Implementação de Algoritmo de Compressão e Descompressão de Dados para Modelo de Co-processamento baseado em FPGA's

Bruno Pimentel, Joel Arrais

***Resumo* – O artigo descreve resultados obtidos no âmbito de um projecto final do curso de LECT (Licenciatura em Engenharia de Computadores e Telemática). O projecto consiste na concepção e optimização de co-processadores baseados em FPGA's para a resolução de tarefas computacionalmente intensivas (tais como, resolução de problemas combinatórios, utilizando matrizes discretas).**

Têm já sido feitos estudos para abordar o problema da transferência de grandes quantidades de dados para um co-processador baseado numa FPGA e em receber os resultados, podendo estes ser armazenados de forma bastante compacta. Neste projecto, como forma de reduzir o tempo gasto na comunicação, foi implementada uma técnica de compressão e descompressão de dados, através da qual, um computador anfitrião pode comprimir dados e transferi-los para um co-processador baseado numa FPGA. Uma vez no co-processador, a informação pode ser manipulada directamente ou após ser descomprimida.

O artigo descreve os algoritmos de compressão e descompressão baseados no método de Huffman, ligeiramente alterado, a sua implementação na linguagem C++ e os circuitos baseados em FPGA's que suportam a necessária descompressão. Todo o *hardware* foi concebido através da especificação na linguagem Handel-C e implementado na FPGA *XC2V1000-4* da família Virtex-II, a qual integra a placa protótipo RC200 da Celoxica.

***Abstract* – The paper describes some results obtained within a final year project for LECT (Licenciatura em Engenharia de Computadores e Telemática). The project is devoted to the design and optimization of FPGA-based co-processors for data extensive applications (such as solving combinatorial problems over discrete matrices). One particular problem that has been studied in detail is transferring high volume of data to an FPGA-based co-processor and receiving the results from the co-processor, which can be coded very compactly. In order to reduce communication time, a technique of data compression/decompression has been employed in such a way that a host computer compresses data and transfers them to an FPGA-based co-processor. The latter either decompresses them or handles the compressed records. The paper describes compression/decompression algorithms based on the slightly modified Huffman method, their implementation in C++ language and FPGA-based circuits that provide for the relevant decompression. All hardware has been designed from specification in Handel-C and implemented in FPGA *XC2V1000-4* of Virtex-II family available on a RC200 prototyping board of Celoxica.**

I. INTRODUÇÃO

Na realização de tarefas de grande complexidade, a utilização de co-processadores com características mais adequadas tem como objectivo reduzir o tempo da sua realização. Assim, para tirar partido da eficiência que o *hardware* permite, os computadores de uso geral podem ser ligados a plataformas de características reconfiguráveis, como as que se baseiam em FPGA's (*Field Programmable Gate Array*). Perante tarefas que implicam grandes quantidades de dados, a largura de banda do canal de transferência pode ser um factor limitativo da taxa de realização das tarefas. Para não comprometer a utilidade do co-processamento, a comunicação pode tornar-se mais rápida através da compressão dos dados, num modelo como o ilustrado na fig. 1.

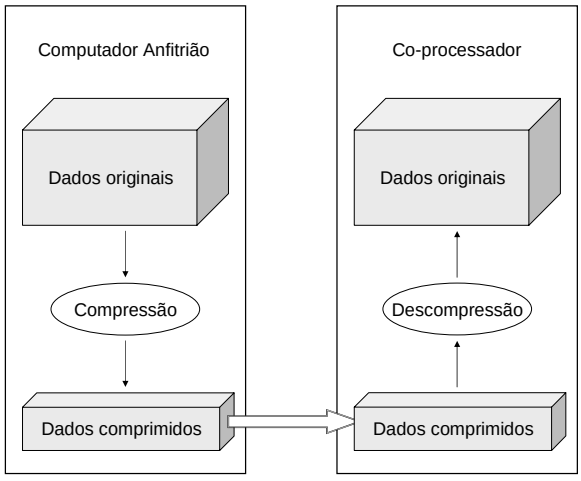

Fig. 1 – Modelo de co-processamento

Depois de os dados comprimidos se encontrarem no co-processador, existe a possibilidade de operar directamente sobre eles ou de os descomprimir previamente. No presente trabalho, foi abordada a segunda alternativa e, como co-processador, foi utilizada a placa RC200 [1,2] da Celoxica.

