,4,5], mas em modelos mais simples, cabendo-nos saber que componentes seriam necessários e como adapta-los às circunstâncias que se pretendiam.

Um possível modo de abordar o problema seria dividi-lo em quatro partes.

1) A primeira onde se fizesse a interface com o computador PC,

2) numa segunda onde a preocupação principal seria armazenar os dados que viessem do computador PC,

3) outra onde se deveria programar a MEFR com os dados provenientes do computador PC,

4) e por último ter-se-ia de retornar os dados para o computador PC provenientes da FPGA.

Inicialmente após uma abordagem superficial ao projecto pensaríamos que o projecto seria bastante simples e que bastaria:

- no que diz respeito a entrada de dados na placa criar um componente com os pinos da porta paralela correspondentes,
- para guardar os dados que viessem do computador PC, criar um bloco que seria um conjunto de registos,
- para reprogramar a MEFR utilizar-se memórias de *Dual Port RAM*, ou outras memórias em que se pudesse escrever no seu conteúdo,
- para a saída de dados bastaria criar um componente que fizesse interface entre a placa (FPGA) e o computador PC.

Contudo, após um estudo mais aprofundado sobre o assunto facilmente se verifica que o problema não seria assim tão simples, pois poderiam surgir problemas como: a sincronização entre o computador PC e a placa (FPGA); modo de controlar as operações que pudessem ser efectuadas sobre a placa.

Tendo em conta estes tipos de problemas, a solução para a resolução do problema teve de ser muito bem pensada.

III. ARQUITECTURA BÁSICA

O modo que se encontrou para resolver o problema foi dividir o sistema em duas partes: uma parte de *software* desenvolvido num PC, usando a linguagem C++; e outra efectua o controlo em *hardware* na FPGA (ver fig. 1).

No que diz respeito ao hardware foi necessário criar uma MEF, utilizando memórias do tipo *ROM*, onde implementasse um algoritmo que permitisse controlar todo o conjunto de operações que seriam executadas na placa (FPGA). Esse algoritmo permite receber e gravar o vector e o tamanho, reprogramar a MEFR, iniciá-la e retornar o resultado da mesma.

Para tal efeito foi necessário criar alguns componentes, tais como, uma MEF que implementasse o algoritmo; criou-se uma MEFR, que pudesse ser alterada de acordo com a operação que se pretendesse efectuar, tendo em conta uma unidade de execução que também teve que ser criada; foi criado registos que permitiriam guardar valores; houve também a necessidade de criar uma unidade de controlo, que fizesse a interligação entre a máquina de estados finitos e os outros componentes;

A Fig. 1 mostra a estrutura geral do sistema implementado. A MEF implementa um algoritmo que permite efectuar cinco operações, sendo estas:

- escreve vector;
- escreve tamanho;
- gravar memórias da MEFR;
- executar MEFR
- retornar resultado.

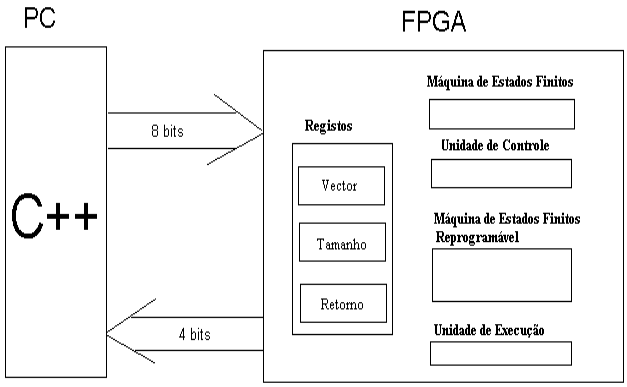

Fig. 1 – Estrutura geral do sistema implementado

O algoritmo é composto por 18 estados (ver fig. 2), sendo estes o menor número de estados que implementam o pretendido de uma forma compreensível, simples e que garantem que possa existir sincronização entre o computador PC e a placa com FPGA.

Para a melhor compreensão do que cada estado significa, passa-se a apresentar uma pequena descrição dos mesmos:

A0 – Espera: é um estado onde se aguarda que o bit de sinalização por parte do computador PC venha a '0';

A1 – é um estado intermédio;

A2 – Vector: é o estado que "define" a escrita do vector;

A3 – Actualiza: é um estado onde se espera pelos valores correctos por parte do computador PC;

A4 – Escreve: escreve os dados no respectivo registo;

A5 – Tamanho: "define" a escrita do tamanho;

A6 – estado intermédio;

A7 – Memória: grava no registo qual o bloco de memória que se pretende alterar;

A8 – Actualiza: é um estado onde se espera pelos valores correctos por parte do computador PC;

A9 – estado intermédio;

A10 – Escreve Endereço;

A11 – Escreve Dados;

A12 – Inicia MEFR;

A13 – Espera: espera que a MEFR termine;

A14 – Retorno: retorna a primeira parte do resultado da operação efectuada sobre o vector;

A15 – Grava MEFR: grava os dados que estão nos registos para as memórias;

A16 – Inicia: é o início;

A17 – Retorna2: retorna a segunda parte do resultado da operação efectuada sobre o vector;

A MEFR é um circuito que a nível de componentes só difere da MEF por a sua memória não ser *ROM* mas sim *Dual Port RAM* com a finalidade de poder ser alterado o seu conteúdo (ver detalhes em [6,7]). A saída no nosso caso é de 5 bits, pois é o suficiente para os algoritmos que se pretendem implementar.

