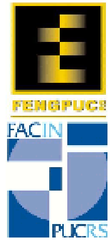
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE ENGENHARIA

FACULDADE DE INFORMÁTICA

CURSO DE ENGENHARIA DE COMPUTAÇÃO



ADAPTANDO SISTEMAS PARA OPERAR COM REDES INTRACHIP: DECODIFICADOR M-JPEG/HERMES

AUGUSTO LANGE

VINICIUS PESSIL BOHRER

VINÍCIUS SANTOS DA SILVA

TRABALHO DE CONCLUSÃO DO

CURSO DE ENGENHARIA DE COMPUTAÇÃO

ORIENTADOR: PROF. DR. NEY LAERT VILAR CALAZANS

Porto Alegre
2010

AUGUSTO LANGE

VINICIUS PESSIL BOHRER

VINÍCIUS SANTOS DA SILVA

ADAPTANDO SISTEMAS PARA OPERAR COM REDES INTRACHIP: DECODIFICADOR M-JPEG/HERMES

Trabalho de conclusão de curso de graduação apresentado nas Faculdades de Engenharia e de Informática da Pontifícia Universidade Católica do Rio Grande do Sul, como requisito parcial para obtenção do grau de Engenheiro de Computação.

ORIENTADOR: PROF. DR. NEY LAERT VILAR CALAZANS

Porto Alegre

2010

Dedicamos este trabalho a todos que nos deram força e motivação, não só na conclusão deste trabalho, mas durante toda a jornada acadêmica. Obrigado familiares, namoradas, amigos e Deus.

AGRADECIMENTOS

Eu, Augusto Lange, devo agradecer à minha família, que sempre estiveram presentes nos momentos importantes da minha vida, me auxiliando a traçar metas e alcançá-las. Agradeço a minha namorada e os meus amigos, pela compreensão devido as minhas ausências e prometo que serão compensados depois desta jornada. Agradeço também aos colegas de TCC que não só deram suporte nesta fase, mas certamente o farão a vida inteira.

Eu, Vinicius Pessil Bohrer, agradeço à minha família, que esteve sempre presente me indicando o melhor caminho a seguir. Agradeço, em especial, minha namorada, por me apoiar e ser muito paciente comigo nos momentos mais críticos deste trabalho. Não posso esquecer-me de agradecer meus grandes amigos pelos incentivos e pelos poucos momentos sociais que eu tive. Agradeço, principalmente, aos meus amigos, Augusto e Vinícius, pelas risadas essenciais ao longo deste trabalho.

Eu, Vinícius Santos da Silva, agradeço a dedicação e carinho da minha família, em especial minha mãe e minha avó, que estiveram sempre presentes na minha jornada. Agradeço também a minha namorada e aos amigos pelos incentivos e pela compreensão nos momentos de ausência. Por fim, e não menos importante, agradeço aos amigos, e não só colegas de TCC, que possibilitaram a realização deste trabalho.

O “grupo A2V” agradece a todos envolvidos na realização deste trabalho. Principalmente a Deus. Agradecemos aos avaliadores Fernando Moraes e César Marcon, pela disponibilidade de avaliar e corrigir o nosso trabalho, agradecimentos em especial ao orientador Ney Calazans, pelo apoio e orientação no desenvolvimento deste trabalho e também pela sua paciência em corrigir as deficiências do grupo. Somo gratos a todos os amigos que ganhamos na faculdade, nos juntando aos que já se formaram e desejando força para os que continuam na jornada de conclusão do curso.

Temos o destino que merecemos.
O nosso destino está de acordo com os
nossos méritos.

ALBERT EINSTEIN

RESUMO

Cada vez mais, projetos de sistemas embarcados necessitam de mais recursos integrados em equipamentos cada vez mais compactos, gerando a necessidade de empregar na sua construção sistemas integrados em um único *chip* (em inglês, os chamados *systems on a chip* ou SoCs). O aumento de recursos integrados dentro de um único chip decretou a necessidade de refinar os métodos de interconexão destes, visando garantir eficiência de interação entre recursos e com o meio externo, reduzir ou pelo menos controlar a energia dissipada na arquitetura de comunicação, entre outras características. Neste contexto desenvolveu-se uma nova área, a de redes intrachip (em inglês, *Networks on Chip* ou NoCs), uma alternativa bem sucedida para as limitações físicas que vinham comprometendo a escalabilidade e o desempenho de arquiteturas de comunicação internas em SoCs complexos. Tais redes são tipicamente compostas por roteadores e enlaces locais a uma região do *chip* que, de forma estruturada, provêem maior desempenho no processamento de aplicações paralelas e ainda ao menor consumo de energia.

O objetivo deste Trabalho de Conclusão é propor um esforço inicial de adaptação de um sistema convencional, neste caso, um decodificador M-JPEG para operar no contexto de uma arquitetura de SoC que empregue uma NoC como arquitetura de comunicação intrachip. Este sistema respeita os requisitos do decodificador original, incluindo o requisito crítico deste componente, a vazão de dados especificada. A NoC HERMES, implementada pelo GAPH (Grupo de Apoio ao Projeto de Hardware) da PUCRS, foi escolhida para este fim. Ao contrário de outros projetos, este não visa ganhos de desempenho, mas sim analisar se o comportamento da NoC HERMES é adequado para dar suporte aos requisitos do módulo M-JPEG.

O decodificador M-JPEG foi aproveitado do projeto disponível no *Opencores*. Ele é originalmente descrito na linguagem VHDL e sua distribuição original foi prototipada em um FPGA Virtex-II Pro com sucesso, usando os ambientes EDK e ISE da Xilinx. A frequência de operação é 100 MHz. O módulo M-JPEG também utiliza uma memória externa de 256 MBytes para armazenar o vídeo que o decodificador processa. O decodificador é formado por núcleos IPs, incluindo os componentes do processo de decodificação de vídeo e/ou imagens e também uma lógica de controle de busca dados da imagem (através de um processador embarcado PowerPC do FPGA Virtex-II Pro) e a lógica de envio de dados para uma interface VGA. Estes IPs estão conectados segundo uma estrutura *pipeline* que utiliza um controle de fluxo do tipo *handshake* entre cada par de interfaces. A NoC HERMES é uma rede intrachip simples, formada por roteadores e enlaces físicos bidirecionais entre estes. Esta NoC usa controle de fluxo do tipo *handshake* baseado em créditos, emprega modo de chaveamento *wormhole*, transmite pacotes divididos em *flits* e seus roteadores contêm uma lógica de controle compartilhada por um conjunto de até cinco portas de entrada e saída. Sua utilização é facilitada pelo uso do ambiente parametrizável de geração automática de NoCs denominado ATLAS, também de autoria do GAPH e disponibilizado como código aberto no site do grupo de pesquisa.

No processo de integração proposto e realizado neste trabalho, desenvolveu-se interfaces que fazem conexão entre IPs e roteadores da NoC, que interpretam os protocolos de ambos os lados (NoC e IP). O fato de o decodificador ter estrutura *pipeline* facilitou a implementação das interfaces. O desafio principal consistiu em sincronizar os diferentes controles de fluxo envolvidos.

Os resultados desta nova arquitetura são analisados através de simulações na ferramenta Modelsim. Comparam-se os tempos de execução e verifica-se a correta funcionalidade, quando comparado ao projeto original do decodificador.

Palavras-chave: Decodificação, M-JPEG, SoCs, NoCs, HERMES, FPGA.

ABSTRACT

The design of embedded systems requires an increasing number of integrated resources inside equipments increasingly more compact, creating the need to employ in their construction systems on a chip (SoCs). The increase in the number of integrated resources establishes the need to refine the intrachip interconnection methods, in order to ensure an efficient interaction between internal resources and the external environment, reduce, or at least control the energy dissipated in the communication architecture, among other features. In this context appeared the new field of networks on chip (NoCs). NoCs were developed to provide a successful alternative to the physical limitations that compromise the scalability and performance of complex SoCs. These networks are typically composed of routers and links local to a region of the chip that, in a structured form, provide a higher performance for parallel applications and lower power consumption.

The objective of this end of term work is to propose an initial effort to adapt a conventional system, in this case, an M-JPEG decoder, to operate in the context of a SoC architecture that employs a NoC as on chip communication architecture. This system respects the requirements of the original decoder, including the critical part of this component, i.e. the specified data flow figures. The HERMES NoC, implemented by the GAPH (Grupo de Apoio ao Projeto de Hardware) PUCRS, was elected as the NoC to employ. Unlike other projects, this one is not intended to achieve performance gains, but just to analyze if the behavior of the HERMES NoC is adequate to support the requirements of a module such as the M-JPEG.

The M-JPEG decoder originated from an OpenCores project. It is originally described in the VHDL language and the original distribution has been successfully prototyped in a Virtex-II Pro Xilinx FPGA, using Xilinx EDK and ISE design environments. The operating frequency is 100 MHz. The M-JPEG module also uses an external memory of 256 MBytes to store the video file to decode. The decoder consists in a set of IP cores, including the components of the decoding process of video and/or pictures and also a control logic fetches data from the image (via an embedded PowerPC processor inside the FPGA) and the logic of sending data to a VGA interface. These IPs are interconnected using a pipeline structure that uses a handshake flow control between each pair of interfaces.

The HERMES NoC is a simple network on chip, composed of routers and bidirectional physical links between these. This NoC uses credit based flow control, employs a wormhole switching mode and transmits packets divided into flits. Its routers contain a control logic shared by a set of up to five bidirectional ports. The utilization of HERMES is facilitated by the use of the ATLAS parameterizable environment for automatic generation of NoCs. This tool is also authored by the GAPH and is available as open source at the site of the group.

In the proposed integration process carried out in this work, a set of dedicated interfaces that provide the connection between IPs and the HERMES routers. These interfaces interpret the protocols on both of its sides (NoC and IP). The fact that the decoder has a pipeline structure facilitated the implementation of the interfaces. The main challenge was to synchronize the different flows of control involved.

The results of this implementation were analyzed by RTL simulation using the Modelsim tool. A preliminary evaluation of the comparison of execution time verified the correct functionality, when compared to the original project of the decoder.

Key-Words: Decoding, M-JPEG, SoCs, NoCs, HERMES, FPGA

LISTA DE ILUSTRAÇÕES

Figura 2.1 - Os passos essenciais na codificação JPEG. 19
Figura 2.2 - Blocos de uma imagem JPEG. 20
Figura 2.3 - Tipos de amostragem. 20
Figura 2.4 - *Stream* de blocos 8x8 de componentes Y, Cb e Cr. 21
Figura 2.5 - Estrutura dos componentes utilizados no processo de codificação....... 22
Figura 2.6 - Diagrama Simplificado de um decodificador baseado em DCT............ 25
Figura 2.7 - Decodificação de entropia em um *stream* de bits. 26
Figura 2.8 - Exemplo de arquitetura NoC e comunicação com os IPs do sistema. .. 29
Figura 2.9 - Exemplo de interface física entre roteadores. 32
Figura 2.10 - Arquitetura do roteador HERMES. B indica os *buffers* de entrada. 33
Figura 2.11 - Roteador HERMES com dois canais virtuais por canal físico............. 33
Figura 2.12 - Tabela de roteamento. a) chaveamento; b) mudança da tabela. 34
Figura 2.13 - Estrutura da topologia Malha 2D 3x3. 35
Figura 2.14 - Janela inicial do ambiente ATLAS. 36
Figura 2.15 - Janela principal da ferramenta de geração da NoC HERMES. 36
Figura 2.16 - Interface entre roteadores com o (a) *Handshake* e (b) *Credit Based*. . 37
Figura 2.17 - Janela principal da ferramenta *Traffic Generation*. 38
Figura 2.18 - Configuração padrão para tráfego (a)uniforme; (b)normal; (c)pareto. . 39
Figura 2.19 - Janela principal para controlar a ferramenta NoC Simulation............. 39
Figura 2.20 - Interface para a seleção do cenário de tráfego a ser analisado. 40
Figura 2.21 - Interface principal da ferramenta *Traffic Measurer*. 40
Figura 2.22 - Relatório da análise dos canais (enlaces) da rede. 41
Figura 2.23 - Relatório de pacotes ordenados por valores de latência. 41
Figura 2.24 - Relatório global. 42
Figura 3.1 - Diagrama de blocos da arquitetura Decodificador-Controle. 44
Figura 3.2 - Estrutura do MyIPIF. 45
Figura 3.3 - Máquina de estados finita para iniciar um ciclo de leitura. 46
Figura 3.4 – Estrutura genérica dos sinais de controle de fluxo para os componentes do *pipeline* JPEG. 47
Figura 3.5 - Forma de onda típica para os sinais de *handshake*. 47
Figura 3.6 - Componentes e conexões básicas de controle da taxa de quadros...... 48

Figura 3.7 - Estrutura e sinais do componente *Input Buffer*..................................... 49
Figura 3.8 - Máquina de estados finita utilizada na decodificação de entropia (simplificada)... 50
Figura 3.9 - Realocação dos dados em registradores de deslocamento. 52
Figura 3.10 - Controle de fluxo e *pipeline* originais do componente IDCT. 53
Figura 3.11 - Controle de fluxo e *pipeline* do componente IDCT modificados. 53
Figura 4.1 - Modelo implementado na primeira fase de integração JPEG-HERMES.57
Figura 4.2 - Modelo a ser implementado na segunda fase de integração JPEG-HERMES. 58
Figura 4.3 - Fluxo básico de atividades das interfaces. 60
Figura 4.4 - Diagrama de blocos da interface entre *UpSampling* e a NoC. 61
Figura 4.5 - Máquina de estados da interface *UpSampling*. 62
Figura 4.6 - Lógica do *ready_o* para o controle de fluxo com o *UpSampling*............ 63
Figura 4.7 - Diagrama de blocos da interface entre *YCbCr2RGB* e a NoC. 64
Figura 4.8 - Máquina de estados da interface *YCbCr2RGB*. 65
Figura 5.1 - Prototipação inicial de um vídeo em M-JPEG. 67
Figura 5.2 - A imagem Lenna.jpg, usada para validar a decodificação. 68
Figura 5.3 - Início dos *logs* de decodificação.. 69
Figura 5.4 - Comparação do início de uma transação para o decodificador original e com NoC. 69
Figura 5.5 - Tempo de Envio de cada dado.. 70
Figura 5.6 - Comparação entre duas Transações. 70
Figura 5.7 - Envio de dados sentido Interface-Roteador... 71
Figura 5.8 - Interfaces e roteadores.. 72
Figura 5.9 - Final do arquivo de *log* gerado pelo *testbench*...................................... 72
Figura 5.10 - Formas de onda indicando o final da decodificação. 73

LISTA DE TABELAS

Tabela I - Características dos processos sugeridos pela norma JPEG [JPE92]....... 18
Tabela II - Exemplo de Tabela de *Huffman* DC. 24
Tabela III - Exemplo de Tabela de *Huffman* AC. 24

LISTA DE SIGLAS

ASIC	Application-Specific Integrated Circuit
AVI	Audio Video Interleaved
DCT	Discrete Cosine Transform
DDR	Double Data Rate
DFT	Discrete Fourier Transform
DMA	Direct Memory Access
DSP	Digital Signal Processing
EDK	Embedded Development Kit
EOI	End Of Image
EOI	End Of Block
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GAPH	Grupo de Apoio ao Projeto de Hardware
GIF	Graphics Interchange Format
HeMPS	HERMES Multiprocessor System
HOL	Head of Line
IBM	International Business Machines
IDCT	Inverse Discrete Cosine Transform
IP	Intellectual Property
ISE	Integrated Software Environment
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union
JFIF	JPEG File Interchange Format
JPEG	Joint Photographic Experts Group
LUT	Lookup Table
MPSoC	Multiprocessor System on Chip
M-JPEG	Motion JPEG
MOS	Metal Oxide Semiconductor
NoC	Network on Chip
OPB	On-Chip Peripheral Bus
OSI	Open Systems Interconnection

PLB	Processor Local Bus
PowerPC	Power Optimization with Enhanced RISC – Performance Computing
QoS	Quality of Service
RGB	Red (R), Green (G) and Blue (B) Color System
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System on a Chip
SOI	Start of Image
TCL	Tool Command Language
TDM	Time-Division Multiplexing
UART	Universal Asynchronous Receiver/Transmitter
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
XMD	Xilinx Microprocessor Debug
XUP	Xilinx University Program
XUPV2P	Xilinx University Program Virtex®-II Pro
YCbCr	Luminance (Y) and Chrominance (CbCr) Color System
ZRL	Zero Run Length

SUMÁRIO

1 INTRODUÇÃO 14
2 REFERENCIAL TEÓRICO 16
2.1 M-JPEG 16
2.1.1 Formatos dos Contêineres 16
2.1.2 JPEG 17
2.2 SYSTEM ON A CHIP 27
2.3 NETWORKS ON CHIP 28
2.3.1 Arquiteturas de NoCs 29
2.3.2 Topologias 30
2.3.3 Protocolos 31
2.3.4 A NoC HERMES 32
2.3.5 O Ambiente ATLAS 35
2.4 PLATAFORMAS DE PROTOTIPAÇÃO DE HARDWARE 42
2.4.1 A Plataforma XUP-V2PRO 43
3 DECODIFICADOR M-JPEG 44
3.1 COMPONENTES (MÓDULOS) 44
3.1.1 MyIPIF 45
3.1.2 JPEG Decoder 48
3.1.3 VGA 55
4 ARQUITETURA DO DECODIFICADOR M-JPEG/HERMES 57
4.1 DESCRIÇÃO 57
4.2 GERAÇÃO DA NOC 58
4.2.1 Especificação 59
4.2.2 Interfaces 59
5 VALIDAÇÃO 66
5.1 PROTOTIPAÇÃO INICIAL 66
5.2 SIMULAÇÃO COM A NOC 68
5.3 DIFICULDADES ENCONTRADAS 73
6 CONSIDERAÇÕES FINAIS 75
7 REFERÊNCIAS BIBLIOGRÁFICAS 76

1 INTRODUÇÃO

O mercado tecnológico amadureceu, e atualmente as pessoas exigem qualidade e desempenho com um custo cada vez menor. Para atender a esta demanda, o mundo é surpreendido com tecnologias inovadoras onde produtos eletrônicos menores, com mais funções e mais baratos são criados. Os codificadores e decodificadores (codecs) de vídeo acompanham esta evolução. Codificadores são utilizados para transformar sinais analógicos do ambiente (normalmente imagem e som) em sinais digitais, fazendo uso de um padrão, visando uma melhor qualidade e/ou menor tamanho de armazenamento. Os decodificadores, por sua vez, realizam o processo inverso do codificador, utilizando métodos similares para reconstruir a informação original. Tais sistemas favorecem desde produtoras cinematográficas a usuários domésticos, que hoje podem, com uma máquina fotográfica ou celular, armazenar informação e utilizá-la repetidas vezes em seu computador pessoal, televisor ou mesmo dispositivo portátil.

Os avanços da tecnologia *Very Large-Scale Integration* (VLSI) propiciaram o desenvolvimento do processo de criação de circuitos integrados, culminando na combinação de milhões de transistores inseridos em um único *chip*, incluindo processadores e blocos de memórias [CAR09b]. Esses sistemas recebem o nome de *System-on-Chip* (SoC). Um dos grandes desafios do processo de projeto de SoCs é conseguir implementar uma estrutura de alocações de tarefas que consiga utilizar toda a capacidade de processamento do mesmo. Neste contexto, surgem as Redes Intrachip (*Networks on Chip* - NoCs), arquitetura de comunicação construída no interior de um circuito integrado, compostas por roteadores e outros elementos de rede capazes de prover um alto desempenho no processamento de aplicações paralelas de diversos tipos e ainda um menor consumo de energia.

O ambiente ATLAS [GAP01], proposto pelo Grupo de Apoio ao Projeto de Hardware (GAPH), automatiza vários processos relacionados ao fluxo de projeto de algumas das NoCs propostas pelo grupo, dentre elas a NoC HERMES, que é uma infraestrutura utilizada para a geração de NoCs para diferentes topologias, tamanhos de *flit*, profundidade do *buffer* e algoritmos de roteamento.

O objetivo primário deste trabalho foi contribuir para o desenvolvimento de uma aplicação demonstrativa da NoC HERMES, baseada em um decodificador de vídeo JPEG (M-JPEG) [MAN08]. Foi possível avaliar alguns dos compromissos de implementação de uma arquitetura de comunicação para interligar um conjunto de módulos de hardware, que neste caso são partes do processo de decodificação de vídeo. Os resultados foram analisados através de simulação com a ferramenta Modelsim. O desempenho e a funcionalidade do sistema foram demonstrados.

O aprendizado ao longo do curso de Engenharia de Computação raramente envolveu lidar com aplicações completas e complexas como esta, tornando este trabalho uma ótima oportunidade para fixar os conceitos aprendidos durante todo o curso, pois incluem desenvolvimentos em *hardware* e *software*, numerosos conceitos de arquitetura de processadores, sistemas embarcados, processamento digital de sinais, microeletrônica e principalmente a prototipação de um projeto deste porte em FPGA.

Percebe-se que estas áreas são de importância acadêmica e profissional, agregando conhecimento aos integrantes deste grupo de TCC, além de envolver diversos tipos de tecnologias de ponta. Lidar com elas simplificou a compreensão das tecnologias atuais, e também qualificou os autores para participar da criação de novas tecnologias.

O referencial teórico será abordado no Capítulo 2, cobrindo definições para os conteúdos envolvidos no desenvolvimento deste projeto, entre eles: a tecnologia M-JPEG e a Norma JPEG e suas implementações, conceitos fundamentais de SoCs, redes intrachip (NoCs), em particular a NoC HERMES e o Ambiente ATLAS, e conceitos fundamentais relacionados à plataforma de prototipação utilizada. No Capítulo 3 será apresentado como o decodificador M-JPEG do

OpenCores [MAN08] foi implementado. A integração das tecnologias é detalhada no Capítulo 4, e os resultados obtidos são apresentados no Capítulo 5.

2 REFERENCIAL TEÓRICO

Este Capítulo apresenta as principais definições relacionadas aos assuntos que são parte do desenvolvimento deste Trabalho de Conclusão. A Seção 2.1 aborda a tecnologia M-JPEG, baseado na Dissertação de Mestrado de Sebastian Manz [MAN08]. O foco é colocado na implementação, e na definição do padrão M-JPEG. Serão apresentadas técnicas de codificação e decodificação de imagens JPEG, em acordo com a norma CCITT pertinente [JPE92].

A Seção 2.2 aborda um conjunto de conceitos relacionados a sistemas integrados em um único chip (SoCs). Em seguida, na Seção 2.3 apresenta-se o conceito de redes intrachip (NoCs), a NoC HERMES e o ambiente ATLAS. Finalmente, a Seção 2.4 expõe sucintamente plataformas de prototipação de *hardware* com ênfase naquela que é utilizada neste projeto.

2.1 M-JPEG

Motion JPEG é um nome informal para uma classe de formatos de vídeo onde cada quadro ou campo entrelaçado de uma sequência de vídeo digital é comprimido separadamente como uma imagem JPEG. Originalmente desenvolvida para aplicações multimídia em PCs, onde formatos mais avançados se sobressaíram, M-JPEG é usado atualmente por muitos dispositivos portáteis de captura de vídeo, como câmeras digitais[WIK01].

Se um decodificador JPEG é projetado para decodificar múltiplas imagens em sequência sem perda de desempenho, então ele é adequado para ser transformado em um decodificador *Motion* JPEG. De qualquer maneira, duas observações devem ser colocadas. Primeiro, é importante saber como o filme foi armazenado, de forma que imagens JPEG possam ser extraídas. Segundo, para exibir os dados decodificados em um monitor VGA, alguns dados de imagem podem ser decodificados várias vezes, a fim de respeitar a taxa de atualização do monitor.

2.1.1 Formatos dos Contêineres

Um arquivo contêiner [MAN08] combina diferentes tipos de dados de um filme. Estes tipos podem ser de vídeo e áudio, mas também informações adicionais como legendas ou uma segunda trilha de áudio (e. g. em uma língua diferente) são comuns. Para ter todas as informações necessárias simultaneamente armazenadas no arquivo, os diferentes tipos de dados são intercalados. Informações simples adicionais, como a taxa de quadros do vídeo ou os *codecs* de áudio e vídeo utilizados, geralmente são armazenadas em um cabeçalho ou às vezes ao final do arquivo (como um *trailer*).

Existem muitos formatos de contêineres de vídeo, alguns dos mais conhecidos estão listados abaixo:

- *Audio Video Interleaved* (.avi): Um formato bem estabelecido, apesar de ser uma tecnologia ultrapassada;
- MPEG-4 *File Format* (.mp4): O formato padrão para *streams* de vídeos MPEG-4;
- *Matroska Media Container* (.mkv): Um formato de contêiner que possui código livre;
- Ogg (.ogg): O formato padrão para *streams*, amplamente utilizado na Internet.

Para implementar um decodificador de vídeo, pelo menos um formato de contêiner deve ter suporte embutido no sistema. O decodificador deve analisar o cabeçalho e extrair informações importantes. O *stream* de vídeo deve ser decodificado de acordo com os parâmetros definidos no cabeçalho. No entanto, para que o contêiner possa ser utilizado, o *stream* de entrada deve suprir algumas restrições para ser interpretado no decodificador proposto.

A extração do *stream* de vídeo do contêiner é realizada pela análise de marcadores, estes, auxiliam no controle das informações contidas em uma imagem JPEG (e.g. EOI e SOI), uma lista com os principais marcadores é disponibilizada no ANEXO A. Logo, outro tipo de dado não poderá conter estes marcadores. Por existir a possibilidade de encontrar alguns destes marcadores em um *stream* de áudio, o arquivo não pode ter nenhum tipo de dado de áudio.

2.1.2 JPEG

O termo JPEG é um acrônimo para *Joint Photographic Experts Group*[1] e foi especificado por uma comissão conjunta entre a *International Organization for Standardization* (ISO) e a *International Telecommunication Union* (ITU-T) [2], através da recomendação T.81 em 1992.

A norma JPEG [JPE92] define técnicas para compactar, descompactar e armazenar dados de imagens. Esta tecnologia é necessária e cobre um vasto campo de aplicações. É um grande desafio implementar tudo que é previsto na norma JPEG. Assim, a ITU-T definiu um conjunto de quatro diferentes tipos de processos de codificação: processo básico, processo baseado em DCT (*Discrete Cosine Transform*) estendido, processo sem perdas e processo hierárquico.

Com exceção do processo sem perdas, os outros processos são chamados de compressão com perdas. Isto significa que informações não redundantes são removidas dos dados e a imagem resultante não poderá ser restaurada exatamente como a imagem original. No entanto, se o processo de codificação for executado corretamente, a imagem descompactada será muito similar à imagem original.

A compressão JPEG funciona muito bem para imagens com tons contínuos como fotografias de paisagens. Para imagens com grandes áreas de cores exatamente iguais, por exemplo, diagramas gerados por computador, outras técnicas de compressão, como as desenvolvidas seguindo o padrão *Graphics Interchange Format* (GIF), geralmente resultam em melhores taxas de compressão apesar de serem processos sem perda.

O formato de arquivo amplamente utilizado - geralmente com o sufixo jpg - tem a sua própria norma, chamada *JPEG File Interchange Format* (JFIF) [HAM92], que possui uma série de restrições. JFIF é compatível com a especificação oficial do JPEG, mas não faz parte dele.

2.1.2.1 Métodos de Compressão

Como explicado anteriormente, existem quatro métodos de compressão que foram estabelecidos pela norma JPEG, conforme a Tabela I: Processo básico, Processo baseado em DCT estendido, Processo sem perdas e Processo hierárquico.

Apenas o Processo básico é amplamente usado, e as técnicas mais modernas de compressão de vídeo (como M-JPEG, MPEG-1/ MPEG -2) são fortemente baseadas nele.

---

[1] *ISO/IEC Joint Technical Committee 1, Subcommittee 29, Working Group.*

[2] *Anteriormente CCITT (Comité Consultatif International Téléphonique et Télégraphique).*

Tabela I - Características dos processos sugeridos pela norma JPEG [JPE92].

Processo básico	Processo baseado em DCT Estendido
• Processo baseado em DCT • Imagem fonte: amostras de 8 bits dentro cada componente • Sequencial • Codificação de *Huffman*: 2 tabelas de AC e 2 tabelas de DC • Decodificadores devem processar buscas com 1, 2, 3 e 4 componentes • Buscas intercaladas e não intercaladas	• Processo baseado em DCT • Imagem fonte: amostras de 8 bits e de 12 bits • Sequencial ou progressiva • Codificação de Huffman ou aritmética: 4 tabelas de AC e 4 tabelas de DC • Decodificadores devem processar buscas com 1, 2, 3 e 4 componentes • Buscas intercaladas e não intercaladas
Processo sem perdas	**Processo hierárquico**
• Processo previsível (não baseado em DCT) • Imagem fonte: amostras de P bits ($2 \leq P \leq 16$); • Sequencial • Codificação de Huffman ou aritmética: 4 tabelas de DC • Decodificadores devem processar buscas com 1, 2, 3 e 4 componentes • Buscas intercaladas e não intercaladas	• Múltiplos quadros (não diferencial e diferencial) • Baseado em DCT estendida ou no processo sem perdas • Decodificadores devem processar buscas com 1, 2, 3 e 4 componentes • Buscas intercaladas e não intercaladas

2.1.2.2 Codificação

Primeiramente a imagem é transformada para o modo de cor YCbCr, separando a luminosidade (Y) da informação de cor (Cb/Cr). Então, a imagem, agora pode ser separada em blocos menores, as MCUs, que representam um bloco 2x2 de blocos de 64 *pixels* (8x8). A partir destes blocos menores, os componentes das cores são reduzidos em resolução espacial em um passo denominado *amostragem*. Aplicando a Transformada Discreta do Cosseno (DCT) os blocos são mapeados para o domínio frequência, onde as frequências mais altas são removidas no passo de *quantização*. Depois, organizam-se os coeficientes restantes no passo de *ordenação zigue-zague*. O vetor de oito bits resultante está bem preparado agora para a *codificação de entropia*, utilizando codificação aritmética e um algoritmo de *Huffman*. A Figura 2.1 demonstra estes passos de codificação do JPEG que serão detalhados nas Seções a seguir.

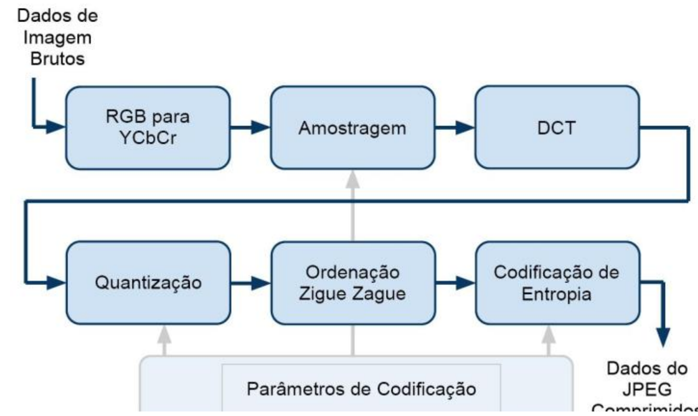


Figura 2.1 - Os passos essenciais na codificação JPEG.