A. Objectivos

O trabalho desenvolvido visa, fundamentalmente, os seguintes objectivos:

- Seleccionar o algoritmo de compressão e descompressão a implementar;
- Implementar o algoritmo de compressão seleccionado em C++;
- Implementar um mecanismo de suporte à recursividade em *hardware* na linguagem Handel-C [3];
- Implementar o algoritmo de descompressão em Handel-C, recorrendo à recursividade.

Para além dos objectivos propostos, após a conclusão do compressor em C++, foi realizado um descompressor compatível na mesma linguagem, para testar a primeira aplicação.

Os fluxos de operações de alto nível que descrevem o funcionamento de algumas componentes desenvolvidas são ilustrados através de diagramas de actividades, de acordo com o especificado na UML (*Unified Modeling Language*) [4].

II. Compressão / Descompressão de dados

Na escolha do algoritmo a implementar [5,6], foram considerados três requisitos: ser eficiente, de modo a fornecer factores de compressão elevadas; ser simples, para comprimir e descomprimir em tempo reduzido; ser implementável em *hardware* com recursos limitados. Foi adoptada a versão adaptativa do algoritmo de compressão de Huffman [7], por verificar estes requisitos em medida considerável.

No sentido de melhorar o factor de compressão, este algoritmo foi objecto de algumas adaptações, nomeadamente a integração de um mecanismo que tira partido da redundância inerente às repetições consecutivas de determinadas sequências de *bits* do ficheiro original. O mecanismo consiste em detectar estes casos, indicar a sequência de *bits* em causa, uma única vez, e indicar o número de repetições. Assim, na contagem das ocorrências, apenas são consideradas as ocorrências não consecutivas. Em seguida, é descrito o funcionamento da aplicação desenvolvida, ilustrando todos os passos envolvidos na criação dos ficheiros comprimidos.

Considerando o exemplo do ficheiro na fig. 2-a, o primeiro passo consiste em determinar o número de ocorrências e o tamanho da sequência mais longa de repetições consecutivas de cada caracter, criando a tabela de frequência presente na fig. 2-b. Seguidamente, para cada caracter, é criado um nó com o respectivo número de ocorrências. Recorrendo ao algoritmo de ordenação considerado em [8], é gerada uma árvore (fig. 3-c) cuja organização exprime a ordenação dos nós por número de ocorrências: ramo esquerdo para valores menores ou iguais; ramo direito para valores maiores. Esta árvore, designada de árvore de ordenação, permite obter a árvore

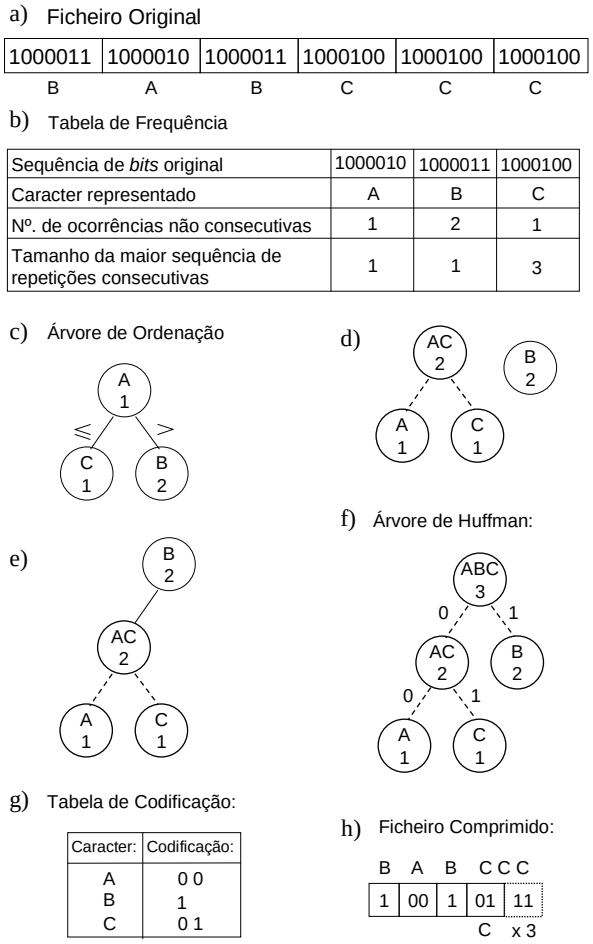

Fig. 2 – Exemplo de aplicação do algoritmo de compressão

de Huffman, através de algumas transformações cuja descrição se segue.

