

Leandro da Silva Freitas

Projeto em Nível RT de IPs Digitais

Florianópolis

Junho 2009

Leandro da Silva Freitas

Projeto em Nível RT de IPs Digitais

Monografia submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Luiz Cláudio Villar dos Santos

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Florianópolis

Junho 2009

Monografia de graduação sob o título “*Projeto em Nível RT de IPs Digitais*”, defendida por Leandro da Silva Freitas e aprovada em 06 de julho de 2009, em Florianópolis, Santa Catarina, pela banca examinadora constituída pelos professores:

Prof. Dr. Luiz Cláudio Villar dos Santos
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. José Luís Almada Güntzel
Universidade Federal de Santa Catarina
Membro da Banca

Prof. Dr. Olinto José Varela Furtado
Universidade Federal de Santa Catarina
Membro da Banca

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Abstract

1	Introdução	p. 9
1.1	Paradigmas contemporâneos de projeto	p. 10
1.2	Aplicações embarcadas contemporâneas	p. 11
1.3	Trabalho anterior	p. 12
1.4	Escopo deste trabalho	p. 13
2	Breve revisão bibliográfica	p. 14
2.1	Linguagens de descrição de hardware	p. 14
2.2	Protocolo de comunicação AMBA AXI	p. 15
2.3	Trabalho relacionado	p. 16
3	Metodologia	p. 18
3.1	Verificação funcional	p. 18
4	Desenvolvimento em nível RT	p. 21
4.1	DCT 2-D	p. 21
4.1.1	Arquitetura do IP	p. 21

4.1.2	Interface de comunicação	p. 22
4.2	DCT 1-D	p. 22
4.2.1	Arquitetura do módulo	p. 23
4.2.2	Arquitetura do <i>buffer</i> ping-pong	p. 24
4.2.3	Arquitetura do multiplicador	p. 25
4.2.4	Interface de comunicação	p. 27
4.3	<i>Buffer</i> de transposição	p. 29
4.3.1	Arquitetura do módulo	p. 29
5	Resultados da síntese do IP	p. 31
5.1	Configuração experimental	p. 31
5.2	Resultados para os principais componentes	p. 31
5.2.1	DCT 1-D	p. 31
5.2.2	<i>Buffer</i> de transposição	p. 33
5.3	Resultados globais	p. 34
6	Verificação funcional	p. 35
7	Conclusões e perspectivas	p. 38
7.1	Trabalhos em andamento	p. 38
7.2	Trabalhos futuros	p. 38
	Anexo A – Algoritmo DCT 1-D	p. 39
	Anexo B – Imagem utilizada para estimativa de consumo de energia	p. 41
	Referências Bibliográficas	p. 43

Lista de Figuras

1.1	<i>Design gap</i>	p. 9
1.2	<i>Time to market</i>	p. 10
3.1	Estrutura final do <i>testbench</i>	p. 19
4.1	Arquitetura do IP DCT 2-D	p. 21
4.2	Roteamento dos sinais de interface do IP DCT 2-D	p. 22
4.3	Arquitetura do módulo DCT 1-D	p. 23
4.4	Arquitetura do <i>buffer</i> ping-pong	p. 25
4.5	Interface do multiplicador	p. 26
4.6	Arquitetura do multiplicador	p. 26
4.7	Interface do módulo DCT 1-D	p. 27
4.8	Cenário 1: <i>valid</i> antes de <i>ready</i>	p. 28
4.9	Cenário 2: <i>ready</i> antes de <i>valid</i>	p. 28
4.10	Cenário 3: <i>valid</i> junto com <i>ready</i>	p. 28
4.11	Cenário 3: transferência em rajada utilizando o sinal <i>last</i>	p. 29
4.12	Interface do <i>buffer</i> de transposição	p. 30
4.13	Arquitetura do <i>buffer</i> de transposição	p. 30
6.1	Passo 1: estrutura “modelo de referência”	p. 36
6.2	Passo 2: estrutura “modelo de referência duplo”	p. 36
6.3	Passo 3: estrutura “emulação do DUV”	p. 36
B.1	Imagem “mibench_small” (contida no benchmark MiBench)	p. 41
B.2	Imagem “mibench_large” (contida no benchmark MiBench)	p. 42

Lista de Tabelas

4.1	Diferença da largura de palavra entre os módulos DTC 1-D	p. 24
4.2	Constantes utilizadas pelo multiplicador	p. 27
5.1	Resumo da síntese da DCT 1-D para o Cyclone II EP2C35F672C6	p. 32
5.2	Estimativa do consumo de energia da DCT 1-D para Cyclone II EP2C35F672C6	p. 32
5.3	Resumo da síntese do <i>buffer</i> de transposição para o Cyclone II EP2C35F672C6	p. 33
5.4	Estimativa do consumo de energia do <i>buffer</i> de transposição para Cyclone II EP2C35F672C6	p. 33
5.5	Resumo da síntese da DCT 2-D para Cyclone II EP2C35F672C6	p. 34
5.6	Estimativa do consumo de energia da DCT 2-D para Cyclone II EP2C35F672C6	p. 34

Resumo

O aumento da complexidade dos sistemas embarcados e a redução do *time to market* coloca em evidência a necessidade do surgimento de novos paradigmas de projeto de sistemas eletrônicos. Não é adequado, por exemplo, utilizar metodologia de desenvolvimento de sistemas digitais de aplicação específica para desenvolver *Systems-on-Chip* (SoCs). Os elevados custos de produção das máscaras para circuitos integrados e preparo da produção, faz com que a indústria se interesse por abordagens de projeto que conduzam à melhoria de eficiência e redução de custo. Uma dessas abordagens é o projeto baseado em plataforma.

O projeto baseado em plataforma consiste, basicamente, na adoção de uma plataforma-alvo, ou seja, uma arquitetura de referência associada ao reuso de componentes pré-projetados. Normalmente, tais componentes são desenvolvidos por terceiros, por isso são conhecidos como blocos de propriedade intelectual (IP). Essa plataforma pode então ser adaptada à aplicação específica através da inclusão ou remoção de IPs.

Este trabalho apresenta o processo de desenvolvimento de um IP que implementa a função DCT 2-D (*Discrete Cosine Transform* em duas dimensões), amplamente utilizada para o processamento de sinais e imagens. O fluxo de projeto parte de uma representação executável do sistema (no estilo TLM) contendo um modelo funcional em alto nível do IP até seu protótipo em FPGA. Este trabalho também apresenta o processo de verificação funcional utilizado durante o desenvolvimento, assim como a caracterização do IP em termos de desempenho, consumo de energia e número de componentes utilizados.

Abstract

The growth of the complexity of embedded systems and the reduction of the time to market require new electronic systems design paradigms. It is not adequate, for instance, to use the ASIC design methodology to develop Systems-on-Chip (SoCs). The high cost of the masks and the cost of production set-up are biasing manufacturers toward any design approaches leading to higher efficiency and lower costs. One of those approaches is the platform-based design.

The platform-based design consists, basically, on the adoption of a reference architecture combined with the reuse of pre-designed components. Usually, those components are designed by third-party and for that reason they are known as Intellectual Property Blocks (IPs). The platform can then be adapted to a specific application by the insertion or remotion of IPs.

This work presents the design process of an IP that implements the discrete cosine transform function in two dimensions (DCT 2-D), widely used for image and signal processing. The design flow starts from an executable representation of a system (in the TLM style) containing a high level functional model of the IP down to its FPGA prototype. This work also shows the functional verification methodology used during the process as well as the IP characterization in terms of performance, energy consumption and component usage.

1 Introdução

A indústria de semicondutores vem vivendo há cerca de uma década a revolução dos sistemas integrados (SoCs) (MARTIN; CHANG, 2003). Um SoC é um sistema computacional ou qualquer outro sistema eletrônico completo em um único chip, podendo conter componentes tais como processadores, memórias e barramentos, geralmente utilizado em sistemas embarcados. Sistemas embarcados são sistemas que não podem ser programados pelo usuário, por serem pré-programados para executarem uma tarefa específica, e que estão instalados dentro do equipamento ao qual servem. Os componentes de um SoC dedicados a tarefas específicas, tais como periféricos e aceleradores de funções críticas, são denominados blocos de *propriedade intelectual* (IPs). A Figura 1.1 ilustra o impacto da utilização de IPs no projeto de SoCs.

A taxa de crescimento da complexidade dos projetos de SoCs é aproximadamente 58% ao ano, enquanto que a produtividade dos projetistas aumenta em torno de 21% ao ano (BELANOVIC et al., 2003). Essa diferença entre o aumento de complexidade e produtividade é conhecida como *design gap*. Uma das principais soluções para a redução do *design gap* é o reuso de componentes.