A unidade de execução, foi implementada em VHDL. Caso se pretenda alterar o algoritmo que está implementado na MEFR tem que se ter em conta o funcionamento da unidade de execução.

O bloco de registos é onde ficam guardados todos os dados que se pretendem armazenar, como:

- bloco de memória (qual o bloco de memória da MEFR que se deve alterar; ver detalhes em [1,6,7]),
- endereço (qual endereço do bloco de memória da MEFR que se deve alterar),
- dados (quais os dados do endereço do bloco de memória da MEFR que se deve alterar),
- vector (vector com que se pretende trabalhar (conjunto de 4 blocos de 6 bits))
- tamanho do vector,
- retorno da MEFR.

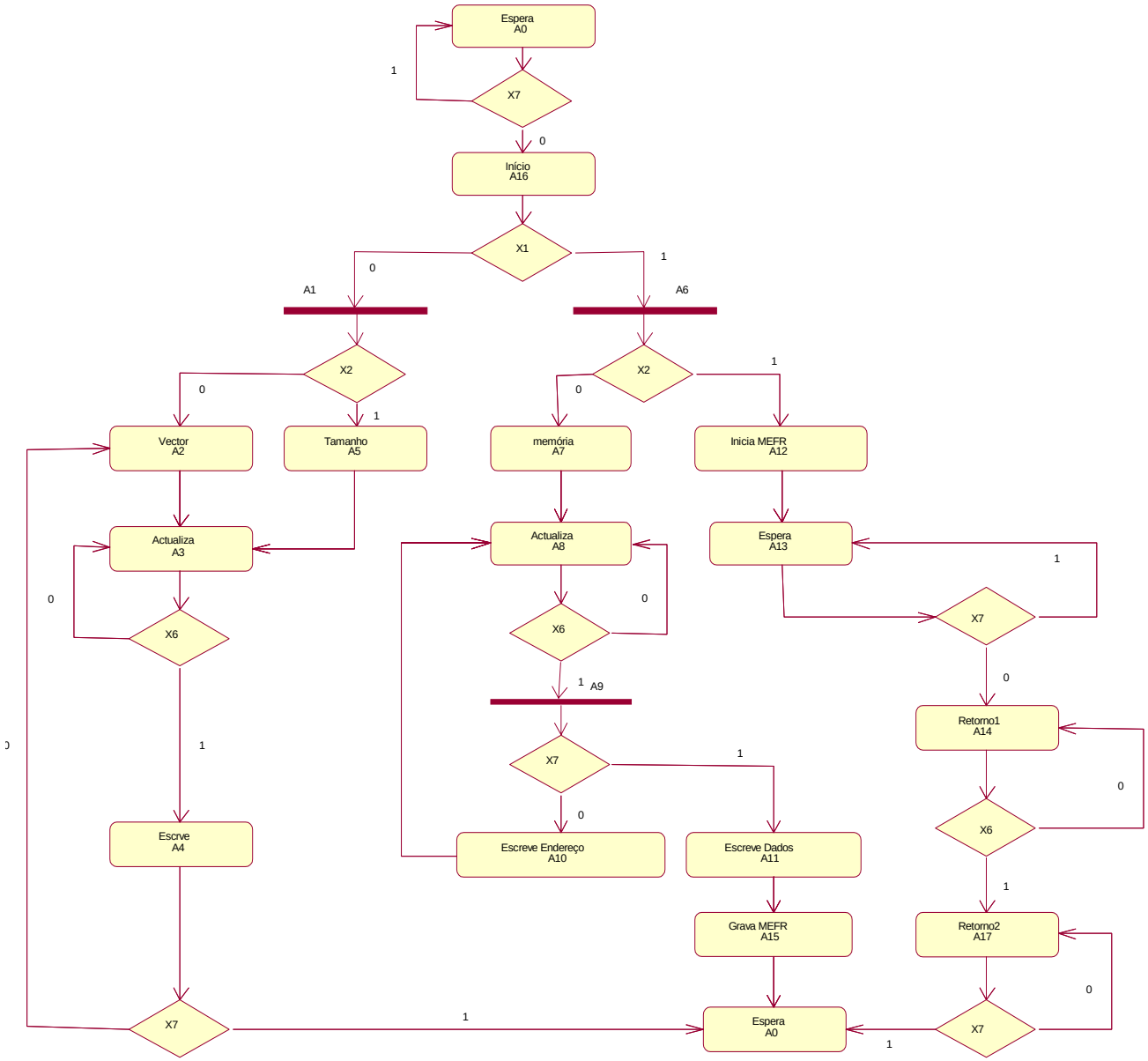

Fig. 2 – Algoritmo implementado na MEF

Neste bloco de registos é onde vão ser guardados: o tamanho do vector e o vector, provenientes da interface com o PC; o resultado da operação efectuada sobre o vector, proveniente da unidade de execução; o bloco de memória; o endereço e os dados. Estes conjuntos de vários registos estão representados na fig. 3 A unidade

de controlo de acordo com o estado da MEF activa ou não determinados componentes, e também de acordo com os sinais provenientes de outros componentes altera o estado da mesma.