2.1.2.2.1 RGB para YCbCr

Uma imagem digital pode ser capturada através de *scanners*, câmeras digitais ou câmeras analógicas ligadas ao computador através de placas digitalizadoras. A câmera digital e o *scanner* contêm sensores que captam a luz emitida ou refletida dos objetos, e a decompõe nos seus componentes fundamentais: vermelho (R), verde (G) e azul (B).

A imagem capturada é bidimensional e pode ser dividida em seus componentes fundamentais, que são chamados de *Pixels*. *Pixel* é o menor componente de uma imagem digital. Sua representação é a de um ponto, porém, com um tamanho definido, ou seja, não é o conceito matemático de ponto com tamanho nulo. Este ponto eventualmente pode ter um formato circular, retangular ou quadrado, e é formado por 3 componentes fundamentais: vermelho, verde e azul (RGB) [MAN08].

Estes *pixels* são enviados para um conversor analógico/digital, e quantizados, ou seja, os valores dos componentes fundamentais RGB, que teoricamente poderiam variar de zero a infinito, são convertidos para valores inteiros, por exemplo, entre 0 e 255, o que possibilita guardar a representação de cada componente em apenas oito bits.

Em uma segunda etapa os componentes “RGB” são convertidos para componentes de luminância (“Y”) e crominância (“Cb” e ”Cr”). A luminância é uma escala de representação de tons de cinza, enquanto a crominância são duas escalas numéricas, que juntas representam as cores. A escala de luminância, por convenção (ITU-R 601)[3], é quantizada com valores entre “16” e “235”, sendo que, o valor “16” representa o negro absoluto e o valor “235” representa o branco absoluto. As duas escalas de crominância contêm valores que variam entre “16” e “240” (128 sendo o valor central). Os valores não utilizados, dos “256” possíveis, servem como códigos de controle [MEL99].

As matrizes de conversão entre os componentes “RGB” e os componentes “YCbCr” são dadas abaixo:

$$\begin{aligned} Y &= 0{,}299 * R + 0{,}587 * G + 0{,}114 * B \\ Cb &= -0{,}1687 * R - 0{,}3313 * G + 0{,}5 * B + 128 \\ Cr &= 0{,}5 * R - 0{,}4187 * G - 0{,}0813 * B + 128 \end{aligned} \quad (2.1)$$

$$\begin{aligned} R &= Y + 1{,}402 * (Cr - 128) \\ G &= Y - 0{,}34414 * (Cb - 128) - 0{,}71414 * (Cr - 128) \\ B &= Y + 1{,}772 * (Cb - 128) \end{aligned} \quad (2.2)$$

[3] *ITU-R Recommendation BT.601, abreviado como Rec. 601 ou BT.601 (ou como o seu antecessor, CCIR 601,) é uma norma publicada pela União Internacional de Comunicação (do Inglês, International Telecommunication Union - Radiocommunications sector)*

2.1.2.2.2 Amostragem

Após a aquisição e a conversão analógica/digital, o processo de amostragem reduz somente os componentes de crominância[4]. Assim são criadas as MCUs, formadas por 4 blocos de 8x8 *pixels*, como indica a Figura 2.2.

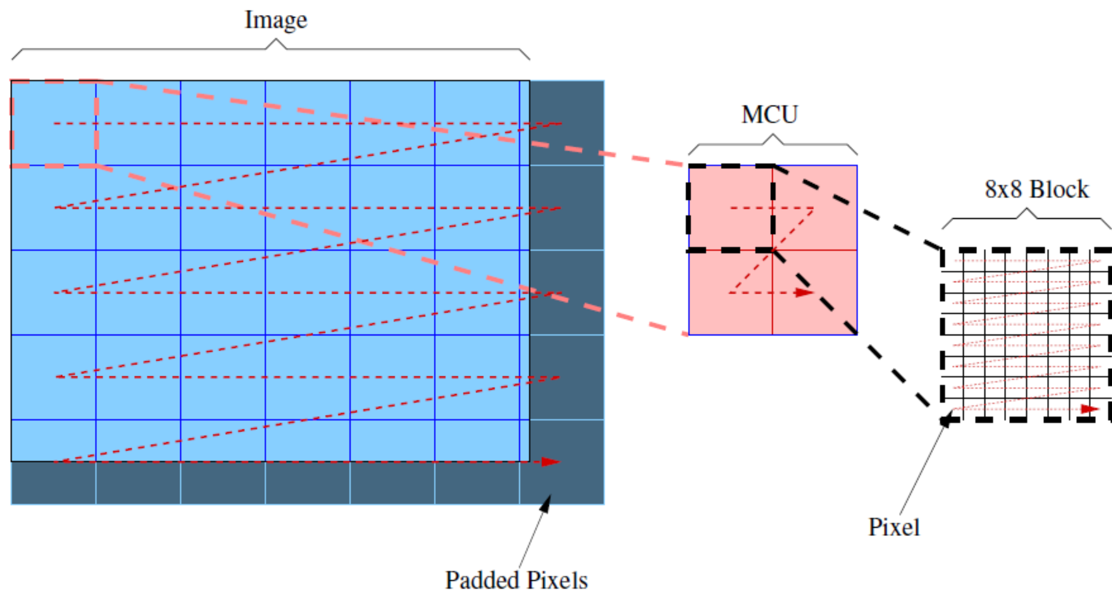


Figura 2.2 - Blocos de uma imagem JPEG.

Dependendo do tipo de amostragem utilizado, a MCU será construída de forma diferente. Para o tipo “4:2:0”, a cada quatro blocos 8x8 Y (luminância), será feita uma média dos componentes de crominância, gerando somente um bloco de crominância Cb e outro de crominância Cr. Ao gerar uma amostragem para o tipo “4:2:2”, a cada dois blocos 8x8 Y, serão gerados um bloco de crominância Cb e outro de crominância Cr. Caso seja utilizada amostragem do tipo “4:4:4”, não haverá perda de informações neste processo como ocorre na amostragem dos demais tipos, pois a cada bloco 8x8 Y, são gerados um bloco de crominância Cb e outro de crominância Cr, ou seja, a imagem não terá nenhum tipo de compressão. A Figura 2.3 ilustra alguns dos diferentes tipos de amostragem, onde o símbolo “X” representa os componentes Y e o símbolo “O” representa os componentes CbCr.

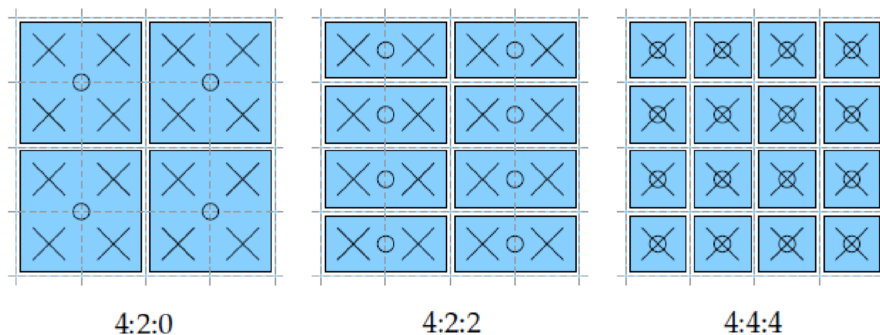


Figura 2.3 - Tipos de amostragem.

Visto que a imagem necessita ser dividida em blocos 8x8 para a utilização ao longo da codificação, o codificador se encarrega de redimensionar a altura e a largura para que ambas sejam múltiplas de oito, Figura 2.2.

---

[4] *O olho humano é mais sensível à luminância (tonalidade de cinza) do que à crominância (cores), o que permite maior taxa de compressão de informações de crominância sem que esta perda seja percebida pelo espectador.*

Para o próximo processo, a aplicação da DCT, será formado um *stream* de blocos 8x8, que constam dos blocos dos componentes da luminância e dos blocos de crominância Cb e Cr, idem à Figura 2.4.

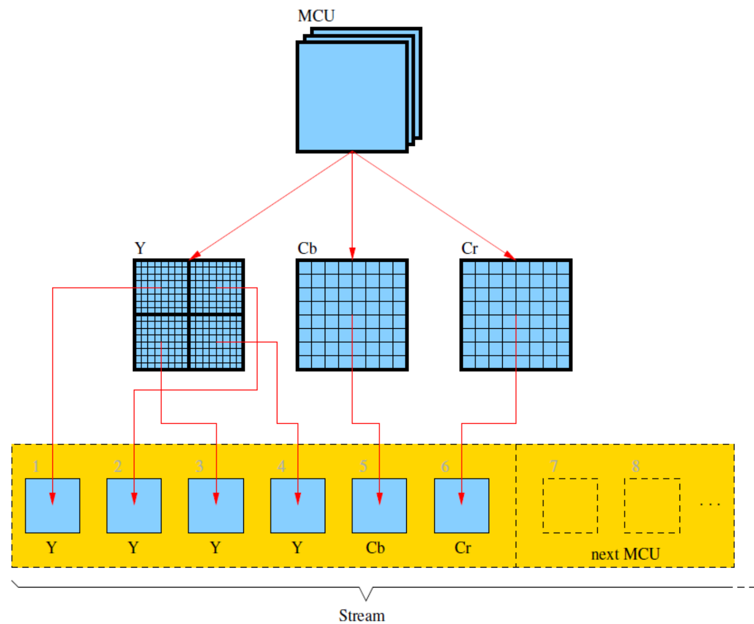


Figura 2.4 - *Stream* de blocos 8x8 de componentes Y, Cb e Cr.

2.1.2.2.3 Transformada Discreta do Cosseno

O algoritmo de compressão JPEG divide a imagem em uma matriz de luminância e duas matrizes de crominância. As três matrizes descrevem a imagem, tendo cada uma o tamanho de 8x8 *pixels*. Com isto, têm-se várias matrizes de 64 *pixels*, conhecidas como valores amostrados (*sampled values*). Sobre estas matrizes é aplicado o algoritmo DCT, gerando outras matrizes, denominadas de matrizes de coeficientes DCT, cuja maioria dos elementos tem valor igual a 0. Esse processo translada a informação do domínio tempo para o domínio frequência [AHM74].

A Transformada Discreta de Cosseno converte uma matriz de valores altamente correlacionados, e com uma distribuição de probabilidade uniforme, em um conjunto de valores menos correlacionados e com uma distribuição de probabilidade não uniforme. Este algoritmo converte uma matriz numérica 8x8 de precisão de 8 bits para outra matriz 8x8 com uma precisão de 11 bits. Em teoria o processo de aplicação da DCT pode ser sem perda, mas informações da matriz resultante serão sempre perdidas devido à limitação da precisão computacional.

O valor médio da matriz é chamado de componente DC, e está localizado no canto superior esquerdo da matriz de Coeficientes DCT. Os outros coeficientes são denominados coeficientes AC e representam os valores das pequenas variações de tonalidade e coloração do bloco.

A DCT, em termos de compressão, gera uma compactação equivalente ao da Transformada Discreta de Fourier (DFT). No entanto, no processo inverso, ao se aplicar a anti-transformada da DCT consegue-se uma aproximação melhor dos coeficientes originais do que a anti-transformada da DFT.

A transformada Discreta de Cosseno para matrizes de duas dimensões e tamanho de 8x8 pode ser expressa pela seguinte equação:

$$F(u,v) = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\left(\frac{2x+1}{16} u\pi\right) \cos\left(\frac{2x+1}{16} v\pi\right) \tag{2.3}$$

Onde,

$$\begin{cases} C_u, C_v = \frac{1}{\sqrt{2}} \; para \; u, v = 0, \\ C_u, C_v = 1 \; caso \; contrário \end{cases}$$

A equação (2.3) é aplicada para cada valor $f(x,y)$, que representa o valor do *pixel* na posição (x,y) da matriz, onde as resultantes obtidas por $F(u,v)$ representam os coeficientes DCT, formando assim, um novo bloco 8x8 de componentes DCT que será utilizado no processo de quantização. A Figura 2.5 (a) mostra um exemplo de bloco 8x8 composto de coeficientes DCT preparado para o próximo passo.

496	-42	33	17	-20	43	44	-20
160	26	29	7	54	7	12	-18
25	49	-15	32	48	-28	44	20
41	27	-35	20	3	12	-16	9
-39	11	30	-31	26	-9	15	-24
23	-23	23	-9	13	26	12	11
15	17	-13	10	5	-34	5	43
-10	12	8	-22	6	38	-46	8

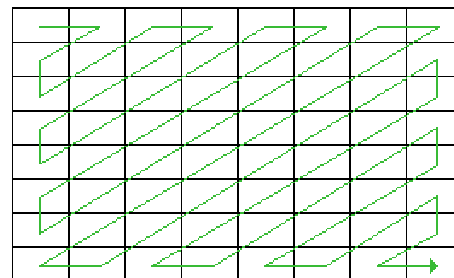
(a) Coeficientes DCT

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

(b) Tabela de Quantização

29	-2	1	0	0	0	0	0
9	1	1	0	1	0	0	0
1	2	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(c) Coeficientes DCT quantizados



(d) Ordenação Zig-Zag

29	-2	9	1	1	1	0	1	2	1	0	0	0	0	0	0	...

(e) Ordenação ZigZag

Figura 2.5 - Estrutura dos componentes utilizados no processo de codificação.

2.1.2.2.4 Quantização

A matriz resultante da DCT terá o mesmo tamanho que a original, resultando em pouca ou quase nenhuma compactação, possuindo a maioria dos coeficientes com valores próximos a 0 e poucos com valor 0. A compactação virá a partir da aplicação do processo denominado *quantização*. Neste processo de compressão, os elementos menos importantes da matriz de coeficientes DCT serão descartados.

A vantagem da representação no domínio frequência é que, diferente do domínio tempo antes da DCT, nem todos os valores têm a mesma importância para a qualidade visual da imagem. Remover os componentes com altas frequências irá reduzir o nível de detalhes, mas a estrutura geral continuará, pois esta é definida pelos componentes de baixas frequências.

$$QF(u,v) = round\left(\frac{F(u,v)}{Q(u,v)}\right) \tag{2.4}$$

Utilizando a equação (2.4), a matriz de coeficientes DCT é dividida por outra matriz (chamada de tabela de quantização), como na Figura 2.5 (b), que transforma todos os elementos em uma nova matriz, agora com componentes quantizados. Não existe informação perdida pela divisão dos coeficientes, mas o resultado será arredondado para o inteiro mais próximo. Quanto maior o divisor, mais as informações dos coeficientes serão posicionadas nas casas decimais depois da vírgula, sendo portanto descartadas pela operação de arredondamento. Ao se codificar uma imagem, a tabela de quantização é armazenada no cabeçalho da imagem precedido pelo marcador DQT, assim podendo ser restaurada no processo de decodificação.

Este processo irá acentuar as características da matriz de coeficientes DCT forçando todos os elementos a aproximarem-se do valor 0, como mostra a Figura 2.5 (c).

2.1.2.2.5 Ordenação Zigue-zague

A matriz de valores quantizados é lida em forma zigue-zague, da esquerda para direita e de cima para baixo, fazendo com que os primeiros elementos lidos sejam os valores não nulos. Como resultado, se obtém um único vetor com os coeficientes da tabela de quantização reorganizados, os coeficientes diferentes de zero encontram-se no início do vetor e os coeficientes 0 estão ao final do mesmo vetor conforme a Figura 2.5 (e).

2.1.2.2.6 Codificação de Entropia

Após a leitura em zigue-zague, os coeficientes serão rearranjados para possibilitar uma nova compressão denominada codificação de entropia. Esta codificação utiliza três técnicas: *Huffman Encoding*, *run length encoding* e *variable length encoding*. Estas técnicas são do tipo sem perda, ou seja, não acarretam perda de informações. Logo, esta nova compressão não afetará a qualidade da imagem [TAN97].

Ao tratar desta codificação existem alguns tópicos que devem ser lembrados:

- O primeiro coeficiente do vetor recebido é chamado DC, os coeficientes restantes recebem o nome de AC[5];
- O primeiro coeficiente (DC) é o valor médio do bloco original, antes do processo da DCT;
- Existe uma ligação entre o componente DC do vetor atual e os componentes DC dos vetores vizinhos;
- É comum que o componente DC possua valores elevados. Ele é o componente mais importante e, portanto, o menos reduzido no processo de quantização;
- A maioria dos coeficientes 0 aparece no final do vetor;
- A maioria dos coeficientes com valores diferentes de 0 tem valores pequenos.

O coeficiente DC será decodificado de maneira diferente em relação aos coeficientes AC. Respeitando a relação entre os blocos vizinhos, apenas para primeiro bloco o coeficiente DC será totalmente processado. Para os próximos blocos será realizada a diferença entre o coeficiente DC do bloco atual e o coeficiente DC do bloco anterior, conforme o exemplo abaixo. Este processo é aplicado para cada componente independentemente.

Utilizando o vetor resultante da Figura 2.5 (d), exemplifica-se o processo utilizado para a codificação dos coeficientes DC:

$$29, -2, 9, 1, 1, 1, 0, \ldots$$

[5] *Referência à eletrônica, onde "DC" significa corrente continua e "AC" corrente alternada. O primeiro coeficiente é o valor médio do bloco original e é o único que contribui igualmente para todos os pixels.*

Supondo que o bloco decodificado anterior do mesmo componente obteve o coeficiente DC 23, então a diferença a ser decodificada é $29 - 23 = 6$, resultando o vetor:

$$6, -2, 9, 1, 1, 1, 0, \ldots$$

O próximo passo é fazer a codificação dos componentes AC utilizando o algoritmo *run length encoding*, onde cada coeficiente AC diferente de zero recebe a informação de quantos zeros precedem o coeficiente AC. Desta forma, os zeros precedentes podem ser removidos. Caso o número de zeros precedentes exceda 15, é utilizado um código especial para os 16 zeros precedentes, "ZRL" *zero run length*, as descrições dos marcadores encontram se no ANEXO B.

Não existem zeros precedentes ao coeficiente DC, mas por outro lado, para os coeficientes AC, zero é um valor que deve ser levado em consideração. Os coeficientes com valor zero encontrados após o último coeficiente AC diferente de zero recebem um código especial, "EOB" ou *end of block.*

$$\begin{array}{ccccccccc} 6, & -,d & 9, & 1, & 1, & 1, & \underbrace{0,1,} & \underbrace{0,0,0,0,0,1,} & \underbrace{0,0,0,\ldots} \\ [\;]6, & [0]-,d & [0]9, & [0]1, & [0]1, & [0]1, & [1]1, \cdots & [5]1, & [eob] \end{array}$$

O próximo processo, *variable length encoding*, altera o resultado do processo de *run length encoding* para a representação em forma binária, adicionando, antes do valor, a quantidade mínima necessária de bits para que o coeficiente seja representado. Valores negativos são representados em complemento de um (invertem-se os bits).

$$\begin{array}{cccccccccc} 6, & -10 & 9, & 1, & 1, & 1, & \underbrace{0,1,} & & \underbrace{0,0,0,0,0,1,} & \underbrace{0,0,0,\ldots} \\ [\;]6, & [0]-10 & [0]9, & [0]1, & [0]1, & [0]1, & [1]1, & \ldots & [5]1, & [eob] \\ [\;3]110, & [0\;2]01, & [0\;4]1001, & [0\;1]1, & [0\;1]1, & [0\;1]1, & [1\;1]1, & & [5\;1]1, & [0\;0] \end{array}$$

Após os coeficientes codificados corretamente, o processo *Huffman encoding* é responsável por codificar, em forma binária, os códigos que precedem os coeficientes. Para isso, têm-se um exemplo de uma tabela de *Huffman* DC e uma tabela de *Huffman* AC.

Tabela II - Exemplo de Tabela de *Huffman* DC.

Tabela DC		
Huffman code	Valor original	
⋮	⋮	⋮
110	$0x3$	[3]
⋮	⋮	⋮

Tabela III - Exemplo de Tabela de *Huffman* AC.

Tabela AC		
Huffman code	Valor original	
00	$0x00$	$[eob]$
01	$0x01$	[0 1]
100	$0x02$	[0 2]
101	$0x11$	[1 1]
1100	$0x04$	[0 4]
⋮	⋮	⋮
1111110	$0x51$	[5 1]
⋮	⋮	⋮

Assim pode-se reconstruir o vetor de bits final a partir de:

$$\begin{array}{cccccccccccccccccc}
 & 6, & & -00 & & 9, & & 1, & & 1, & & 1, & 0, & 1, & & \underbrace{0,0,0,0,0,} & 1, & \underbrace{0,0,0,\ldots} \\
[\ ] & 6, & [0] & -00 & [0] & 9, & [0] & 1, & [0] & 1, & [0] & 1, & [1] & 1, & & [5] & 1, & [eob] \\
[\ 3] & 110, & [0\ 2] & 01, & [0\ 4] & 1001, & [0\ 1] & 1, & [0\ 1] & 1, & [0\ 1] & 1, & [1\ 1] & 1, & \ldots & [5\ 1] & 1, & [0\ 0] \\
 & & & & & & & & & & & & & & & & & \\
110 & 110 & 100 & 01 & 1100 & 1001 & 01 & 1 & 01 & 1 & 01 & 1 & 101 & 1 & & 1111110 & 1 & 00
\end{array}$$

O vetor de bits final, neste caso é:

$$11011010 \quad 00111001 \quad 00101101 \quad 10111011 \quad 11111101 \quad 00$$

Pelo processo da codificação, 64 bits de dados de entrada foram codificados em menos de seis bytes. As tabelas da *Huffman* utilizadas na codificação devem ser armazenadas no cabeçalho da imagem. Para isso, um código especial é utilizado, o DHT. Após ele é armazenado em uma tabela de *Huffman* de cada vez. Para o método de compressão utilizado, o Processo básico, cada cabeçalho deve conter duas tabelas de *Huffman* DC e duas tabelas de *Huffman* AC.

2.1.2.3 Decodificação

O objetivo do processo de decodificação é reconstruir uma imagem a partir de qualquer imagem codificada com uma precisão adequada, através de parâmetros e limites definidos pela aplicação [JPE92]. Na decodificação, cada bloco 8x8 *pixels* é processado por um conjunto de métodos sequenciais (cf. Figura 2.6) usando o cabeçalho fornecido pela imagem e que será descritos nas Subseções seguintes.

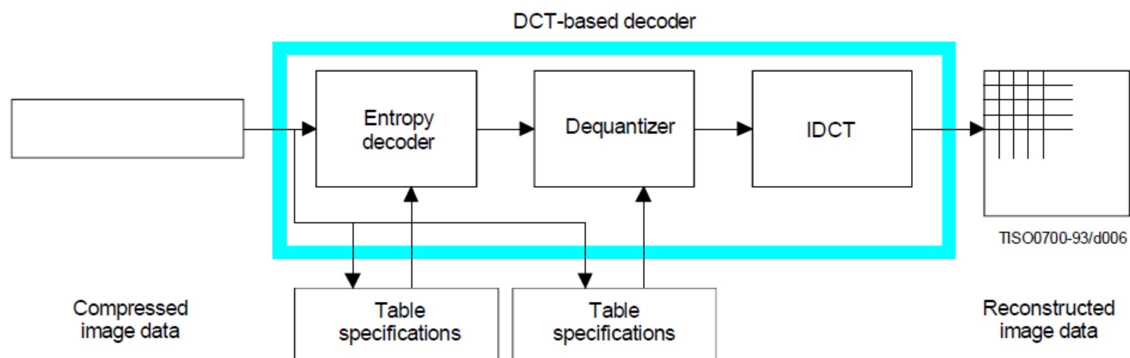


Figura 2.6 - Diagrama Simplificado de um decodificador baseado em DCT.

2.1.2.3.1 Decodificação de entropia

Similar à codificação de entropia, a decodificação de entropia também pode ser considerada como um processo de dois passos. A partir dos coeficientes AC e DC retirados de cada bloco 8x8, estes coeficientes são decodificados, com base nas tabelas *Huffman*[6] identificadas no cabeçalho da imagem, na ordem em que foram armazenadas pelo processo de zigue-zague ao codificar a imagem. Inicia-se com o método de decodificação diferencial e depois com o método de decodificação *run length* [MAN08].

Este processo recebe como entrada um *stream* de bits. Nele estão contidos os coeficientes de cada bloco 8x8, que será reconstruído nesta etapa para a utilização dos próximos processos[7]. Utilizando a Figura 2.7 como referência, observa-se que o *stream* de bits é percorrido até que um

---

[6] Para o processo básico, são utilizadas duas tabelas de *Huffman* DC e duas tabelas de *Huffman* AC

[7] Método inverso ao utilizado na ordenação zigue-zague.

código de *Huffman* válido seja encontrado (representado em azul). O código é consultado na tabela de *Huffman* utilizada, que retorna a quantidade de zeros precedentes e quantos bits o valor codificado necessita. Primeiramente, o número de zeros é escrito. Após, a quantidade de bits necessária é lida e o valor resultante é escrito (em vermelho).

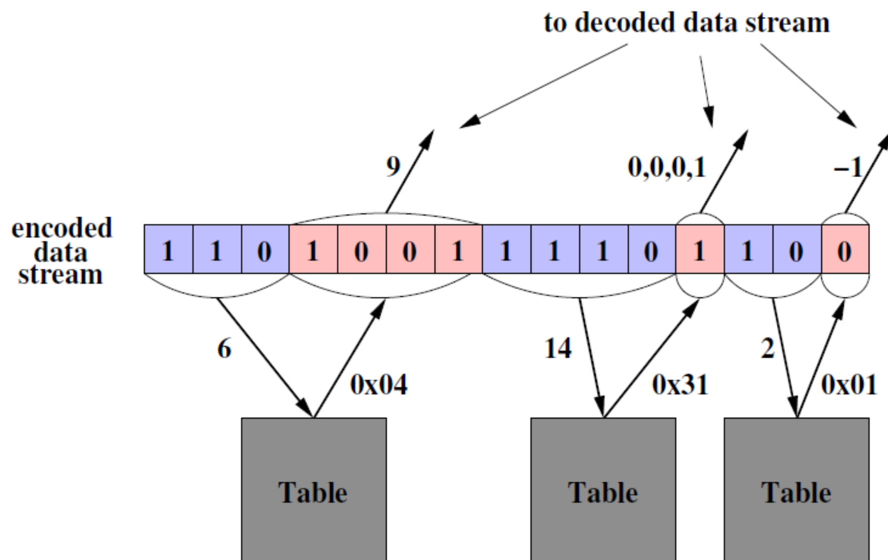


Figura 2.7 - Decodificação de entropia em um *stream* de bits.

Na decodificação diferencial, é necessário obter o valor original do coeficiente DC antes da codificação. Para isso, faz-se o cálculo da soma entre o coeficiente DC atual e o coeficiente DC obtido no bloco 8x8 anterior. Após decodificar o coeficiente DC, o método *run length* é utilizado para decodificar os coeficientes AC. Eles são decodificados diretamente, utilizando a mesma ordem do processo de ordenação zigue-zague na codificação. Quando o marcador EOB é encontrado, os coeficientes restantes são zerados.

Quando o bloco 8x8 está completamente reestruturado, ele está pronto para o processo de desquantização.

2.1.2.3.2 Desquantização

Após receber o bloco 8x8 decodificado pelas tabelas de *Huffman*, o bloco passa por uma normalização onde é aplicada a seguinte equação[8]:

$$R_{vu} = Sq_{vu} * Q_{vu} \tag{2.5}$$

A equação (2.5) tem como resultado o R_{vu}, que representa o valor do coeficiente DCT. Ele é obtido através da multiplicação do coeficiente DCT quantizado pelo valor correspondente na tabela de quantização, a qual é obtida por meio da leitura do cabeçalho da imagem [JPE92].

O decodificador deve ser capaz de utilizar até quatro tabelas de quantização.

2.1.2.3.3 IDCT

A transformada inversa do cosseno (IDCT) corresponde a uma transformação matemática utilizando funções básicas do cosseno que converte um bloco 8x8 de coeficientes DCT quantizados em um bloco 8x8 amostrado através da equação:

---

[8] Dependendo do arredondamento utilizado na quantização, é possível que o coeficiente desquantizado fique fora dos limites esperados.

$$f(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C_u C_v F(u,v)\cos\left(\frac{2x+1}{16}u\pi\right)\cos\left(\frac{2x+1}{16}v\pi\right) \quad (2.6)$$

Onde,

$$\begin{cases} C_u, C_v = \frac{1}{\sqrt{2}}\ para\, u, v = 0 \\ C_u, C_v = 1\ caso\ contrário \end{cases}$$

Após a aplicação da IDCT os sinais do bloco 8x8 resultante têm os seus níveis deslocados (*level-shifted*), convertendo a saída para uma representação sem sinal (*unsigned*). Para uma precisão de 8 bits o deslocamento de nível é realizado adicionando-se 128, caso necessário. As saídas amostradas devem ser restringidas para permanecerem dentro dos limites da precisão utilizada (0 a 255 para a precisão de 8 bits) [JPE92].

2.1.2.3.4 Amostragem

Apesar de não constar na Figura 2.6 como um método de decodificação, o processo da amostragem ainda é utilizado na decodificação. Ele geralmente é implementado juntamente com o processo da IDCT, mas também é possível fazer com que este seja um processo separado.

O objetivo da amostragem é similar ao da ordenação zigue-zague do processo da decodificação de entropia. Os blocos 8x8 resultantes do processo da IDCT são agora reunidos, onde o *stream* de blocos 8x8 com as informações de cores (YCbCr) originais da imagem é reorganizado e transformado em MCUs. Após isso a imagem pode ser totalmente reconstruída [MAN08].

Para que seja possível apresentar uma imagem JPEG totalmente reconstruída, é necessária que seja realizada a troca de cores novamente, pois a maioria dos dispositivos trabalha no formato de cores RGB e não no formato YCbCr. Assim, a equação (2.2) é utilizada, tornando agora possível encaminhar a imagem para um dispositivo de exibição de imagens, como um monitor VGA, display LCD, etc.

Com os passos da decodificação realizados, o que se tem é uma imagem JPEG reconstruída e que se assemelha muito com a original, o fato de não ser idêntica se deve a alguns passos na codificação da imagem que possuem em seus algoritmos internos perda de dados devido às limitações computacionais (e.g. operações matemáticas com grande mantissa).

2.2 System on a chip

O termo *System on a chip* (SoC) está relacionado ao conceito de um grande número de componentes em um circuito integrado, formando um sistema completo ou quase, com elementos funcionais complexos e heterogêneos dentro de um único circuito integrado (*chip*), com a finalidade de serem mais rentáveis, pois aumentam o rendimento da fabricação e são mais confiáveis [BER02]. Nestes sistemas, integra-se um conjunto de componentes como um processador e blocos de memórias, interfaces de E/S (entrada e saída) e interfaces de rede. Espera-se que em um futuro breve seja possível integrar centenas de tais componentes em um só circuito integrado. Cada vez mais, os projetos de sistemas embarcados necessitam de mais integração e comunicação, gerando assim necessidade de SoCs cada vez mais complexos.