O uso de IPs para o desenvolvimento de SoCs permite ao projetista concentrar-se no sistema

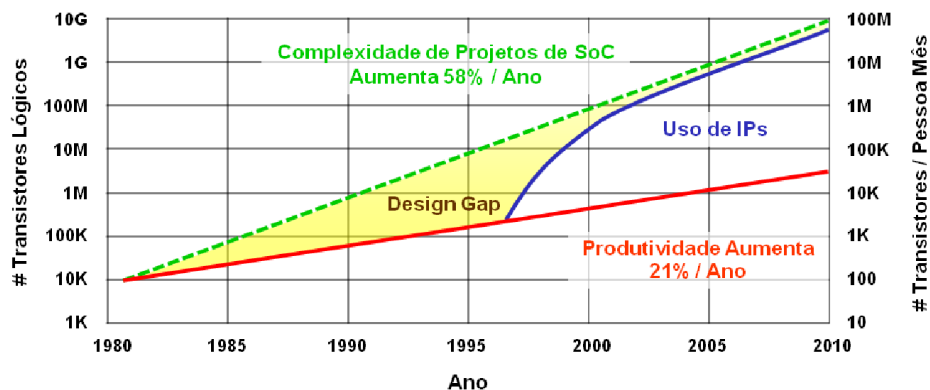


Figura 1.1: *Design gap*
 Fonte: Adaptado de Sematech (2009)

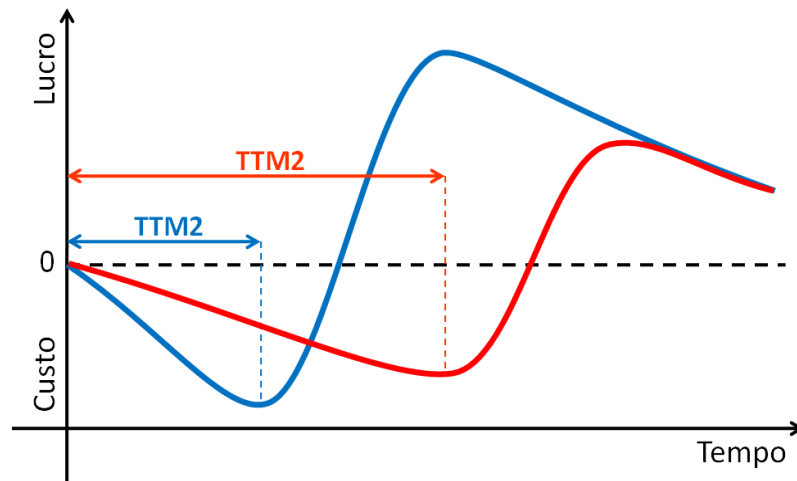


Figura 1.2: *Time to market*

de forma mais ampla sem se preocupar com os detalhes de cada componente individualmente. Outro benefício decorrente do aumento da produtividade é a redução do *time to market*. Caso o tempo entre o início do projeto de um produto e o seu lançamento para o mercado seja muito grande, ou pelo menos, se for maior do que o concorrente, isso se traduzirá em perda de representatividade do produto do mercado e a conseqüente diminuição do lucro. Na Figura 1.2, o projeto 1 lançou seu produto no tempo TTM(1) e, sem concorrência, teve a oportunidade de atingir uma grande margem de lucro até o tempo TTM(2), momento da chegada da concorrência ao mercado.

1.1 Paradigmas contemporâneos de projeto

O objetivo global no projeto de sistemas eletrônicos é balancear os custos de produção com o custo e tempo de desenvolvimento considerando questões de performance e funcionalidade. Devido ao elevado custo das fases de projeto, integração, processamento e verificação no ciclo de vida dos SoCs, a indústria está interessada em abordagens de projeto que levem ao aumento da eficiência e diminuição dos custos. Uma dessas abordagens é o Projeto Baseado em Plataforma ou *Platform-Based Design* (PBD), onde plataformas já integradas e verificadas servem como base para famílias de produtos derivados (BAILEY; MARTIN; ANDERSON, 2005).

Segundo Sangiovanni-Vincentelli e Martin (2001), o PBD tem como motivação, além dos fatores já citados, o crescimento do custo de produção das máscaras de circuitos integrados para tecnologias CMOS nanométricas e do custo engenharia não-recorrente. Isto motiva os projetistas a se utilizar de projetos que possam ser montados rapidamente a partir do reuso

de componentes pré-projetados e pré-caracterizados, e que tenham alto volume de produção garantido, a partir de um único conjunto de máscaras. O projeto resulta simplificado com a utilização de uma arquitetura de referência para sistematizar reuso de IPs.

Os blocos de uma plataforma podem ser descritos em diversos níveis de abstração: nível elétrico, nível lógico e nível de transferência entre registradores (RTL). Esse último vinha sendo mais usado como ponto de partida para o projeto de circuitos integrados dedicados (ASICs). Em RTL, são modelados todos os componentes no nível de registradores, incluindo unidades aritméticas/lógicas (*arithmetic/logic units*, ALUs), registradores, memórias, multiplexadores e decodificadores. Geralmente, um componente é descrito em RTL como sendo formado por um *datapath* comandado por uma unidade de controle. Modelos descritos nesse nível de modelagem invariavelmente possuem precisão de ciclos. A síntese automática partindo de modelos RTL não é mais um desafio (MARWEDEL, 2006).

Ainda devido ao aumento da complexidade dos SoCs, aumentou a necessidade da indústria por um nível mais alto de descrição dos sistemas eletrônicos, levando ao surgimento da modelagem em nível de sistema eletrônico (ESL), que possibilita a descrição de módulos em diversos estilos incluindo a modelagem em nível de transações ou *Transaction-Level Modeling* (TLM). Em TLM, são abstraídos os detalhes de comunicação entre os componentes da plataforma. A comunicação é modelada simplesmente através de “canais”, enquanto as transações se realizam invocando funções de interface desses canais modelados. Detalhes de comunicação e computação desnecessários são omitidos em TLM e podem ser adicionados posteriormente. TLMs aumentam a velocidade de simulação e permitem a exploração e validação de alternativas de projeto em um nível mais alto de abstração (CAI; GAJSKI, 2003). Além disso, uma representação TLM permite que se antecipe o desenvolvimento de software dependente de hardware (código de inicialização, drivers, tratadores de E/S, etc.). Por viabilizar uma representação executável oficial da plataforma-alvo, o estilo TLM permite que o co-projeto de software e hardware não requeira prototipação de hardware para viabilizar o desenvolvimento de software dependente de hardware. Nesse contexto, requer-se a prototipação apenas para a validação funcional dos novos IPs projetados e não dos IPs reusados da plataforma.

1.2 Aplicações embarcadas contemporâneas

Atualmente, as áreas de aplicação mais desafiadoras para sistemas embarcados são eletrônica automotiva, multimídia e telecomunicações. Os veículos estão recebendo um número cada vez maior de unidades de controle eletrônicas (ECUs). Um veículo moderno contém entre uma

dúzia e uma centena de ECUs de diferentes domínios de aplicação, entre controle de tempo real severo e informação-entretenimento (SANGIOVANNI-VICENTELLI; NATALE, 2007).

No setor de entretenimento produz-se bilhões de unidades de vídeo digital a cada ano, incluindo tocadores de mídia portáteis, câmeras digitais, TVs digitais, entre outros. Muitos desses equipamentos têm requisitos de processamento semelhantes aos *desktops* de hoje, porém com métricas de potência e custo inferiores (TALLA; GOLSTON, 2007).

A primeira geração de telefones celulares utilizava transmissão analógica com microcontroladores simples para realizar tarefas básicas, porém, os celulares modernos confiam aos microprocessadores tarefas importantes como compressão de voz e o processamento de sinal de banda básica (*baseband*), além de recursos adicionais como agenda telefônica e multimídia (WOLF, 2007; RAMACHER, 2007).

1.3 Trabalho anterior

No escopo do trabalho de conclusão de curso de (VOLPATO; ECCO, 2007) iniciou-se o desenvolvimento do IP descrito nesta monografia. Para a seleção da funcionalidade do IP, foi realizado um “profiling” do software de compressão JPEG contido no benchmark MiBench usando a ferramenta gprof. O MiBench (GUTHAUS et al., 2001) é um benchmark que reúne um conjunto de aplicações embarcadas divididas em seis grupos, cujos alvos são áreas específicas do mercado de aplicações embarcadas: controle automotivo e industrial, consumidor, automação de escritório, rede, segurança e telecomunicações. Dentro do grupo *consumidor* está implementado o compressor JPEG analisado.

O gprof é o profiler que compõe o *GNU Binary Utilities*, uma coleção de ferramentas para a manipulação de código em diferentes formatos mantida pela GNU. “Profiling” possibilita ao analista saber em que partes o programa analisado gastou o seu tempo de execução e rastrear informações sobre invocações de métodos. Essa informação pode mostrar que partes do programa estão mais lentas do que o esperado e que são fortes candidatas à reescrita para torná-las mais eficientes ou, neste caso, candidatas a serem implementadas em hardware para acelerar a execução da aplicação em um sistema embarcado.

Como o profiler utiliza informações coletadas durante a execução do programa, caso algumas funcionalidades da aplicação não sejam estimuladas elas não serão captadas durante o “profiling”. A implementação do compressor JPEG contida no MiBench (release 6, de 7 de fevereiro de 1996, do “The Independent JPEG Group”) foi aplicada a um determinado conjunto de imagens de diferentes tamanhos, padrões, em cores e em tons de cinza juntamente com o

gprof para realizar o “profiling” da execução do software.

Dentre as funções que se destacaram, a *forward_DCT* foi escolhida por consumir aproximadamente 36% do tempo de processamento e ter um potencial de reuso maior do que as outras. O resultado do trabalho foi uma plataforma descrita no estilo TLM, contendo um processador PowerPC, um bloco de memória e um IP interconectados através de um barramento, e a caracterização do comportamento funcional da interface do IP, definido por um conjunto de estímulos aplicado às suas entradas e o respectivo conjunto de resultados às suas saídas. Tal caracterização será utilizada para o projeto e validação do IP em nível RT objetivando a prototipação primeiro em FPGA e depois em silício.

1.4 Escopo deste trabalho

Esta monografia relata o projeto de um IP, partindo de uma descrição TLM de um sistema integrado onde o IP é representado através de um modelo atemporal descrito em SystemC. A aplicação-alvo é a codificação de imagens JPEG e a função crítica escolhida para ser implementada no IP é a transformada discreta do cosseno em duas dimensões (DCT 2D), utilizada para transformar a representação da informação do domínio espacial para o domínio das frequências.