Nas figs. 2-c, 2-d, 2-e e 2-f, as ligações que representam os ramos das árvores de ordenação e de Huffman estão representadas por linhas contínuas e descontínuas, respectivamente. O processo é cíclico e o primeiro passo consiste em retirar da árvore de ordenação os dois nós com menor valor. Estes nós são agregados num novo nó, ao qual se atribui o valor da soma dos números de ocorrências dos nós anteriores, obtendo um ramo na árvore de Huffman (figs. 2-c e 2-d). O novo nó é inserido na árvore de ordenação (fig. 2-e) e todo este conjunto de operações é repetido até que existam apenas ligações da árvore de Huffman (fig. 2-f). Este método de transformação é descrito com maior detalhe e acompanhado de um exemplo mais complexo, por V. Sklyarov, no artigo "*FPGA-based implementation of recursive algorithms*" [9].

Fazendo uso dos valores 0 e 1, para distinguir o ramos da esquerda e da direita em cada nó da árvore de Huffman, é possível construir a tabela de codificação dos caracteres apresentada na fig. 2-g. Para tal, é extraída a sequência de valores que se obtém no caminho desde a raiz até à folha de cada caracter. Como exemplo, a codificação *01* para o

caracter *C* justifica-se pelo facto de a folha correspondente se encontrar no ramo direito do nó *AC* que, por sua vez, pertence ao ramo esquerdo do nó *ABC*.

Finalmente, recorrendo à tabela de codificação, é possível realizar a tradução dos caracteres originais pelos códigos correspondentes.

Na descompressão, é realizada a tradução no sentido inverso, utilizando, novamente, a tabela de codificação. Em caso de sequência de repetições consecutivas, é lido o tamanho da mesma, de modo a poder reconstitui-la.

III. IMPLEMENTAÇÃO DE COMPRESSÃO / DESCOMPRESSÃO EM SOFTWARE

Até aqui, foi considerado o caracter como elemento básico da compressão, cujo tamanho é oito *bits*. No entanto, foi introduzido o conceito de símbolo, enquanto elemento básico de compressão de tamanho não pré-definido, possibilitando uma eventual optimização do factor de compressão.

A versão adaptativa realiza uma optimização ao algoritmo de Huffman original, através da definição da codificação individual dos símbolos para cada ficheiro a comprimir. Uma vez que esta codificação é necessária ao descomprimir, definiu-se que a árvore de Huffman gerada na compressão seria incluída no início do ficheiro, ficando toda a informação necessária à descompressão a ele confinada. Sendo assim, antes de iniciar a implementação do algoritmo, especificou-se o formato a utilizar no armazenamento da árvore, assegurando a compatibilidade entre as componentes.

Para especificar a estrutura de armazenamento da árvore de Huffman no ficheiro comprimido, foi utilizada a norma BNF (Backus Naur Form) [10]. Na fig. 3 é apresentada uma versão simplificada da especificação definida, tendo alguns parâmetros sido concretizados.

Segundo esta especificação, a árvore de Huffman é constituída por um ramo, podendo cada ramo conter uma folha ou um par de ramos. As folhas são constituídas por um símbolo e pelo número de *bits* que vão indicar o número máximo de repetições consecutivas. Neste exemplo, foram atribuídos quatro *bits* para o tamanho máximo de repetições consecutivas e oito para o de símbolo.

<Bit>::=0|1

<Simbolo>::=<Bit><Bit><Bit><Bit><Bit><Bit><Bit><Bit>

<TamanhoRepeticao>::=<Bit><Bit><Bit><Bit>

<Folha>::=<TamanhoRepetição><Simbolo>

<Ramo>::=0<Folha>|1<Ramo><Ramo>

<ArvoreHuffman>::=<Ramo>

Fig. 3 – Estrutura de armazenamento da árvore de Huffman

A. Compressor

O processo de compressão é constituído pelas seguintes fases: definição da codificação a utilizar através da análise do ficheiro original, exportação da árvore de Huffman e tradução efectiva do ficheiro.

Como referido, a codificação atribuída a cada símbolo do ficheiro original baseia-se no número de vezes que nele ocorre. Assim, é feita uma primeira leitura para o levantamento desta informação e para verificar o maior número de ocorrências consecutivas (NOC) de cada símbolo.