SoCs não estão somente em computadores domésticos, mas também em vários dispositivos eletrônicos do cotidiano. Eles são o elo entre as empresas de sistemas embarcados e a fábricas de semicondutores. Normalmente, SoCs são projetados para dispositivos adequados para aplicações

específicas com desempenho alvo específico. MPSoCs, por outro lado, são SoCs multiprocessados projetados para aplicações tipicamente com maior poder computacional [WOL05].

Pode-se dizer que um SoC compõe-se de núcleos de Propriedade Intelectual (IPs), arquiteturas de interconexão e interfaces para dispositivos periféricos[GUP97]. A arquitetura de interconexão inclui interfaces físicas e mecanismos de comunicação que permitem a comunicação entre os componentes do SoC.

Tradicionalmente, a arquitetura de interconexão empregava fios dedicados ou barramentos compartilhados. Fios dedicados são eficazes para sistemas com um pequeno número de núcleos, mas o número de fios ao redor do núcleo aumenta à medida que a complexidade do sistema cresce. Além disso, tal forma de interconexão possui baixa escalabilidade e flexibilidade.

Um barramento compartilhado é um conjunto de fios comuns a vários núcleos, sendo naturalmente escalável e reutilizável quando comparado aos fios dedicados. No entanto, barramentos não permitem que mais de uma comunicação ocorra ao mesmo tempo, todos os núcleos compartilham a mesma largura de banda de comunicação no sistema e a escalabilidade é limitada a algumas dezenas de núcleos [KJS02][BEN01][GUE00]. Utilizar barramentos separados, interconectados por pontes ou arquiteturas de barramentos hierárquicos, pode reduzir algumas destas limitações, uma vez que diversos barramentos podem contribuir para diferentes necessidades de largura de banda, protocolos e ainda aumentam o paralelismo da comunicação. No entanto, a escalabilidade continua a ser um problema para arquiteturas de barramento hierárquico, pois a comunicação entre módulos conectados a dois barramentos distintos pode gerar um grau de contenção por recursos significativos.

Neste contexto, aplicando conceitos herdados das áreas de sistemas distribuídos e redes de computadores para sistemas embarcados, emergiu no início deste século uma nova estrutura de interconexão, a chamada Redes Intrachip (NoCs). NoCs são compostas por componentes ativos dedicados ao processo de comunicação dentro do *chip*, comumente denominados roteadores. Os elementos processadores de um SoC, os chamados Núcleos IP conectam-se aos roteadores através de interfaces de rede. Roteadores por sua vez ligam-se entre si por enlaces de comunicação (fios). Assim, a comunicação é realizada de uma forma mais estruturada e escalável [BEN02]. Redes Intrachip são abordadas na próxima Seção, detalhando arquiteturas, topologias e protocolos, além da abordar especificamente a NoC HERMES, desenvolvida pelo grupo de pesquisa GAPH, que é utilizada neste trabalho.

2.3 Networks on Chip

Redes intrachip (do Inglês *Networks on Chip* - NoCs) são uma alternativa bem sucedida para as limitações físicas que comprometem a escalabilidade e o desempenho do processamento de aplicações paralelas [RGW01]. NoCs são consideradas a solução para as restrições existentes nas arquiteturas de interconexão devido às seguintes características: (i) eficiência energética e confiabilidade [BEN01], (ii) escalabilidade da largura de banda, quando comparado às arquiteturas de barramento tradicionais, (iii) reutilização, (iv) decisões de roteamento distribuído [BEN02] [GUE00].

Congestionamentos em NoCs reduzem o desempenho geral do sistema. Este efeito é particularmente forte nas redes que possuem um único *buffer* associado a cada canal de entrada, o que simplifica o projeto do roteador, mas impede pacotes de compartilhar um canal físico em qualquer instante de tempo. Um canal pode ser multiplexado em n canais virtuais para reduzir este problema. Canais virtuais usam vários *buffers* para cada canal físico, potencialmente elevando a disponibilidade de recursos para as mensagens que trafegam na rede.

A comunicação ponto a ponto em um sistema é realizada através da troca de mensagens entre IPs. De acordo com o conceito de redes de pacotes [DYN02], muitas vezes, a estrutura de mensagens particulares não é adequada para fins de comunicação. Um pacote é uma forma padrão para representar informações de forma adequada para a comunicação. Um pacote pode

corresponder a uma fração, uma ou mesmo várias mensagens. No contexto de NoCs, pacotes são frequentemente, uma fração de uma mensagem. Pacotes são muitas vezes compostos por um cabeçalho, uma carga útil e uma cauda. Para garantir o funcionamento correto durante transferências de mensagens, uma NoC deve evitar fenômenos indesejáveis tais como *deadlock*, *livelock* e *starvation* [DYN02]. *Deadlock* pode ser definido como uma dependência cíclica entre nós da rede que solicitam acesso para utilizar recursos de modo que nenhum progresso a partir do estabelecimento da dependência cíclica pode ser obtido, não importando em qual sequência os eventos ocorram. Assim, um pacote não pode mais progredir. *Livelock* refere-se a pacotes que percorrem caminhos circulares na rede, sem nunca fazer progresso efetivo em direção ao seu destino. *Starvation* consiste na não alocação de um recurso devido à postergação indefinida de acesso ao mesmo, sendo uma falha comum em protocolos de arbitragem mal elaborados [FRE09].

2.3.1 Arquiteturas de NoCs

A arquitetura de uma NoC é formada basicamente por três tipos de elementos: roteadores, interfaces e enlaces de comunicação. Cada elemento possui funções para garantir a integridade da operação da rede, ou seja, garantir a comunicação entre todos os IPs que necessitem se comunicar. A Figura 2.8 ilustra um exemplo de uma NoC.

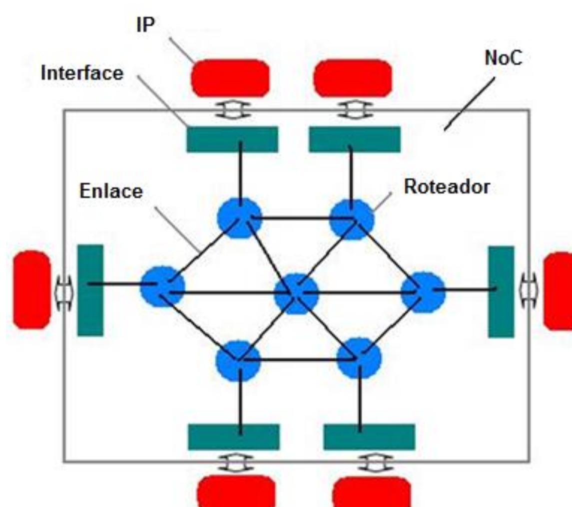


Figura 2.8 - Exemplo de arquitetura NoC e comunicação com os IPs do sistema.

As interfaces realizam a comunicação com os IPs que desempenham as funções de processamento de informação no SoC. Obviamente, redes de interconexão são indispensáveis para qualquer arquitetura de SoC complexa. Do ponto de vista de funcionalidade, as redes são compostas por serviços e pelo sistema de comunicação. Serviços como a transmissão com garantia de integridade de dados e garantias de latência, são considerados essenciais para SoCs [RGR03]. A implementação destes serviços é muitas vezes baseada no uso de pilhas de protocolo, como o proposto no modelo de referência OSI da ISO [DAY83].

O sistema de comunicação, em contrapartida, é o que dá suporte à transferência de informações da origem para o destino. O sistema de comunicação permite que cada núcleo envie pacotes para todos os outros núcleos na estrutura NoC. Sua estrutura é um conjunto de roteadores interligados por enlaces de comunicação.

2.3.2 Topologias

A forma como os roteadores são ligados define a *topologia da rede*. De acordo com a topologia, redes podem ser classificadas em uma das duas classes principais: estática e dinâmica [HP96] [HWA92].

Nas redes estáticas, cada nó possui conexões fixas ponto a ponto para um determinado número de outros nós. Hipercubo, Anel, Malha, Toro e Árvore Gorda são exemplos de topologias de redes estáticas. A topologia de rede predominante na literatura é a Malha 2D. A razão para esta escolha decorre de suas três vantagens: a implementação facilitada utilizando as tecnologias planares atuais para circuitos integrados, a simplicidade da estratégia de roteamento XY e a escalabilidade da rede. Outra abordagem é usar a topologia Toro 2D, para reduzir o diâmetro da rede [MBV02]. Um problema das topologias Malha e Toro é a latência da rede associada quando nodos distantes necessitam se comunicar.

O mais importante é encontrar uma topologia que se adapte aos requisitos do sistema, em relação a desempenho, confiabilidade e custo.

Redes dinâmicas utilizam canais de comunicação que pode ser (re) configurados em tempo de execução. Barramentos e chaves *crossbar* são exemplos de redes dinâmicas.

O mecanismo de comunicação, o modo de roteamento e os algoritmos de roteamento são função da topologia da rede e são usados para compor os serviços prestados pela NoC.

O mecanismo de comunicação especifica como as mensagens evoluem através da rede. Dois métodos de transferência de mensagens são o roteamento de circuitos e o roteamento de pacotes [HWA92]. No roteamento de circuitos, um caminho é estabelecido antes dos pacotes serem enviados através da reserva de uma sequência de canais entre a origem e o destino. O caminho assim formado é chamado de conexão. Após estabelecer uma conexão, mensagens podem ser enviadas através dos canais alocados. O uso destes para outras comunicações é negado até que um procedimento de desconexão seja efetuado. No roteamento de pacotes, os pacotes são transmitidos sem necessidade de procedimentos prévios de estabelecimento da conexão.

A roteamento de pacotes requer o uso de um modo de roteamento, que define como os pacotes passarão através dos roteadores. Os modos mais importantes são: *store-and-forward*, *virtual cut-through* e *wormhole* [NI93]. No modo *store-and-forward*, um roteador não pode reencaminhar um pacote até que ele tenha sido completamente recebido. Cada vez que um roteador recebe um pacote, seu conteúdo é examinado para decidir o que fazer o que implica em uma latência específica para cada roteador, esta latência se acumula ao longo do caminho percorrido pelo pacote. No modo *virtual cut-through*, um roteador pode encaminhar um pacote, logo que o roteador próximo dê a garantia de que o pacote será aceito completamente [RGW01]. Assim, é necessário um *buffer* para armazenar pelo menos um pacote completo, da mesma forma que no modo *store-and-forward*, mas neste caso com ganhos para a latência da comunicação. O modo *wormhole* é uma variante do modo *virtual cut-through* que desacopla o tamanho do *buffer* do tamanho da mensagem. Um pacote é transmitido entre os roteadores em unidades chamadas *flits* (dígitos de controle de fluxo, do inglês *flow control digits*, a menor unidade de controle de fluxo). Apenas o cabeçalho do *flit* tem as informações de roteamento. Assim, o resto dos *flits* que compõem um pacote deve seguir o mesmo caminho reservado para o cabeçalho. O tamanho do *flit* é um importante parâmetro quantitativo de roteadores de uma NoC.

A estratégia de armazenamento temporário a ser usada pelo roteador é outro parâmetro importante, pois podem ocorrer problemas de bloqueio HOL (*Head-of-Line*). A maioria das NoCs empregam *buffers* de entrada com estrutura de fila. O uso de uma única fila por entrada leva à menor sobrecarga de área, justificando a escolha. No entanto, uma fila de entrada única é mais suscetível ao problema de bloqueio HOL [RGR03]. Para não ocorrer este problema, filas de saída podem ser usadas, ou os já mencionados canais virtuais [FOR02], com uma maior sobrecarga de área, uma vez que isto aumenta o número total de filas no roteador. Pode-se também utilizar uma solução intermediária, filas de entrada e saída, que combinam as vantagens de ambas as estratégias.

O compromisso entre a quantidade de contenção da rede, latência dos pacotes e sobrecarga de área depende do tamanho das filas.

O algoritmo de roteamento define o caminho que será percorrido por um pacote entre a origem e o destino. De acordo com o local onde são tomadas as decisões de roteamento, é possível classificá-los em roteamento na origem ou distribuído [DYN02]. No roteamento na origem, todo o caminho é decidido no roteador de origem, enquanto no roteamento distribuído cada roteador recebe um pacote e decide para qual direção enviá-lo. De acordo com a forma como um caminho é definido para transmitir pacotes, o roteamento pode ser classificado como determinístico ou adaptativo. No roteamento determinístico, o caminho é definido apenas pelos endereços de origem e de destino. Já em algoritmos adaptativos, o caminho pode ser função do tráfego da rede ou outras grandezas de interesse [DYN02] [NI93]. Esta última classificação de roteamento pode ser subdividida em algoritmos parcialmente ou totalmente adaptativos. O roteamento parcialmente adaptativo usa apenas um subconjunto dos caminhos físicos disponíveis entre origem e destino. Os algoritmos totalmente adaptativos podem usar qualquer um dos caminhos existentes entre origem e destino.

O suporte a qualidade de serviço (QoS) é uma característica fundamental de algumas NoCs. A forma mais encontrada de garantir a QoS em NoCs é através do uso do roteamento de circuitos. A desvantagem desta abordagem é que a largura de banda pode ser desperdiçada se o caminho de comunicação não é utilizado durante todo o período em que a conexão está estabelecida. Além disso, uma vez que a maioria das abordagens combina roteamento de circuitos, com técnicas de melhor esforço, isto traz como consequência o aumento da sobrecarga de área do roteador. Canais virtuais são uma maneira de obter QoS sem comprometer a largura de banda, especialmente quando combinados com a técnica de multiplexação por divisão do tempo (TDM). Esta técnica evita que pacotes fiquem bloqueados por longos períodos, uma vez que *flits* de entradas diferentes de um roteador são transmitidos de acordo com uma alocação de tempo pré-definida associada a cada saída do roteador.

O tamanho do roteador pode ser função da utilização de canais virtuais. Não utilizar canais virtuais pode levar a uma menor área para o roteador. Isto indica a necessidade de estabelecer uma relação entre o tamanho do roteador e sobrecarga de área causada pela arquitetura de comunicação do SoC. O tamanho do roteador, o tamanho do *flit* (ou seja, a largura do canal de comunicação) e o número de portas do roteador são outros valores fundamentais para permitir a estimativa da sobrecarga de área e o desempenho máximo esperado para a comunicação intrachip. Estimativas de desempenho máximo de um roteador são obtidas com frequência a partir do produto de três valores: número de portas de roteador, tamanho do *flit* e a frequência de operação, no caso de um roteador implementado como um circuito síncrono.

2.3.3 Protocolos

O protocolo de roteamento usado em uma NoC depende da topologia da rede. Este protocolo é um conjunto de regras para o transporte de dados através da rede, com o objetivo de garantir o sucesso da entrega dos pacotes em qualquer situação.

Em NoCs a camada física corresponde à forma de comunicação entre os roteadores em *hardware*, como exemplificado na Figura 2.9, a interface *handshake* na NoC HERMES (retirada do modelo de roteador com interface *handshake* da NoC HERMES [MOR04]).

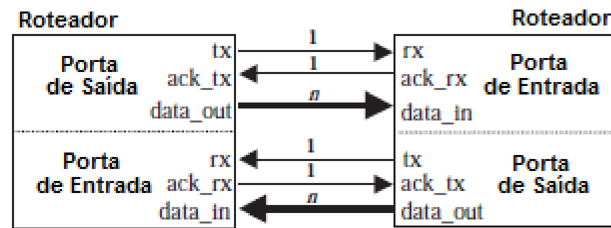


Figura 2.9 - Exemplo de interface física entre roteadores.

A largura do barramento de dados físico deve ser escolhida em função dos recursos disponíveis para roteamento e da memória disponível para implementar sistemas de armazenamento. A porta de saída no exemplo é composta pelos seguintes sinais: (1) Tx: sinal de controle que indica a disponibilidade de dados, (2) Data Out: dados a serem enviados, (3) Ack tx: sinal de controle que indica a recepção de dados bem-sucedida. A porta de entrada no exemplo é composta pelos seguintes sinais: (1) Rx: sinal de controle que indica a disponibilidade de dados, (2) Data In: dados a serem recebidos; (3) Ack rx: sinal de controle que indica a recepção de dados bem-sucedida.

Para implementar a camada de enlace de dados pode-se utilizar os protocolos como *handshake* e baseado em créditos. O protocolo *handshake* consiste de um acordo entre o transmissor e o receptor através de linhas de controle para liberação de envio de pacotes. Quando o transmissor precisa enviar dados para um roteador vizinho, ele coloca os dados no sinal de saída de dados e aciona o sinal tx. Uma vez que o roteador vizinho armazena os dados a partir do sinal de dados de entrada, este aciona sinal rx ack, e a transmissão está completa.

No protocolo baseado em créditos, o receptor anuncia a disponibilidade de espaço livre no *buffer* de entrada para que o transmissor saiba que há espaço disponível para a recepção de dados de pacotes.

A camada de rede relaciona-se com a troca de pacotes. Esta é responsável pela segmentação e remontagem de *flits*, roteamento ponto a ponto entre os roteadores e gerenciamento de contenção. Normalmente utiliza-se a técnica de roteamento de pacotes para estas funções em NoCs.

A camada de transporte é responsável por estabelecer a comunicação fim a fim entre origem e destino. Serviços como a segmentação e remontagem dos pacotes são essenciais para proporcionar uma comunicação confiável [DYN02]. Esta comunicação é implementada nas interfaces dos núcleos IPs locais.

A seguir, apresentam-se alguns conceitos sobre a NoC HERMES [MOR04] e o ambiente ATLAS, ambos desenvolvidos pelo grupo GAPH.

2.3.4 A NoC HERMES

A HERMES [MOR04] é uma NoC e uma infraestrutura associada, utilizada para gerar NoCs usando algumas topologias como malha 2D e toro 2D e diversos valores parametrizáveis, incluindo tamanho de *flit*, profundidade de *buffers* e algoritmos de roteamento.

A denominação NoC HERMES é empregada para se referir à NoC implementada com a infraestrutura HERMES e para os outros componentes da rede, como roteadores e *buffers*. Com esta infraestrutura é tipicamente possível implementar os três níveis inferiores do Modelo de referência OSI da ISO.

A NoC HERMES emprega modo de chaveamento *wormhole* e transmite pacotes divididos em *flits*. O tamanho do *flit* é parametrizável em tempo de projeto, e o tamanho máximo da carga útil de um dado pacote é de $2^n - 1$ (onde n é o tamanho do *flit*, em bits). O primeiro e o segundo *flits* de um pacote são informações de cabeçalho, sendo respectivamente, o endereço do roteador de destino, e o número de *flits* de carga útil do pacote.

A NoC HERMES pode empregar estratégias de controle de fluxo *handshake* e baseada em créditos. A implementação de Canais Virtuais necessariamente emprega controle de fluxo baseado em créditos.

O roteador da NoC HERMES contém uma lógica de controle única e um conjunto de até 5 portas bidirecionais: Leste, Oeste, Norte, Sul e Local. A lógica de controle implementa a arbitragem e o algoritmo de roteamento. A porta Local estabelece a comunicação entre o roteador e o seu núcleo IP local. As outras portas do roteador estão conectadas a roteadores vizinhos. A estrutura geral do roteador HERMES é apresentada na Figura 2.10. Cada porta contém dois canais físicos unidirecionais (incluindo linhas de dados e controles).

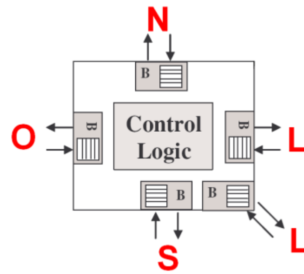


Figura 2.10 - Arquitetura do roteador HERMES. B indica os *buffers* de entrada.

Um canal físico pode dar suporte a múltiplos canais virtuais multiplexados (*lanes*). A Figura 2.11 apresenta um roteador HERMES com duas *lanes* por canal físico. A porta local do roteador não é multiplexada, porque se presume que o núcleo conectado a esta porta não é capaz de receber ou enviar mais de um pacote simultaneamente. Apesar da multiplexação de canais físicos poderem aumentar o desempenho do roteamento [RGW01], é importante manter um compromisso entre desempenho, complexidade e área de roteador.

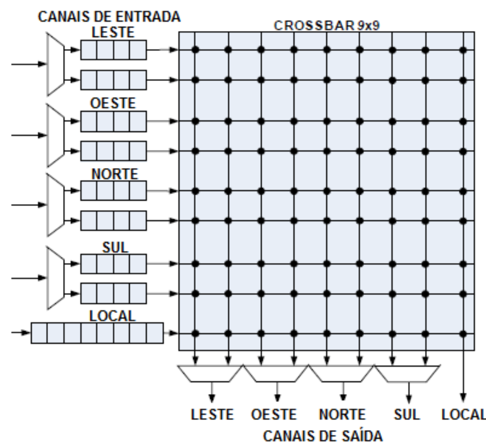


Figura 2.11 - Roteador HERMES com dois canais virtuais por canal físico.

Cada porta de entrada tem um *buffer* para armazenamento temporário do *flit*, com uma profundidade d. Quando n *lanes* são utilizados, um *buffer* com $\frac{d}{n}$ profundidade é associado a cada *lane*. A porta de entrada recebe os *flits*, e diminui o número de créditos do *lane*. Quando uma porta de saída transmite um *flit*, este *flit* é removido do *buffer* e o número de créditos é incrementado. A disponibilidade de espaço é informada a roteadores vizinhos através do sinal "*credit_o*".

Múltiplos pacotes podem chegar simultaneamente em um determinado roteador. A arbitragem centralizada do *round-robin* garante acesso de apenas um destes pacotes por vez à lógica de roteamento. A prioridade de um *lane* é em função do último *lane* que teve uma requisição de roteamento concedido.

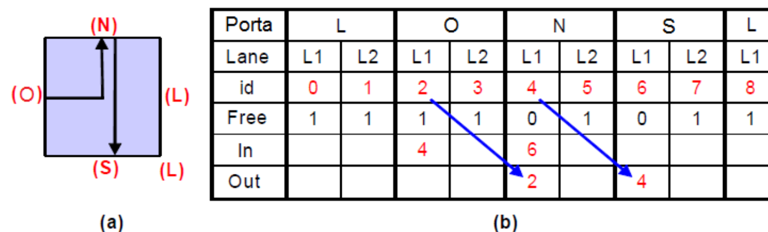
Se a requisição do pacote de entrada é concedida pelo árbitro, um algoritmo de roteamento é executado para ligar a porta de entrada à porta de saída correta. Existem pelo menos quatro

algoritmos de roteamento disponíveis para uso (a seleção é realizada em tempo de projeto): um determinístico (ordenado por dimensão ou XY) e três parcialmente adaptativos (*West First*, *North Last* e *Negative First*). Quando o algoritmo retorna uma porta de saída ocupada, o *flit* de cabeçalho, bem como todos os *flits* subsequentes do pacote serão bloqueados.

Quando o algoritmo de roteamento encontra uma porta de saída livre, a conexão entre o *lane* de entrada e o *lane* de saída é estabelecida e uma tabela de roteamento é atualizada. Três vetores compõem a tabela de roteamento: *in*, *out* e *free*. O vetor *in* conecta um *lane* de entrada a um *lane* de saída. O vetor *out* conecta um *lane* de saída a um *lane* de entrada. O vetor *free* é responsável por modificar o estado do *lane* de saída de livre (1) para ocupado (0). Os vetores in e out recebem um identificador construído pela combinação do número da porta (np) e o número do *lane* (nl), conforme apresentado na seguinte equação:

$$id = (np\ x\ n^{\underline{o}}\ de\ lanes) + nl \qquad (2.7)$$

As portas Leste, Oeste, Norte, Sul e Local são numeradas de 0-4 respectivamente. Os *lanes* L1, L2 ... Ln são numerados de 0 a n-1, respectivamente. Considere-se um roteador com dois *lanes* por canal físico. O *lane* L1 da porta Norte tem o identificador 4 ($((2 \times 2) + 0)$) e o *lane* L2 tem o identificador 5 ($((2 \times 2) + 1)$). A tabela de roteamento apresentada na Figura 2.12 (b) representa a situação ilustrada na Figura 2.12 (a). Considerando a porta Norte. O *lane* de saída L1 estará como ocupado (*free*='0') e estará sendo conduzido pelo *lane* de entrada L1 da porta Oeste (out='2'). O *lane* de entrada L1 está conduzindo o *lane* de saída L1 da porta Sul (in='6'). A tabela de roteamento contém redundâncias de informações, mas esta organização é útil para melhorar a eficiência do algoritmo de roteamento, conduzindo a um acesso mais eficiente e paralelo às informações de conexão e roteamento entre portas de um dado roteador.



Porta	L		O		N		S		L
Lane	L1	L2	L1	L2	L1	L2	L1	L2	L1
id	0	1	2	3	4	5	6	7	8
Free	1	1	1	1	0	1	0	1	1
In			4		6				
Out					2		4		

Figura 2.12 - Tabela de roteamento. a) chaveamento; b) mudança da tabela.

Após o roteamento, a porta de saída é responsável pela alocação da largura de banda entre os n *lanes*. Cada *lane*, se possuir *flits* para transmitir e créditos para a transmissão, usa pelo menos $\frac{1}{n}$ da largura de banda do canal físico. Se um único *lane* satisfizer esta condição, ele usará toda a largura de banda do canal físico.

A conexão é finalizada após a transmissão de todos os *flits* que compõem o pacote. O roteador possui um contador de *flits* para cada *lane* de saída. O contador de um *lane* específico é inicializado quando o segundo *flit* de um pacote chega, indicando o número de *flits* do *payload*. O contador é decrementado para cada *flit* subsequente. Quando o valor do contador chega à zero, a conexão é finalizada e o vetor *free* correspondente à posição do *lane* de saída vai para (*free*='1').

O roteador pode simultaneamente receber requisições para estabelecer até cinco conexões. A lógica de arbitragem é utilizada para garantir acesso a uma porta de saída quando uma ou mais portas de entrada requisitam uma conexão ao mesmo tempo. Um esquema de arbitragem dinâmico baseado em prioridades é utilizado. A prioridade de uma porta é atribuída com base na última porta que teve o roteamento concedido (a já mencionada estratégia *round-robin*). Por exemplo, se a porta de entrada local (índice 4) foi a última porta a ter o roteamento concedido, a porta Leste (índice 0) vai ter a prioridade máxima, seguida pelas portas Oeste, Norte, Sul e Local. Este método garante que todas as portas de entrada eventualmente tenham suas requisições atendidas, evitando a ocorrência de *starvation*. A lógica de arbitragem espera quatro ciclos de relógio para tratar de uma nova requisição, sendo este tempo necessário para o roteador executar o algoritmo de roteamento.

A taxa de transferência dos pacotes pelo roteador HERMES é dada pela equação, que apresenta a latência mínima para transferir um pacote da origem para o destino, em ciclos de relógio.

$$latência\ mínima = \left(\sum_{i=1}^{n} R_i\right) + P \qquad (2.8)$$

Nesta equação:

• n é o número de roteadores no caminho (origem e destino incluídos);

• R_i é a latência do roteador (arbitragem e roteamento), pelo menos 6 ciclos de relógio nesta implementação;

• P é o tamanho do pacote.

Para um roteador com 5 portas, *flits* de 8 bits e *clock* de 25MHz, a vazão máxima do roteador HERMES é 1Gbits/s, (5 portas x 8 bits x 25MHz).

A topologia da NoC HERMES é definida pela estrutura de conexão dos roteadores, sendo que esta assume que cada roteador tem um conjunto de portas bidirecionais ligadas a seus roteadores vizinhos e ao seu núcleo IP. A topologia mais comumente utilizada é a malha 2D. Nesta topologia, cada roteador possui um número diferente de portas, dependendo da sua localização na rede, como mostrado na Figura 2.13, onde os endereços dos roteadores são indicados pelas posições X, Y na rede e C indica os núcleos IP locais. Como exemplo, o roteador central (índice 11) possui todas as 5 portas. Já os roteadores nos cantos da rede como o de índice 20, possuem apenas 3 portas.

O roteador HERMES pode utilizar as topologias Toro 2D ou outras similares. No entanto, a modificação da topologia implica em alterações nas ligações entre os roteadores e, mais importante, no algoritmo de roteamento.

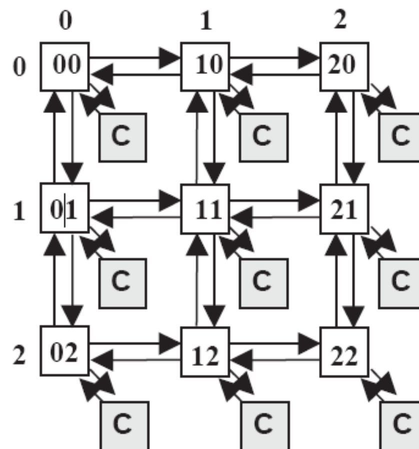


Figura 2.13 - Estrutura da topologia Malha 2D 3x3.

2.3.5 O Ambiente ATLAS

O ambiente ATLAS [GAP01], proposto pelo grupo de pesquisa GAPH automatiza vários dos processos relacionados ao fluxo de projeto de algumas das NoCs propostas pelo grupo, dentre elas várias versões da NoC HERMES.

Esta ferramenta de código aberto é de um dos únicos ambientes desta natureza de domínio público. A partir do controle de *downloads* do ambiente no site do grupo GAPH, sabe-se que existem interessados no ATLAS em 29 países dos 5 continentes.

A Figura 2.14 ilustra a interface principal do ambiente ATLAS, a qual permite invocar ferramentas que compõem as etapas do fluxo de projeto. A interface do ambiente ATLAS é dividida em três blocos: (1) barra de *menus*, (2) ferramentas e (3) desenvolvedor.

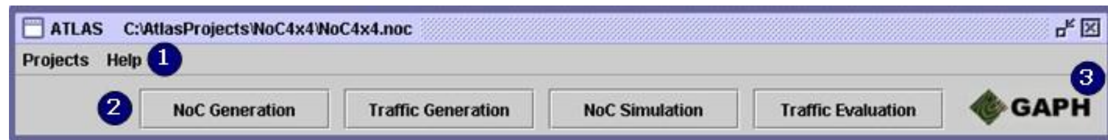


Figura 2.14 - Janela inicial do ambiente ATLAS.

O *menu Projects* permite ao usuário criar ou abrir um projeto de NoC. No caso de criação de um novo projeto, o usuário deve selecionar o tipo de NoC a ser criado. Das mais de uma dezena de tipos de NoCs já criados pelo GAPH apenas cinco estão integradas na última versão do ambiente: HERMES[MOR04], MERCURY[BAS06], HERMES-TU[SCH07], HERMES-TB[SCH07] e HERMES-SR .

Abaixo, detalhada-se as etapas de uso do ambiente para criar e avaliar instâncias de NoCs.

2.3.5.1.1 NoC Generation – MAIA

A parametrização dos roteadores e a geração manual da rede a partir destes é um processo passível de erros, devido à grande quantidade de fios que ligam os roteadores entre si e ao núcleo local. A automatização desse processo foi a principal motivação para o desenvolvimento do ambiente ATLAS, que inicialmente recebeu a denominação de ferramenta MAIA.

A ferramenta *NoC Generation* gera a descrição em VHDL de todos os módulos que compõem a NoC, podendo gerar também um *testbench* básico, um conjunto de arquivos descritos em SystemC e/ou VHDL.