Em seguida, propõe-se uma micro-arquitetura para o IP, para a qual será obtida uma descrição RTL sintetizável. A micro-arquitetura será validada experimentalmente através da simulação de sua descrição RTL (Verilog), utilizando-se a sua descrição atemporal executável (SystemC) como modelo de referência, estimuladas com exatamente o mesmo conjunto de sinais.

Através da prototipação em FPGA, o IP será caracterizado em termos de seu desempenho (frequência de operação e *throughput*) e consumo de energia/potência, utilizando ferramentas comerciais de automação de projeto eletrônico ou EDA (*Electronic Design Automation*).

2 *Breve revisão bibliográfica*

O objetivo deste capítulo é apresentar um panorama sobre os principais conceitos, linguagens e técnicas envolvidos neste trabalho. Além disso, analisa-se o trabalho mais diretamente relacionado com o projeto descrito nesta monografia. Não é a intenção apresentar uma pesquisa em profundidade sobre algoritmos de DCT ou técnicas alternativas, uma vez que se trata de uma implementação a partir de um algoritmo pré-escolhido.

2.1 Linguagens de descrição de hardware

As linguagens utilizadas para a descrição de hardware são denominadas *hardware description languages* (HDL). Até a década de 1980, a maioria dos projetos utilizavam linguagens gráficas para o desenvolvimento de sistemas digitais. Entretanto, também é possível desenvolver sistemas digitais utilizando linguagens de descrição de hardware textuais. A vantagem das linguagens textuais em relação às linguagens gráficas é que com elas é possível representar facilmente computações complexas, incluindo variáveis, *loops*, parâmetros e recursividade. Em virtude disso, a medida em que os projetos de hardware foram se tornando mais complexos, as HDLs textuais foram substituindo as linguagens gráficas praticamente em sua totalidade (MARWEDEL, 2006).

A principal diferença entre as linguagens de desenvolvimento de software e as HDLs é a necessidade de descrever concorrência entre componentes de hardware distintos. Além disso, projetos de hardware também necessitam de modelagem com noção de tempo.

As HDLs se tornaram populares com o aparecimento da linguagem VHDL e sua concorrente Verilog (IEEE, 2001). Em VHDL, a modelagem do paralelismo entre dois módulos é feita utilizando o conceito de processos. Cada processo modela um componente do hardware potencialmente concorrente. Para projetos mais simples apenas um processo é suficiente, já os mais complexos podem exigir um maior número de processos para a sua modelagem. Os processos se comunicam através de sinais, que quase sempre correspondem a conexões físicas (fios).

Verilog era uma linguagem proprietária criada por Thomas e Moorby (1991) e comprada pela Cadence. Posteriormente, com o sucesso da linguagem VHDL, a Cadence decidiu disponibilizar a linguagem Verilog para padronização. Como no VHDL, os projetos em Verilog são descritos como um conjunto de processos concorrentes e interconectados. Existem, entretanto, áreas onde Verilog é menos flexível e privilegia funcionalidades mais confortáveis. Verilog tem, por exemplo, suporte nativo à lógica multivalorada (Alta impedância, “Valor indefinido” e os dois valores da lógica booleana 0 e 1) enquanto que, para implementar a mesma funcionalidade, VHDL depende da disponibilidade de bibliotecas. Entretanto, em VHDL é possível instanciar todos os *full-adders* de um somador através de uma interação não sendo necessário instanciá-los individualmente, como é feito em Verilog.

Com a necessidade de se integrar HDLs aos *testbenches* (normalmente escritos em linguagens específicas para o fim, como Vera, ou até mesmo em linguagens de propósito geral, como C ou C++), em 2002 surge um novo padrão de linguagem. As versões de Verilog a partir de 3.0, também conhecidas como SystemVerilog (SYSTEMVERILOG, 2009), incorporaram suporte a modelagem em nível de transações, chamadas para funções C/C++/SystemC (e vice-versa), suporte nativo a asserções, entre outros. SystemVerilog foi considerada a primeira linguagem de desenvolvimento e verificação de hardware (HDVL).

2.2 Protocolo de comunicação AMBA AXI

O protocolo AMBA (*Advanced Microcontroller Bus Architecture*) é uma especificação de barramentos “on-chip” que é amplamente utilizado para interconectar IPs em projetos de SoCs. Um projeto típico baseado em AMBA consiste em um barramento principal de alta performance e um barramento periférico cujo desempenho é um pouco mais moderado (PASRICHA; DUTT; BEN-ROMDHANE, 2004). Na especificação AMBA 3 (ARM, 2009) é definido um conjunto de 4 protocolos que cobrem os requisitos de tráfego de dados “on-chip” dos componentes de processamento intensivo de dados requerendo um grande fluxo de dados, comunicação de baixa largura de banda requerendo baixa contagem de portas e potência, testes “on-chip” e acesso para depuração. Os quatro protocolos definidos na especificação AMBA 3 são apresentados a seguir:

AXI (*Advanced eXtensible Interface*)

É um protocolo focado no projeto de sistemas de alta frequência e alta performance (ARM, 2003).

AHB (*Advanced High-performance Bus*)

Provê uma interconexão eficiente entre periféricos simples em um subsistema de frequência

única, onde a performance do protocolo anterior não é necessária.

APB (*Advanced Peripheral Bus*)

Suporta transações de baixa largura de banda necessárias para acessar registradores de configuração em periféricos e o fluxo de dados através de periféricos de baixa largura de banda.

ATB (*Advanced Trace Bus*)

Especificação que, para a realização de *traces*, adiciona uma interface independente dos dados tratados (*data-agnostic interface*).

2.3 Trabalho relacionado

O nome “JPEG” é abreviação de *Joint Photographic Experts Group* (JPEG, 2009), nome do comitê criador do padrão de compressão de imagens JPEG. O padrão JPEG define métodos para a compressão de imagens com perdas e sem perdas. Compressão de imagens com perdas significa que, durante a compressão da imagem, algumas informações são descartadas objetivando a abreviação da quantidade de dados a ser transferida/armazenada posteriormente. O método foi desenvolvido visando a aplicação em imagens fotográficas ou gráficos realistas (imagens sem fronteiras definidas entre as cores) e não fornece um resultado satisfatório quando aplicado a desenhos, figuras textuais e/ou icônicas. O modo de compressão sem perdas é pouco utilizado por atingir taxas de compressão menores em comparação a outros métodos já existentes, como o GIF (INCORPORATED, 1987) e PNG (ROELOFS, 2009).

A dissertação de mestrado de (AGOSTINI, 2002) apresentou três arquiteturas para compressão JPEG para aplicação em sistemas de monitoramento de trânsito: um compressor JPEG para imagens em tons de cinza, um compressor JPEG para imagens coloridas e uma conversão de espaço de cores de RGB para YCbCr. O padrão JPEG define quatro modos de operação: seqüencial, progressivo, hierárquico e sem perdas. As arquiteturas dos compressores JPEG descritas implementam a compressão no modo de operação *baseline*, que se enquadra no modo de operação seqüencial. Segundo Agostini (2002), o modo *baseline* é o modo que reúne o menor número de requisitos para que se considere uma compressão como compressão JPEG, sendo o mais simples e um dos mais utilizados na prática. A implementação do compressor JPEG, operando no modo *baseline*, foi dividida em cinco operações principais, descritas a seguir:

Conversão do espaço de cores

É a primeira operação a ser realizada no compressor JPEG para imagens coloridas (quando

essas imagens estiverem no espaço de cores RGB). Os componentes R, G e B possuem um elevado grau de correlação, o que dificulta o processamento independente de cada um. A compressão JPEG é muito mais eficiente para os espaços de cores do tipo luminância e crominância (YCbCr).

Downsampling

Assim como a anterior, a etapa de *downsampling* só é aplicada a imagens coloridas. Nesta etapa, explora-se o fato de que o olho humano é mais sensível à luminância da imagem do que à crominância. O que se faz é eliminar parte das informações de crominância visando aumentar a taxa de compressão da imagem.

DCT-2D

Primeira operação realizada no compressor JPEG para imagens em tons de cinza, a transformada discreta do cosseno em duas dimensões é utilizada para transformar a representação das informações que compõem a imagem do domínio espacial para o domínio da frequência.

Quantização

Juntamente com a etapa de *downsampling*, a etapa de quantização é a principal responsável pelas taxas de compressão atingidas pelo JPEG. Nesta etapa, componentes de mais alta frequência, que contribuem pouco com a informação contida na imagem, são atenuados ou até mesmo descartados.

Codificação de entropia

A codificação de entropia consiste na aplicação de técnicas de compressão sem perdas nas matrizes de coeficientes já quantizados, viabilizando uma representação da imagem com menor número de bits.

O elemento de principal interesse para esta monografia é a DCT 2-D. A arquitetura implementada baseia-se no princípio da separabilidade da transformada discreta do cosseno, que determina que uma aplicação da DCT 2-D é equivalente à aplicações da DCT unidimensional sobre as linhas e sobre as colunas da matriz de entrada. Dessa forma, a arquitetura da DCT 2-D é composta por dois módulos DCT 1-D e um módulo intermediário que faz a transposição da matriz para a aplicação da segunda DCT unidimensional. Maiores detalhes da arquitetura serão tratados no Capítulo 4 desta monografia.

3 *Metodologia*

3.1 Verificação funcional

A metodologia VeriSC (2009) é uma metodologia para a verificação funcional de componentes digitais síncronos, através da comparação do dispositivo sob verificação (DUV) com seu modelo de referência, permitindo a criação do ambiente de verificação (*testbench*) antes mesmo da implementação do DUV. Essa metodologia foi desenvolvida a partir de seu uso prático no âmbito de edição anterior do projeto Brazil-IP (2009) e foi objeto de tese de doutorado.