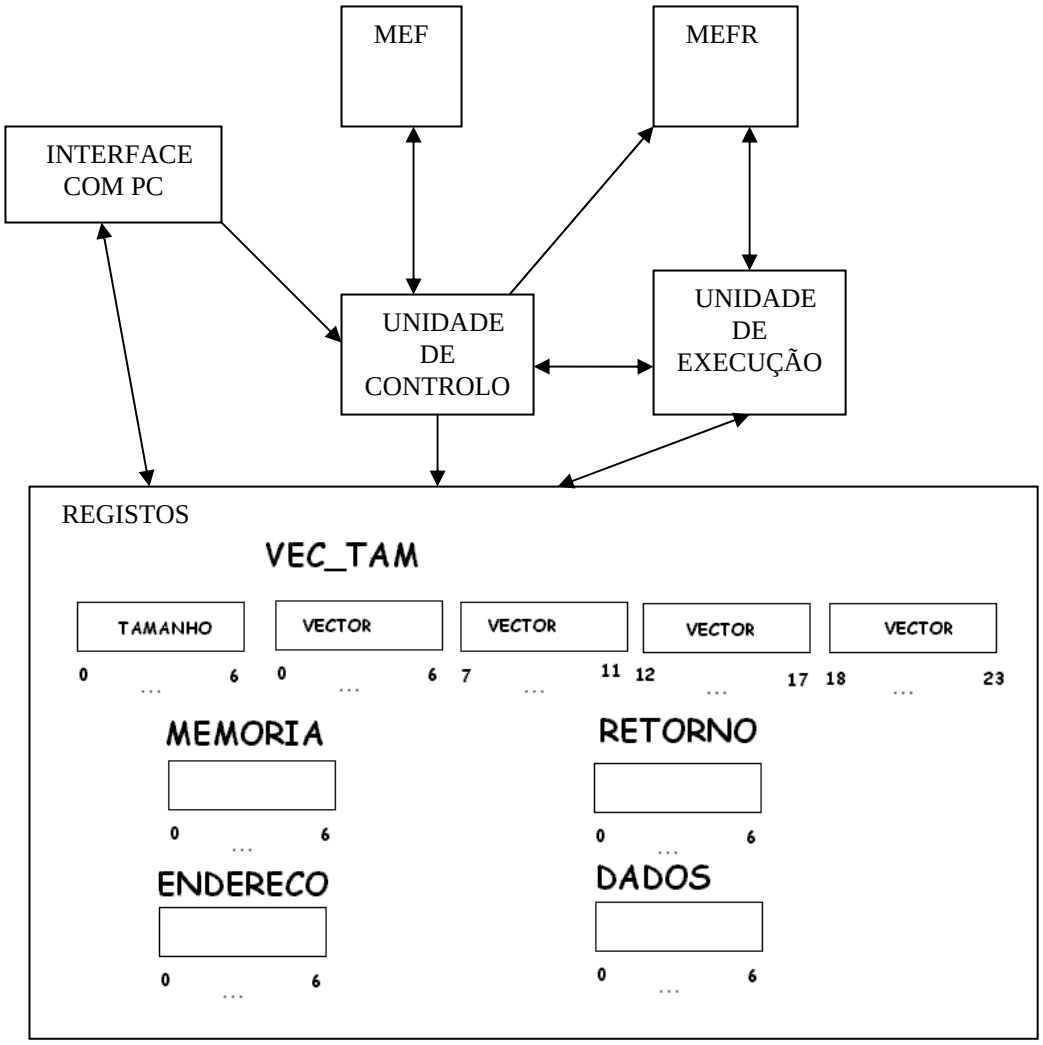

Fig. 3 – Organização do bloco de Registos e sua comunicação com os outros componentes

A Fig.4 pretende mostrar como a parte nuclear dos componentes se interligam.

O conjunto dos blocos OSC4 e C84CE são utilizados para gerar um sinal de relógio com baixa frequência.

O bloco FSMR_R_4_MACRO (H2) é a MEF e implementa o algoritmo da fig.2. Este comunica com a unidade de controlo através das variáveis XX0,...,XX7 (variáveis de entrada) e FSMR_SAIDA0,.., FSMR_SAIDA7 (variáveis de saída).