Os NOC's máximos vão ser utilizados na determinação do número de *bits* necessário para armazenar o tamanho das repetições consecutivas de cada símbolo, que foram detectadas nos dados comprimidos. O diagrama de actividades que consta da fig. 4 descreve o funcionamento desta análise.

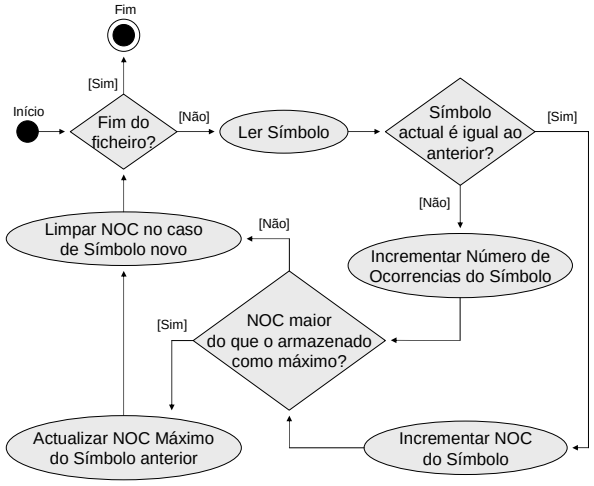

Fig. 4 – Diagrama de actividades do processo de análise do ficheiro original

Iterativamente até ao fim do ficheiro, é lido um símbolo que é comparado com o anterior. Se estes símbolos forem iguais, o NOC actual (variável auxiliar) é incrementado; se se tratar de um novo símbolo, incrementa-se o número de ocorrências deste.

De seguida, o NOC actual é comparado com o NOC máximo do símbolo em causa (armazenado na tabela de frequência). No caso de o primeiro ser superior, o segundo é actualizado.

No passo seguinte, o NOC actual é inicializado, no caso de se tratar de um símbolo novo, uma vez que se trata do começo de uma nova sequência.

A informação obtida da anterior análise do ficheiro de entrada é utilizada para a criação da árvore de ordenação. Para tal, cada símbolo com número de ocorrências positivo dá origem a um nó que é inserido na árvore de ordenação [8]. Como ilustrado nas figs. 2c-f, esta árvore é convertida na árvore de Huffman, a partir da qual se extrai a codificação de cada símbolo, populando a tabela de codificação.

A exportação da árvore de Huffman consiste na invocação de uma função recursiva, reinvocando-se para exportar cada um dos ramos até às folhas, e verificando o formato que foi previamente especificado (fig. 3).

Por último, é realizada a tradução dos símbolos do ficheiro original para a codificação definida, através de uma segunda leitura. O diagrama de actividades que consta da fig. 5 descreve os passos envolvidos neste processo. Cada símbolo lido é comparado com o anterior para verificar se se trata da mesma sequência de repetição. Em caso afirmativo, o NOC é incrementado e passa-se ao símbolo seguinte. Ao detectar-se uma nova sequência, são exportados a codificação do símbolo anterior e o NOC obtido, sendo ambos actualizados para a sequência seguinte.

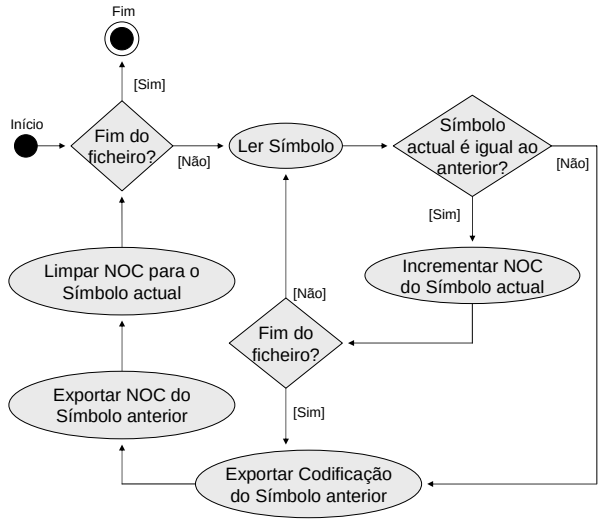

Fig. 5 - Diagrama de actividades do processo de tradução na compressão

B. Descompressor

O processo de descompressão é constituído pela importação da árvore de Huffman e pela tradução dos códigos nos símbolos do ficheiro original.