A metodologia VeriSC baseia-se na adoção de um modelo de referência atemporal (também chamado de *golden model*) que é uma representação executável da especificação do comportamento do DUV. Essa metodologia apoiou-se no uso das seguintes ferramentas: SystemC (SYSTEMC, 2008a), SystemC Verification Library (SYSTEMC, 2008b) e em uma ferramenta de geração semi-automática de testbenches denominada eTBc (“Easy TestBench Creator”) (ETBC, 2008). Posteriormente, em virtude da facilidade que proporciona para o desenvolvimento de *testbenches*, adotou-se a linguagem SystemVerilog (substituindo o SystemC e o restante das bibliotecas) e a metodologia passou se denominar “Brazil-IP Verification Metodology” (BVM).

Para a realização da verificação funcional necessita-se do DUV, de um modelo de referência que implemente de forma ideal as funcionalidades desejadas para o DUV e de um *testbench*. As características desejáveis em um *testbench* são:

- Auto-verificação;
- Modelagem em TLM;
- Utilização de estímulos aleatórios;
- Orientação à cobertura.

A Figura 3.1 ilustra o *testbench* utilizado pela metodologia BVM, cujos componentes são descritos a seguir:

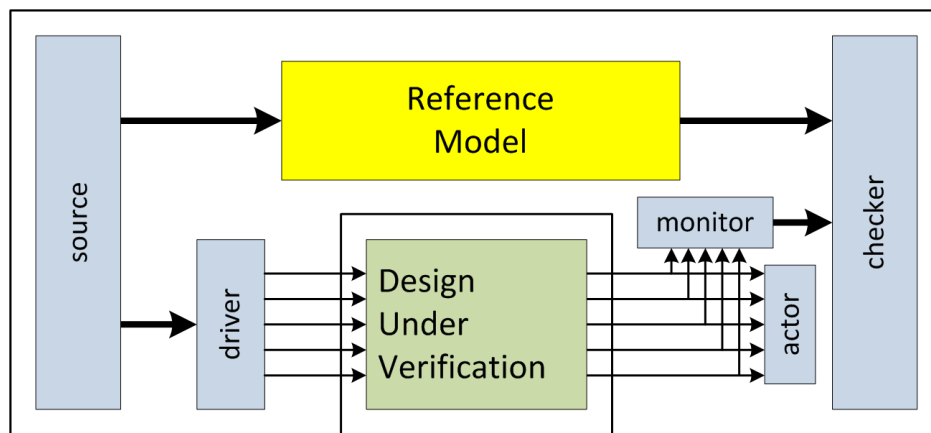


Figura 3.1: Estrutura final do *testbench*
 Fonte: Adaptado de Brazil-IP (2009)

Modelo de referência

O *modelo de referência* incorpora as funcionalidades desejadas para o DUV. Implementado em nível mais alto de abstração, pode ser escrito em qualquer linguagem de programação que se comunique com SystemVerilog (normalmente C ou C++). É desejável que o grupo de pessoas envolvidas no desenvolvimento do *modelo de referência* e o grupo de pessoas envolvidas com o DUV sejam distintos, para evitar que erros cometidos em ambas as unidades passem despercebidos pelo *testbench*. Aqui, é comum o uso de software desenvolvido por terceiros.

Source

Responsável pela geração de estímulos na forma de transações, tanto para o *modelo de referência* quanto para o *driver* (que posteriormente irá repassá-los ao DUV), visando a maior cobertura possível. A geração dos estímulos neste módulo ocorre de três diferentes maneiras: estímulos reais; estímulos aleatórios, testando o DUV frente a sequências de estímulos inusitadas; e valores limites (“corner cases”).

Checker

Recebe as transações de saída tanto do *modelo de referência* quanto do DUV. É um módulo reutilizável, pois sua implementação independe do DUV.

Driver

O *driver* recebe os estímulos gerados pelo *source* em nível de transação, transforma-os para o nível de sinais (“bit true”) e repassa-os para o DUV utilizando o protocolo de comunicação do DUV especificado nos requisitos.

Monitor

Este módulo faz a transformação oposta à feita pelo *driver*. Ele recebe as saídas do DUV em nível de sinais e repassa esses valores ao *checker* já na forma de transações.

Actor

Na metodologia VeriSC original, além de fazer a transformação das informações que saíam do DUV no nível de sinais para o nível de transações, o *monitor* também implementava o protocolo de comunicação com o DUV. Na metodologia BVM, a parte de comunicação foi extraída do *monitor* e atribuída a um módulo novo chamado *actor*.

DUV

O dispositivo sob verificação propriamente dito.

4 *Desenvolvimento em nível RT*

Neste capítulo serão apresentados os detalhes do projeto e implementação do IP em nível RT.

4.1 DCT 2-D

4.1.1 Arquitetura do IP

Após pesquisa sobre o funcionamento de uma DCT bi-dimensional e comparação de algumas arquiteturas propostas na literatura (BHASKARAN; KONSTANTINIDES, 1999), foi definida a implementação baseada na arquitetura proposta por Agostini (2002), apresentada na Figura 4.1. Nas seções subsequentes serão apresentadas as arquiteturas de cada um dos módulos que compõem o IP da DCT 2-D.

A computação básica de uma DCT em duas dimensões é a transformação de uma unidade de dados (uma matriz de 8x8 pixels) para o domínio da frequência. Para a aplicação da DCT 2-D em cada uma dessas unidades, a arquitetura apresentada por Agostini (2002) baseia-se numa propriedade dessa transformação, o princípio da separabilidade, que estabelece que o cálculo pode ser realizado através de duas aplicações da DCT unidimensional, a primeira sobre as linhas da matriz 8x8 e a segunda sobre as colunas da matriz resultante da primeira. A arquitetura proposta é composta por 3 sub-módulos síncronos (Figura 4.1), sendo dois deles responsáveis pela aplicação da DCT unidimensional e o módulo intermediário, denominado

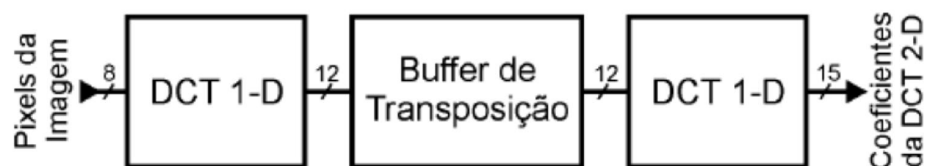


Figura 4.1: Arquitetura do IP DCT 2-D

Fonte: Agostini (2002)

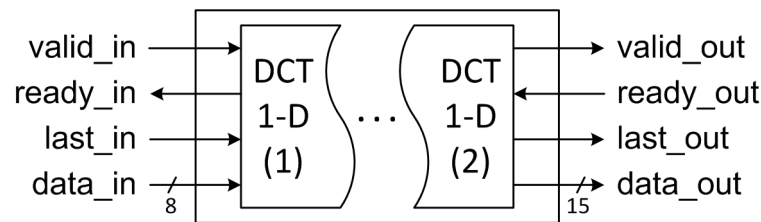


Figura 4.2: Roteamento dos sinais de interface do IP DCT 2-D

“*buffer* de transposição”, é quem permite que a segunda transformada opere sobre as colunas da matriz.

Com uma simples realimentação, seria possível fazer o cálculo da DCT 2D utilizando-se apenas um bloco de DCT unidimensional. Porém, a exemplo do que foi feito por Agostini (2002), optou-se pelo uso de dois módulos separados como medida para aumentar o desempenho, assim como pelo desenvolvimento de duas arquiteturas diferentes para cada um dos blocos, com o objetivo de minimizar o uso dos recursos de *hardware*. A descrição da DCT 2-D resultou em 1545 linhas de código Verilog.

4.1.2 Interface de comunicação

O módulo principal da DCT 2-D, que agrega os demais sub-módulos, possui como interface de entrada de dados os seguintes sinais: *valid_in*, *ready_in*, *last_in* e *data_in*, além dos sinais de sincronização *clock* e *reset*. Todos os sinais de entrada são roteados para a interface de entrada do primeiro sub-módulo de DCT unidimensional, como mostra a Figura 4.2. Além disso, os sinais *clock* e *reset* são roteados para os dois sub-módulos restantes. A interface de saída da DCT 2-D é constituída pelos seguintes sinais: *valid_out*, *ready_out*, *last_out* e *data_out*. Todos os sinais de saída são roteados para a interface de saída do segundo sub-módulo de DCT unidimensional (Figura 4.2).

Este módulo não possui uma unidade de controle. Como todos os sinais da interface são roteados para algum sub-módulo DCT unidimensional, a coordenação do processamento dos dados é feita através do protocolo de comunicação entre os sub-módulos e o ambiente externo. Os detalhes do protocolo de comunicação são apresentados na Seção 4.2.4.