O bloco UNIDADE_CONTROLO (U2) é a unidade de controlo, este comunica com a MEF, através de: ME0,..,ME7 (variáveis de entrada) e OME0,..,OME7 (variáveis de saída); comunica com a unidade de execução através de: MER (variáveis de entrada), servindo esta para indicar que a unidade de execução terminou a operação sobre o vector, e ACTIVARMER (variáveis de saída), que serve para iniciar a unidade de execução; comunica com o bloco de registos através das variáveis de saída: OVEC_TAM, serve para activar o conjunto VEC_TAM composto por cinco registos (ver fig. 3), OENDERECO, serve para activar o conjunto ENDERECO composto por um registo (ver fig. 3), ODADOS, serve para activar o conjunto DADOS composto por um registo (ver fig. 3), ORETORNO, serve para activar o conjunto RETORNO composto por um registo (ver fig. 3), OMEM, serve para activar o conjunto MEMÓRIA composto por um registo (ver fig.3), COUNT0,..,COUNT2, serve para escolher dentro do conjunto VEC_TAM qual o registo que se pretende seleccionar; comunica com a MEFR através de OFUTURA que é uma variável de saída, a qual quando activada vai permitir que o conteúdo dos blocos de memória da MEFR seja alterado; comunica com a interface do PC através de: IPC0,..,IPC7 (variáveis de entrada) e ACK (variáveis de saída) sendo esta última utilizada para a sinalização.

- O bloco UNIDADE_EXECUCAO (U1) é a unidade de execução, este comunica com a MEFR, através de: Y0,..,Y7 (variáveis de entrada) e X1,..,X6 (variáveis de saída); comunica com a unidade de controlo através de: WR (variáveis de entrada), que serve para iniciar a unidade de execução e READY (variáveis de saída), servindo esta para indicar que a unidade de execução terminou a operação sobre o vector; comunica com o bloco de registos através das variáveis de entrada: SZ e V, que são respectivamente o tamanho e o vector. E através das variáveis de saída: READY e RES, servindo esta última para devolver o resultado da operação efectuada sobre o vector.

O bloco FSMR_RAM_MACRO (U3) é a MEFR e implementa os algoritmos das operações que se pretendem efectuar sobre o vector. Este comunica com a unidade de execução através das variáveis XX1,...,XX6 (variáveis de entrada) e FSMR_REP_SAIDA0,..,FSMR_SAIDA5 (variáveis de saída).

De um modo geral é possível ver a importância da unidade de controlo, pois esta tem ligação com a maior parte dos componentes.