Os arquivos VHDL sintetizáveis são adequados como entrada para ferramentas clássicas de síntese de FPGA ou de ASICs. Os arquivos VHDL sintetizáveis e os arquivos *testbench* são especialmente projetados para entrada no simulador Modelsim SE da Mentor, usando co-simulação VHDL/SystemC. Ao exibir a janela de controle desta ferramenta (via botão *NoC Generation*, índice 2 da Figura 2.14) surge a janela do gerador da NoC HERMES, ilustrada na Figura 2.15.

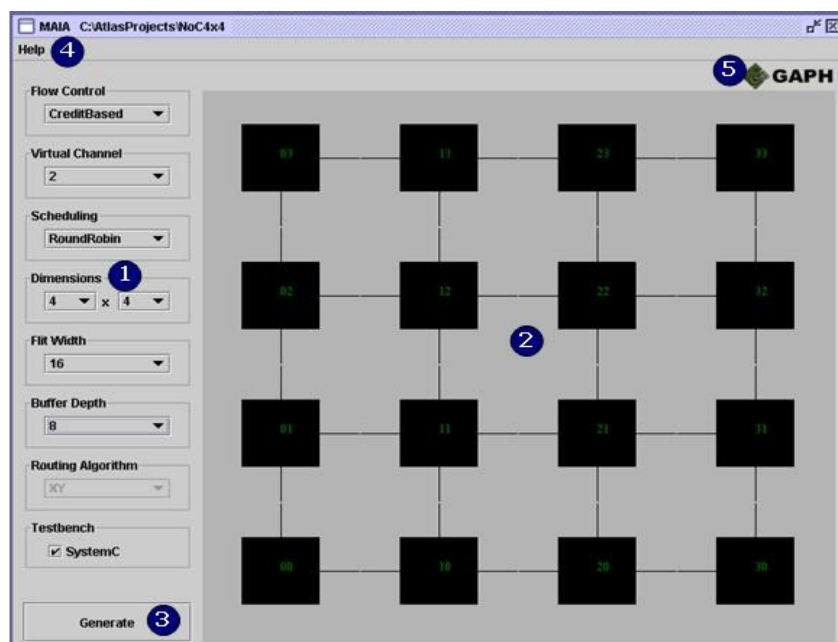


Figura 2.15 - Janela principal da ferramenta de geração da NoC HERMES.

Na ferramenta de geração da NoC HERMES, o usuário pode configurar os parâmetros contidos na barra de parametrização (índice 1 na Figura 2.15), que são:

***Controle de Fluxo*:** O controle de fluxo é um mecanismo que possibilita ao transmissor saber se o receptor está habilitado a receber dados (o receptor pode estar com os *buffers* de recepção cheios ou com algum problema momentâneo que o impossibilita de receber dados). Como já

discutido anteriormente, MAIA possui duas opções de controle de fluxo: *handshake* e *baseado em créditos*. A diferença entre estes controles de fluxo é ilustrada pelo esquema das duas interfaces da Figura 2.16**.**

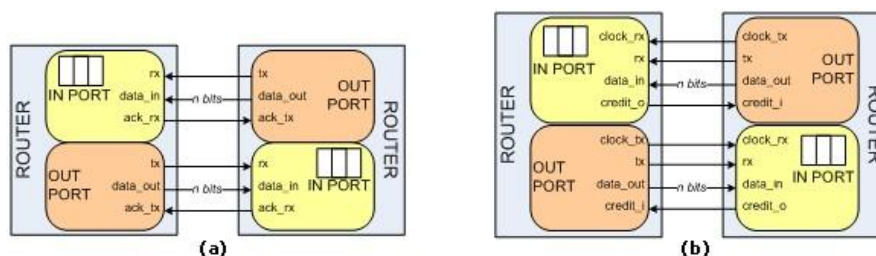


Figura 2.16 - Interface entre roteadores com o (a) *Handshake* e (b) *Credit Based*.

Canais Virtuais**:** Um canal físico pode ser compartilhado por canais virtuais. Um canal virtual possui seu próprio *buffer*, mas compartilha a largura de banda do canal físico com outros canais virtuais. Na rede HERMES, apenas o controle de fluxo *credit based* foi estendido para atender a este requisito [MEL05]. Portanto, o uso de canais virtuais está limitado ao uso do controle de fluxo *credit based*. Na versão atual da ATLAS, um canal físico pode ser compartilhado por 1 (sem canais virtuais), 2 ou 4 canais virtuais.

Escalonamento**:** Quando um canal físico é multiplexado por canais virtuais, é necessário uma política de escalonamento que divida a largura de banda do canal físico entre os canais virtuais. A MAIA tem duas opções de escalonamento: *Round-Robin* e *Priority*.

- *Round-Robin*: divide a largura de banda do canal físico em um certo número de partes iguais, número dado pelo total de canais virtuais com dados para transmissão em um dado enlace físico.
- *Priority*: cada canal virtual que compartilha o canal físico recebe uma prioridade estática. Quando mais de um canal virtual tem dados para transmitir, o canal virtual com prioridade maior utilizará 100% da largura de banda do canal físico, mesmo se os outros canais virtuais estão esperando há mais tempo.

Dimensões**:** A dimensão da rede é dividida em dois parâmetros que representam o número de roteadores na coordenada X e o número de roteadores na coordenada Y. MAIA gera redes malha 2D com dimensões m x n, sendo m e n valores entre 2 e 16 roteadores em saltos discretos.

Largura do Flit**:** A largura do canal define o número bits de dados que são transmitidos/recebidos em um canal físico. Na HERMES a largura do canal corresponde a um *flit* (*flow control unit* - menor unidade sobre o qual se realiza o controle de fluxo). Na versão atual da ATLAS o *flit* possui 8, 16, 32 ou 64 bits.

Profundidade do Buffer**:** Define o número de *flits* que o *buffer* pode armazenar. ATLAS atualmene permite a escolha de 4, 8, 16 ou 32 *flits*. Profundidades maiores reduzem a contenção, aumentando o desempenho global e resultando em maior consumo de área. Recomenda-se o uso de *buffers* de 8 ou 16 *flits*.

Algoritmo de Roteamento**:** O algoritmo de roteamento define como é realizada a transferência de dados entre um roteador origem e um roteador destino. Atualmente, ATLAS dá suporte aos algoritmos de roteamento XY, *West-First*, *Negative-First* e *North-Last*. As operações destes algoritmos de roteamento são descritas, por exemplo, em [GLA94].

Testbench**:** ATLAS gera *testbenches* descritos em SystemC/VHDL que possibilitam a validação da rede gerada. Os *testbenches* consistem de código para leitura de arquivos de entrada, escritas em arquivos de saída e injeção de dados na rede. Cada arquivo de entrada possui uma descrição de pacotes que um núcleo IP local deve enviar para a rede. Cada arquivo de saída descreve os pacotes que chegaram a um determinado núcleo, bem como dados de latência dos mesmos na rede.

Ao estar satisfeito com as parametrizações o usuário pode clicar no botão *Generate* (índice 3 da Figura 2.15) para gerar as descrições VHDL sintetizáveis da NoC e os arquivos de controle de simulação (*testbench*).

2.3.5.1.2 Traffic Generation - Traffic Mbps.

A segunda ferramenta do ambiente ATLAS, *Traffic Generation*, pode gerar arquivos de tráfego que simulam o comportamento de módulos processadores conectados a algum ponto da NoC, e que durante a simulação produzem tráfego sintético de dados a ser injetado na rede. A Figura 2.17 apresenta a janela principal desta ferramenta. Para utilizá-la, o usuário deve clicar no botão *Traffic Generation* na janela principal da ATLAS (Figura 2.14).

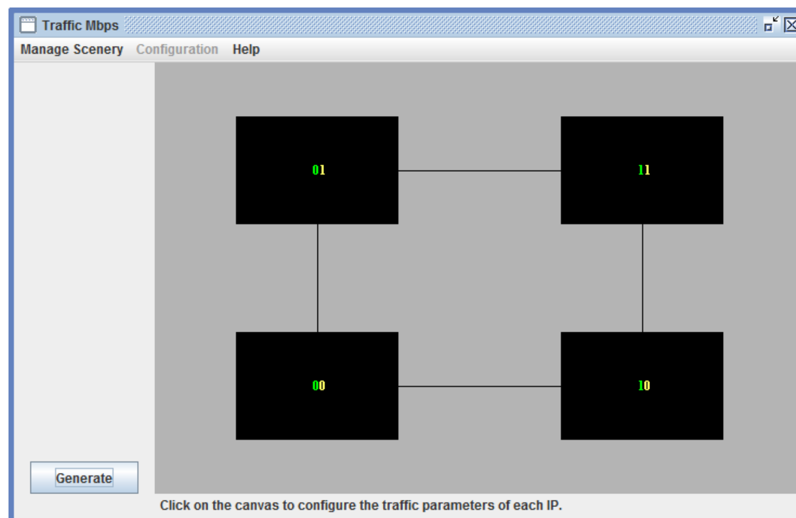


Figura 2.17 - Janela principal da ferramenta *Traffic Generation*.

Criar ou abrir um cenário de tráfego é o primeiro passo na ferramenta *Traffic Generation*. Sem este passo, as outras opções da ferramenta não são habilitadas. Para criar um novo cenário de tráfego, o usuário deve clicar sobre o *menu Manage Scenery* e selecionar a opção *New Scenery*. Pode-se também abrir um cenário de tráfego já existente.

O *menu Configuration* (Figura 2.18) permite ao usuário definir a Configuração Padrão dos parâmetros utilizados pelas interfaces de rede durante a geração do tráfego. Esta configuração é chamada de padrão porque é atribuída a todas as interfaces dos roteadores da rede. Esta configuração pode ser mudada individualmente para cada roteador. A Figura 2.18 apresenta os três possíveis formatos de janelas para a escolha da configuração padrão. Os seis parâmetros apresentados na parte superior de todas as janelas são independentes. Por outro lado, os parâmetros apresentados na parte inferior das janelas dependem da escolha do tipo de distribuição (respectivamente uniforme, normal ou pareto *on*/*off*).

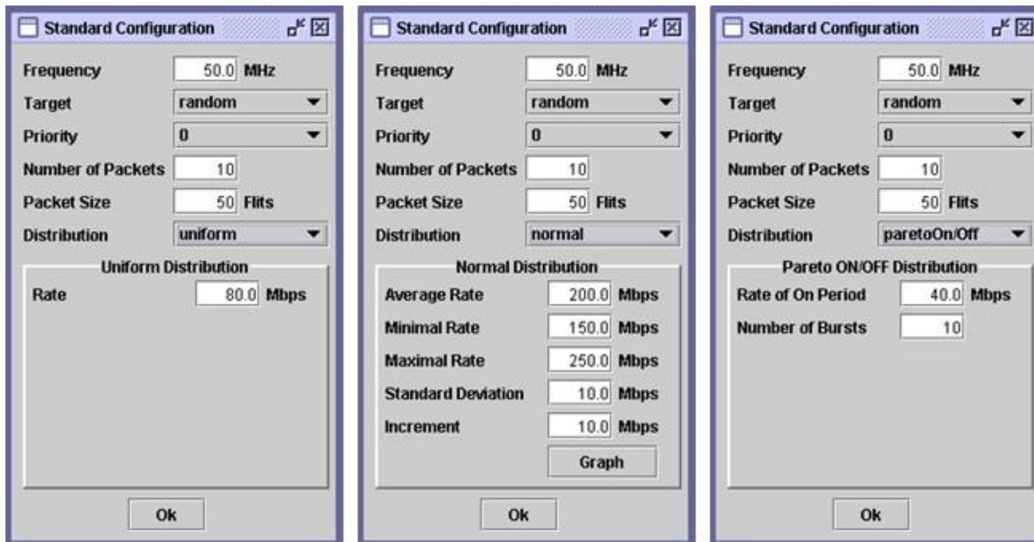


Figura 2.18 - Configuração padrão para tráfego (a)uniforme; (b)normal; (c)pareto.

2.3.5.1.3 NoC Simulation - ModelSim®

A terceira ferramenta do ambiente provê controle da simulação da NoC, executada usando o simulador comercial ModelSim da Mentor. Se o usuário não tem acesso a este simulador ou se ele deseja utilizar outro simulador, a fase de simulação não deve ser executada dentro do ambiente ATLAS. Tudo que é necessário é um ambiente de simulação que aceite VHDL e SystemC. Ajuda se o simulador aceitar *scripts* escritos na linguagem de programação Tcl. Assim, os *scripts* gerados automaticamente dentro da ATLAS podem também ser utilizados. Na interface (Figura 2.19), o usuário configura a simulação da NoC. Esta janela é dividida em quatro blocos: (1) parametrização, (2) simulação, (3) *help* e (4) desenvolvedor.

Figura 2.19 - Janela principal para controlar a ferramenta NoC Simulation.

Na Parametrização, escolher o nome do cenário que deseja simular (pode existir qualquer número deles), o caminho do diretório onde o simulador ModelSim foi instalado que deve conter o subdiretório, que possui o arquivo "modelsim.exe", o tempo de simulação da rede (um número inteiro em milissegundos) e se os resultados da simulação devem gerar dados para permitir a avaliação interna da rede ou não. A avaliação interna consiste em coletar dados que possibilitem analisar as ligações que conectam os roteadores entre si, enquanto a avaliação externa consiste em analisar somente o comportamento das portas locais da rede.

A simulação começa quando o usuário clica sobre o botão Ok (índice 2 da Figura 2.19). A *NoC Simulation* utiliza *scripts* para manipular os arquivos do cenário de tráfego, invocar o simulador ModelSim, compilar e simular a NoC.

2.3.5.1.4 Traffic Evaluation - Traffic Measurer

A última ferramenta do ambiente, *Traffic Evaluation* foi desenvolvida para facilitar a análise dos resultados gerados durante a simulação da rede. O primeiro passo nesta ferramenta é a seleção do cenário de tráfego a ser analisado. O usuário seleciona o cenário de tráfego para análise e o abre (Figura 2.20).

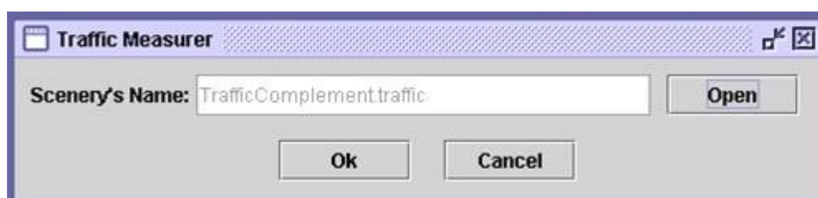


Figura 2.20 - Interface para a seleção do cenário de tráfego a ser analisado.

Após a seleção do cenário de tráfego, uma janela especial de controle é apresentada. Esta interface (Figura 2.21) é dividida em três blocos básicos: (1) barra de *menus*; (2) barra de parametrização; e (3) área de visualização que permite ao usuário identificar a dimensão da rede, os endereços e as conexões dos roteadores.

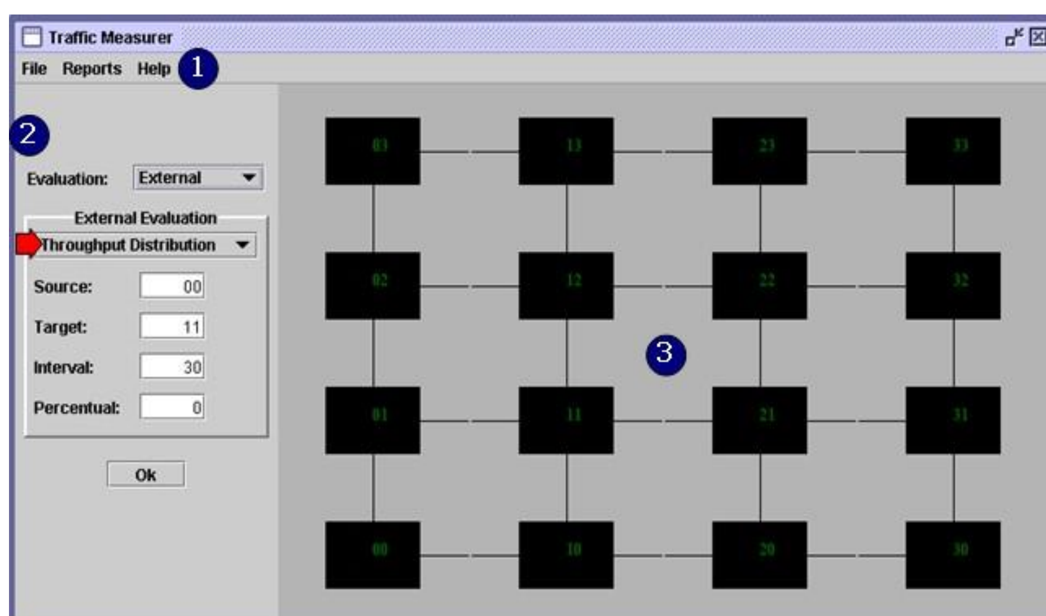


Figura 2.21 - Interface principal da ferramenta *Traffic Measurer*.

Na barra de *menus*, o usuário pode selecionar diversos parâmetros associados ao funcionamento externo ou interno dos serviços de entrega de mensagens da rede. Isto inclui, mas não é limitado à avaliação de dados de vazão e latência para pontos específicos da rede ou para suas interfaces externas. Para visualizar tais resultados, é possível gerar automaticamente diferentes tipos de gráficos, tabelas e dados textuais. Isto permite a avaliação de desempenho da NoC de forma detalhada por ambos, especialistas de redes ou usuários da rede.

A barra de *menus* possui três submenus:

O submenu *Reports* possui três tipos de relatórios:

***Links Analysis Report*:** Exibe um relatório (Figura 2.22) contendo informações (por exemplo, número de pacotes transmitidos, número de bits por ciclo de relógio) sobre todos os canais (enlaces) da rede. O relatório contendo as informações sobre os canais da rede faz parte da avaliação interna. Portanto, o relatório *Links Analysis Report* somente é disponibilizado quando a opção *Internal Evaluation* é selecionada na fase de simulação.

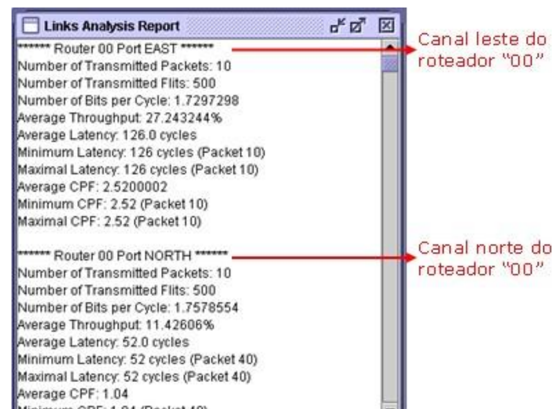
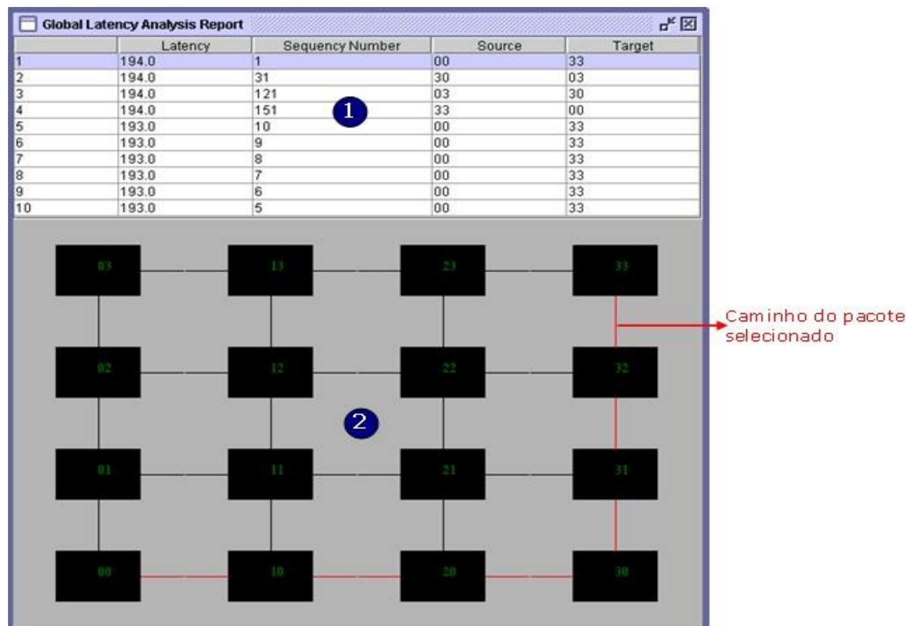


Figura 2.22 - Relatório da análise dos canais (enlaces) da rede.

Latency Analysis Report**:** Exibe uma lista ordenada dos pacotes com maior ou menor latência. O relatório de latência é dividido em dois blocos: (i) tabela (índice 1 da Figura 2.23) e (ii) área de visualização (índice 2 da Figura 2.23). A tabela informa a latência, o número de sequência e os núcleos origem e destino dos pacotes que compõem a lista. A área de visualização possibilita ao usuário verificar o caminho percorrido pelo pacote selecionado na tabela. Para selecionar um pacote, o usuário deve clicar sobre a linha da tabela que contém as informações sobre o mesmo.



	Latency	Sequency Number	Source	Target
1	194.0	1	00	33
2	194.0	31	30	03
3	194.0	121	03	30
4	194.0	151	33	00
5	193.0	10	00	33
6	193.0	9	00	33
7	193.0	8	00	33
8	193.0	7	00	33
9	193.0	6	00	33
10	193.0	5	00	33

Figura 2.23 - Relatório de pacotes ordenados por valores de latência.

Global Report**:** apresenta um resumo da avaliação externa. Este resumo é apresentado no formato HTML, conforme apresentado na Figura 2.24. Os resultados da avaliação externa são agrupados em fluxos. Um fluxo é um conjunto de pacotes que possuem o mesmo núcleo origem e destino.

Hermes NoC
Flow Control: CreditBased (1 cycle per flit)
Virtual Channels: 1
Scheduling: RoundRobin
Dimensions: 4 x 4 (16 routers)
Flit Width: 16
Buffer Depth: 16
Routing Algorithm: AlgorithmXY

		Generated			Measured									Graphs	
			Throughput (%)	Latency (clock cycles)		Throughput (%)				Latency (clock cycles)					
Source	Target	Packets	Average	Ideal	Packets	Average	Standard Deviation	Minimum	Maximal	Average	Standard Deviation	Minimum	Maximal	Latency Distribution	Throughput Distribution
00	33	10	10.0	84.0	10	10.0	0.0	10.0	10.0	193.1	0.30	193.0	194.0	VIEW	VIEW
10	23	10	10.0	74.0	10	10.0	0.0	10.0	10.0	133.1	0.30	133.0	134.0	VIEW	VIEW
20	13	10	10.0	74.0	10	10.0	0.0	10.0	10.0	133.1	0.30	133.0	134.0	VIEW	VIEW
30	03	10	10.0	84.0	10	10.0	0.0	10.0	10.0	193.1	0.30	193.0	194.0	VIEW	VIEW
01	32	10	10.0	74.0	10	10.0	0.0	10.0	10.0	131.1	0.30	131.0	132.0	VIEW	VIEW
11	22	10	10.0	64.0	10	10.0	0.0	10.0	10.0	70.1	0.30	70.0	71.0	VIEW	VIEW
21	12	10	10.0	64.0	10	10.0	0.0	10.0	10.0	70.1	0.30	70.0	71.0	VIEW	VIEW
31	02	10	10.0	74.0	10	10.0	0.0	10.0	10.0	131.1	0.30	131.0	132.0	VIEW	VIEW
02	31	10	10.0	74.0	10	10.0	0.0	10.0	10.0	131.1	0.30	131.0	132.0	VIEW	VIEW
12	21	10	10.0	64.0	10	10.0	0.0	10.0	10.0	67.1	0.30	67.0	68.0	VIEW	VIEW
22	11	10	10.0	64.0	10	10.0	0.0	10.0	10.0	67.1	0.30	67.0	68.0	VIEW	VIEW
32	01	10	10.0	74.0	10	10.0	0.0	10.0	10.0	131.1	0.30	131.0	132.0	VIEW	VIEW
03	30	10	10.0	84.0	10	10.0	0.0	10.0	10.0	193.1	0.30	193.0	194.0	VIEW	VIEW
13	20	10	10.0	74.0	10	10.0	0.0	10.0	10.0	133.1	0.30	133.0	134.0	VIEW	VIEW
23	10	10	10.0	74.0	10	10.0	0.0	10.0	10.0	133.1	0.30	133.0	134.0	VIEW	VIEW
33	00	10	10.0	84.0	10	10.0	0.0	10.0	10.0	193.1	0.30	193.0	194.0	VIEW	VIEW
		160			160										

1 2 3 Dados gerados 4 Dados coletados 5 Gráficos

Figura 2.24 - Relatório global.

Na barra de parametrização, o usuário define como a avaliação dos resultados será realizada a partir da configuração dos campos contidos na barra de parametrização (índice 2 da Figura 2.21). O parâmetro principal a ser configurado é o tipo de avaliação (*Evaluation*). Este parâmetro é dito principal porque a configuração dos outros parâmetros depende dele. Existem dois tipos de avaliação: avaliação externa e avaliação interna.

2.4 Plataformas de Prototipação de Hardware

A prototipação do sistema proposto será realizada em um (FPGAs), da Xilinx, inicialmente empregando a plataforma XUP-V2PRO, produzida pela empresa Digilent.

Uma *plataforma de prototipação de hardware* é um sistema composto de uma ou mais placas de circuito impresso sobre a qual é possível construir um protótipo do *hardware* de um sistema que será implementado em algum produto. A este *hardware* de prototipação normalmente associam-se diversos outros recursos, tais como cabos de comunicação ou interfaces que permitem controlar/comandar a plataforma a partir de um computador hospedeiro, uma diversidade de interfaces de comunicação com dispositivos periféricos, tais como mouse, teclado, até discos ou redes de computadores e *software* de apoio. Praticamente toda plataforma de prototipação de *hardware* moderna é construída em torno de circuitos integrados reconfiguráveis, dispositivos cuja funcionalidade em *hardware* pode ser definida pelo usuário final, através do preenchimento de uma memória interna de controle que define a funcionalidade do dispositivo reconfigurável. Dentre os diversos tipos de circuitos integrados reconfiguráveis, destacam-se os FPGAs baseados em RAM, inventados na década de 80 pela Xilinx. Estes dispositivos apresentam capacidade de serem reconfigurados infinitas vezes e possuem hoje densidades equivalentes a dezenas de milhões de portas lógicas.

Dada as enormes flexibilidade e capacidade de FPGAs, outro uso significativo de plataformas de prototipação de *hardware* é em treinamento, durante o aprendizado de projeto de circuitos.

2.4.1 A Plataforma XUP-V2PRO

O *Xilinx University Program* (*XUP) Virtex-II Pro Development System* [XIL01] fornece uma plataforma de prototipação de *hardware* avançada (na época de seu lançamento), que consiste em um FPGA da família Virtex-II Pro rodeada por uma coleção abrangente de componentes periféricos que podem ser usados para criar um sistema razoavelmente complexo. A placa é recomendada pelo XUP para todos os níveis de programas educacionais em engenharia.

A *V2-Pro* dá suporte a ferramentas de projeto difundidas, incluindo a *ISE Foundation*, o *ChipScope-Pro*, o *Embedded Developer's Kit* (EDK), e o *System Generator*.

Mesmo existindo plataformas mais atuais como a *Virtex4* e a *Virtex5*, a escolha pela utilização da Virtex-II Pro se da pelo fato de o Decodificador M-JPEG original já ter sido desenvolvido nesta plataforma, necessitando assim de poucas modificações para o seu pleno funcionamento, além do projeto já contar com a UCF pra a *Virtex-II Pro*. Outro fato importante para manter a mesma plataforma foi evitar possíveis incompatibilidades das versões dos IPs.

3 DECODIFICADOR M-JPEG

Este Capítulo apresenta em detalhes como o decodificador M-JPEG foi implementado por [MAN08], discutindo todos os componentes na Seção 3.1. Para alguns componentes, é importante ter o embasamento teórico da compressão JPEG, apresentada no Capítulo 2.

Todo o sistema original está descrito em VHDL. Usou-se a ferramenta ModelSim para e simulação e a ferramenta ChipScope para a verificação em hardware. Para a prototipação, utilizou-se o ambiente EDK 8.2 da Xilinx, sendo que a geração dos *buffers* utilizou o software CoreGen, da mesma empresa.

O *hardware* decodifica os dados de imagens comprimidas com o processo básico JPEG. Os diferentes estágios do processo de decodificação foram integrados em um *pipeline,* o que permite o processamento de múltiplos blocos de imagem simultaneamente. Este *hardware* foi descrito para o FPGA Virtex-II Pro XC2VP30, utilizando frequência de operação de 100 MHz. O desempenho e precisão são comparáveis a um decodificador desenvolvido em uma aplicação para PC com relógio de 1.5 GHz [MAN08].

3.1 Componentes (Módulos)

A Figura 3.1 ilustra como o sistema principal de decodificação, cuja maior parte é o decodificador JPEG, foi desenvolvido.

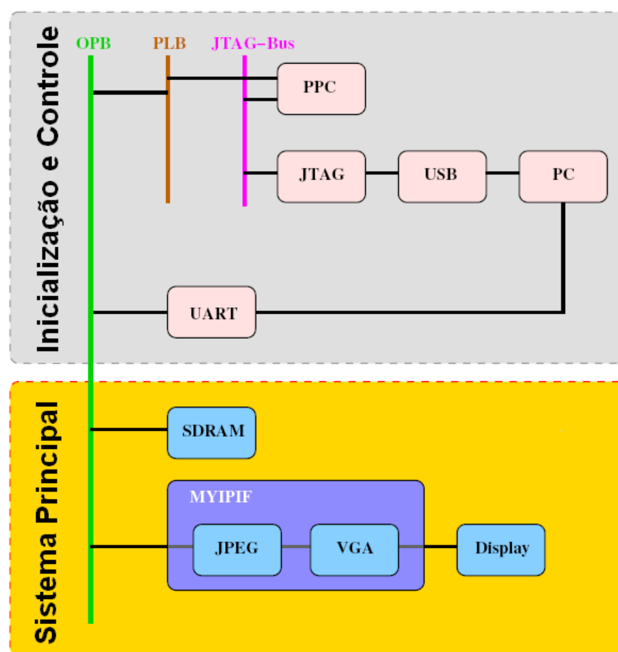


Figura 3.1 - Diagrama de blocos da arquitetura Decodificador-Controle.

Como mostrado na Figura 3.1, o sistema é controlado por um processador PowerPC, onde se faz a inicialização e o controle do sistema, que está conectado a um PC através de uma interface JTAG via cabo USB. Por outro lado, o PowerPC conecta-se ao restante do *hardware* via o barramento Core Connect PLB, que estase conecta ao barramento Core Connect OPB através de uma ponte (*plb2opb_bridge*). Os dados de imagem ou vídeo são primeiramente gravados na memória SDRAM externa através de comunicação serial (via UART) ou USB.