4.2 DCT 1-D

Como já foi dito anteriormente, para a minimização do uso de recursos optou-se pelo desenvolvimento de duas arquiteturas para os módulos da DCT 1-D. As duas arquiteturas distinguem-

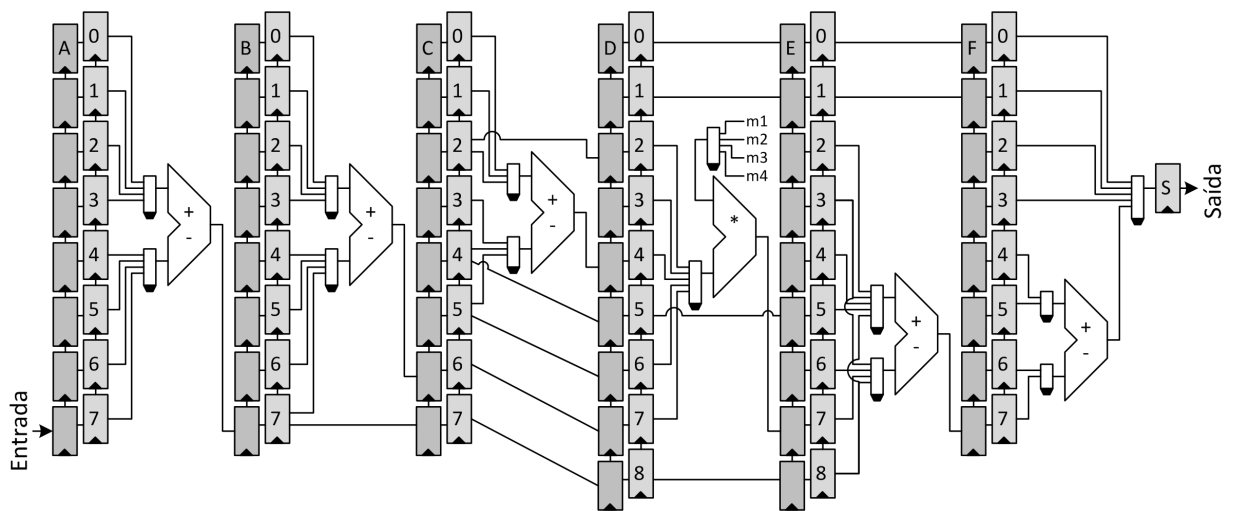


Figura 4.3: Arquitetura do módulo DCT 1-D

Fonte: Adaptado de Agostini (2002)

se basicamente pelo diferente número de bits de representação de sinais no *datapath*.

4.2.1 Arquitetura do módulo

Para o cálculo da DCT em uma dimensão foi implementado o algoritmo proposto por Kovac e Ranganathan (1995), corrigido por Agostini (2002), que pode ser encontrado no Anexo A. A arquitetura aqui utilizada é a arquitetura proposta por Agostini (2002), apresentada na figura 4.3, com algumas modificações. O algoritmo implementado possui seis passos independentes entre si, característica que foi aproveitada ao desenvolver-se um *pipeline* de, também, seis estágios, sendo que em cada um deles existe apenas uma unidade aritmética. Cinco desses estágios possuem um somador/subtrator e apenas um deles possui um multiplicador. Cada estágio do *pipeline* leva oito ciclos de *clock* para completar as suas operações, sendo que o número máximo de adições/subtrações no mesmo estágio é oito e o número máximo de multiplicações em um mesmo estágio é cinco.

Na Figura 4.3, elementos de oito bits entram a uma taxa de um elemento por ciclo de *clock*. Esses dados vão sendo armazenados em um *buffer* do tipo ping-pong de modo a serem disponibilizados à primeira unidade aritmética para que seja realizada a primeira etapa o cálculo da DCT. Todos os estágios do *pipeline* são precedidos por um *buffer* ping-pong, que armazena os resultados da etapa anterior do cálculo e os disponibiliza para o próximo estágio.

No sexto estágio do *pipeline*, alguns elementos são gerados em série e outros paralelamente. Para que seja feita a coordenação de qual elemento deve estar disponível na saída do módulo a cada ciclo de *clock*, a saída da unidade aritmética, assim como algumas das saídas do

buffer ping-pong, fica ligada a uma das entradas do multiplexador, que determina qual desses elementos será disponibilizado na saída do módulo. A função do registrador que fica conectado à saída do multiplexador é estabilizar o valor da saída do módulo no intervalo entre cada borda de *clock*.

A diferença entre as arquiteturas das DCTs unidimensionais está no número de bits necessários para representar os elementos manipulados, o que se reflete nas larguras de palavras dos elementos da arquitetura (unidades lógicas e aritméticas, registradores, multiplexadores etc.) e também no desempenho de cada um dos módulos. A Tabela 4.1 sumariza as diferenças do número de bits na entrada de cada estágio entre o primeiro e o segundo módulos da DCT 1-D.

Tabela 4.1: Diferença da largura de palavra entre os módulos DTC 1-D

Estágio	Operação	Nº de bits (entrada)	
		1ª DCT 1-D	2ª DCT 1-D
1	adição/sub.	8	12
2	adição/sub.	9	13
3	adição/sub.	10	14
4	multiplicação	11	15
5	adição/sub.	11	15
6	adição/sub.	12	15

O gerenciamento desta arquitetura é feito por um bloco de controle constituído de uma máquina de estados com 9 estados. Esse bloco de controle implementa os protocolos de comunicação na entrada e na saída do módulo, assim como a sincronização dos *buffers* ping-pong e a seleção de operações e operadores em cada estágio do *pipeline*. A latência da arquitetura da DCT unidimensional é de 47 ciclos de *clock*, já incluindo a escrita no registrador de saída no último estágio do *pipeline*.

4.2.2 Arquitetura do *buffer* ping-pong

O *buffer* ping-pong é basicamente uma unidade de armazenamento que recebe dados em série e os devolve em paralelo. Ele é composto por duas fileiras de registradores classificados por função, como pode ser visto na Figura 4.4. Na primeira fileira ficam os registradores do tipo “ping”. A cada ciclo de *clock*, cada registrador do tipo “ping” envia o dado armazenado para o próximo registrador da fila e armazena o dado que lhe foi enviado pelo registrador anterior, com exceção do primeiro e último registradores. Um armazena o dado disponível na entrada do *buffer* e o outro descarta o dado que armazenava, respectivamente. Cada vez que o sinal de controle “wren” (*write enabled*) é ativado, cada valor armazenado em um registrador “ping” passa para um registrador do tipo “pong”. Os registradores do tipo “pong” mantêm esses valores

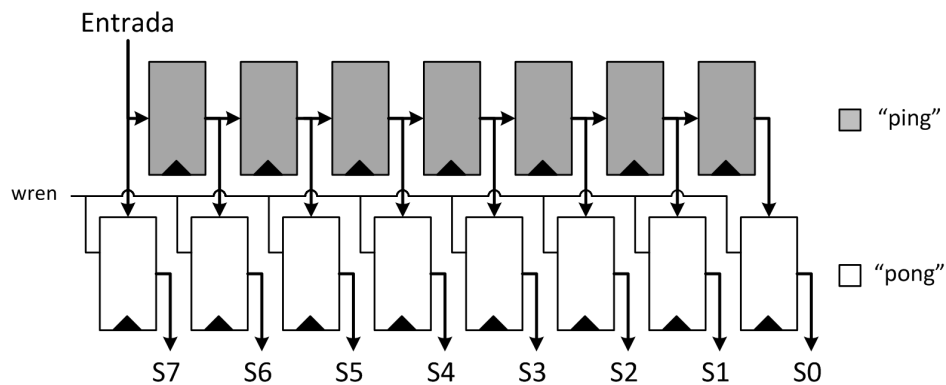


Figura 4.4: Arquitetura do *buffer* ping-pong
 Fonte: Adaptado de Agostini (2002)

fixos na saída até a próxima ativação do sinal de controle.

Como se pode perceber na Figura 4.4, a fila de registradores “pong” possui um elemento a mais do que a fila de registradores “ping”. Isso é uma otimização que foi feita para economizar um ciclo de *clock* a cada transferência de estágio do *pipeline* da DCT 1-D. Em vez de passar por um registrador “ping” e somente no próximo ciclo de *clock* ser transferido para um registrador “pong”, sempre que o sinal de controle “wren” é ativado, o valor na entrada é armazenado diretamente em um registrador “pong”.

4.2.3 Arquitetura do multiplicador

Os operadores de multiplicação da DCT unidimensional foram decompostos em somas de deslocamentos como otimização para desempenho. O processo de multiplicação implementado envolve quatro deslocamentos, seis concatenações, três adições e um truncamento. Como no algoritmo da DCT 1-D só são efetuadas multiplicações por quatro constantes, essas constantes foram integradas à arquitetura do multiplicador por meio dos *barrel-shifters* (deslocadores), sendo necessária apenas uma entrada de dados, além dos sinais de controle para esta unidade, como mostra a Figura 4.5.

Por ter uma complexidade maior, se comparada ao somador/subtrator (afinal, só no multiplicador são utilizados três somadores), e conseqüentemente, por ser um potencial caminho crítico na arquitetura da DCT 1-D, a arquitetura do multiplicador foi dividida em dois estágios de *pipeline*, como pode ser observado na Figura 4.6. Dessa forma, o período do *clock*, que deveria ter uma duração que comportasse a propagação de um sinal do início ao final da arquitetura do multiplicador, passando por dois somadores, agora tem que ser suficiente para que o sinal se propague do início a um dos registradores de *pipeline* ou de um dos registradores de *pipeline*