A importação da árvore de Huffman consiste em interpretar a sequência binária que precede os dados comprimidos, no ficheiro. Para tal, é usada uma função recursiva, que se reinvoca para a importação de cada um dos ramos, até às folhas, tendo em conta o formato especificado para o efeito (rever fig. 3).

A tabela de codificação, obtida com a interpretação da árvore de Huffman, é utilizada na reconstituição dos símbolos originais. Este processo, ilustrado na fig. 6, inicia-se com a leitura de um código que corresponde a um símbolo. Se esse símbolo envolver NOC, é lido o seu valor do ficheiro comprimido; caso contrário, assume-se que é unitário.

Por último, o símbolo em causa é repetido no ficheiro de saída o número de vezes indicado pelo valor de NOC. O ciclo repete-se até que o ficheiro comprimido termine.

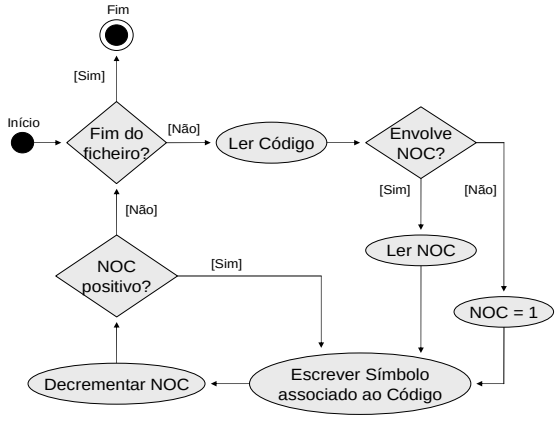

Fig. 6 - Diagrama de actividades do processo de tradução na descompressão

IV. IMPLEMENTAÇÃO DE DESCOMPRESSÃO EM HARDWARE

Esta etapa consiste na implementação do algoritmo de descompressão apresentado anteriormente para a placa RC200, a qual possui a FPGA Virtex II *XC2V1000-4*. Para isso, desenvolveu-se um algoritmo em Handel-C [3] que realiza uma descompressão idêntica à que foi feita em C++. Porém, os algoritmos criados numa e noutra linguagem revelam diferenças significativas. Estas diferenças assentam no facto de se passar de um paradigma orientado aos objectos para um paradigma funcional e de especificação de *hardware*. A possibilidade de execução de operações em paralelo permitiu optimizar o processo de descompressão, aumentando a velocidade de execução. No entanto, a diferença mais significativa entre as duas implementações proveio da utilização do modelo RHFSM (*Recursive Hierarquichal Finite State Machine*) [11]. Com este modelo, ultrapassou-se a ausência de mecanismos de suporte à recursividade das linguagens de especificação de *hardware*. O capítulo V aborda a implementação deste modelo.

O funcionamento global do circuito descompressor está ilustrado na fig. 7. O cabeçalho do ficheiro comprimido (fig. 7-a) é lido através do Gestor de Leitura, para o *parser* que interpreta a árvore de Huffman e reconstrói a Tabela de Codificação (figs. 7-a, 7-c, 7-d e 7-e). A segunda fase da descompressão consiste em ler os dados comprimidos através do Gestor de Leitura e traduzi-los nos caracteres originais, recorrendo à Tabela de Codificação (figs. 7-b, 7-c e 7-f). Os caracteres obtidos são escritos, reconstituindo o ficheiro original, através do Gestor de Escrita (figs. 7-h e 7-g).

V. IMPLEMENTAÇÃO DO MODELO RHFSM

Tal como referido anteriormente, a linguagem Handel-C não disponibiliza mecanismos que suportem recursividade. Assim, o *parser* do descompressor

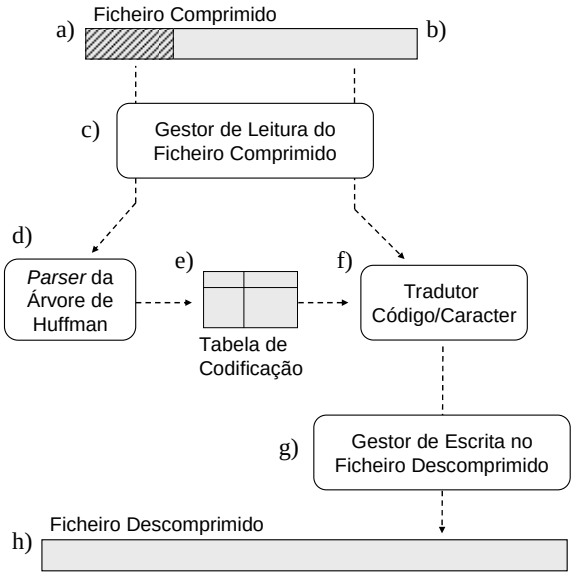

Fig. 7 – Circuito descompressor em Handel-C

desenvolvido em Handel-C foi implementado, recorrendo a um método baseado no modelo RHFSM [11].