A interface MyIPIF busca dados de imagem da memória SDRAM para fazer a decodificação e enviar ao monitor. O sistema de decodificação é implementado pelo componente JPEG dentro da MyIPIF. Os dados de imagem são lidos a partir da SDRAM e recebidos através do barramento OPB em palavras de 32 bits. O OPB é configurado para operar em modo rajada, fornecendo uma nova palavra a cada ciclo de relógio, até que o *buffer* de entrada esteja completamente cheio. Novos dados não são solicitados até que a lógica do *buffer* de entrada solicite novos dados. Desta forma, na maioria das vezes, o OPB estará livre para ser usado, o que facilita atender os requisitos em eventuais modificações realizadas no sistema (e.g.: adaptação do JPEG para operar como MPEG). O componente JPEG ou JPEG *Decoder* será detalhado na Seção JPEG Decoder (3.1.2), junto com os processos de decodificação.

O decodificador deve fornecer os dados para a interface VGA de forma sincronizada. Portanto, o decodificador tem que ser rápido o suficiente para sempre fornecer dados válidos para a VGA, que opera a 60 Hz. Por conseguinte, uma imagem precisa ser decodificada em menos de $\frac{1}{60}$ s.

Como mostrado na Figura 3.1, a interface MyIPIF abrange os componentes do decodificador e a VGA, bem como o controle de fluxo entre os componentes. A próxima Seção detalha estrutura e função desta interface.

3.1.1 MyIPIF

A interface MyIPIF, como abordado anteriormente, é dividida em dois componentes, o JPEG *Decoder*, que possui todos os processos necessários para a decodificação JPEG e o componente VGA, responsável por exibir as imagens decodificadas em um monitor. Sua estrutura está representada na Figura 3.2.

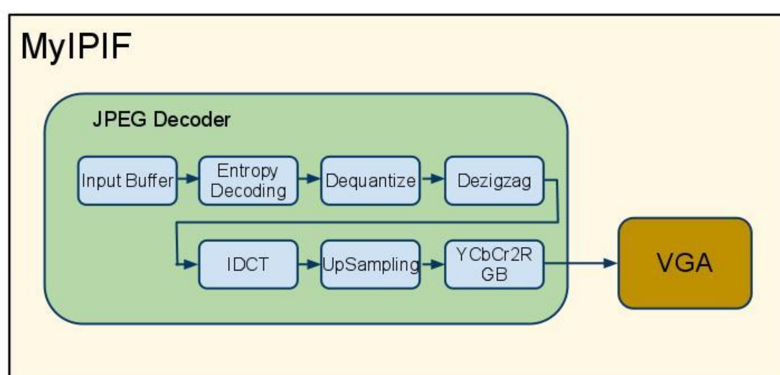


Figura 3.2 - Estrutura do MyIPIF.

MyIPIF possui uma máquina de estados finita responsável pela leitura dos dados da memória externa SDRAM, ilustrada na Figura 3.3.

No estado *idle*, a entidade "escuta" o OPB, à espera de comandos de configuração. Tão logo o decodificador requisite novos dados, aciona-se o sinal *jpeg_ready*, que irá fazer a máquina ir para o próximo estado. No entanto, com o comando *go*[9] recebido da aplicação, o usuário pode forçar a máquina de estados a ficar no estado *idle*. No estado *Master Read* 1 a entidade solicitará o barramento ao árbitro OPB. Se o barramento for concedido (*OPB_MGrand*), a máquina de estados muda para o estado *Master Read* 2, onde ele solicita dados da SDRAM até que o *buffer* de entrada esteja cheio e *jpeg_ready* recebe o valor zero, fazendo com que a máquina de estados retorne ao estado *idle*. Enquanto estiver no estado *Master Read* 1, o OPB está bloqueado para outros componentes.

[9] Quando conectado ao PC via UART, o usuário pode enviar comandos para a aplicação. A lista de comandos completa encontra-se no ANEXO C.

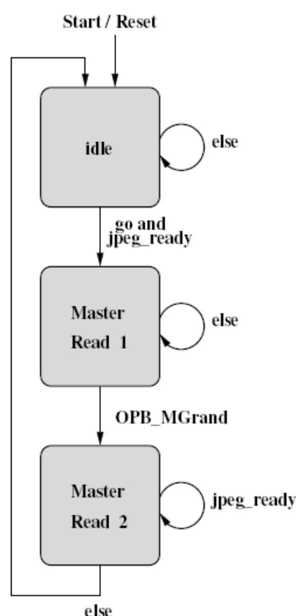


Figura 3.3 - Máquina de estados finita para iniciar um ciclo de leitura.

O barramento OPB fornece os dados em palavras de 32 bits. O primeiro dado é indicado pelo endereço fornecido pela MyIPIF. Após isto, a cada ciclo de relógio envia-se o próximo dado de 32 bits da sequência. Quando o *buffer* de entrada do decodificador está totalmente preenchido, a máquina de estados memoriza o último endereço lido. Então, na próxima vez que os dados forem solicitados pelo decodificador, a máquina de estados retoma a leitura a partir desta posição. No entanto, se o decodificador sinalizar que recebeu o marcador de final da imagem (EOI), o ciclo seguinte começa a ler a partir do endereço base novamente.

O próximo passo, depois de ler os dados fornecidos pelo barramento OPB, é decodificar estes dados, processo realizado pelo componente JPEG *Decoder*.

3.1.1.1 Controle de fluxo

Ao utilizar um controle de fluxo, a aplicação garante que somente os dados válidos sejam processados e que nenhum dado seja perdido ou reprocessado. Isto torna a aplicação mais segura e mais rápida. Para esclarecer melhor o controle de fluxo, as próximas seções descrevem os métodos empregados para tal.

3.1.1.1.1 Pipeline

Como visto no Capítulo 2, a decodificação JPEG compreende uma série de processos também utilizados na decodificação de um vídeo M-PEG. Estes processos, além de serem sequenciais, são independentes, fazendo com que sejam adaptados para uso de *pipeline* no tratamento da informação de imagens. Para que a estrutura *pipeline* funcione corretamente, deve-se considerar que um *pipeline* balanceado é aquele que consegue manter o mesmo tempo de processamento para todos os seus componentes. Também é necessário informar a cada componente em que estado se encontra o decodificador, esta informação é chamada de *contexto*.

O *contexto* pode indicar alguma informação do cabeçalho ou se algum marcador foi detectado, por exemplo, um EOI. Quando uma nova imagem entra no *pipeline,* outra se encontra em decodificação. Ou seja deve-se poder processar duas imagens simultaneamente. Ou seja, necessita-se manter dois cabeçalhos distintos, cada um com as suas próprias tabelas de *Huffman* e tabelas de quantização. Assim *contexto* é um *indicador*, responsável por selecionar em todos os componentes do *pipeline* qual cabeçalho usar.

3.1.1.1.2 Handshake

Os estágios do decodificador utilizam um controle de fluxo simples, onde cada componente possui dois sinais de controle, *datavalid* e *ready*. O sinal *datavalid* indica que o transmissor possui dados válidos para envio, e o sinal *ready* indica que o receptor pode receber este dado.

A Figura 3.4 indica genericamente onde estão localizados os sinais de *handshake*.

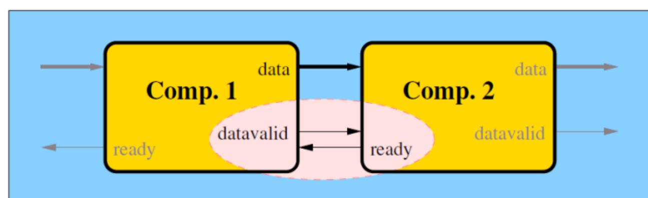


Figura 3.4 – Estrutura genérica dos sinais de controle de fluxo para os componentes do *pipeline* JPEG.

O sinal *datavalid* é gerado pelo componente anterior no fluxo de dados, enquanto o sinal *ready* é gerado pelo componente posterior. Quando os dois sinais se encontram em "1" (Comp1. *datavalid*='1' e Comp2. *ready*='1'), dados são transferidos e o processamento no Comp. 2 é iniciado. Este fluxo de dados segue o diagrama de tempos genérico representado na Figura 3.5.

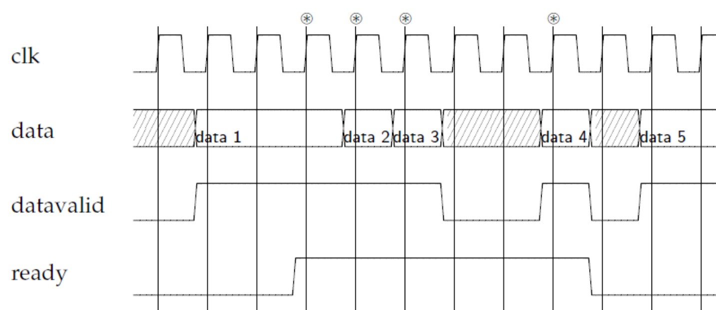


Figura 3.5 - Forma de onda típica para os sinais de *handshake.*

3.1.1.1.3 Frame Rate e Slow control

Um decodificador JPEG pode ser utilizado para M-PEG. No entanto, devem-se considerar dois aspectos: a taxa de quadros do vídeo gravado pode não ser a mesma taxa necessária para o monitor e, deve-se supor que nenhum dado foi perdido no *buffer* de entrada.

Um monitor VGA exibe imagens a uma taxa de 60 Hz. Este *hardware* não possui memória intrachip suficiente para armazenar um quadro completo, então o decodificador deve ser capaz de gerar uma nova imagem a cada $\frac{1}{60}s$. Se a taxa de quadros é diferente de 60 Hz, alguns quadros deverão ser decodificados mais de uma vez.

Para repetir um quadro, usa-se o endereço SDRAM dele que havia sido armazenado no início do processamento. Assim o quadro pode ser buscado da memória quantas vezes forem necessárias para ser reprocessado. Esta implementação dá suporte a vídeos com a taxa de quadros de $\frac{60Hz}{n}$ com $n \in \mathbb{N}$, isto é, a taxa de repetição para cada quadro deve ser igual.

A Figura 3.6 ilustra os componentes e as conexões básicas necessárias para buscar um *stream* e ajustar a taxa de quadros. Um sinal reseta simultaneamente uma fila (Fifo 1) do *buffer* de entrada para descartar dados inválidos. Assim, a próxima imagem não pode ser buscada enquanto o sinal *JPEG_EOI* do *JPEG-decoder* estiver ativado, pois parte da imagem seria descartada também. Uma detecção lógica de EOI e SOI percorre os dados de entrada por esses marcadores. A lógica para calcular o endereçamento deve proceder de duas maneiras após um marcador EOI ser detectado, dependendo se o quadro deverá ser repetido ou não:

1) Se o quadro não deve ser repetido, o *stream* é analisado até um marcador SOI ser encontrado. Então o *OPB_address* da palavra de 32 bits contendo o marcador SOI é

armazenado. Se o *JPEG_EOI* de um quadro anterior não foi recebido ainda, o endereço será o valor armazenado quando o *JPEG_EOI* foi recebido. Caso contrário, dados são perdidos, devido ao *reset* do *buffer* de entrada;

2) Se o quadro necessita ser repetido, o decodificador aguarda pelo *JPEG_EOI* e então reseta o *OPB_address* para o valor previamente armazenado.

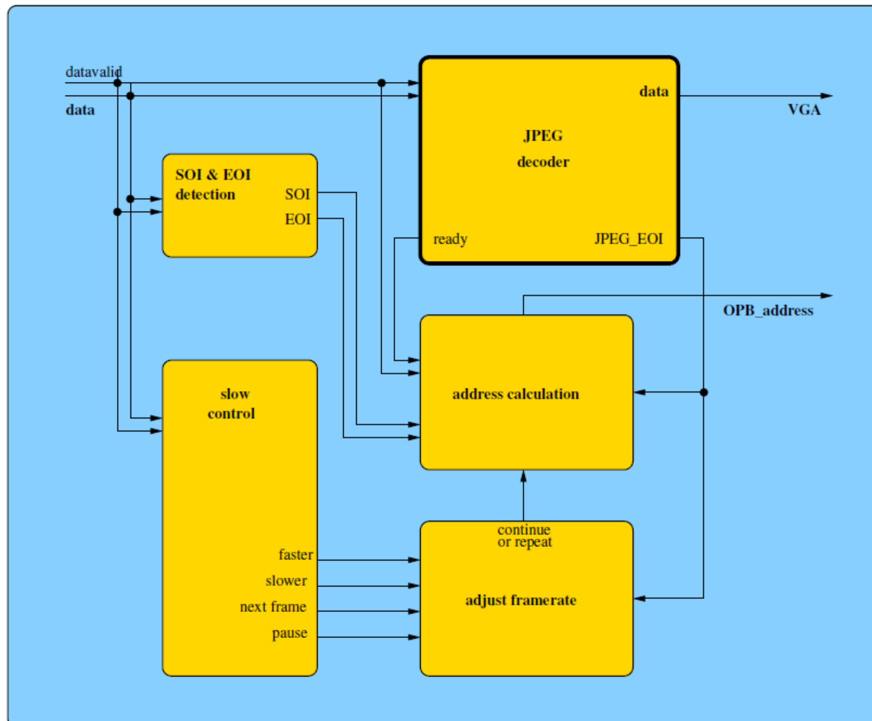


Figura 3.6 - Componentes e conexões básicas de controle da taxa de quadros.

3.1.2 *JPEG Decoder*

O decodificador possui uma entidade *top*, responsável pela comunicação externa, e que encapsula as conexões para os seus componentes. Os dados de entrada são de 32 bits e são decodificados em três dados de saída de 8 bits (um para cada componente RGB). Esta entidade também possui algumas informações sobre o método de amostragem, como largura, altura e o marcador EOI para fazer a sincronização com a entidade VGA. Um segundo marcador EOI é mandado para a lógica OPB, para indicar que todos dados da imagem foram lidos, e que a lógica OPB pode começar a retransmitir os dados da imagem.

Cada componente é responsável por uma parte do processo de decodificação, possuindo funcionalidades específicas. Estes processos são ilustrados na Figura 3.2 e são sete, descritos cada um em uma das a seguir.

3.1.2.1.1 Input Buffer

O *buffer* de entrada foi projetado como uma FIFO de duas fases, com uma lógica de pré-análise dos dados de entrada, conforme a Figura 3.7. Os dados provenientes do barramento OPB são coletados em uma primeira FIFO (Fifo 1), onde os dados de entrada de 32 bits são separados em blocos de 8 bits para a pré-análise (check_FF). Uma vez que os dados são armazenados em endereços sequenciais, estes podem ser buscados em rajada.

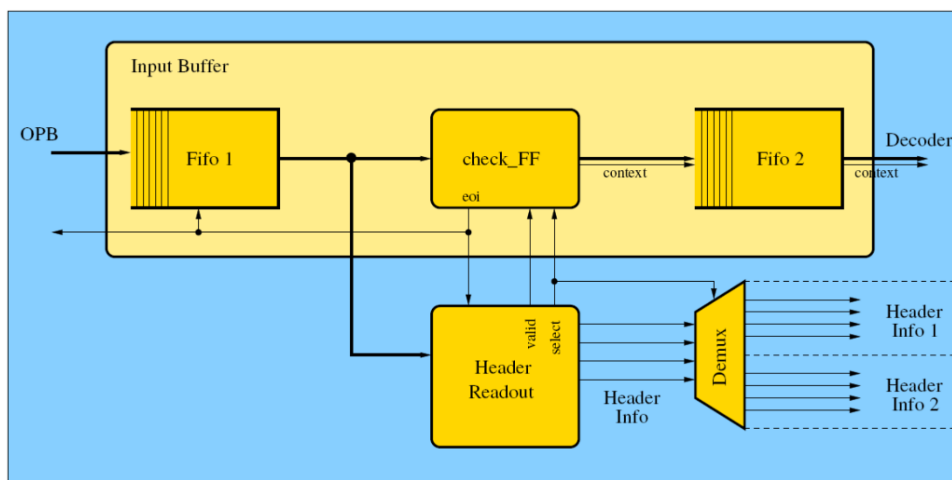


Figura 3.7 - Estrutura e sinais do componente *Input Buffer*.

No entanto, após o fim da imagem, os dados não são mais válidos e devem ser descartados. O bloco Check_FF espera pelo marcador EOI, e quando este for detectado, os dados inválidos serão descartados pelo Fifo 1. Além disso, um novo ciclo de decodificação é iniciado a partir da indicação da interface MyIPIF para retransmitir a imagem inteira e reativar o *header readout* 1.

Para decodificar a mesma imagem JPEG repetidamente, seria suficiente ler o cabeçalho ao iniciar o processo de decodificação, e então armazenar as informações para que tais dados fossem utilizados no restante do processo. Contudo, na decodificação *Motion-JPEG*, existem várias imagens diferentes a serem processadas sucessivamente, e o cabeçalho de cada uma delas deve ser lido e armazenado. Assim, as informações do cabeçalho não são somente lidas uma vez, mas a cada ciclo elas precisam ser relidas. Desta forma, o decodificador está preparado para o *Motion-JPEG*. Por outro lado, muitos estágios do *pipeline* necessitam das informações do cabeçalho e este não pode ser atualizado enquanto esses componentes estão ocupados. Esvaziar o *pipeline* antes de decodificação da próxima imagem seria muito ineficiente. Assim, o decodificador provê espaço para dois conjuntos de dados usados para o armazenamento dos cabeçalhos, eliminando a necessidade de esvaziar o *pipeline* antes de decodificar a imagem seguinte.

O *buffer* de entrada de dois estágios, combinado com os dois conjuntos de informações de cabeçalho, permite processar o cabeçalho de uma imagem $n + 1$, enquanto o resto do decodificador ainda está trabalhando sobre a imagem n. A leitura de múltiplos cabeçalhos em si não adiciona nenhum atraso, uma vez que é feita em paralelo com o resto do processo de decodificação.

Existe outra tarefa importante feita durante esta pré-análise. Qualquer ocorrência de 0xFF no *bitstream* comprimido que tenha saído do codificador com resultado 0xFF00 será substituída na pré-análise por apenas 0xFF.

3.1.2.1.2 Entropy Decoding

O último passo da codificação é a codificação de entropia. Portanto este será o primeiro passo da decodificação. A codificação de entropia é uma combinação de diversas técnicas de codificação, como o *run length encoding*, *variable length encoding* e uma forma especial da *Huffman encoding*. A decodificação de entropia é dividida em dois passos. Primeiro, a informação necessária para as técnicas de *run length decoding* e *variable length decoding* deve ser decodificada, primeiramente aplicando-se um algoritmo de *Huffman*. Somente depois as técnicas de *run length decoding*, para os valores zerados, e *variable length decoding*, para os valores não zerados, podem ser aplicadas.

A Figura 3.8 mostra uma versão simplificada da máquina de estados finita para a decodificação de entropia. Para manter a clareza alguns sinais menos relevantes são omitidos.

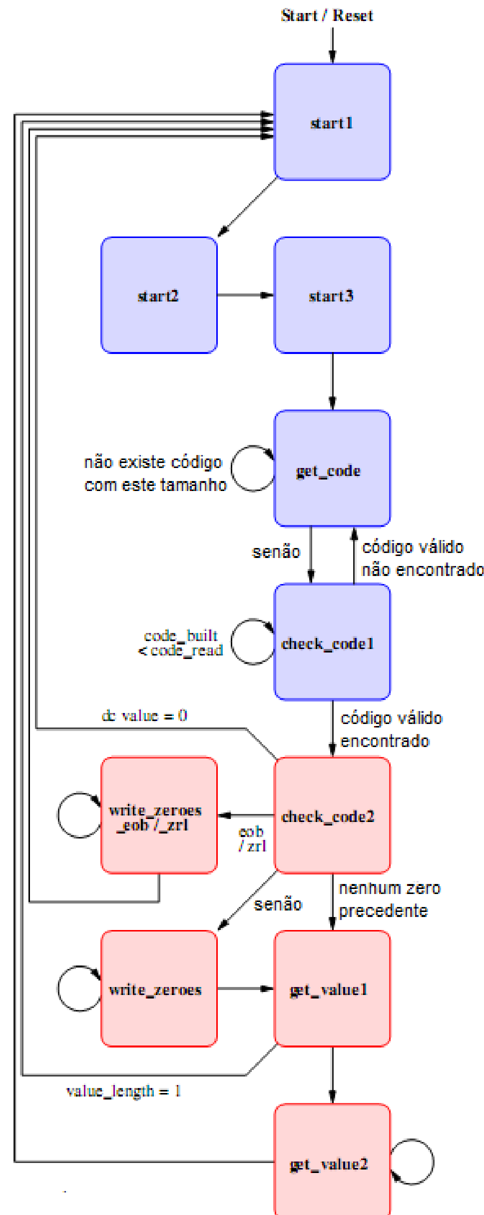


Figura 3.8 - Máquina de estados finita utilizada na decodificação de entropia (simplificada).

Como entrada, o decodificador necessita dos dados bit a bit, mas o *buffer* de entrada contém os dados em blocos de 8 bits. Modificando a fila "FIFO 2" para uma FIFO 8-para-1 assimétrica, pode resultar em problemas devido ao *contexto*, que deve ser enviado em conjunto com os dados. Então uma "FIFO 2" de 12 bits de largura é utilizada e o *contexto* de 4 bits e os dados de 8 bits são escritos como uma palavra de 12 bits nesta FIFO. Na parte da leitura, os dados são armazenados em um registrador de deslocamento de 8 bits, que a seguir são deslocados bit a bit.

Um caso especial pode ocorrer se um marcador JPEG (EOI) for inserido no *stream* comprimido. Marcadores são alinhados em bytes, portanto pode ser necessário remover bits inválidos que estejam antes do marcador. Estes bits removidos estão todos em '1' e, portanto não existe nenhum código de *Huffman* válido de acordo com as especificações. Se encontrado, a máquina de estados continua a decodificar até que o último bit armazenado seja processado e então resseta logo que o marcador seja lido (o marcador é detectado pelo *buffer* de entrada, o qual então aciona a *flag context*). Desta maneira nenhum código válido é perdido.

Existem quatro tipos de tabelas de *Huffman*, uma diferente para os coeficientes DC, outra AC e outras para os componentes de cores, a tabela Y do brilho, e a CbCr da crominância. As tabelas são armazenadas no cabeçalho. As informações sobre os códigos das tabelas de *Huffman* são armazenados em 16 bytes específicos. O primeiro byte é o número de códigos com tamanho 1, o

segundo byte representa o número de códigos com o tamanho 2 e assim em diante. Em seguida, são seguidos pelos valores que são representados pelos códigos. Os códigos necessitam que sejam remontadas pelo decodificador, pois se encontram criptografados. Iniciando com o menor tamanho, o próximo código válido é sempre o menor valor possível para que não tenha o código anterior como prefixo.

Os próximos parágrafos descrevem os estados *start1*, *start2*, *start3*, *get_code* e *check_code1* da Figura 3.8, juntamente com os sinais que estes manipulam e as transições de estado possíveis.

Dois sinais de 16 bits são utilizados nesta parte da máquina de estados, *code_read* e *code_build*. Eles são comparados e se forem iguais isto indica que um código de *Huffman* foi encontrado. Se necessário, o *code_read* é agrupado com o dado de entrada e o *code_build* é montado sucessivamente de acordo com as regras. Para diferenciar as tabelas DC e AC, um contador chamado *ac_dc_counter* é utilizado, para diferenciar os componentes de brilho dos componentes de crominância. Uma máquina de estados finita foi implementada separadamente, mas esta depende da amostragem.

No primeiro estado (*start1*), os registradores armazenam os endereços das tabelas e os códigos recebem zero. Nos próximos dois estados é necessário aguardar a BRAM armazenar as tabelas com dados válidos.

No estado *get_code* o tamanho de um possível código de *Huffman* é acrescido em um bit. Um bit é lido da entrada do registrador de deslocamento e concatenado no registrador *code_read*. Um zero é concatenado no registrador *code_build*. O número de códigos de *Huffman* com o tamanho atual é lido da tabela de *Huffman* correta e memorizado no registrador *symbols*. Se este valor é zero, a máquina de estados permanece no mesmo estado por mais um ciclo de relógio para concatenar outro bit.

A seguir, no estado *check_code1*, os valores de *code_read* e *code_build* são comparados, o que indica se o código de *Huffman* obtido é válido. O sinal *code_build* é incrementado até ser encontrado um código válido ou o número de códigos testados ser igual ao valor do registrador *symbols*.

O byte decodificado pela parte de *Huffman* consiste de dois *nibbles* com significados distintos. O primeiro representa a informação de quantos zeros existem antes do valor a ser decodificado, enquanto o segundo representa o número de bits necessários para representar o valor a ser decodificado.

A tabela de *Huffman* armazena dois bytes especiais (0x00: EOB e 0xF0: ZRL) que são tratados diferentemente. O marcador ZRL representa 16 zeros, enquanto o marcador EOB indica que os próximos valores do bloco são zeros.

Os próximos parágrafos explicam os estados *write_zeroes_eob*, *write_zeroes_zrl*, *check_code2*, *write_zeroes*, *get_value1* e *get_value2*, juntamente com os sinais e transições afetados por estes.

Dois contadores são utilizados nesta parte da máquina de estados, um para os zeros a serem escritos, e outro para os bits a serem lidos dos valores não zerados.

Para decodificar o coeficiente DC, é necessário o coeficiente DC do bloco anterior, três vetores de registradores são utilizados para armazenar os coeficientes DC, um para cada componente de cor. A seleção do registrador do componente de cor é realizada por uma máquina de estados finita.

O estado *check_code2* interpreta um código válido, consultando a tabela de *Huffman* apropriada. O código pode ser um código especial, como os marcadores EOB ou ZRL, ou uma combinação do número de zeros precedentes e o número de bits necessário para o valor a ser decodificado. Caso o código encontrado seja identificado como um marcador especial, EOB ou ZRL, o próximo estado será o *write_zeroes_zrl*, que escreve 16 zeros, ou o *write_zeroes_eob*, que é encarregado de zerar os demais coeficientes do bloco 8x8 atual.

Caso o código seja o número de zeros precedentes ao valor que será decodificado, o próximo estado é o *write_zeroes*, que simplesmente escreve a quantidade de zeros conforme o número recebido.

Caso o código recebido seja o número que representa o valor a ser decodificado, o próximo estado é o *get_value1* onde um bit é lido. Se o número for zero, então o valor a ser decodificado será negativo e este bit e o próximo devem ser invertidos para recuperar o valor original. Os próximos bits necessários para a decodificação do valor correto são lidos e escritos no estado *get_value2*.

3.1.2.1.3 Dequantize

Uma vez que as tabelas de quantização são armazenadas no cabeçalho em ordem zigue-zague, a desquantização é aplicada antes do processo de dezigue-zague. Isso permite que as tabelas possam ser lidas e utilizadas sem necessidade de uma reordenação. Elas estão armazenadas em registradores de deslocamento circulares, onde o registrador de saída alimenta o *buffer* de entrada. O registrador deve ter 8 bits de comprimento e 64 bits de profundidade para armazenar uma tabela. Para o decodificador descrito, os registradores foram instanciados utilizando um gerador de componentes CoreGen, para ter acesso às vantagens de uso de primitivas otimizadas da Xilinx como a SRL16, que é um registrador de deslocamento de 16 bits construído com uma única LUT.

Para a multiplicação necessária ao processo de desquantização, utiliza-se novamente o CoreGen. Que emprega a primitiva de hardware MULT18x18 do FPGA.

A informação sobre qual método de amostragem é utilizado serve para selecionar a tabela correta. A multiplicação de um byte é realizada em um ciclo.

3.1.2.1.4 Dezigzag

Neste processo são utilizados dois registradores de deslocamento: o primeiro armazena os dados de entrada. Logo que um bloco inteiro é lido, ele é mapeado para o segundo registrador de deslocamento, na ordem inversa à de zigue-zague, que por sua vez armazena os dados ordenados enquanto o primeiro registrador inicia o procedimento com o próximo bloco de dados, segundo o esquema mostrado na Figura 3.9.

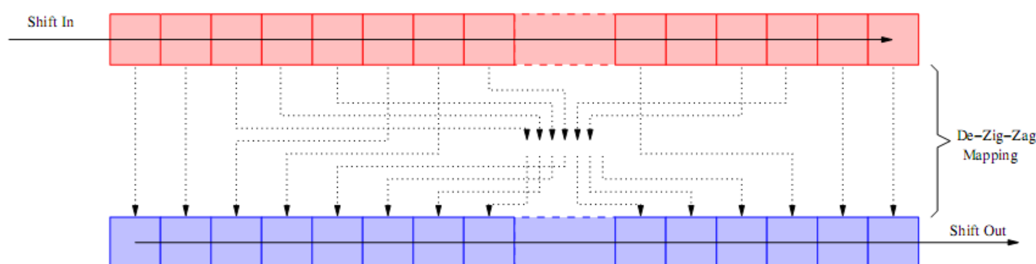


Figura 3.9 - Realocação dos dados em registradores de deslocamento.

Registradores de deslocamento são preferíveis ao uso de RAM distribuída, pois esta última gasta muitos recursos para a lógica de acesso aleatório, que neste caso é desnecessária. Blocos de memória (BRAMs) também utilizam lógica de acesso aleatório, mas podem compensar isto armazenando quantidades de dados bem maiores. De qualquer maneira, estes blocos de memórias só estão disponíveis em unidades grandes para o porte das memórias necessárias no decodificador, como as BRAMs de 18 Kbits na Virtex II-Pro, um recurso valioso que poder ser grandemente desperdiçado ao se utilizar deste para armazenar apenas 128 bytes de dados.

O *contexto* necessita de uma consideração especial, uma vez que o componente é internamente modelado como um *pipeline* de dois estágios. O registrador de deslocamento da entrada é preenchido enquanto se armazena simultaneamente no registrador de deslocamento os dados lidos anteriormente. Então, o componente armazena temporariamente dois conjuntos de dados, eventualmente com uma informação de contexto diferente para cada um. Portanto os dois

conjuntos de dados são necessários. O conjunto 1 é copiado para o conjunto 2 ao mesmo tempo em que os dados são mapeados.

3.1.2.1.5 IDCT

Anteriormente, aplicando o processo da DCT a imagem foi transformada do domínio tempo para o domínio frequência. Agora, aplicando a IDCT, a imagem codificada no domínio frequência, retornará para o domínio tempo.

A Xilinx disponibiliza um núcleo IDCT via CoreGen. Ele é utilizado porque possui vantagem sobre a primitiva de multiplicação, MULT18x18 da FPGA utilizada. Este núcleo é implementado como um modelo *pipeline*, mas não gera nenhum controle de fluxo reverso. Uma vez que o bloco de dados é lido, os dados serão disponibilizados depois do tempo de cálculo. Se a próxima entidade não está pronta ainda para receber dados, ele pode ser perdido. Para garantir que isso não ocorra um *wrapper* foi implementado para o núcleo IDCT e o sinal de fluxo *ready* do próximo componente na sequência foi adaptado. O sinal *upsampling-ready* é redefinido quando menos de um bloco pode ser armazenado no *buffer*. O sinal *upsampling-ready* é utilizado diretamente como o sinal *IDCT-ready*. Então, o segundo bloco não é totalmente lido, e indiretamente, pausa o núcleo IDCT. Isso só pode ser feito porque o núcleo IDCT inicia repassando o primeiro bloco antes que o segundo esteja completamente lido, mesmo que sempre existam dados de entrada válidos. O núcleo IDCT para um FPGA Virtex-II Pro lê um bloco 8x8 *pixels* completo em 92 ciclos e necessita da mesma quantidade de ciclos para escrever um bloco 8x8 *pixels*. A latência do núcleo é sempre de 144 ciclos (maior que 92 ciclos). O tempo entre a última entrada e a primeira entrada é de 144-92=52 (menor que 92 ciclos). Configurar o sinal *upsampling-ready* de 64 bytes para zero muito cedo resultará em um controle de fluxo reverso. O processo da transição de dados original é representado na Figura 3.10, o processo implementado é representado na Figura 3.11.