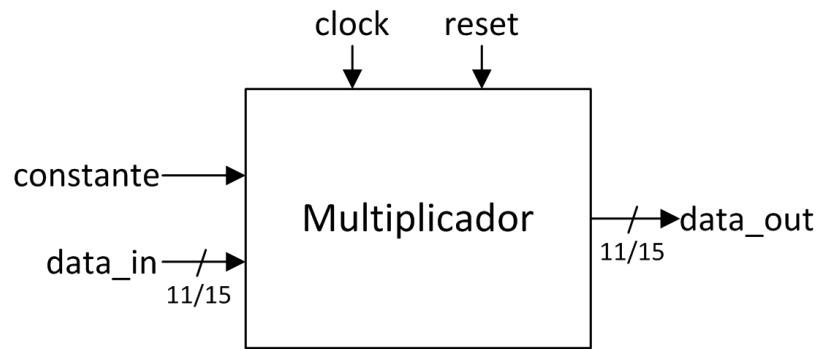


Figura 4.5: Interface do multiplicador

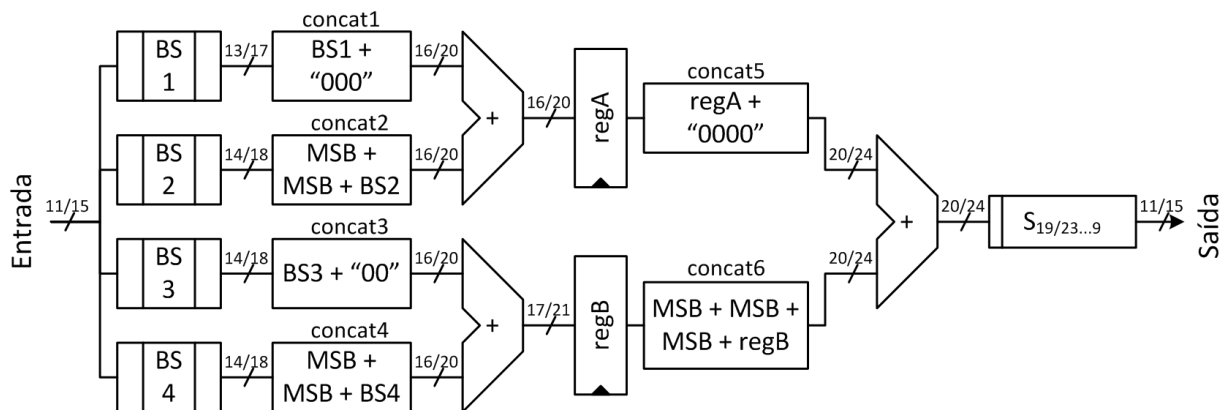


Figura 4.6: Arquitetura do multiplicador
Fonte: Adaptado de Agostini (2002)

até o final da arquitetura passando, em ambos os casos, por apenas um somador.

Na Figura 4.6, cada conexão tem dois indicadores de largura em termos de número de bits. Assim como na Figura 4.7, o primeiro e o segundo números representam a largura das palavras dos multiplicadores do primeiro e do segundo módulos da DCT 1-D, respectivamente.

Como já foi dito anteriormente, as multiplicações do algoritmo da DCT 1-D envolvem sempre uma entre quatro constantes. Isso permitiu que, em vez de n deslocadores para implementar essa multiplicação onde n é a largura de palavra da entrada do multiplicador, neste caso 11 ou 15 bits, fossem utilizados apenas quatro *barrel-shifters* cujos deslocamentos são pré-determinados de acordo com a constante utilizada. É importante ressaltar que não existe uma ligação direta entre as quatro constantes e os quatro *barrel-shifters*, essa minimização do número de *barrel-shifters* só foi possível em virtude do arredondamento das constantes da multiplicação, apresentados na Tabela 4.2.

Tabela 4.2: Constantes utilizadas pelo multiplicador

Constante	Valor decimal	Aproximação utilizada	
		Binário	Decimal
$m1 = \cos(4\pi/16)$	0,707107	0,101101000	0,703125
$m2 = \cos(6\pi/16)$	0,382683	0,011000101	0,384766
$m3 = \cos(2\pi/16) - \cos(6\pi/16)$	0,541196	0,100010101	0,541016
$m4 = \cos(2\pi/16) + \cos(6\pi/16)$	1,306563	1,010011000	1,296875

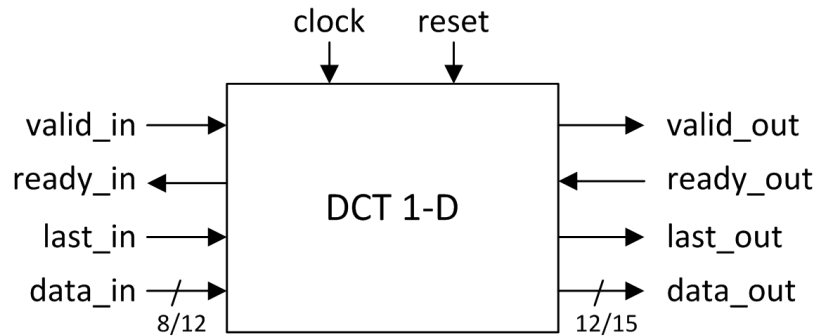


Figura 4.7: Interface do módulo DCT 1-D

4.2.4 Interface de comunicação

As interfaces destes módulos são exatamente idênticas à da DCT 2-D. Além dos sinais de sincronização *clock* e *reset*, os sinais de entrada são: *valid_in*, *ready_in*, *last_in* e *data_in*, e os sinais de saída: *valid_out*, *ready_out*, *last_out* e *data_out*. A Figura 4.7 apresenta a interface da DCT unidimensional. Nela, junto aos sinais onde o número de bits entre os dois módulos é diferente, apresentam-se dois números. O primeiro número refere-se ao número de bits do sinal no primeiro módulo e o segundo refere-se ao número de bits do sinal no segundo módulo.

O protocolo de comunicação deste módulo é baseado no protocolo AMBA AXI. Ele foi implementado da mesma forma tanto para entrada de dados quanto para saída, e a sua especificação está descrita a seguir.

Atuam no protocolo de comunicação os seguintes sinais:

valid É acionado pela fonte de dados para indicar que a informação desejada encontra-se disponível.

ready É acionado pelo destino para indicar que a informação foi aceita, ou que está preparado para receber uma nova informação. Essa diferenciação é identificada pelo contexto.

last Durante uma rajada de dados, a fonte deve acionar este bit para indicar que o dado sendo enviado atualmente é o último da série.

data É neste sinal onde todo o tráfego de informações ocorre.

As Figuras 4.8 até 4.10 apresentam diferentes cenários de comunicação. Na Figura 4.8, a fonte apresenta a informação e coloca o sinal *valid* em nível lógico alto. A informação permanece estável até que o destino acione o sinal *ready*, indicando que a informação foi aceita. A transferência ocorre quando ambos os sinais, *valid* e *ready*, encontram-se em nível lógico alto.

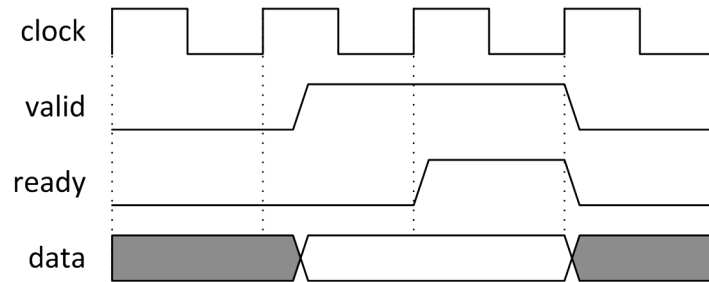


Figura 4.8: Cenário 1: *valid* antes de *ready*

No cenário apresentado na Figura 4.9, o destino coloca o sinal *ready* em nível lógico alto antes da informação estar disponível. Isso indica que o destinatário pode aceitar a informação assim que ela estiver disponível. Novamente, a transferência ocorre no momento em que os sinais *valid* e *ready* encontram-se em nível lógico alto.

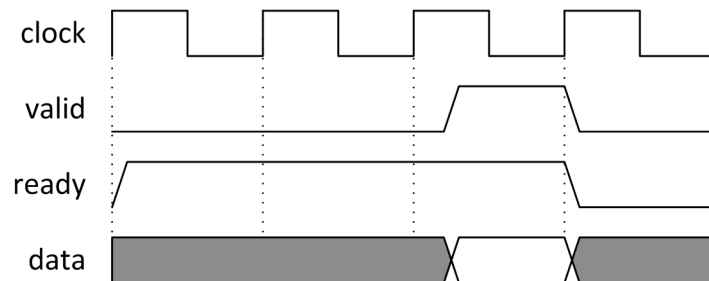


Figura 4.9: Cenário 2: *ready* antes de *valid*

Na Figura 4.10, fonte e destino indicam, no mesmo ciclo, que ambos estão prontos para a transferência da informação. Nesse caso, a transferência ocorre imediatamente.

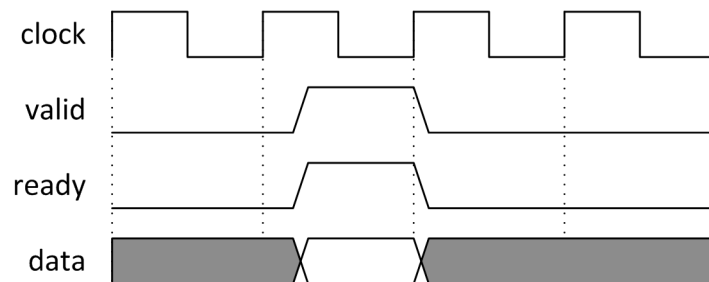


Figura 4.10: Cenário 3: *valid* junto com *ready*

A Figura 4.11 apresenta um exemplo de transferência de dados em rajada cujo cenário de transferência de dados é equivalente ao cenário apresentado na Figura 4.8. Porém, neste exemplo, ao invés de um único elemento sendo transferido por *handshake*, transfere-se uma rajada de dados e utiliza-se o sinal *last* para indicar o fim da sequência.