A. Biblioteca de suporte ao modelo RHFSM

O modelo RHFSM baseia-se na concepção dos módulos recursivos do algoritmo na forma de máquinas de estados finitos, nas quais se atribui a execução de um subconjunto atómico de instruções a cada estado. A sua utilização implica a fusão do código útil com as funções que gerem os dados de suporte às transições.

Para permitir alguma abstracção no que diz respeito a esta gestão, foi criada uma biblioteca simples e reutilizável, denominada RHFSM, que inclui sete primitivas em Handel-C de suporte às transições entre estados e módulos:

- *RHFSM_init( )*, que inicia os dados internos para se estar apto a usar a RHFSM;
- *RHFSM_go_to(next_state)*, que permite transitar para o estado *next_state* do módulo activo;
- *RHFSM_jump_to(next_module, next_state)*, que permite transitar para o estado *next_state* do módulo *next_module*;
- *RHFSM_end_module( )*, que permite a saída do módulo activo, regressando ao módulo anterior ou terminando a RHFSM;
- *RHFSM_state( )*, que devolve o estado activo;
- *RHFSM_module( )*, que devolve o módulo activo;
- *RHFSM_done( )*, que indica se a RHFSM terminou.

B. Concepção da RHFSM do parser

Estando o suporte à recursividade definido, o passo seguinte consistiu em modelar o *parser* na forma de uma máquina de estados finitos hierárquica. Como foi referido, a finalidade do *parser* é interpretar a árvore de Huffman que se encontra no início do ficheiro, armazenada de acordo com o definido em código BNF (fig. 3).

Para uma análise pormenorizada da implementação do *parser*, o código Handel-C respectivo está disponível no *site* da *WebCT* [12]. No entanto, é oportuno abordar uma perspectiva global do seu funcionamento. Na fig. 8 é descrito o fluxo do *parser*, através dos estados e transições que compõem o módulo implementado. O processo é iniciado, invocando este módulo (fig. 8).

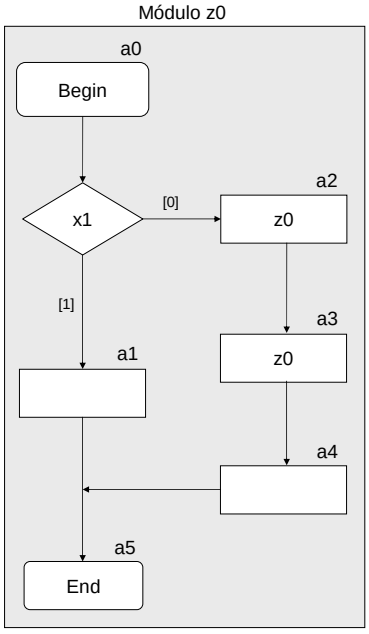

Fig. 8 - Módulo da RHFSM do *parser*

No estado *a0*, é lido um nó do ficheiro comprimido. Na condição *x1*, é verificado se o nó é uma folha ou um ramo.

No caso de ser folha, transita-se para o estado *a1*, no qual se extrai o número de *bits* para o NOC e o símbolo que foram armazenados na folha. Seguidamente, transita-se para o estado terminal *a5*, após o qual, se retorna ao nível hierárquico superior.

Se na condição *x1*, por outro lado, se verificar que o nó lido representa um ramo, transita-se para o estado *a2*. Neste estado, é reinvocado o módulo *z0*, descendo-se de nível hierárquico, para realizar o *parsing* do ramo esquerdo do nó.

Após regressar desta sub-tarefa, retoma-se o nível hierárquico anterior e transita-se para o estado *a3*. Neste estado, repete-se a invocação do módulo *z0*, para realizar o *parsing* do ramo direito.

Ao concluir-se a importação do segundo ramo deste nó, regressa-se ao estado *a4*, realizando algumas operações auxiliares, e transita-se para o estado terminal *a5*.