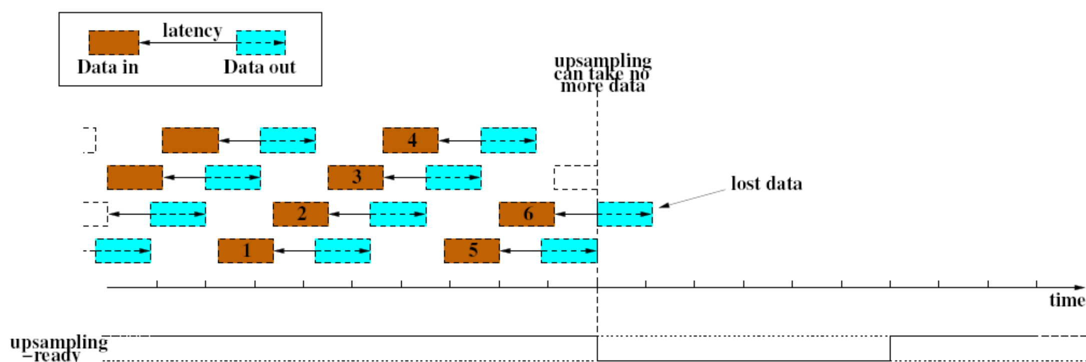


Figura 3.10 - Controle de fluxo e *pipeline* originais do componente IDCT.

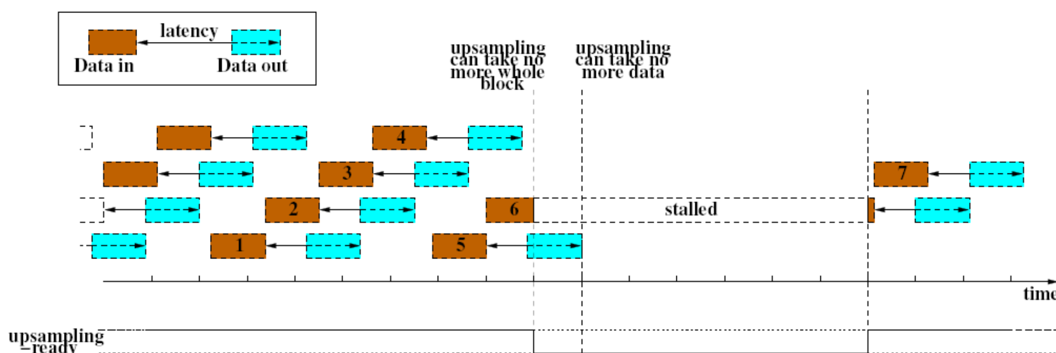


Figura 3.11 - Controle de fluxo e *pipeline* do componente IDCT modificados.

Os coeficientes de entrada são sinais de 12 bits, como especificados pelo padrão JPEG. Internamente, os dados devem ser representados por mais bits, para reduzir o erro de

arredondamento. Neste decodificador, utiliza–se internamente sinais de 15 bits de precisão, ficando assim dentro das especificações do padrão IEEE 1180-1990 que descreve a precisão para transformações de IDCT [IEE91].

Uma vez que o núcleo IDCT pode conter mais de um (no máximo três) conjunto com dados diferentes, o *contexto* correto deve ser selecionado. Existem dois contadores de 8 bits, *in_counter* e *out_counter* e quatro conjuntos de *contextos*. O contador *in_counter* é incrementado se um dado é lido, o contador *out_counter* é incrementado caso o dado seja repassado. Ambos contadores possuem 2 bits a mais do que o necessário para cada bloco. Assim, os dois bits mais significativos podem ser interpretados como um bloco contador de 2 bits. Os dados de entrada são armazenados no conjunto de contexto indicado pelos 2 bits mais significativos do *in_counter*, o *contexto* fornecido em conjunto com o dado de saída é tomado de acordo com os 2 bits mais significativos do *out_counter*.

Devido ao algoritmo utilizado, os dados decodificados são repassados em ordem de coluna ao invés de ordem de linha necessária. Uma vez que na próxima entidade, a amostragem, os dados são reordenados de qualquer maneira, isto não representa um problema, pois na amostragem, a matriz 8x8 *pixels* é transposta em um passo, retornando à ordenação original.

3.1.2.1.6 UpSampling

O olho humano é mais sensível para perceber diferenças no brilho do que nas cores. No processo de amostragem da codificação, os componentes de cores (CbCr) são armazenados com uma resolução reduzida. A amostragem da decodificação interpola os elementos de cor para as suas resoluções originais.

Semelhante ao componente *dezigzag*, o componente de amostragem reorganiza os dados em uma ordem diferente. Portanto, é necessário armazenar o bloco de dados inteiro. Ele é implementado do mesmo modo que o componente *dezigzag*, utilizando dois registradores de deslocamento. O *pipeline* interno de dois estágios justifica-se devido ao já mencionado problema do *contexto*. Dois conjuntos de *contextos* são necessários. O primeiro conjunto é copiado para o segundo conjunto ao mesmo tempo em que os dados são copiados, pois a informação do *contexto* influencia o tamanho dos *buffers* e a ordem em que estes são mapeados.

A ordem de entrada é de um bloco 8x8 *pixels* após o outro, até que uma MCU seja completada. A saída deste componente será um bloco MCU inteiro na ordem habitual, da esquerda para a direita e de cima para baixo. Ainda, ao se mapear da entrada para a saída, os dados necessitam ser reorganizados para compensar a ordem de saída em colunas do componente IDCT.

Diferente do componente *dezigzag*, o qual deve ser capaz de armazenar um bloco 8x8 *pixels*, este componente armazena uma MCU inteira. Dependendo do método de amostragem utilizado, uma MCU pode conter até seis blocos 8x8 *pixels* no *buffer* de entrada e doze blocos 8x8 *pixels* no *buffer* de saída. A implementação original, com registradores de deslocamento é ineficiente em termos de utilização do dispositivo. O *buffer* deve armazenar $(6 + 12) * 64\ bytes = 1152\ bytes$, pois a primitiva BRAM dupla porta é utilizada. Logo, os dados são mapeados em diferentes endereços para escrita e leitura, semelhante a como os dados são organizados no componente VGA.

O controle de fluxo no *upsampling* deve ser ligeiramente diferente daquele dos demais componentes, caso contrário as vantagens do modelo de *pipeline* interno do componente IDCT são perdidas. O sinal *ready* deve receber zero quando o espaço do *buffer* ficar com menos de 64 bytes, o que significa que não é possível mais armazenar um bloco completo de 8x8 *pixels*.

3.1.2.1.7 YCbCr2RGB

A compressão JPEG utiliza o formato de cores YCbCr, enquanto a VGA utiliza o formato de cor RGB. Este componente é responsável por transformar o formato dos dados YCbCr em RGB.

O dado de saída do componente IDCT é deslocado, de modo que do valor seja subtraído 128. Então os valores de 8 bits ficam em uma faixa entre -128 e 127, em complemento de dois, ao invés da faixa de 0 a 255. As fórmulas abaixo indicam o que é necessário para cada componente de cor. Ou seja, o componente Y é corrigido simplesmente adicionando 128 aos seus coeficientes. Para aplicar a fórmula, o componente Y é multiplicado por 1024 e o mesmo é realizado para os fatores que os componentes de cor multiplicam. Os fatores são arredondados para o próximo inteiro, fazendo com que as fórmulas (3.1) realizem operações somente sobre inteiros.

$$\begin{aligned} R &= 1024 * (Y_{IDCT} + 128) & & + 1436 * (Cr_{IDCT}) \\ G &= 1024 * (Y_{IDCT} + 128) & - 352 * (Cb_{IDCT}) & - 731 * (Cr_{IDCT}) \\ B &= 1024 * (Y_{IDCT} + 128) & + 1815 * (Cb_{IDCT}) & \end{aligned} \quad (3.1)$$

Contudo, depois da transformação, os valores RGB podem não estar dentro da faixa de 0 a 255. Devido aos erros de arredondamento, no processo da IDCT, os valores RGB resultantes deste processo devem ser menores que 0 e maiores que 255. A norma JPEG já prevê este problema. Ela exige que os valores RGB sejam ajustados para a faixa de 0 a 255. Assim, valores negativos são convertidos para 0 e os maiores que 255 são convertidos para 255.

A implementação deste teste de limites não é difícil. Infelizmente, a ferramenta de síntese não foi capaz de realizar a síntese lógica com as restrições temporais para a FPGA Virtex-II Pro. Basicamente as primitivas de multiplicação, utilizadas para a multiplicação na fórmula de transformação de cores, resultaram em latência demasiada. Então este passo é dividido em dois ciclos de relógio. No primeiro, a fórmula é aplicada. No segundo, os limites são verificados e os resultados são divididos por 1024, o qual é necessário para as operações com inteiros.

O controle de fluxo e o gerenciamento de *contexto* deste componente são simples. O sinal *ready* vindo de fora do decodificador alimenta diretamente o componente de *upsampling* e também é conectado ao pino de *clock enable* do flip-flop utilizado. O sinal *datavalid* e o *contexto* do componente anterior sofrem um atraso de dois ciclos, mantendo-os sincronizados com os dados.

3.1.3 VGA

VGA (*Video Graphics Array*) é um padrão analógico de conexão de vídeo. Portanto o componente VGA deve gerar sinais analógicos. Isso é feito por um conversor digital-analógico (DAC) instalado na placa de desenvolvimento. O DAC exige os seguintes sinais de entrada: os três componentes de cor RGB (8 bits cada), os sinais de sincronismo horizontal e vertical (1 bit), um sinal de *blank* (1 bit, substituindo a entrada RGB) e um sinal de *clock* (operando na frequência de mudança de *pixel*). Sua saída são os cinco sinais analógicos necessários para operar um monitor VGA, três sinais para os componentes de cores RGB e dois para o sincronismo vertical e horizontal.

Para operar o DAC no padrão mais comumente usado (resolução de 640x480 *pixels* e taxa de atualização de 60 Hz), um novo domínio de *relógio* é introduzido. Um novo *pixel* é exibido a cada $\frac{1}{25}\,\mu s$. O *clock* do DAC deve operar em 25 MHz. O *clock* é gerado pelo componente *vga_signals*, descrito em VHDL. Esse componente também gera os sinais de sincronismo horizontal e vertical, e o *blank*. Adicionalmente para os sinais VGA, um contador de linhas e um de *pixels* são fornecidos para indicar a posição do *pixel* atual na tela. Como todos os sinais são gerados de forma independente do resto do sistema, os dados a serem exibidos devem ser sincronizados com os sinais. Isso é alcançado usando uma *DualPort Block* RAM que também é usada para armazenar e reorganizar os dados.

A figura presente no ANEXO D ilustra o componente VGA cujos dados são passados para o novo domínio de *clock* (25 MHz) através de uma *DualPort Block* RAM. Alguns sinais que são gerados no domínio de *clock* de 25 MHz necessitam ser utilizados no domínio de 100 MHz, e vice-versa. Então eles têm que ser sincronizados para o domínio do outro *relógio*. Os sinais gerados a partir do domínio de 100 MHz que são necessários no domínio de 25 MHz são críticos, pois nesta passagem de um domínio de maior frequência a um domínio de frequência menor, pode ocorrer a

perda de sinais. No entanto, estes sinais (*width*, *height* e *sampling*) não são voláteis, de modo que este não é um problema real.

Como mencionado anteriormente, pode não haver memória suficiente disponível *on-chip* para armazenar uma imagem completamente decodificada. Então, o decodificador precisa decodificar dados em sincroniismo com a tela. Caso o decodificador atrase o envio dos dados, não existirá um dado válido para ser exibido. Para tornar mais confiável essa sincronização, é necessário armazenar alguns dados decodificados no componente VGA. Existe ainda outra razão para se armazenar dados entre o JPEG *Decoder* e o componente VGA: a ordem com que os *pixels* são decodificados a partir de um arquivo JPEG não é a mesma em que os *pixels* serão exibidos. Isso é um problema grave, pois a leitura e escrita na SDRAM são em ordens diferentes, e o barramento OPB não pode ser usado no modo rajada para leitura e escrita.

As leituras e escritas de diferentes endereços são calculadas separadamente. Para definir rigorosamente os espaços de endereços separados para leitura e gravação, o sinal *memory_select* é usado. Este sinal é usado como bit mais significativo (MSB) do endereço de leitura e sua negação como MSB do endereço de escrita. O sinal alterna após a 16ª linha, quando o método de amostragem 4:2:0 é usado, e após a 8ª linha, quando um outro método de amostragem é usado, pois a altura de um MCU na amostragem 4:2:0 é de 16 linhas, para os outros métodos de amostragem é de 8 linhas.

Cálculo de escrita de endereço - Os dados decodificados devem ser apresentados em uma ordem definida. Dados perdidos resultam em uma imagem desalinhada. Quando uma imagem inteira foi decodificada, a escrita deve encerrar à espera de um novo quadro VGA livre. Para isto, os sinais EOI, *line_count* e *height* são usados. O uso de EOI sincroniza o fluxo de dados, no caso de uma imagem inesperadamente ter se desalinhado no processo. A escrita de endereços é incrementada quando os sinais de controle de fluxo indicam que dados válidos foram entregues.

Cálculo de leitura de endereço - Os dados são armazenados no sentido de MCUs e não linha de *pixels* após linha de *pixels*. Os dados de uma linha de *pixel* de uma imagem são armazenados em espaços de endereços espalhados. No entanto, a dispersão é bem definida. O endereço correto de leitura pode ser calculado a partir dos sinais *pixel_count* e *line_count*. Se o *pixel* exibido no monitor VGA não está dentro da área coberta pela imagem, os valores de saída RGB são ajustados para zero, ao definir como ‘1’ o sinal *blank* do DAC.

4 ARQUITETURA DO DECODIFICADOR M-JPEG/HERMES

Este Capítulo descreve a principal contribuição do presente trabalho, a arquitetura do decodificador de vídeo M-JPEG/HERMES. Ele é o resultado da integração da NoC HERMES, descrita no Capítulo 2, com o decodificador M-JPEG, descrito no Capítulo 3. As adaptações necessárias são detalhadas no decorrer desse Capítulo.

4.1 Descrição

A integração entre a NoC HERMES e o decodificador se dá em duas fases de desenvolvimento. A primeira fase é uma prova de conceito, na qual se utiliza somente dois módulos do processo de decodificação JPEG e se insere entre eles dois roteadores da rede HERMES, a fim de analisar a capacidade dos roteadores para atender aos requisitos temporais. Caso esta se revele satisfatória, a segunda fase é a integração total dos módulos com a NoC. A segunda fase foi apenas analisada e não implementada neste trabalho.

Na primeira fase gerou-se com o ambiente ATLAS uma NoC HERMES 2x2, descartando-se os dois roteadores do topo desta instância (*Top-Left* e *Top-Right*). Os módulos JPEG selecionados para conectar via NoC nesta fase são *UpSampling* e *YCbCr2RGB* escolhidos por que estes módulos se comunicam em um intervalo de tempo bem definido e o número de dados enviados é sempre o mesmo, facilitando assim a implementação. A comunicação realiza-se através dos roteadores *Bottom-Left* (BL) e o *Bottom-Right* (BR). Como o protocolo de comunicação usado na NoC é naturalmente diferente do protocolo do decodificador, é necessário implementar interfaces de rede para adequar uma estrutura à outra. Este novo sistema, que abrange as interfaces está ilustrado na Figura 4.1.

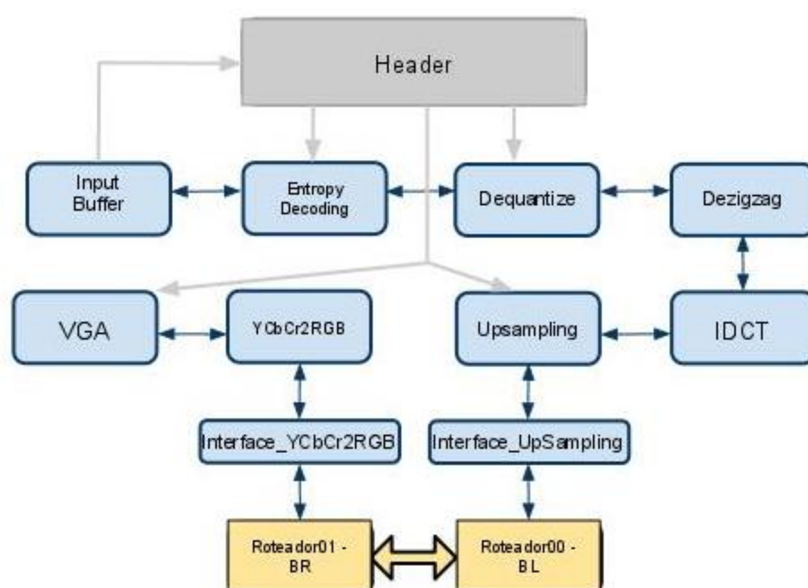


Figura 4.1 - Modelo implementado na primeira fase de integração JPEG-HERMES.

Após a conclusão da primeira fase, propõe-se, como atividade futura, gerar uma NoC maior, pois o sistema completo requer no mínimo uma NoC 2x4 para suprir os 7 módulos da entidade JPEG *Decoder*, mostradas na Seção 3.1.2. Também se deve implementar cada interface e analisar

novamente se esta estrutura respeita os requisitos temporais do decodificador. O resultado esperado da segunda fase está ilustrado na Figura 4.2.

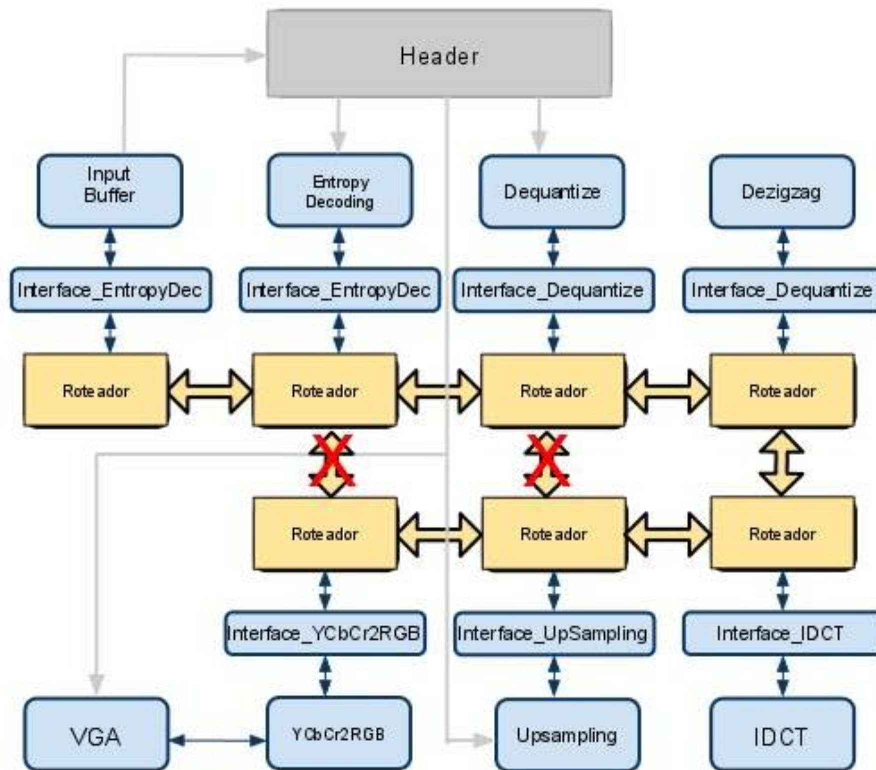


Figura 4.2 - Modelo a ser implementado na segunda fase de integração JPEG-HERMES.

Neste caso, percebe-se que não existe roteador para a VGA, pois como visto na Seção 3.1.3 este processo não faz parte do componente do JPEG *decoder*. Assim, define-se então que este não participa do roteamento e recebe diretamente os dados enviados pelo módulo YCbCr2RGB.

As ligações entre os roteadores centrais da NoC não são utilizadas pois o processo de decodificação é sequencial e em um primeiro momento não é necessário a troca de dados entre eles, como por exemplo, entre o *Dequantize* e o *UpSampling*.

4.2 Geração da NoC

A geração da NoC é uma etapa importante para este trabalho. O acréscimo desta estrutura pode implicar limitações de lógica da plataforma XUP V2-PRO. Antes da geração, é necessário fazer uma análise de todos os processos do decodificador e assim adaptar a rede aos requisitos do sistema. Deve-se analisar, por exemplo, como é feito o controle de fluxo entre os componentes, bem como o tamanho dos dados que estão trafegando e a velocidade de transmissão. A partir destas informações, é possível gerar a NoC considerando sua especificação e limitações. Como o objetivo deste projeto não é implementar uma NoC, e sim inserir uma que se adapte ao decodificador já implementado, foi utilizado o ambiente ATLAS para gerá-la facilmente.

ATLAS é capaz de gerar uma NoC automaticamente via uma interface gráfica, onde o usuário seleciona o tipo de NoC que deseja gerar, seu tamanho, tipo de controle de fluxo e algoritmo de roteamento a usar, entre outros parâmetros. Todos os elementos componentes da NoC são gerados pelo ATLAS em VHDL sintetizável, com exceção das interfaces que serão abordadas na Seção 4.2.2.

4.2.1 Especificação

Para o modelo de implementação final, definiu-se gerar uma NoC HERMES com dimensão 2x4 para se adequar com o número de processos da decodificação, com o algoritmo XY pelo simples motivo de que este é o mais simples dos algoritmos. A NoC é gerada com *flits* de 32 bits, pois após uma análise dos dados trocados entre os módulos do decodificador, concluiu-se que a concatenação deles não ultrapassa 32 bits, mas é maior que 16 bits. Por exemplo, no caso da interface entre o componente *UpSampling* e o *YCbCr2RGB*, o *UpSampling* envia os dados de *contexto* (4 bits), *Y* (9 bits), *Cb* (9 bits) e *Cr* (9 bits) para o *YCbCr2RGB*, perfazendo um total de 31 bits. Então, com *flits* de 32 bits podem-se repassar todas as informações de um dado da imagem em exatamente um *flit* de carga útil. Um pacote HERMES pode conter assim um número inteiro de dados de uma imagem de forma natural. Foi escolhido usar pacotes de tamanho fixo de 255 flits na carga útil, para garantir a possibilidade de enviar em cada pacote um MCU completo.

A profundidade do *buffer* foi definida com um tamanho de 4 *flits* para a primeira etapa, não sendo necessário aumentar este valor pois após o sincronismo do tráfego dos dados, verificou-se que o *buffer* nunca fica cheio. Apesar de a HERMES possuir suporte a canais virtuais, foi definido que esta fase não necessita deste recurso, podendo-se assim prescindir de tal sobrecarga da área [MEL05].

O sistema de controle de fluxo entre os roteadores é baseado em créditos, com o objetivo de suprir os requisitos temporais do decodificador, já que em comparação com o *Handshake*, o tipo crédito consome menos ciclos de relógio por dado transmitido. Esse é um dos motivos para a implementação das interfaces, pois os componentes do decodificador se comunicam com *Handshake* e os roteadores da rede baseiam-se em créditos.

4.2.2 Interfaces

Interface de rede é o componente que faz a comunicação entre dois módulos que necessitam trocar informações, mas não compartilham um protocolo comum para envio e recepção destes. Basicamente, interfaces fazem traduções de protocolos de comunicação. No contexto desse projeto, as interfaces de rede fazem a comunicação entre cada roteador HERMES e o seu respectivo processo de decodificação. Esta Seção explica e demonstra a estrutura das interfaces implementadas bem como seu comportamento.

4.2.2.1 Comunicação entre os Elementos da Rede

Todas a interfaces devem ser capazes de reconhecer os dados recebidos da NoC, bem como também os dados recebidos do decodificador. Apesar de suas respectivas peculiaridades, cada componente de decodificação segue o mesmo esquema de controle de fluxo que os outros. Do mesmo modo que os roteadores, que possuem um controle de fluxo único, mas diferente daquele do decodificador. Neste caso, as interfaces possuem uma função importante de manipular sinais de entrada e saída, de modo a garantir toca de informações sem perdas. Como cada módulo do decodificador possui características distintas, cada um deve possuir uma interface específica, capaz de entender e encaminhar os dados para o roteador, bem como também no sentido oposto.

Basicamente, cada interface deve seguir o fluxo de atividades definido na Figura 4.3.

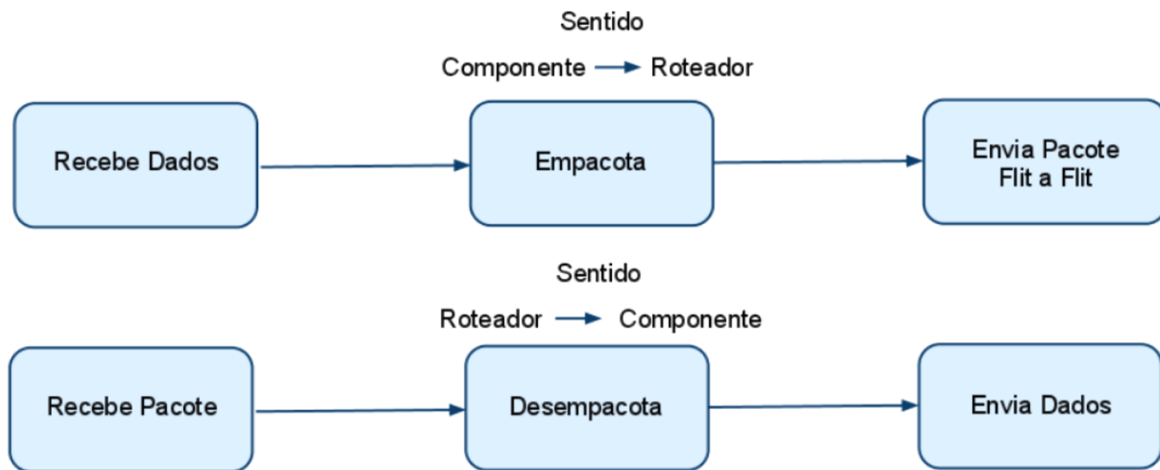


Figura 4.3 - Fluxo básico de atividades das interfaces.

Analisando o funcionamento da NoC HERMES e de cada processo do decodificador, pode-se facilmente projetar cada interface seguindo o fluxo indicado na Figura 4.3. No sentido componente-roteador, a interface deve receber os dados do componente, acondicionar os mesmo na carga útil de uma ou mais pacotes, criando cada pacote com as informações de controle (cabeçalho, tamanho e carga útil) e enviando-os *flit* a *flit* para o roteador local. O desafio é sincronizar o recebimento dos dados do IP em relação ao envio, pois o roteador interrompe a recepção de dados quando não possui mais espaço no *buffer* de entrada. Neste momento a interface deve indicar ao IP para parar de enviar dados.

No sentido roteador-componente, a interface deve receber primeiramente, as informações de controle do pacote (cabeçalho e tamanho) para então receber os *flits* de carga útil, desempacotá-los e então encaminhá-los ao componente JPEG.

Primeiramente, foi feita uma análise dos sinais de entrada e saída dos componentes envolvidos no sistema de decodificação com a NoC HERMES, juntamente com as particularidades e limitações de cada um. A partir desta análise foi possível elaborar um diagrama de blocos ilustrando como é feita a ligação componente-roteador. Cada interface possui um diagrama de blocos diferente, não sendo considerados os sinais ligados ao componente *Header Readout*, como também não foram considerados outros sinais que não são relevantes para as interfaces. As Seções seguintes somente descrevem os módulos implementados, deixando o tratamento dos módulos restantes como trabalho futuro.

4.2.2.2 Interface UpSampling

O módulos *UpSampling* possui, fundamentalmente três sinais de saída e um de entrada com informações referentes à imagem que está sendo decodificada. Os sinais restantes são para controle do *pipeline* e do decodificador. Os sinais *datavalid* e *ready* servem ao controle de fluxo e os sinais de *contexto* para indicar a situação geral do decodificador. O roteador possui os sinais já mostrados no Capítulo 2. A interface entre o módulo *UpSampling* e a NoC possui a estrutura mostrada na Figura 4.4.

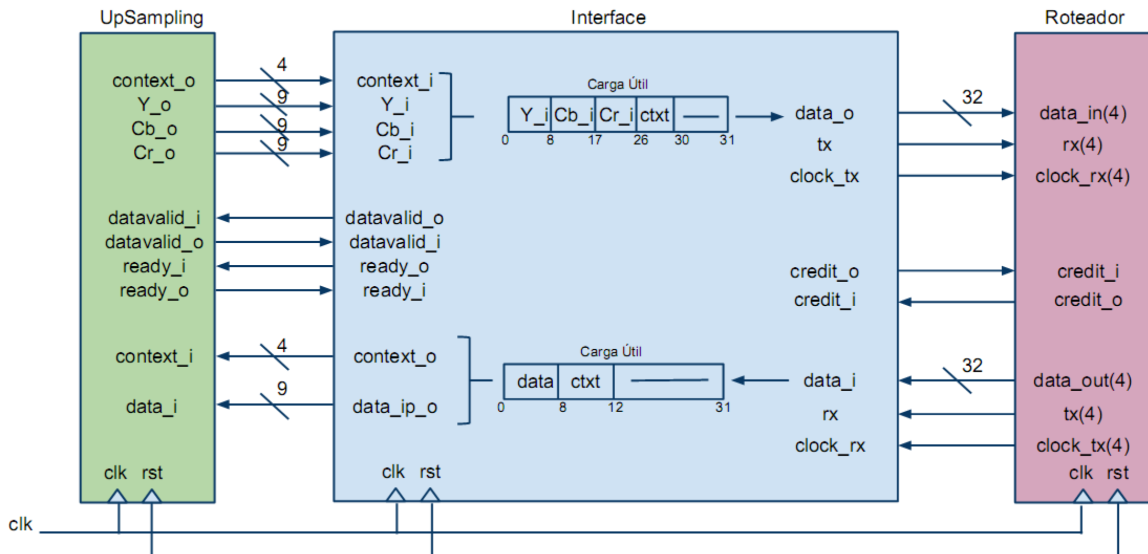


Figura 4.4 - Diagrama de blocos da interface entre *UpSampling* e a NoC.

UpSampling recebe blocos de 8x8 *pixels* do módulo IDCT. O número de blocos varia de acordo com o tipo de amostragem, mas o componente sempre tem como resultado uma MCU 2x2 de blocos 8x8, ou seja, um bloco com tamanho $16x16 = 256$ *pixels*. Primeiramente, o componente armazena os blocos 8x8 que recebe. Após o processamento, o componente envia a MCU para o componente *YCbCr2RGB* em modo rajada e em seguida fica a espera de mais blocos 8x8 vindos do componente anterior (IDCT), para então iniciar um novo processamento.