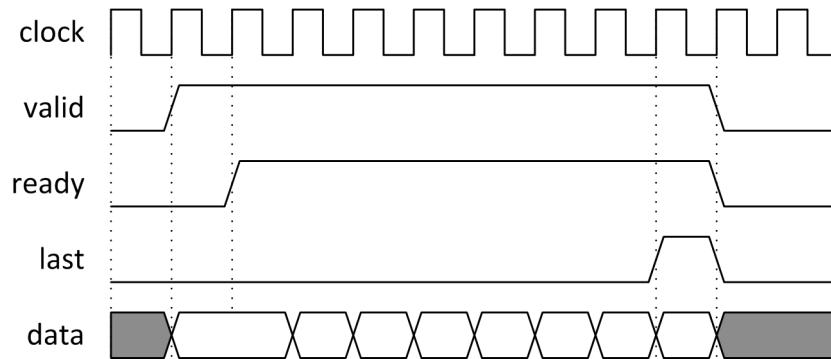


Figura 4.11: Cenário 3: transferência em rajada utilizando o sinal *last*

Planeja-se a inserção de um sinal na interface que indique quando o IP não estiver sendo utilizado pela arquitetura como medida de economia de energia. Esse comportamento é previsto pelo protocolo AMBA AXI.

4.3 *Buffer* de transposição

O papel do *buffer* de transposição é, como o próprio nome diz, transpor uma matriz quadrada de 64 elementos, ou seja, trocar as linhas da matriz por suas colunas. Aqui não se executa nenhum tipo de operação com os elementos manipulados, apenas armazenamento e retorno desses valores. Dessa forma, o número de bits na saída do módulo não se altera em relação à entrada. Assim como todos os outros módulos, a interface do *buffer* de transposição possui, além dos sinais de sincronização *clock* e *reset*, os sinais de entrada: *valid_in*, *ready_in*, *last_in* e *data_in*, e os sinais de saída: *valid_out*, *ready_out*, *last_out* e *data_out*, como pode ser visto na Figura 4.12.

4.3.1 Arquitetura do módulo

A arquitetura do *buffer* de transposição, apresentada na figura Figura 4.13, foi projetada utilizando-se duas memórias RAM que intercalam suas funções a cada 64 elementos recebidos (8x8 elementos). As duas funções desempenhadas pelas memórias são receber elementos da porta de entrada e armazená-los linha a linha e devolver os elementos armazenados coluna a

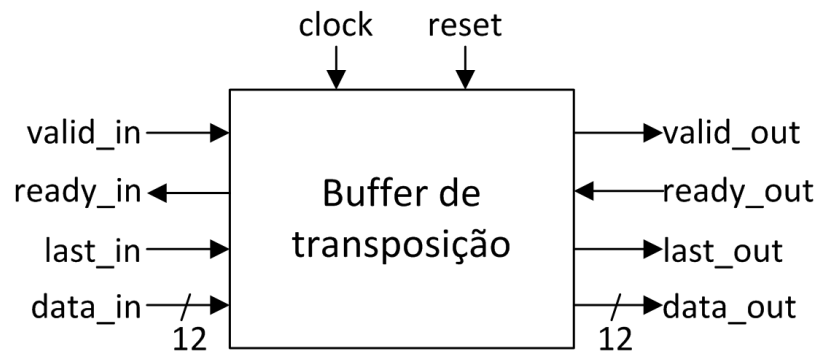


Figura 4.12: Interface do *buffer* de transposição

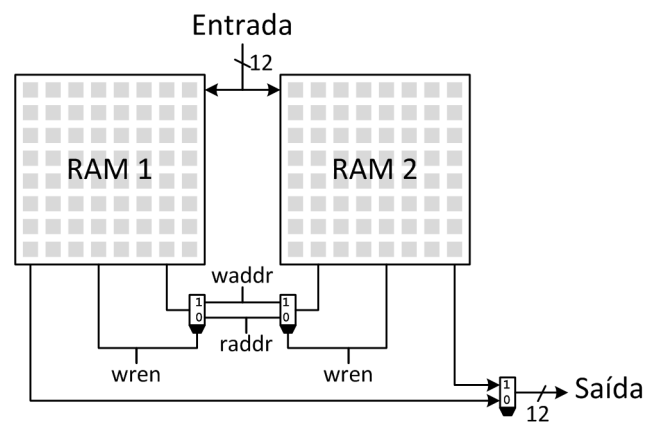


Figura 4.13: Arquitetura do *buffer* de transposição

coluna. A arquitetura desenvolvida para o *buffer* de transposição possui uma latência de 64 ciclos de *clock*.

5 *Resultados da síntese do IP*

5.1 **Configuração experimental**

Todos os resultados apresentados neste capítulo são referentes à síntese do IP, direcionada ao dispositivo FPGA Cyclone II EP2C35F672C6 (ALTERA CORPORATION, 2008) da Altera, utilizando a ferramenta Quartus II versão 9.0 (com licença acadêmica) também da Altera. A escolha do dispositivo Cyclone II foi feita por questões de disponibilidade.

As estimativas do consumo de energia foram feitas também utilizando a ferramenta Quartus II, nas mesmas configurações descritas anteriormente e para o mesmo dispositivo, com a inserção de vetores de estímulos gerados pela ferramenta ModelSim Starter Edition versão 6.4a da Altera. A imagem fonte para a geração dos vetores de estímulos encontra-se no Anexo B.

5.2 **Resultados para os principais componentes**

Esta seção apresenta os resultados da síntese em FPGA, que consiste em um resumo do número de componentes utilizados assim como o número de componentes disponíveis no dispositivo alvo, e as estimativas de consumo de energia para cada um dos sub-módulos do IP. Tanto para a síntese quanto para a estimativa do consumo de energia, os sub-módulos foram considerados elementos independentes e seus sinais de interface foram interpretados pelo sintetizador como pinos no processo da síntese, o que influenciou bastante no consumo de energia total estimado. Dessa forma, a dissipação térmica de entrada e saída pode ser desconsiderada nos sub-módulos para compreender melhor o seu real impacto no contexto do IP.

5.2.1 **DCT 1-D**

Os resultados das sínteses de ambos os módulos da DCT unidimensional são apresentados na Tabela 5.1. Nota-se a diferença do número de componentes utilizados por cada um dos módulos em virtude da diferença do tamanho dos seus *datapaths*. Foram utilizados alguns bits

de blocos de memória na síntese dos módulos DCT 1-D apesar de não haver nenhuma memória na definição das arquiteturas. Esses bits de memória são resultado da síntese dos *buffers* ping-pong que fazem a divisão do *pipeline*.

Tabela 5.1: Resumo da síntese da DCT 1-D para o Cyclone II EP2C35F672C6

Componente	DCT 1-D (1)		DCT 1-D (2)	
	Quantidade	% do total	Quantidade	% do total
Registradores	798	2%	1098	3%
ALUTs combinacionais	518	2%	690	2%
Bits de blocos de memória	93	< 1%	129	< 1%
Pinos	28	6%	36	8%
Pinos virtuais	0	—	0	—
PLLs	0	0%	0	0%

A tabela 5.2 apresenta os dados estimados para o consumo de energia dos módulos DCT 1-D. Nota-se uma pequena diferença em relação a estimativa de dissipação térmica estática do núcleo entre os dois módulos apesar da quantidade de *hardware* consideravelmente maior do segundo em relação ao primeiro. Na estimativa de dissipação térmica dinâmica, por sua vez, essa diferença ficou mais visível. Essa discrepância existe para a dissipação de energia estática contribui todo o *hardware* da FPGA, enquanto na dinâmica influenciam apenas as regiões utilizadas pelo módulo sintetizado.

Tabela 5.2: Estimativa do consumo de energia da DCT 1-D para Cyclone II EP2C35F672C6

	DCT 1-D (1)	DCT 1-D (2)
	Quantidade (mW)	Quantidade (mW)
Dissipação térmica total	158,93	176,21
Dissipação térmica dinâmica no núcleo	11,74	16,06
Dissipação térmica estática no núcleo	80,09	80,14
Dissipação térmica de entrada e saída	67,10	80,01
Confiança da estimativa	baixa	baixa

Ainda na Tabela 5.2, embora não deva ser considerada na análise global do consumo de energia do IP, é interessante fazer uma análise comparativa da dissipação térmica de entrada e saída entre os módulos DCT 1-D. Ao contrário dos elementos do núcleo do dispositivo, qualquer alteração no número de pinos aumenta significativamente o consumo total de energia. A diferença de 8 pinos entre os módulos acarretou em um aumento de 12,91mW na dissipação de entrada e saída. Isso é maior do que toda a energia térmica dissipada dinamicamente pelo menor dos dois módulos.

5.2.2 *Buffer* de transposição

Grande parte do *datapath* do *buffer* de transposição é composta pelas memórias, são 1536 bits para o armazenamento de 128 elementos de 12 bits cada, divididos entre duas matrizes com 64 elementos cada uma, como mostra a Tabela 5.3. Todo o restante dos elementos gerados na síntese do *buffer* de transposição, principalmente os registradores, podem ser considerados pertencentes ao bloco de controle uma vez que, além das memórias, o *datapath* possui somente três multiplexadores 2 para 1.

Tabela 5.3: Resumo da síntese do *buffer* de transposição para o Cyclone II EP2C35F672C6

Componente	Quantidade	% do total
Registradores	103	< 1%
ALUTs combinacionais	96	< 1%
Bits de blocos de memória	1536	< 1%
Pinos	32	7%
Pinos virtuais	0	—
PLLs	0	0%

Pela necessidade de ter que armazenar duas matrizes inteiras ao mesmo tempo, o *buffer* de transposição é um módulo maior do que as DCTs unidimensionais, que processam os elementos em *pipeline* e em momento algum uma matriz inteira encontra-se totalmente contida no seu *datapath*. Entretanto, a estimativa de dissipação térmica estática do núcleo e, principalmente, a dissipação dinâmica do núcleo são menores para o *buffer* de transposição, como mostra a Tabela 5.4. Isso se explica pelo fato de células lógicas terem um comportamento muito mais ativo, ou seja, as suas taxas de chaveamento são muito maiores se comparadas às das células de memória, que tendem a permanecer estáveis (armazenando um mesmo valor) durante um período de tempo relativamente longo.