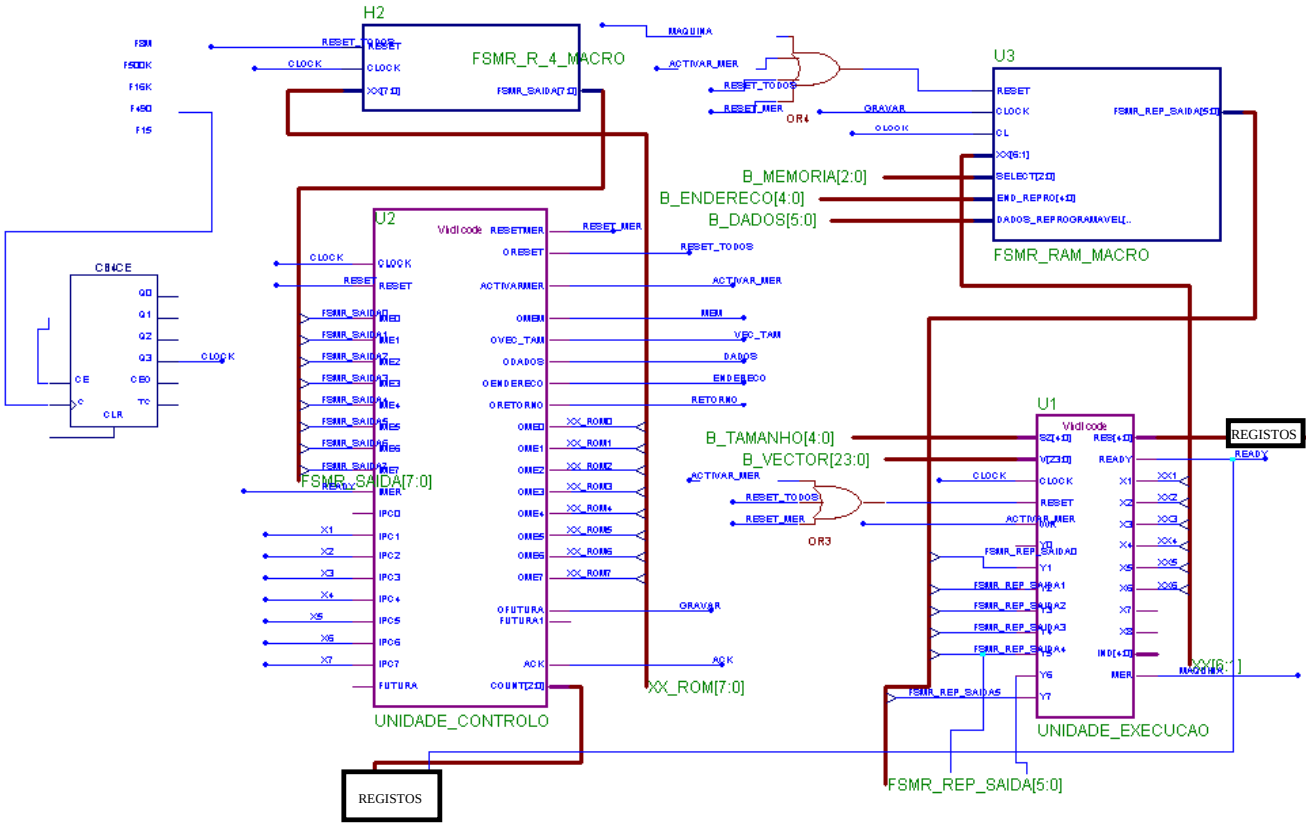

Fig. 4 – Parte dos componentes implementados com o Xilinx

IV. SÍNTESE, MODELAÇÃO E IMPLEMENTAÇÃO DO SISTEMA

A. Interface

A comunicação entre o computador PC e a placa é efectuada através da porta paralela, nomeadamente através dos 8 bits (para tal foi criado um componente que fizesse a interface). Desses 8 bits optou-se por atribuir o bit mais significativo para o sinal *Reset*, o segundo mais significativo para sinalização com a placa, e os restantes para dados (ver fig. 5).

No que diz respeito à comunicação entre a placa e o computador PC, como só 4 bits (ver fig. 6) são usados, optou-se por atribuir o mais significativo para comunicação com o computador PC e os restantes 3 bits para dados.

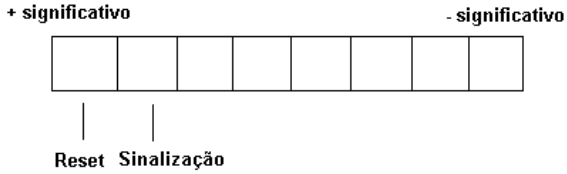

Fig. 5 – Apresentação de dados do computador PC para a placa

Fig,.6 – Apresentação de dados da placa para o computador PC

B. Unidade de execução

A unidade de execução, foi implementada em VHDL, como referido anteriormente. A seguir esta representado o código desta unidade.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity unidade_execucao is
   port( V: in STD_LOGIC_VECTOR (0 TO 23);
                sz: in integer range 0 to 23;
                res: inout integer range 0 to 23;
                clock: in STD_LOGIC;
                reset: in STD_LOGIC;
                wr: in STD_LOGIC;
                ready: out STD_LOGIC;
                y0: in STD_LOGIC;
                y1: in STD_LOGIC;
                y2: in STD_LOGIC;
                y3: in STD_LOGIC;
                y4: in STD_LOGIC;
                y5: in STD_LOGIC;
                y6: in STD_LOGIC;
                y7: in STD_LOGIC;
                x1: out STD_LOGIC;
                x2: out STD_LOGIC;
                x3: out STD_LOGIC;
                x4: out STD_LOGIC;
                x5: out STD_LOGIC;
                x6: out STD_LOGIC;
                x7: out STD_LOGIC;
                x8: out STD_LOGIC
                );
end unidade_execucao;

architecture unidade_execucao_arch of
unidade_execucao is
begin
st: process(clock,reset)
variable vector: STD_LOGIC_VECTOR (0 TO 23);