Como mostrado da Figura 4.4, os dados de entrada *Y*, *Cb*, *Cr* e *context_i* devem ser empacotados pela interface e serem encaminhados, como um *flit* de carga útil, para o roteador. O mesmo ocorre no sentido contrário, o sinal *data_i* recebido do roteador deve ser separado para ser enviado ao *UpSampling*. Segue abaixo a descrição em VHDL.

Empacotar:

```
payload(8 downto 0) <= Y_i;
payload(17 downto 9) <= Cb_i;
payload(26 downto 18) <= Cr_i;
payload(30 downto 27) <= context_i;
```

Desempacotar:

```
data_up_o <= data_i(8 downto 0);
context_o <= data_i(12 downto 9);
```

A partir da análise realizada foi possível implementar a interface *UpSampling* e a sua máquina de estados para o sentido componente-roteador. A Figura 4.5 mostra a descrição da máquina de estados da interface *UpSampling*, que recebe os dados do IP e os envia para o roteador . Contudo, somente as informações mais relevantes para o processo são mostradas, visando preservar a clareza da descrição. Nesta Figura tem-se:

Idle: É o estado de inicialização da máquina. Este estado é atingido quando o sinal *reset* é ativado ou quando a máquina ficar em modo de espera por novos dados. Neste estado, alguns sinais de controle devem ser reiniciados, pois ele é sempre o início de um novo ciclo de recepção/envio de dados. Então, *ready_o*=’1’ e *tx*=’0’, para indicar ao *UpSampling* que a máquina está pronta para receber dados, e também indicar ao roteador que não tem dados para enviar. O contador *counter* é reiniciado. Quando a interface recebe um sinal *ready_i* ativo do *UpSampling*, significa que em seguida se está começando a processar os dados para depois enviá-los. Logo a máquina vai para o estado *WaitCom*.

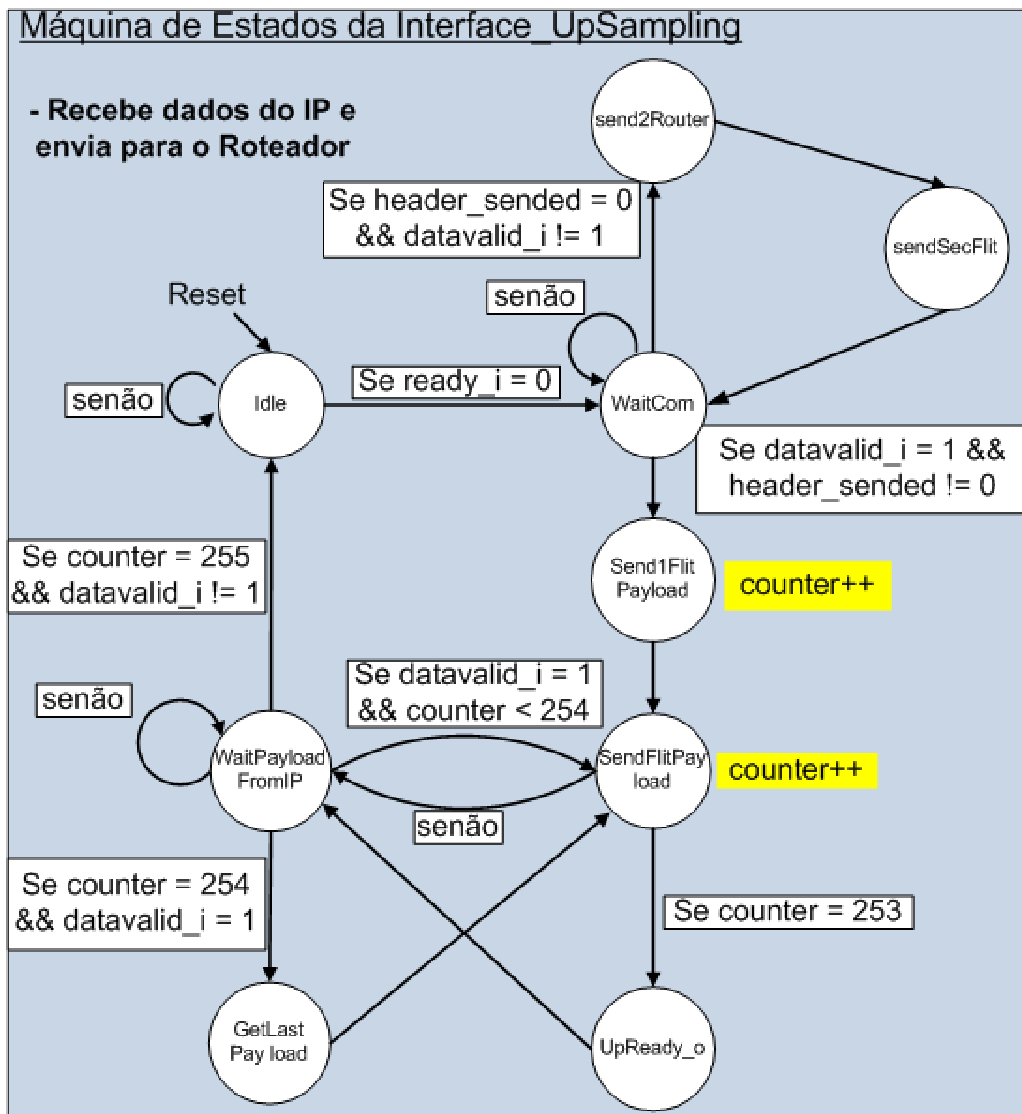


Figura 4.5 - Máquina de estados da interface *UpSampling*.

WaitCom: É o estado que espera o UpSampling indicar quando se vai enviar dados novos. Contudo, antes que isso aconteça, para acelerar o processo, este estado já encaminha para o roteador o cabeçalho e o tamanho de carga útil em sequência. Então, se essas informações não foram enviadas ainda (header_sended=’0’), a máquina passa para o estado Send2Router. Senão este estado fica à espera de dados do UpSampling, indicados por datavalid_i ativo alto. Se isto ocorrer, um flit de carga útil é gerado, para ser enviado no próximo estado, e a máquina passa para o estado Send1FlitPayload.

Send2Router: É o estado que simplesmente envia o cabeçalho para o roteador, indicando *tx*=’1’ e *data_o* recebe o endereço, em 32 bits, do roteador que está conectado no destino, neste caso, o *YCbCr2RGB*. A seguir, a máquina passa para o estado *SendSecFlit*.

SendSecFlit: É o estado responsável por enviar o segundo *flit* de controle do pacote, que indica a quantidade de *flits* que serão enviados a partir deste. Então, neste caso, *data_o* recebe 255 (0xFF) em 32 bits, *tx* continua em ‘1’ e este estado deve indicar que os dados de controle foram enviados, logo, *header_sended*=’1’. Então a máquina passa para o estado *WaitCom* novamente, para esperar os dados.

Send1FlitPayload: Este estado foi criado para enviar o primeiro *flit* de carga útil ao roteador e também pegar o segundo conjunto de dados e empacotar para o próximo estado enviar, pois o componente *UpSampling* envia os dois primeiros dados em sequência e não existiria tempo suficiente para pegar o segundo. Então, este estado indica ao *UpSampling* que a máquina está

ocupada (*ready_o*='0'), fazendo com que não seja enviado o terceiro conjunto de dados no próximo ciclo de relógio, sincronizando a máquina. Portanto, neste estado, *counter* é incrementado em '1' e o *tx*='1', e a máquina passa para *SendFlitPayload*.

SendFlitPayload: Este estado é responsável por enviar (*tx*='1') o *flit* de carga útil criado pelo *WaitPayloadFromIP,* ou pelo *Send1FlitPayload* (segundo conjunto de dados), ou pelo *GetLastPayload* (último conjunto de dados). Indica que a máquina está livre para recepção (*ready_o*='1'), incrementa *counter.* Se *counter* for igual a 253, então o próximo estado é *UpReady_o*. Senão, vai para o estado *WaitPayloadFromIP*.

WaitPayloadFromIP: Este estado é responsável por empacotar os dados recebidos do *UpSampling* para o *SendFlitPayload* enviar. Indica *tx*='0'. Se *counter* for igual a 255 então já foram enviados todos os dados e a máquina passa para o estado *Idle.* Se *counter* for igual a 254, a máquina para para o estado *GetLastPayload.* Contudo, se a máquina receber um novo conjunto de dados (*datavalid_i*='1'), a máquina retorna para *SendFlitPayload.*

UpReady_o: Este estado é responsável por forçar o sinal *ready_o* a ficar em ativo alto, ou seja, este estado funciona em conjunto com o *WaitPayloadFromIP* e com o *GetLastPayload* para resolver aquele mesmo problema do início da transação, que o *UpSampling* envia dois conjuntos de dados em sequência. Porém, neste caso, enviam-se os dois últimos conjuntos de dados em sequência. Coloca-se *tx* em '0' e a máquina passa para *WaitPayloadFromIP*.

GetLastPayload: Este estado é responsável por receber e empacotar o último *flit* de carga útil da transação e passar para o estado *SendFlitPayload.*

O sinal *ready_o* possui uma lógica diferenciada para o controle de fluxo com o componente *UpSampling*, pois a cada dado recebido dele, a interface deve ser capaz de indicar que está ocupada e não pode receber mais dados.

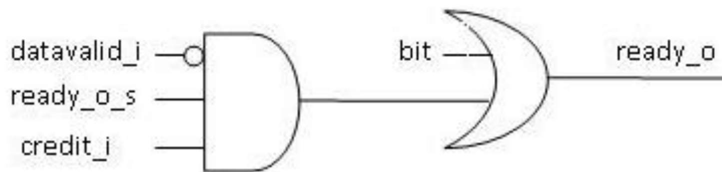


Figura 4.6 - Lógica do *ready_o* para o controle de fluxo com o *UpSampling*.

Como ilustrado na Figura 4.6, a lógica do *ready_o* depende de quatro outros sinais. O sinal *datavalid_i*, em nível alto, indica que a interface está recebendo dados do *UpSampling*. Logo, deve indicar que está ocupada para receber dados. Contudo, mesmo com *datavalid_i* em nível baixo, o *credit_i* também determina se a interface está ocupada, pois se está em nível baixo, indica que o *buffer* do roteador está cheio e não pode mais receber *flits.* Portanto, também não se pode receber dados. O sinal *ready_o_s* é um sinal interno, controlado pela máquina de estados, por exemplo o estado *SendFlitPayload*, que indica *ready_o* para '1'. O sinal *bit s*erve para forçar o *ready_o* a ficar em '0', no caso do estado *UpReady_o*.

4.2.2.3 YCbCr2RGB Interface

O módulo *YCbCr2RGB* possui três sinais de saída e três de entrada com informações referentes a imagem que está sendo decodificada. Os sinais restantes são para controle do *pipeline* e do decodificador. Os sinais *datavalid* e *ready* são para controle de fluxo e os de *contexto* para indicar a situação geral do decodificador. O roteador possui os sinais já mostrados no Capítulo 2.

O diagrama de blocos da interface entre o *YCbCr2RGB* e a NoC possui a interface ilustrada na Figura 4.7.

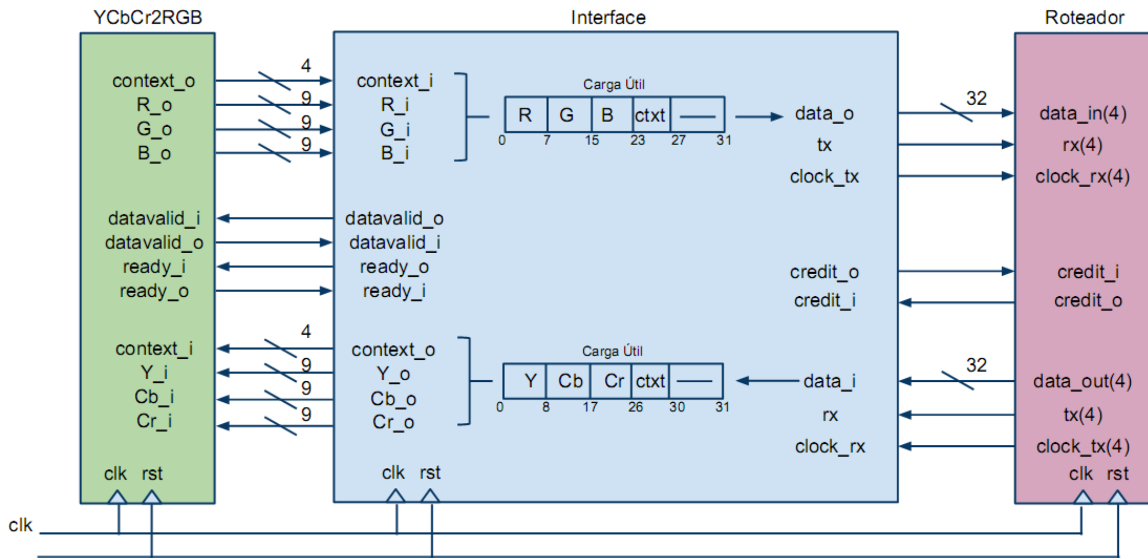


Figura 4.7 - Diagrama de blocos da interface entre *YCbCr2RGB* e a NoC.

O componente *YCbCr2RGB* recebe do *UpSampling* os 256 dados (MCU), que contêm informações de Y, Cb e Cr. Cada *pixel* recebido passa por uma transformação de cor, utilizando a equação (3.1), onde as informações de cores são alteradas do formato YCbCr para o formato RGB, após finalizar o processo em cada *pixel* da MCU recebida, o componente *YCbCr2RGB* envia o resultado para o componente VGA. A lógica de controle de fluxo deste componente é bastante simples. Devido ao grande volume de informações o componente recebe dados ininterruptamente, ou seja, o sinal *ready_i* sempre está em nível lógico alto.

Como mostrado na Figura 4.7, os dados de entrada *R*, *G*, *B* e *context_i* devem ser empacotados pela interface e assim encaminhados, como um *flit* de carga útil, para o roteador. Também, no sentido contrário, o *data_i,* recebido do roteador, deve ser quebrado para ser enviado ao *YCbCr2RGB*. Segue abaixo a descrição em VHDL.

Empacotar:

```
payload(7 downto 0) <= R_i;
payload(15 downto 8) <= G_i;
payload(23 downto 16) <= B_i;
payload(27 downto 24) <= context_i;
```

Desempacotar:

```
Y_o <= data_i(8 downto 0);
Cb_o <= data_i(17 downto 9);
Cr_o <= data_i(26 downto 18);
context_o <= data_i(30 downto 27);
```

A implementação da interface, tornou-se simples a partir desta análise, e a sua máquina de estados para o sentido componente-roteador possui a estrutura mostrada na Figura 4.8. Descreve-se abaixo a operação desta máquina, que representa a máquina de estados da interface *YCbCr2RGB* que recebe os dados do roteador e os envia para o IP. Novamente, somente as informações mais relevantes para o processo serão mostradas.

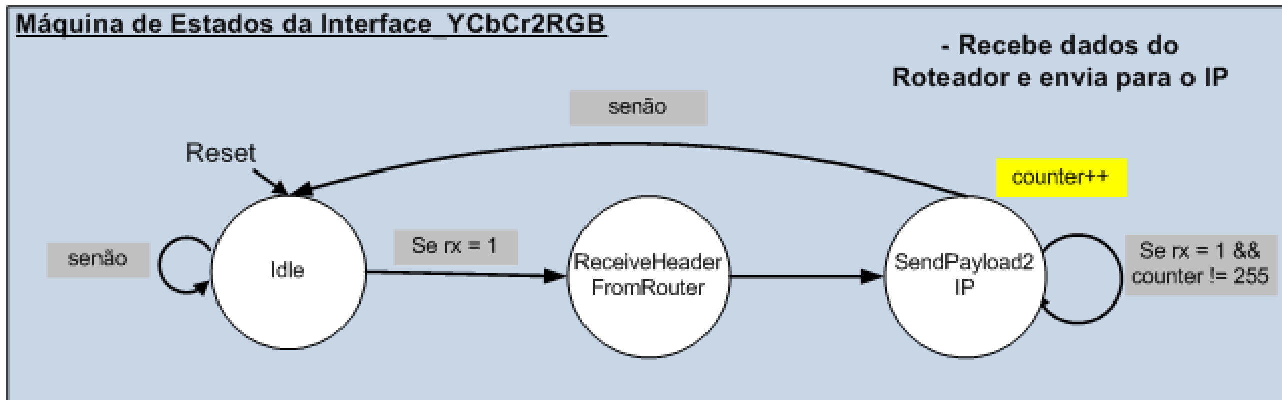


Figura 4.8 - Máquina de estados da interface *YCbCr2RGB*.

Idle: É o estado de inicialização da máquina. Este estado é atingido quando o sinal *reset* é ativado ou quando a máquina fica em modo de espera por novos dados. Neste estado, alguns sinais de controle devem ser reinicializados, pois este é sempre o começo de um novo ciclo de tráfego de dados. Então, *credit_o*='1' e *datavalid_o*='0', para indicar ao roteador que a máquina está pronta para receber dados, e também indicar ao *YCbCr2RGB* que não há dados para enviar. O contador *counter* deve ser reiniciado. Quando a interface receber um sinal *rx* ativo alto do roteador, significa que o último está enviando um *flit* do tipo cabeçalho. Então, a máquina passa para o estado *ReceiveHeaderFromRouter.*

ReceiveHeaderFromRouter: A única funcionalidade deste estado é gastar um ciclo de relógio, pois o dado recebido do roteador é o tamanho em *flits* de carga útil recebido durante a transação. O cabeçalho foi perdido quando a máquina estava em *Idle,* mas estas informações são irrelevantes para a interface. Em seguida, a máquina passa para o estado *ReceivePayloadFromRouter*.

SendPayload2IP: É o estado em que, enquanto o sinal *rx* estiver em '1', se recebe os dados do roteador, desempacota-se estes e os envia ao *YCbCr2RGB,* indicando *datavalid_o* em '1'. A lógica para o sinal de controle *datavalid_o* é que ele é igual ao *rx*, ou seja, quando a interface recebe um dado do roteador (*rx*='1'), a interface pode no mesmo ciclo de relógio enviar o dado para o YCbCr2RGB (*datavalid_o*='1'). A cada envio deve-se incrementar o contador *counter*, e quando este chegar a 255 a máquina retorna ao estado *Idle*. Senão fica-se neste estado a espera de dados do roteador.

5 VALIDAÇÃO

Este Capítulo apresenta-se a validação do trabalho desenvolvido. Duas etapas são fundamentais: A prototipação inicial e a simulação com a NoC. A prototipação inicial foi realizada através da ferramenta ISE da fabricante *Xilinx*. Já a simulação foi realizada com a ferramenta *ModelSim*. É de importância também neste Capítulo a comparação dos resultados obtidos através da simulação do decodificador com NoC e o decodificador original. Uma descrição das dificuldades encontradas também é fornecida ao final do Capítulo, bem como as soluções encontradas para contornar estas dificuldades.

5.1 Prototipação Inicial

Esta etapa, que foi realizada sob o sistema operacional Linux Ubuntu, e consistiu em prototipar o projeto original do *OpenCores*, verificar o seu correto funcionamento e analisar a execução em tempo real. As principais mudanças na estrutura que precisaram ser feitas são relacionadas ao fato do decodificador original ter sido implementado com os ambientes ISE/EDK na versão 8.2 e configurado para utilizar uma memória externa DDR2 SDRAM de 512MB.

O presente trabalho foi modificado para utilizar uma versão mais recente do ambiente ISE/EDK. Foi escolhida a versão 10.1, já que versões mais atuais (e.g. 11.1 e 12) não oferecem mais suporte à família Virtex-II Pro de FPGAs. Uma opção pela redução da memória DDR2 SDRAM para 256MB foi realizada para adaptar o projeto à placa disponível.

A aplicação que executa o decodificador também foi alterada. Essa aplicação foi desenvolvida em C e na versão original o usuário precisa digitar os comandos de execução através de algum *Hyperterminal*, previamente conectado via UART com a FPGA. O comportamento desta aplicação foi alterado para que o usuário não precise inserir um comando para executar a decodificação. Assim, o resultado da decodificação será exibido no monitor após alguns segundos da finalização do *download* do arquivo de imagem JPEG ou vídeo M-JPEG na placa. Apesar da nova aplicação não permitir muita interação com o usuário final, os comandos citados ainda são utilizados pela aplicação. Uma lista de tais comandos e sua funcionalidade está presente no ANEXO C.

A verificação funcional neste caso é visual. É necessário conectar a máquina de desenvolvimento à plataforma XUP V2PRO através da UART ou USB e a um monitor ou projetor através do conector VGA. Para a validação, foi escolhido utilizar a USB como interface de comunicação. Após estas conexões é necessário interagir com o processador Power PC do FPGA e realizar o *download* do arquivo *bitstream* para a FPGA, que contém os dados de configuração do FPGA. Esta etapa pode ser realizada por uma aplicação XMD, dentro do ambiente ISE, ou então usando o software *Impact* do mesmo ambiente. Com o arquivo de *bitstream* carregado no FPGA, é possível realizar o *download* do arquivo que será decodificado.

Os seguintes passos são utilizados para executar a verificação. Para isto é necessário usar o fluxo EDK→Project→EDK Shell, digitando os seguintes comandos:

- **system –f system.make init_bram** – Compila e sintetiza o projeto, gerando todos os arquivos necessários, desde o *netlist* até o *bistream*.
- **system –f system.make download** – Usado para fazer o *download* do *bitstream* para a placa. É necessário para habilitar a conexão com o Power PC.
- **xmd** – Abre o XMD (Xilinx Microprocessor Debug).
- **connect ppc hw** – Conecta-se ao Power PC.

- **dow –data "nome_arquivo.extensão" 0x0** – Faz o *download* do arquivo a ser decodificado para a memória da placa.
- **exit** – Sai do XMD.
- **system –f system.make download** – Após este segundo *download*, da-se início ao processo de decodificação, exibindo os resultados no monitor. Este passo leva aproximadamente 3 segundos para iniciar.

A fim de se evitar a presença de dados espúrios na memória externa da placa, o que pode gerar deformações na imagem exibida, recomenda-se desligar a placa a cada *download*.

Após a prototipação, com os arquivos de *log* gerados pela ferramenta, foi possível coletar algumas informações importantes sobre a lógica de utilização. O número de slices do projeto totalizou apenas 28% dos slices disponíveis no FPGA. O resultado da prototipação inicial, ou seja, a correta decodificação de um vídeo M-JPEG é demonstrada na Figura 5.1.

Figura 5.1 - Prototipação inicial de um vídeo em M-JPEG.

5.2 Simulação com a NoC

Para validar a primeira fase deste projeto, após a validação da prototipação inicial, iniciou-se o processo de estudo e criação de uma NoC que atendesse aos requisitos do decodificador. O maior problema relaciona-se aos requisitos temporais do decodificador, ou seja, conseguir satisfazer as figuras de vazão necessárias para a correta decodificação dos dados, bem como garantir a existência de um dado válido em cada instante de envio à interface VGA.

Assim, utilizando o ambiente ATLAS foi gerada uma NoC HERMES com a seguinte configuração:

- Dimensão: 2x2
- Topologia: Mesh 2D
- Escalonamento: Round-Robin
- Tamanho do *Flit*: 32 bits
- Profundidade do *Buffer*: 4 *flits*
- Algoritmo de Roteamento: Roteamento XY
- Canais Virtuais: 1
- Controle de Fluxo: Baseado em Créditos

Para validar a simulação, foi mantida a frequência de operação original do decodificador, que era de 100 MHz, e utilizado um *testbench* com um ciclo de relógio de 10 ns. O testbench busca a imagem Lenna, reproduzida na Figura 5.2, envia-a para o bloco JPEG *Decoder* para decodificação e salva em um arquivo de *log* os tempos e os resultados (componentes RGB) da decodificação.

Figura 5.2 - A imagem Lenna.jpg, usada para validar a decodificação.

Este arquivo de *log* facilita a validação do sistema, pois basta comparar o *log* da simulação original com o *log* da simulação utilizando a NoC, como mostra a Figura 5.3.

Optou-se por usar uma única imagem, pois leva cerca de 45 minutos para simular a decodificação completa. Assim, para simular 1 segundo de vídeo (24 imagens), demoraria cerca de 18 horas.

Original

1	44340 ns	0x00e1897c
2	44350 ns	0x00e1897c
3	44360 ns	0x00e18882
4	44370 ns	0x00e08781
5	44380 ns	0x00e38877
6	44390 ns	0x00de8372
7	44400 ns	0x00e6887c
8	44410 ns	0x00e4867a
9	44420 ns	0x00e48c7c

M-JPEG HERMES

1	44390 ns	0x00e1897c
2	44400 ns	0x00e1897c
3	44440 ns	0x00e18882
4	44480 ns	0x00e08781
5	44520 ns	0x00e38877
6	44560 ns	0x00de8372
7	44600 ns	0x00e6887c
8	44640 ns	0x00e4867a
9	44680 ns	0x00e48c7c

Figura 5.3 - Início dos *logs* de decodificação.

Como ilustrado na Figura 5.3, pode-se concluir que os dados, resultantes da decodificação, estão de acordo com o original. Nota-se que os dois primeiros dados possuem um atraso de 50 ns, e depois do segundo dado, a saída é gerada a cada 40 ns. Na decodificação original (sem NoC), cada dado é obtido em 10 ns, ou seja, a cada ciclo de relógio. Portanto, percebe-se que os resultados da modificação realizada são quatro vezes mais lentos do que o original.

Este arquivo de *log*, também permite analisar se nenhum dado foi perdido ou enviado incorretamente ou duplicado. O atraso gerado pela inserção da NoC entre os dois componentes não foi possível de ser analisado a nível de execução, pois a nova implementação ainda não foi prototipada corretamente.

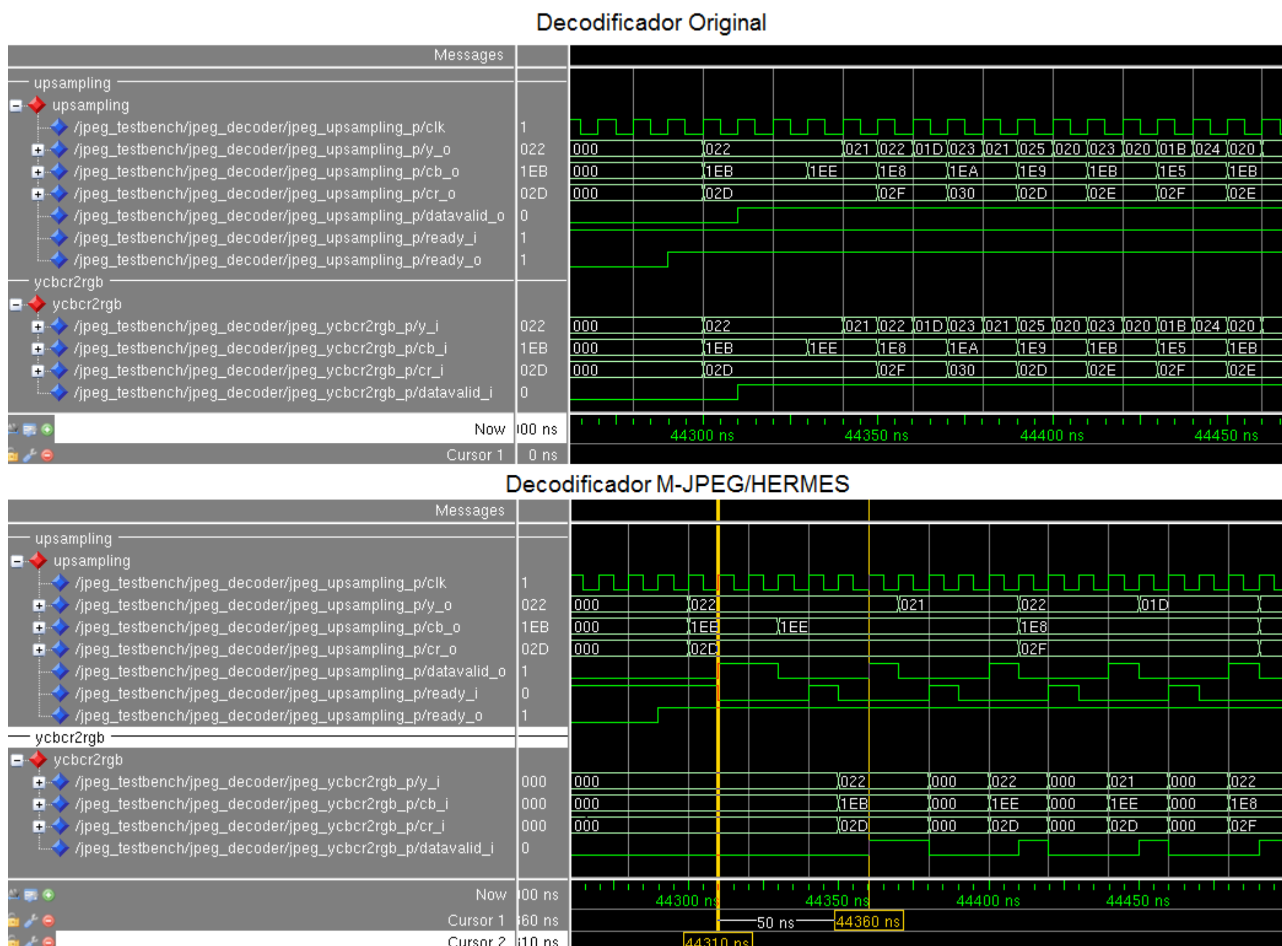


Figura 5.4 - Comparação do início de uma transação para o decodificador original e com NoC.

As formas de onda, ilustradas na Figura 5.4, destinam-se a uma análise do início da transação. Para o decodificador original, o componente *YCbCr2RGB* recebe os dados do

UpSampling no mesmo momento que estes são enviados. Contudo, utilizando a NoC, nota-se que a transação inicial demora 50 ns para alcançar o seu destino.

É importante ressaltar que depois que o fluxo de dados é sincronizado, os dados, utilizando NoC, são enviados a cada 40 ns, como é mostrado na Figura 5.5.