CONCLUSÕES

O recurso a co-processadores baseados em FPGA's constitui uma alternativa relevante na redução do tempo de execução de tarefas computacionalmente intensivas. Quando o volume de dados envolvido nestas tarefas é grande, é importante que a comunicação entre o computador anfitrião e o co-processador não atrase o processo. Neste contexto, a compressão de dados surge como uma forma de optimizar a largura de banda do canal de transmissão.

Foi desenvolvida em C++ uma aplicação de compressão com vista a correr no computador de uso geral anfitrião. Para testar esta aplicação, foi implementado também um descompressor, utilizando a mesma linguagem.

Como co-processador a utilizar no âmbito do projecto, foi adoptada a placa RC200, que integra a FPGA Virtex II *XC2V1000-4*. O Handel-C foi a linguagem utilizada para especificar o circuito descompressor a ser utilizado por este co-processador.

Escolheu-se a versão adaptativa do algoritmo de compressão e descompressão de Huffman, por apresentar factores de compressão elevados e simplicidade e ainda porque a sua implementação em *hardware* não implica demasiados recursos.

Integrou-se um mecanismo adicional que consiste em substituir repetições consecutivas de sequências de *bits* por um único identificador de sequência e um indicador do número de repetições, resultando em factores de compressão superiores.

O processo de compressão completo inclui a análise dos dados a comprimir, a geração da codificação a utilizar, a exportação desta codificação para o ficheiro comprimido e a tradução efectiva dos dados para o mesmo destino.

A descompressão consiste em importar a codificação e em reconstituir os dados originais, através de uma tradução inversa à realizada na compressão.

Foi especificado em BNF o formato de armazenamento da árvore de Huffman, da qual se extrai a codificação utilizada. Esta especificação foi tida em conta na implementação das aplicações de *software* - compressor e descompressor - e do circuito descompressor para a placa RC200, assegurando, deste modo, a compatibilidade entre estes componentes.

Qualquer das implementações compreende operações que envolvem árvores binárias, pelo que foram concebidas com cariz recursivo. Por não haver linguagens de especificação de *hardware* com mecanismos de suporte à recursividade, foi adoptado o modelo RHFSM na implementação do circuito de *parsing* que extrai a árvore de Huffman do ficheiro comprimido.

Criou-se um conjunto de primitivas para manipular os dados que suportam as transições de estado das RHFSM's obtidas. Trata-se de uma biblioteca reutilizável que oferece um nível de abstracção mais elevado.

Futuramente, para colmatar o objectivo da implementação do modelo de co-processsamento adoptado, será desenvolvido um circuito para a resolução de problemas combinatórios, a correr na placa RC200, e um mecanismo de transferência através da porta paralela.

REFERÊNCIAS

[1] "RC200 Hardware and Instalation Manual", Celoxica, 2003.

[2] "RC200 Platform Support Library Reference Manual", Celoxica, 2003.

[3] "Handel-C Language Reference Manual", Celoxica, 2002.

[4] M. Fowler, K. Scott, "UML Distilled: A Brief Guide to the Standard Object Modeling Language", 2nd Edition, Addison Wesley, pp. 129-139.

[5] "Data compression information" [Online, em Janeiro de 2003]: http://datacompression.info.

[6] "Archive compression tests", [Online, em Janeiro de 2004]: http://compression.ca/.

[7] "Lossless Data Compression", [Online, em Janeiro de 2003]: http://www.data-compression.com/lossless.html#huff.

[8] Brian W. Kernighan, Dennis M. Ritchie, "The C Programming Language", Prentice Hall, 1988.

[9] V. Sklyarov, "FPGA-based Implementation of Recursive Algorithms", Elsevier Journal Microprocessors and Microsystems, Abril, 2004.

[10]. "BNF Notation", [Online, em Janeiro de 2003]: http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html.

[11] V. Sklyarov, "Hierarchical Finite-State Machines and Their Use for Digital Control", IEEE Transactions on VLSI Systems, 1999, Vol. 7, No 2, pp. 222-228.

[12] Web Course Tools para a disciplina de Computação Reconfigurável leccionada na Universidade de Aveiro, [Online]: http://webct.ua.pt.

AGRADECIMENTO

Os autores agradecem ao Prof. Valery Sklyarov pela orientação na realização deste trabalho.