variable Size: integer range 0 to 23;
variable Index: integer range 0 to 23;
variable flag: STD_LOGIC;
        begin
                if reset='1' then
                  Index:=0;
                  res<=0;
                  ready<='0';
1 ⟶     elsif clock='0' and clock'event then
                        if wr='1' then
                          vector:=V;
                          Size:=sz;
                        end if;
                        if y1='1' then res<=1;
                        end if;
                        if y2='1' then res<=res+1;
                        end if;
                        if y3='1' then res<=Index;
                        end if;
                        if y4='1' then res<=0;
                        end if;
                        if y5='1' then flag:='1';
                        end if;
                        if y6='1' then flag:='0';
                        end if;
                        if y7='1' then
                        Index:=Index+1;
                        end if;
                        x1<=vector(Index);
                        x2<=flag;
                        if Size=Index then
                                x3<='1';
                                ready<='1';
                          else
                                x3<='0';
                                ready<='0';
                        end if;
2 ⟶             end if;
          end process st;
end unidade_execucao_arch;
```

Inicialmente a unidade de execução mantêm-se inactiva, isto quer dizer, mantêm-se num estado correspondente a reset, e esta necessita que a unidade de controlo active um sinal para que possa sair dessa situação. Esse sinal só é activado pela unidade de controlo quando a MEF salta para o estado A12 (ver fig. 2). Após a unidade de controlo activar esse sinal a MEF mantêm-se nesse estado até que a unidade de execução devolva o sinal de ready a um. Durante o intervalo de tempo em que a unidade de controlo coloca o sinal ACTIVARMER=1 (ver fig. 4) até que a unidade de execução coloca o sinal ready=1, a MEFR executa o algoritmo que tinha sido gravado. Quando a unidade de execução coloca o sinal ready=1 a unidade de controlo faz com que a MEF salte para o estado seguinte e assim volta a colocar o sinal de ACTIVARMER=0 (ver fig. 4), o que faz com que a unidade de execução volte a um estado de reset (inactiva)

A unidade de execução tem como sinais.

V:entrada do vector binário
sz: entrada do tamanho do vector
res: saída do resultado
clock: sinal de relógio

reset: sinal para fazer o reset
wr: sinal que indica que é possível escrever;
ready: indica que terminou a execução do algoritmo
y0,..,y7: sinais de entrada que correspondem à saída da MEFR
x1,..,x8: sinais de saída que correspondem à entrada da MEFR

De uma forma muito sucinta, o código que está entre 1 e 2 vai ser executado cada vez que o clock vem a zero. Nesse instante vai percorrendo os sinais de entrada (ex.: y1..y7,..) e de acordo com os seus valores os sinais de saída (ex.: res,x1,..x3,ready,...) vão sendo definidos. Quando se pretende implementar um novo algoritmo deve-se ter sempre em conta esta unidade de execução. Por exemplo, se num determinado estado se pretender incrementar o sinal de saída (res) da unidade de execução, o novo algoritmo nesse estado, deve activar o sinal de entrada y2.

C. Unidade de controlo

Outro dos principais componentes do projecto é a "Unidade de Controlo", esta foi descrita na linguagem VHDL. Consoante as saídas da MEF a "Unidade de Controlo" pode activar linhas que vão activar outros componentes. Essas saídas da MEF obedecem a definição mostrada na fig. 7

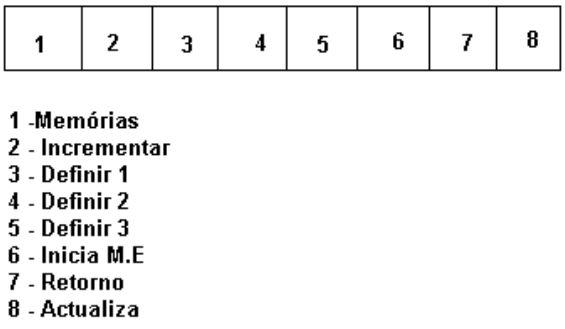

Fig. 7 – Definição das Saídas da MEF

Contudo o seu comportamento não se limita só à leitura dos sinais da saída da MEF e à activação de outros componentes, também lhe cabe definir qual as entradas para a MEF.

D. Reconfiguração