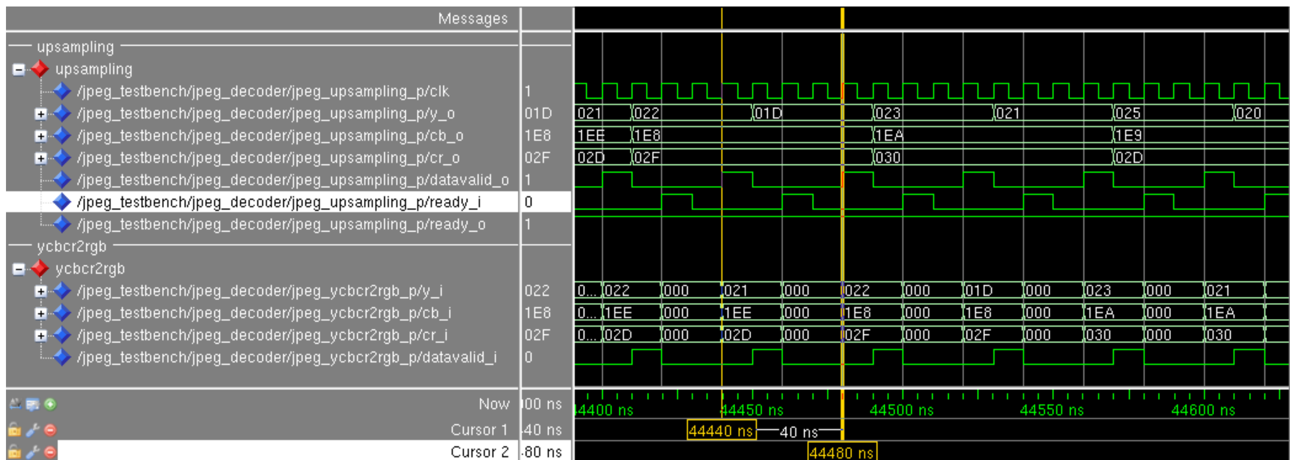


Figura 5.5 - Tempo de Envio de cada dado.

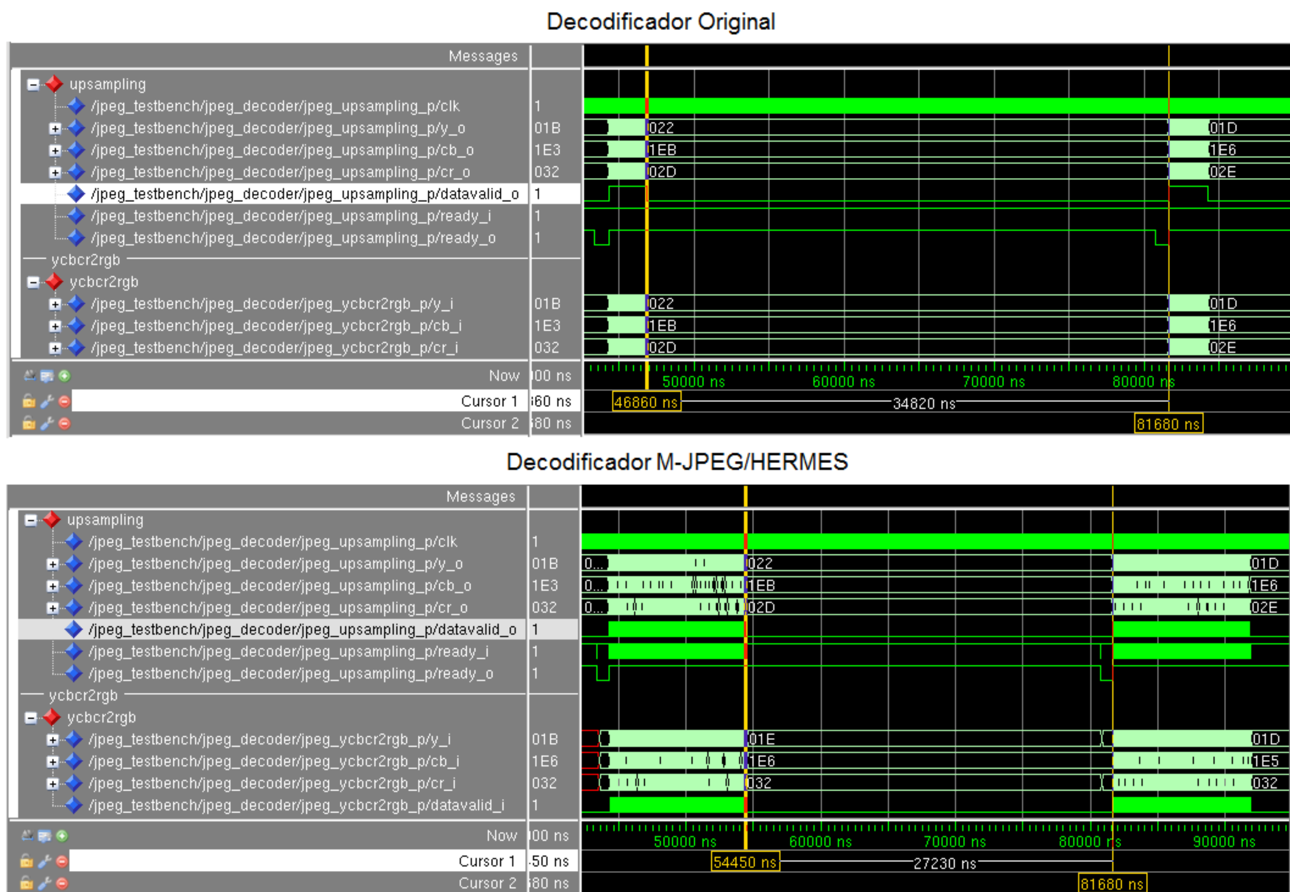


Figura 5.6 - Comparação entre duas Transações.

A Figura 5.6 evidencia uma informação importante em relação às particularidades dos dois componentes em questão, *UpSampling* e *YCbCr2RGB*.

Analisando uma transação completa dos dois componentes, existe uma diferença muito grande entre os tempos. Na solução original, uma transação ocorre em 2.560 ns, enquanto que na solução com NoCs, uma transação acontece em 10.150 ns, ou seja, aproximadamente quatro vezes mais lenta. Contudo, na próxima transação, os dois casos iniciam sempre no mesmo instante, pois neste intervalo, o *UpSampling* fica recebendo blocos de 8x8 *pixels*, enquanto a solução mais lenta

possui tempo suficiente para isso. Este intervalo, conforme na Figura 5.6 possui 27.230 ns. Entretanto, como existe um período longo entre transações, pode-se adiantar que não se espera que haja qualquer tipo de perda na imagem, pois sequências de imagens são tratadas em períodos idênticos (cada processo de decodificação dura 43.9 ms).

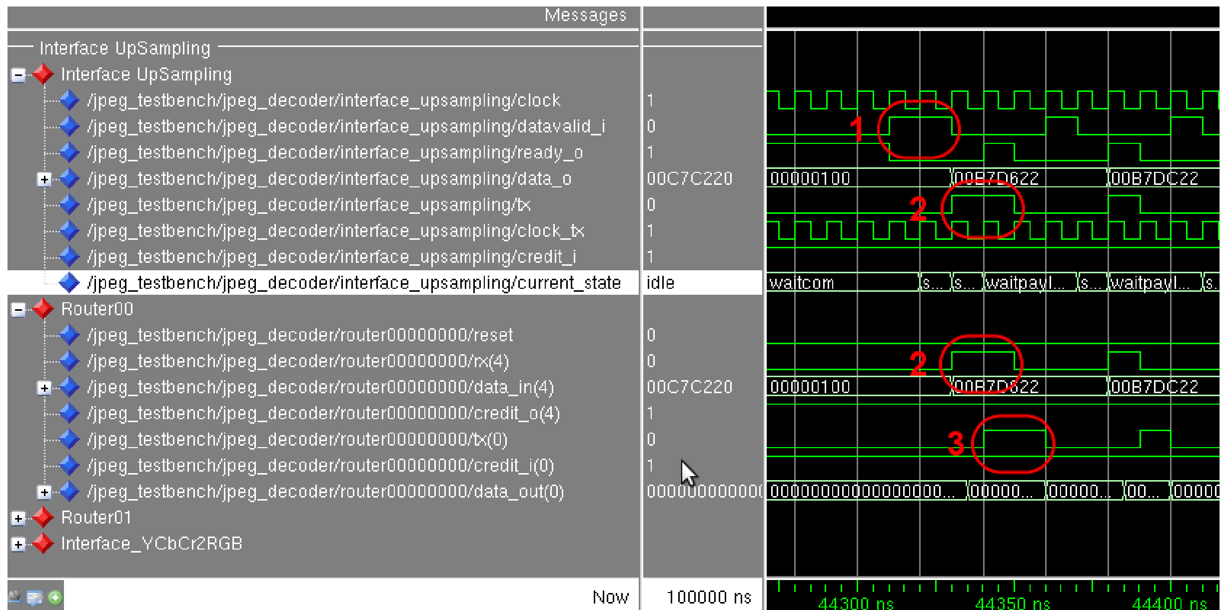


Figura 5.7 - Envio de dados sentido Interface-Roteador.

A Figura 5.7 demonstra o envio de dados do componente *UpSampling* até o primeiro roteador. Assim seguem as seguintes situações:

1. Nesta marcação, a interface está recebendo os dois primeiros dados em sequência, como descrito na Seção 4.2.2.2. Logo após, os mesmos são enviados;
2. Esta marcação demonstra que a interface indica *tx* em ‘1’, e a cada ciclo de relógio é enviado um dado para o roteador, este por sua vez, está com *rx* também em ‘1’, o que indica que está recebendo os dados normalmente;
3. A marcação confirma que o roteador esta encaminhando o dado para o outro roteador, pois *tx* está em ‘1’.

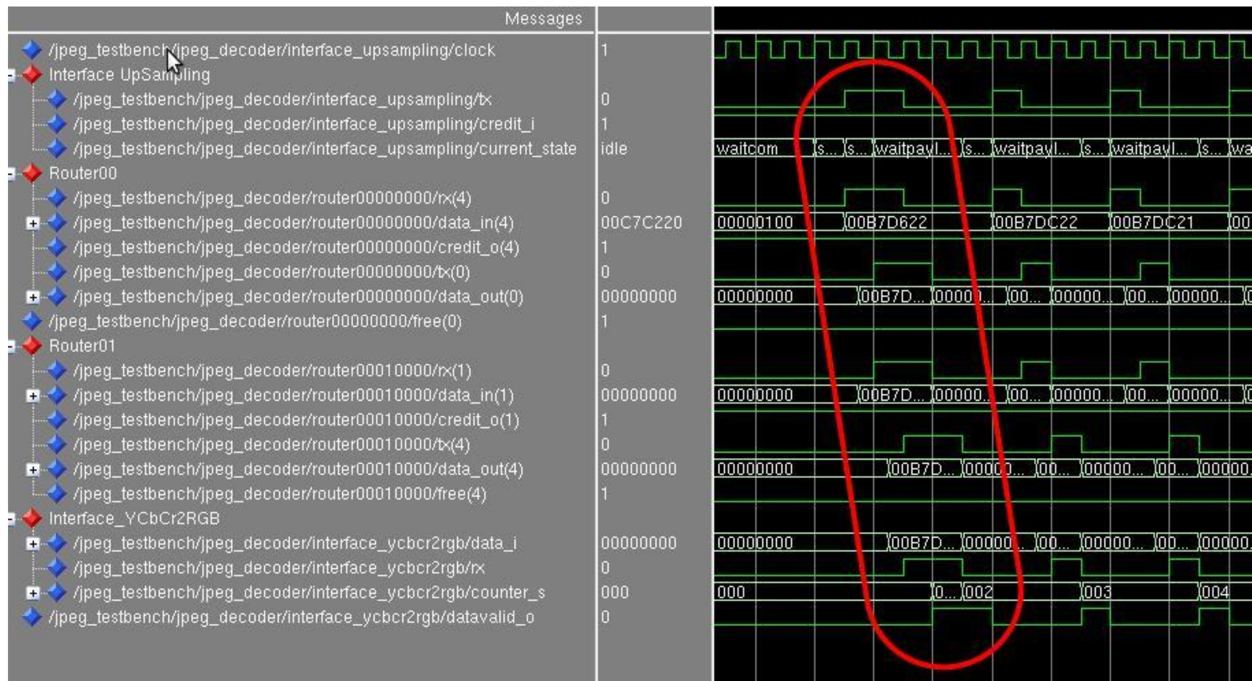


Figura 5.8 - Interfaces e roteadores.

A Figura 5.8 ilustra o fluxo de dados entre interface *UpSampling* e *YCbCr2RGB*. A marcação destaca os dois primeiros dados da transação, mas também pode-se perceber que, logo em seguida, o fluxo é sincronizado os dados são enviados a cada 40 ns.

Original

262136	43872440 ns	0x00813b54
262137	43872450 ns	0x00964258
262138	43872460 ns	0x00933f55
262139	43872470 ns	0x00a54252
262140	43872480 ns	0x00ac4959
262141	43872490 ns	0x00af424b
262142	43872500 ns	0x00b3464f
262143	43872510 ns	0x00b5454f
262144	43872520 ns	0x00b94953

M-JPEG HERMES

262136	43879870 ns	0x00813b54
262137	43879910 ns	0x00964258
262138	43879950 ns	0x00933f55
262139	43879990 ns	0x00a54252
262140	43880030 ns	0x00ac4959
262141	43880070 ns	0x00af424b
262142	43880110 ns	0x00b3464f
262143	43880150 ns	0x00b5454f
262144	43880160 ns	0x00b94953

Figura 5.9 - Final do arquivo de *log* gerado pelo *testbench*.

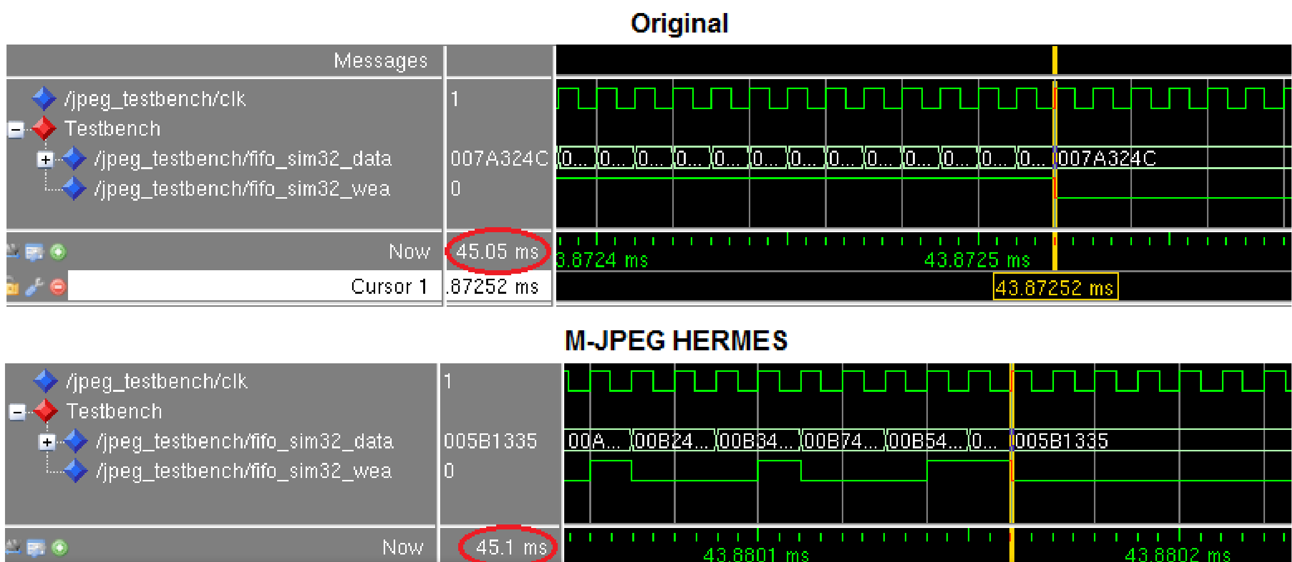


Figura 5.10 - Formas de onda indicando o final da decodificação.

Concluindo a validação do projeto, a Figura 5.9 e a Figura 5.10 ilustram o final da decodificação. No arquivo de *log*, confirma-se a integridade dos dados resultantes e ainda, juntamente com as formas de onda, confirmam-se os períodos finais do processo. Os destaques em vermelho indicam o tempo que foi utilizado na simulação das duas soluções, contudo, em ambos os casos, a decodificação foi finalizada em aproximadamente 43.9 ms.

5.3 Dificuldades Encontradas

Ao longo do projeto existiram diversos tipos de dificuldades, que por sua relevância serão discutidas nesta Seção. As dificuldades listadas e comentadas abaixo, dizem respeito à parte inicial do projeto, onde o decodificador M-JPEG descrito no Capítulo 3 foi prototipado na plataforma XUP V2PRO.

Ambiente EDK: Os autores desconheciam totalmente a funcionalidade básica do ambiente de prototipação de sistemas embarcados EDK. Portanto, foi necessário realizar um longo período de familiarização com este. Este período propiciou a experiência necessária na ferramenta para iniciar o processo de prototipação. Para isso, o orientador recomendou a utilização de tutoriais do ambiente EDK usado em turmas de pós-graduação. Este "Tutorial EDK", pode ser aplicado utilizando a ferramenta em dois sistemas operacionais, Windows e Linux. Porém, ao tentar seguir os seus passos, os resultados esperados não eram obtidos corretamente, o problema então, encontrava-se em diversos fatores. No Windows XP, um problema de memória recorrente de uma sub-aplicação da ferramenta ocorria ao tentar gerar as *libs*. Então, a ferramenta se encerrava automaticamente. Já no Red Hat, o tutorial apresentava problemas no método de criação de um periférico de interrupção.

Problemas na FPGA: Após compilar e gerar os arquivos necessários para realizar a prototipação, o próximo passo era realizar, via conexão USB, o *download* do *bitstream*, então após algumas tentativas fracassadas, notou-se que o erro de prototipação era causado pela placa que estava com defeito e depois pelo não reconhecimento do *driver* USB. A troca da placa e a instalação correta do driver resolveu o problema, mas só depois de vários dias.

SDRAM da FPGA: Como o projeto original foi prototipado com uma memória externa (SDRAM) de 512 MBytes, houve grande dificuldade em alterar o mesmo (dada a não familiaridade com o EDK), pois na plataforma disponível a memória externa disponível era de 256 MBytes. Finalmente, conseguiu-se resolver via EDK este problema de configuração do ambiente do projeto.

Smart Models: Ao tentar simular o projeto após a prototipação, a ferramenta modelsim necessitava de instalação em separado de *smart model* de vários componentes da Xilinx, tema que

os alunos não dominavam, Após estudo, descoberta do motivo do problema e estudo do processo de instalação, uso e utilidade dos *smart models*, pode-se resolver o problema.

Controle de Fluxo: Como descrito anteriormente no Capítulo 3, o controle de fluxo utilizado pelos componentes no projeto original é baseado em *handshake*, mas ao inserir a NoC no projeto, foi necessário realizar algumas alterações no controle de fluxo, pois os roteadores da NoC deviam utilizar (por questões de desempenho) uma comunicação baseada em créditos.

6 CONSIDERAÇÕES FINAIS

Este trabalho propôs uma arquitetura resultante da integração de um decodificador M-JPEG e a NoC HERMES.

As etapas de desenvolvimento ocorreram de formas sequenciais, primeiramente ocorreu a prototipação inicial, o qual contempla a prototipação do decodificador M-JPEG e a obtenção dos resultados, logo após foi realizado o estudo para desenvolver uma arquitetura utilizando a NoCs. Definido isso, iniciou-se a criação e a adaptação dos roteadores no decodificador, devido ao tempo apertado do desenvolvimento, somente dois componentes foram interligados com a NoC. Então, foi possível verificar os novos resultados e criar estimativas futuras, caso fossem criadas as interfaces restantes.

Através da simulação realizada para a validação confirmou-se a integridade dos dados resultantes e ainda, os períodos finais do processo, sendo a decodificação finalizada em aproximadamente 43.9 ms, garantindo a vazão dos dados necessários para a correta decodificação e respeitando então o requisito temporal do decodificador quanto aos seus processos.

Quanto à necessidade de se enviar os dados válidos no tempo correto para a VGA, não obtivemos um resultado conclusivo, pois não realizamos a prototipação final do projeto. Um consenso geral do grupo é de que este tempo ainda esta sendo respeitado, devido à mudança de domínio de frequência de 100 MHz para 25 MHz, realizada no componente VGA, acreditamos que os dados estão sendo enviados para este componente no valor limite de sua operação, onde no decodificador original, estes dados seriam recebidos a cada ciclos de relógio no domínio de 25 MHz, equivalente a 40 ns. Como a saída dos dados do decodificador M-JPEG/HERMES, nunca excede este valor, pois 40 ns é o tempo máximo de cada envio de dados, este valor ainda estaria respeitando o tempo necessário de envio de dados para a VGA.

Por este trabalho ser uma arquitetura proposta, adaptações futuras podem ser implementadas sem muitas dificuldades, assim cito duas possíveis adaptações, sendo que ambas podem ser aplicadas independentemente ou em conjunto. Uma possível adaptação seria alterar os módulos descritos em *hardware* por módulos de mesma funcionalidade descritos em *software* e utilizar a ferramenta de multiprocessada HeMPS, assim seria verificado os resultados da prototipação original com a prototipação com a plataforma HeMPS e constatado qual metodologia apresenta melhor desempenho. Outra possível adaptação seria alterar o decodificador M-JPEG para um decodificador MPEG, pois a norma MPEG para codificação e decodificação de vídeos se baseia fortemente nas técnicas utilizadas pela norma JPEG. A especificação MPEG é um pouco mais restrita que a especificação JPEG, então algumas partes do modelo realizado é mais simplificada do que implementada no decodificador M-JPEG. As tabelas de *Huffman* não são armazenadas no cabeçalho e sim definidas pela especificação MPEG, assim as tabelas deveriam ser descritas em código.

7 REFERÊNCIAS BIBLIOGRÁFICAS

[AHM74] Ahmed, N., Natarajan, T., Rao, K. R. "On image processing and a discrete cosine transform". IEEE Transactions on Computers, vol. 23-1, Jan 1974, pp. 90-93.

[BAS06] Bastos, É. "Uma Rede Intra-chip com Topologia Toro 2D e Roteamento Adaptativo". Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2006. 148 p.

[BEN01] Benini, L.; De Micheli, G. "Powering networks on chips: energy-efficient and reliable interconnect design for SoCs". In: 14th International Symposium on Systems Synthesis, 2001, pp. 33-38.

[BEN02] L. Benini, G. De Micheli, Networks on chips: a new SoC paradigm, IEEE Comput. 35 (1), 2002, pp. 70–78.

[BER02] Bergamaschi, R. A., Cohn, J. "The A to Z of SoCs". In: International Conference on Computer-Aided Design. San Jose, California: IEEE, 2002. pp. 791-798.

[CAR09] Carara, E. A., Oliveira, R. P., Calazans, N. L. V., Moraes, F. G. "HeMPS - A Framework for Noc-Based MPSoC Generation". In: IEEE International Symposium on Circuits and Systems - ISCAS´09, Taipei, 2009. pp. 1345-1348.

[CAR09b] Carvalho, E. L. S. "Mapeamento Dinâmico de Tarefas em MPSoCs Heterogêneos baseados em NoC", Tese de Doutorado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2009, 170p.

[DAY83] Day, J., Zimmermman, H. "The OSI reference model". Proc. IEEE 71 (12), 1983, pp. 1334–1340.

[DIG01] DIGILENT Inc. "Virtex-II Pro Development System". Capturado em: http://www.digilentinc.com/Products/Detail.cfm?Prod=XUPV2P, Abril 2010.

[DYN02] Duato, J., Yalamanchili, S., Ni, L. "Interconnection Networks: An Engineering Approach", Morgan Kaufmann, Los Altos, CA, 2002, 624p, revised edition.

[FOR02] Forsell, M. "A scalable high-performance computing solution for networks on chips". IEEE Micro 22 (5), 2002, pp. 46–55.

[FRE09] de Freitas, H. C. "Arquitetura de NoC Programável Baseada em Múltiplos Clusters de Cores para Suportes a Padrões de Comunicação Coletiva", Tese de Doutorado, Programa de Pós-Graduação em Computação, UFGRS, 2009. 125p.

[FUJ01] FUJTSU. "Fujitsu Scientific & Technical Journal (FSTJ) System-on-a-Chip" Capturado em: http://www.fujitsu.com/global/newspublications/periodicals/fstj/archives/vol42-2.html, Abril 2010.

[GAP01] GAPH - Grupo de Apoio ao Projeto de Hardware. "The GAPH Homepage". Capturado em: http://www.inf.pucrs.br/~gaph, Abril 2010.

[GLA94] Glass, C.; NI, L. "The Turn Model for Adaptive Routing". Journal of the Association for Computing Machinery, v.41-5, 1994, pp. 874-902.

[GUE00] Guerrier. P.; Greiner. A. “A generic architecture for on-chip packet-switched interconnections”. In: Design Automation and Test in Europe (DATE’00), 2000, pp. 250-256.

[GUP97] Gupta, R., Zorian, Y., “Introducing core-based system design, IEEE Des. Test Comput. 14 (4), 1997 pp. 15-25”.

[HAM92] Hamilton, E. “JPEG File Interchange Format”. Capturado em: http://www.jpeg.org/public/jfif.pdf, Technical Report, C-Cube Microsystems, Milpitas, California, Set 1992.

[HP96] Hennessy, J., Patterson, D. “Computer Architecture: A Quantitative Approach”. Morgan Kaufmann, San Francisco, CA, 1996, 760p.

[HWA92] Hwang, K. “Advanced Computer Architecture: Parallelism, Scalability, Programmability”. McGraw-Hill, New York, 1992, 672p.

[IEE91] IEEE - The Institute of Electrical and Electronics Engineers, Inc.. “IEEE Standard Specifications for the Implementations of 8 x 8 Inverse Discrete Cosine Transform”. IEEE, 1991.

[JPE92] JPEG - Joint Photographic Experts Group. “Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines”. Technical Report, Comité Consultatif International Téléphonique et Télégraphique (CCITT), 1992.

[KJS02] Kumar, S., Jantsch, A., Soininen, J. P., Fonsell, M. “A Network on Chip Architecture and Design Methodology”. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI’02), April 2002, pp. 105–112.

[LI00] Li, Ze-Nian, Zhong, William. “CMPT 365 Multimedia systems”. Canada: Simon Fraser University, 2000.

[MAN08] Manz, S. “Development and Implementation of an *MotionJPEG* Capable *JPEG Decoder in Hardware*”, Dissertação de Mestrado, 2008,74p.

[MBV02] Marescaux, T., Bartic, A., Verkest, D., Vernalde, S., Lauwereins, R. “Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs”, In: Field-Programmable Logic and Applications (FPL’02), September 2002, pp. 795-805.

[MEL05] Mello, A.; et al. “Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC”. In: 18th SBCCI, pp. 178-183.

[MEL99] Melo, H. “Compressão digital de vídeo”. Revista de Engenharia de Televisão, 1999.

[MOR04] Moraes, F., Calazans, N., Mello, A., Moller, L., Ost, L. “HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip”. The VLSI Journal, v.38-1, 2004, pp. 69-93.

[NI93] Ni, L., et al., “A survey of wormhole routing techniques in direct networks”. IEEE Comput. 26 (2), 1993, pp. 62–76.

[RHO01] Rhoads, S. “Plasma – most MIPS I(TM) opcodes”. Capturado em: http://opencores.org/project,plasma, Abril 2010.

[SCH07] Scherer Jr., C. “Redes Intrachip com Topologia Toro e Modo de Chaveamento Wormhole: Projeto, Geração e Avaliação”, Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2007. 101 p.

[RGR03] Rijpkema, E., Goossens, K., Rădulescu, A. “Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip”. In: Design, Automation and Test in Europe (DATE’03), March 2003, pp. 350–355.

[RGW01] Rijpkema, E., Goossens, K., Wielage, P. “A router architecture for networks on silicon”. In: 2nd Workshop on Embedded Systems (PROGRESS’2001), Nov. 2001, pp. 181–188.

[TAN97] Tanenbaum, A. S. “Redes de Computadores”. Rio de Janeiro: Editora Campos, 1997.

[TED06] Tedesco, L.; et al. "Application Driven Traffic Modeling for NoCs". In: 19th Symposium on Integrated Circuits and System (SBCCI'06), 2006.

[VID01] VIDEO IMAGING DESIGN LINE. “Picking the right MPSoC-based video architecture: Part 1” Capturado em: http://www.videsignline.com/howto/219400306, Abril 2010.

[WOL05] Wolf, W., Jerraya, A. A. “Multiprocessor Systems-on-chip”. San Francisco: Morgan Kaufmann Publishers, 2005, 602p.

[WOS07] Woszezenki, C. R. “Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2007, 145p.

[WIK01] Wikipedia, a encyclopedia livre. “Motion JPEG” Capturado em: http://en.wikipedia.org/wiki/Motion_JPEG, Maio 2010

[XIL01] XILINX. “XUPV2P Documentation”. Capturado em: http://www.xilinx.com/univ/xupv2p.html, Abril 2010.

ANEXO A

TABELA DE MARCADORES JPEG

Tabela de marcadores JPEG			
Hexadecimal	Abreviatura	Nome Completo	Descrição
0xFFD8	SOI	Start Of Image	Uma imagem JPEG inicia com estes dois bytes. Este marcador não tem carga útil.
0xFFE0	APP0	Application Segment 0	Após o marcador SOI. Ele inicia com a sequência 0x4A46494600, que indica que o arquivo está descrito no formato JFIF. O restante da carga útil contém as informações de versão, densidade nos eixos x e y e uma pré visualização opcional da imagem.
0xFFC0	SOF	Start Of Frame	Inclui a precisão dos dados utilizada (normalmente 8 bits), largura e altura da imagem, número de componentes, fatores de amostragem e a ligação das tabelas de quantização com os componentes de cores.
0xFFDB	DQT	Define Quantization Table	Define onde as tabelas de quantização são armazenadas, normalmente é utilizado um marcador para cada tabela de quantização, mas um marcador DQT as vezes pode armazenar mais de uma tabela.
0xFFC4	DHT	Define Huffman Table	Define onde as tabelas de *Huffman* são armazenadas, normalmente é utilizado um marcador para cada tabela de *Huffman*, mas um marcador DHT as vezes pode armazenar mais de uma tabela.
0xFFFE	COM	Comment	Uma string com terminação em zero.
0xFFDA	SOS	Start Of Scan	Inclui o número de componentes da imagem, a ligação dos componentes com as tabelas de *Huffman*. Este é o último marcador do cabeçalho.
0xFFD9	EOI	End Of Image	Indica o final de uma imagem JPEG. Este marcador não tem carga útil.

ANEXO B

TABELA DE MARCADORES *HUFFMAN*

Tabela de marcadores *Huffman*			
Hexadecimal	Abreviatura	Nome Completo	Descrição
0x00	EOB	End Of Block	Indica que os próximos valores no bloco 8x8 são zeros.
0xF0	ZRL	Zero Run Length	Indica que existem 16 zeros precedentes ao código de *Huffman*, ou seja, se existirem 32 zeros precedentes, dois marcadores ZRL serão utilizados, e.g. "[zrl],[zrl]".

ANEXO C

TABELA DE COMANDOS DO DECODIFICADOR

Tabela de marcadores JPEG		
Comando	Nome do Comando	Descrição
1	Go	Inicia a aplicação.
2	No Go	Pára a decodificação.
5	Reset	Reinicia a decodificação.
14	Pause On	Pausa a decodificação.
15	Pause Off	Continua a decodificação.
16	Next Frame	Quando pausado, decodifica o próximo frame.
17	Faster	Decodifica frames rapidamente.
18	Slower	Decodifica frames com uma taxa reduzida.

ANEXO D

O Diagrama de Blocos do Componente VGA do MJPEG