O algoritmo implementado e descrito na secção III (ARQUITECTURA BÁSICA) permite alterar o vector, o tamanho do mesmo, e principalmente reconfigurar a operação que será efectuada sobre o vector. Estas reconfigurações podem ser descritas da seguinte forma:

D.1 - Escrever vector

Como o vector tem de ter 24 bits e como só no máximo 6 bits podem ser utilizados na entrada da porta paralela (ver fig. 5), são efectuadas quatro sub-operações, sendo em cada sub-operação enviados 6 bits por parte do computador PC (primeiro os 6 bits menos significativos [0..5], depois os 6 bits seguintes [6..11], e assim sucessivamente). A operação de escrita de vectores pode ser apresentada do seguinte modo:

- A placa coloca o bit de sinalização a '1' (ver fig. 6);
- O computador PC envia os 6 bits menos significativos [0..5] simultaneamente com o bit de sinalização a '1' (ver fig. 5);
- A placa grava os 6 primeiros bits no registo (do qual falar-se-à mais à frente);
- A placa coloca o bit de sinalização a '1' (ver fig.6);
- O computador PC coloca o bit de sinalização a '0' (ver fig.5);
- A placa coloca o bit de sinalização a '0' (ver fig.6);
- O computador PC envia os 6 bits [6..11] simultaneamente com o bit de sinalização a '1';
- A placa grava os 6 seguintes bits no registo;
- A placa coloca o bit de sinalização a '1';
- O computador PC coloca o bit de sinalização a '0';
- A placa coloca o bit de sinalização a '0';
- O computador PC envia os 6 bits [12..17] simultaneamente com o bit de sinalização a '1';
- A placa grava os 6 seguintes bits no registo;
- A placa coloca o bit de sinalização a '1';
- O computador PC coloca o bit de sinalização a '0';
- A placa coloca o bit de sinalização a '0';
- O computador PC envia os 6 bits [18..23] simultaneamente com o bit de sinalização a '1';
- A placa grava os 6 últimos bits no registo;

D.2 - Escrever tamanho do vector

Outra operação que pode ser efectuada é a escrita do tamanho do vector. Esta obedece a um algoritmo muito semelhante ao do escreve vector (ver subsecção D.1), só que neste caso só existe a necessidade de guardar 5 bits (pois o tamanho máximo de 24 bits do vector podem ser codificados em 5 bits ($\log_2 24 < 5$)). Para facilitar a correspondência dos bits vindos do computador PC, e também com o objectivo de reutilizar certos componentes anteriormente criados, optou-se por utilizar 6 bits para definir o tamanho.

O algoritmo para gravar o tamanho pode ser descrito da seguinte forma:

- A placa coloca o bit de sinalização a '1' (ver fig. 6);
- O computador PC envia os 6 bits [0..5] simultaneamente com o bit de sinalização a '1' (ver fig. 5);
- A placa grava os 6 primeiros bits no registo.

D.3 - Gravar memórias da MEFR

Outra operação que pode ser efectuada, é a gravação das memórias da MEFR [1,6,7].

A execução de várias vezes desta operação vai permitir gravar um novo algoritmo. Estas operações têm como destinatário a *Dual Port RAM* da MEFR (ver fig. 8)

Como num ciclo de relógio não é possível dar todos os dados necessários, isto é, qual a memória, qual o endereço e quais os dados, para que se possa alterar a *Dual Port RAM* estes vão ter de ser guardados previamente nos registos (ver fig. 3).

Para esta operação optou-se por dividi-la em quatro partes:

1. Trata qual a memória onde se quer escrever (este passo é necessário pois a MEFR é composta por sete memórias [1]: nível 1 - 001, bloco H4; nível 2 - 010, bloco H5; nível 3 - 011, bloco H6; saída – 100, bloco H2 (SAIDA_REPROGRAMACAOSCH); mult8to1 – 101, bloco H9; mult8to1 – 110, bloco H8; mult8to1 – 111, bloco H7), o código do bloco da memória é guardada num registo (ver fig. 3 e fig. 8)
2. Consiste em dizer qual o endereço em que se pretende gravar, o endereço é guardado num registo
3. Quais os dados que se pretendem gravar no endereço especificado anteriormente (ver ponto 2), os dados são guardados num registo
4. Os valores dos registos são gravados na memória

Para efectuar esta operação são necessários mais 3 registos de 6 bits. Apesar dos dados relativos à memória só necessitarem de 3 bits, estes são guardados num registo de 6 bits, para aproveitar os componentes anteriormente criados (ver fig. 3). O algoritmo utilizado para alterar as memórias da MEFR pode ser descrito da seguinte forma:

- O computador PC coloca o bit de sinalização a '0' e indica qual a memória que pretende alterar;
- A placa grava a memória num registo;
- A placa coloca o bit de sinalização a '1';
- O computador PC coloca o bit de sinalização a '1' e envia o endereço para o qual os dados vão ser gravados;
- A placa coloca o bit de sinalização a '0', grava o endereço no registo;
- O computador PC coloca o bit de sinalização a '0';
- A placa coloca o bit de sinalização a '1';
- O computador PC coloca o bit de sinalização a '1' e envia os dados que pretende colocar no endereço;
- A placa coloca o bit de sinalização a '0', grava os dados no registo;
- Os dados que estão nos registos são utilizados para a reprogramação da MEFR.

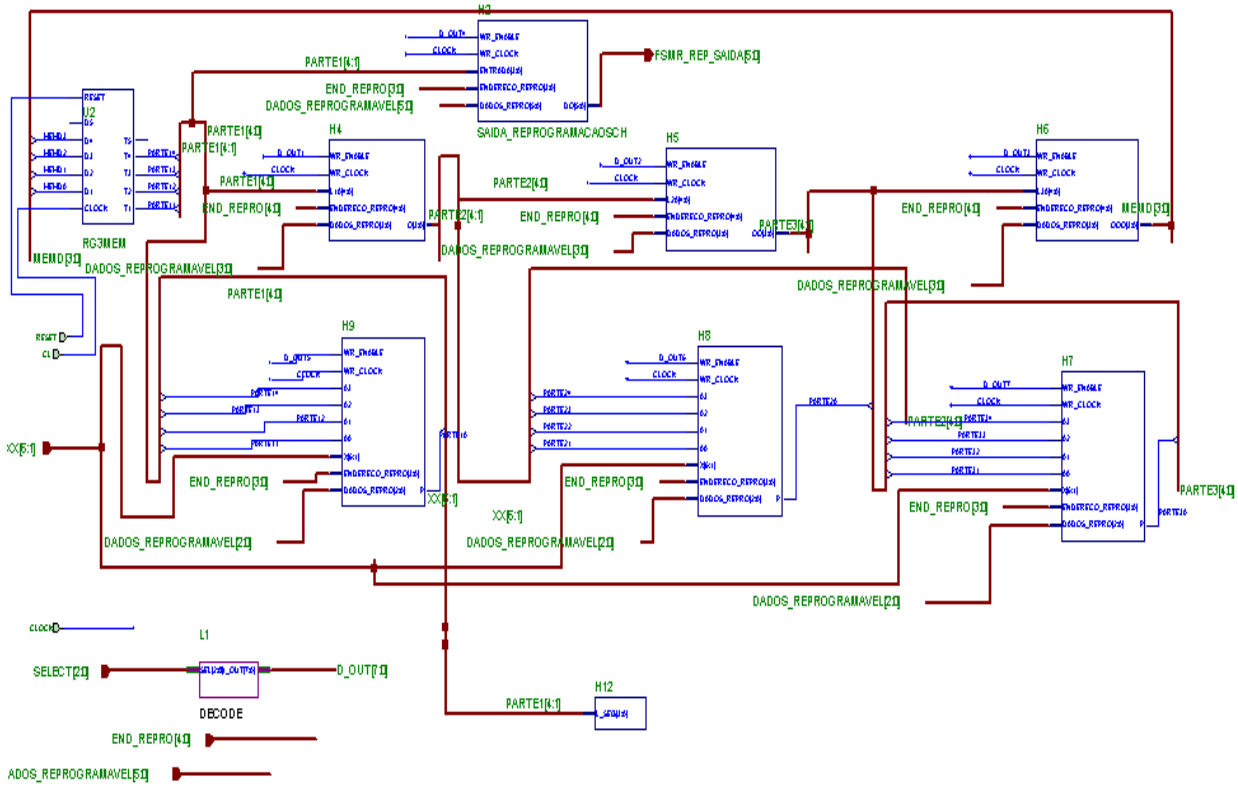

Figura 8 – Implementação da MEFR

D.4 - Executar MEFR

Por último também é possível executar o respectivo algoritmo da MEFR e retornar o resultado da operação que esta efectua sobre o vector. O resultado da operação nos casos considerados é definido em 6 bits, só que só podem ser retornados 3 bits de cada vez (pois um é para a sinalização).

E. A parte de software

A programação do PC, foi feita com recurso ao Microsoft® Visual C++ 6.0 (ver fig. 9).

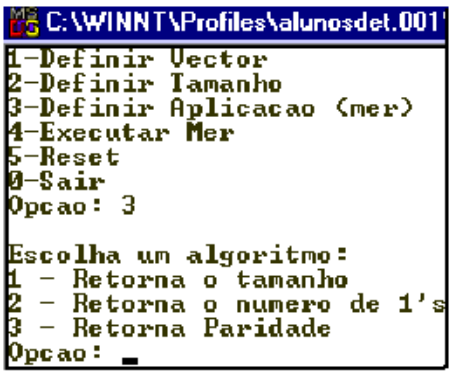

Fig. 9 – Interface do software do utilizador

Ao utilizador é apresentado um menu onde ele(a) pode escolher qualquer uma das opções, tais como: inserir o vector; inserir o tamanho do vector; gravar memórias (se for escolhida esta opção aparecerá um menu com as várias operações disponíveis para executar sobre o vector, por exemplo, contar número de uns); executar MEFR, após a qual retorna o resultado da operação efectuada sobre o vector.

No que diz respeito a comunicação entre o computador PC e a placa (FPGA), esta é efectuada pela porta paralela e respeitando o protocolo definido (ver parte A. Interface, parte D. Reconfiguração).

O projecto que inclui todas as partes consideradas a cima está apresentado na WebCT [5].

VI. IMPLEMENTAÇÃO DE OPERAÇÕES SOBRE VECTORES BOOLEANOS

As operações que se podem efectuar sobre um vector são inúmeras. Alguns exemplos dessas operações são:

- conta número de uns;
- verifica se o número de uns do vector é par ou não.

Supondo que se pretenderia contar o número de uns de um vector, ter-se-ia de implementar um algoritmo que o fizesse, parte desse algoritmo está apresentado na fig. 10.

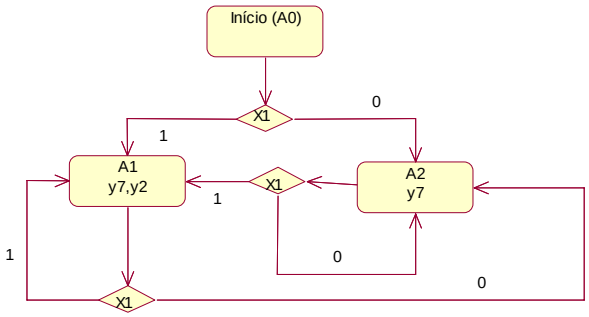

Fig. 10 – Parte do algoritmo que conta o número de uns

Este algoritmo (ver fig. 10) é bastante simples, o qual, de acordo com o valor de X1 (vector[índex]) pode saltar para o estado A1 caso X1=1 activando assim as linhas de entrada da unidade de execução (y2,y7) que para a unidade de execução em causa vai incrementar o resultado de uma unidade (y2), e vai incrementar o índex da posição do vector (y7). Caso o X1 = 0 , salta para o estado A2, a onde este apenas incrementa o índex da posição do vector (y7).

VII. CONCLUSÕES

Com este trabalho conseguiu-se verificar as potencialidades do software utilizado (Xilinx Foundation [2]) bem como da linguagem VHDL. Também mostrou-se como é possível reutilizar o hardware e a facilidade da interligação entre o software e o hardware.

Para o desenvolvimento deste projecto foi importante os conhecimentos adquiridos em outras disciplinas em anos anteriores como Arquitectura de Computadores, Sistemas Digitais, Paradigmas de Programação I [4].

REFERÊNCIAS

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

[2] Xilinx, The programmable logic data book, Xilinx, 2000.

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

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

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

[6] 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.

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