Tabela 5.4: Estimativa do consumo de energia do *buffer* de transposição para Cyclone II EP2C35F672C6

	Quantidade (mW)
Dissipação térmica total	135,61
Dissipação térmica dinâmica no núcleo	5,68
Dissipação térmica estática no núcleo	80,01
Dissipação térmica de entrada e saída	49,93
Confiança da estimativa	baixa

5.3 Resultados globais

A Tabela 5.5 apresenta o resumo da síntese completa do IP. A síntese contabiliza as células de memória do *buffer* de transposição e as geradas para os buffers de transposição, os registradores que compõem os *datapaths*, os blocos de controle dos sub-módulos e o controle da interface de *low-power*. Também contribuem para a utilização do *hardware* as unidades lógicas e aritméticas e os pinos de entrada e saída de dados, sinais de *handshake* e interface *low-power*.

Tabela 5.5: Resumo da síntese da DCT 2-D para Cyclone II EP2C35F672C6

Componente	Quantidade	% do total
Registradores	2001	6%
ALUTs combinacionais	1309	4%
Bits de blocos de memória	1758	< 1%
Pinos	35	7%
Pinos virtuais	0	—
PLLs	0	0%

A estimativa de consumo de energia do IP é apresentada na Tabela 5.6. Nota-se aqui a pouca diferença da dissipação térmica estática do núcleo entre o IP e os sub-módulos analisados separadamente em virtude da contribuição de todos os elementos da FPGA para esses resultados, mesmo os que não compõem o dispositivo configurado, enquanto na estimativa de dissipação térmica dinâmica do núcleo percebe-se uma grande diferença em relação as unidades analisadas até agora.

Tabela 5.6: Estimativa do consumo de energia da DCT 2-D para Cyclone II EP2C35F672C6

	Quantidade (mW)
Dissipação térmica total	189,81
Dissipação térmica dinâmica no núcleo	29,75
Dissipação térmica estática no núcleo	80,19
Dissipação térmica de entrada e saída	79,87
Confiança da estimativa	baixa

6 *Verificação funcional*

Este capítulo apresenta a abordagem de verificação funcional aplicada para o IP DCT 2-D. Para a verificação, o IP foi dividido em dois níveis de abstração: o primeiro é o próprio módulo da DCT 2-D, que representa o mais alto nível de abstração, e o segundo nível é a DCT 2-D dividida nos dois blocos DCT 1-D e no *buffer* de transposição. Cada um desses módulos é verificado individualmente de acordo com a metodologia BVM, apresentada na Seção 3.1. O modelo de referência utilizado é a modelagem da DCT 2-D em nível de transações de Volpato e Ecco (2007).

O processo de verificação funcional se inicia com a criação do *testbench*. A etapa de criação do *testbench* segue um processo que permite que seus elementos sejam verificados quanto a sua corretude na medida em que vão sendo desenvolvidos. Cada passo do processo de criação do *testbench* é mostrado nas Figuras 6.1 a 6.3.

Utilizando-se apenas o modelo de referência, primeiramente cria-se um pré-*source* e um *sink*, que só aparecem no momento inicial de criação do *testbench* e são subconjuntos de *source* e *checker*, respectivamente. No próximo passo, com a replicação do modelo de referência já é possível a inserção dos elementos *source* e *checker*. No terceiro passo, são inseridos os elementos *driver*, *actor* e *monitor*. Como nesta etapa ainda se trabalha apenas com o modelo de referência, que se comunica através de transações, é necessária a criação de um *actor*, um *monitor* e um *driver* sobressalentes. Na geração de estímulos, o *driver* padrão faz a tradução das transações enviadas pelo *source* para o nível de sinais e o *actor* e *monitor* sobressalentes atuam fazendo novamente a transformação de sinais para transações. Esse mesmo processo se repete do outro lado do *testbench*, entre o modelo de referência e o *checker*.

O processo descrito anteriormente se repete para todos os sub-módulos a serem verificados. Obviamente, para isso utiliza-se o modelo de referência equivalente à tarefa executada em cada um desses blocos. Para a decomposição do modelo de referência existe um passo adicional. Após concluir o primeiro passo para todos os sub-módulos, confronta-se o modelo de referência decomposto com o modelo de referência original. A comparação é feita de maneira semelhante

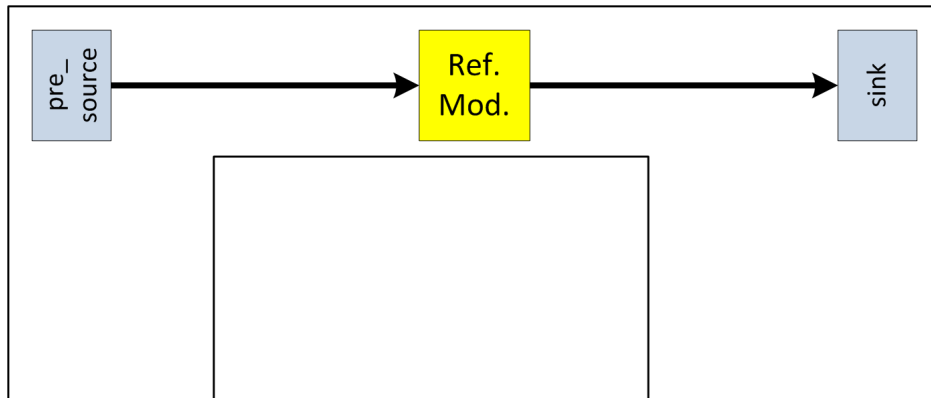


Figura 6.1: Passo 1: estrutura “modelo de referência”

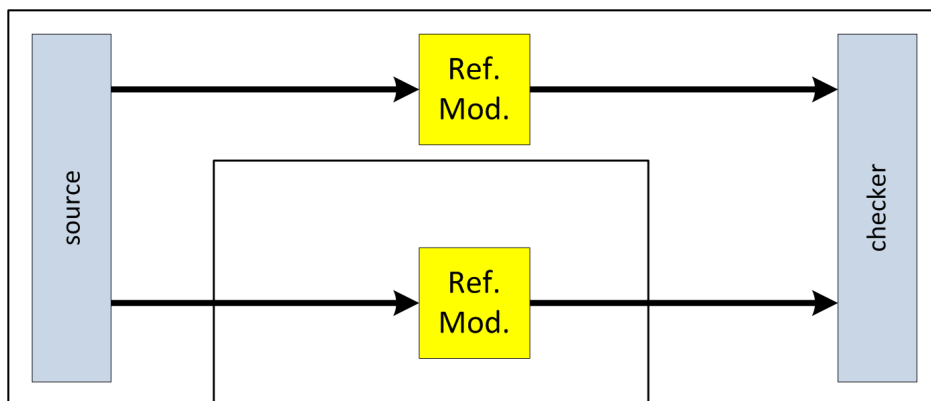


Figura 6.2: Passo 2: estrutura “modelo de referência duplo”

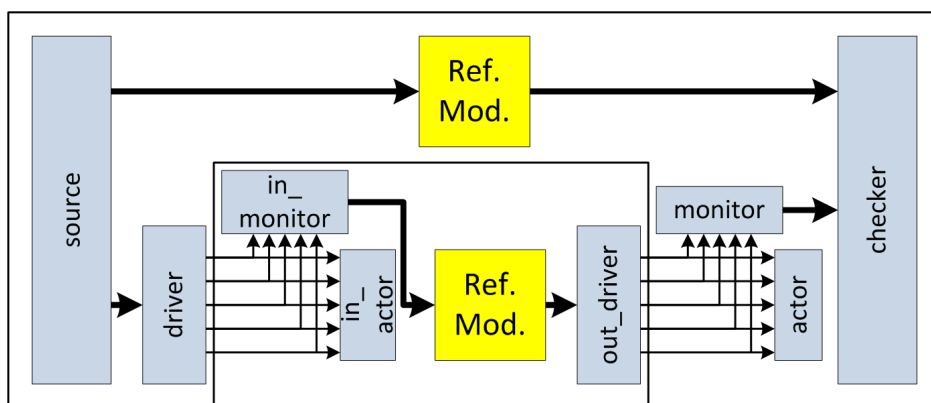


Figura 6.3: Passo 3: estrutura “emulação do DUV”

ao passo dois, porém no lugar da réplica do modelo de referência coloca-se a sua decomposição.

O processo de verificação está em andamento. Existe uma anomalia no algoritmo implementado na DCT 1-D, que ainda não foi detectada. Os resultados obtidos com o IP não conferem com a referência. As imagens geradas pelo compressor JPEG utilizando esse algoritmo apresentam muito ruído.

7 *Conclusões e perspectivas*

O IP desenvolvido e validado será disponibilizado ao domínio público. Dado o reduzido número de IPs disponíveis ao domínio público e o vasto campo de aplicação deste IP, este trabalho efetiva uma contribuição técnica para projetos baseados no reuso de IPs, contribuindo com uma descrição RTL sintetizável com vasta área de utilização.

7.1 Trabalhos em andamento

O algoritmo da DCT 1-D está sendo depurado a fim de detectar e corrigir a anomalia para que o IP passe a gerar os coeficientes corretamente.

Está em fase de implementação uma medida de economia de energia no IP, foram incluídos sinais que implementam um protocolo de economia de energia de modo a permitir o desligamento do *clock* do IP quando o mesmo não estiver em uso. Essa expansão já foi contabilizada nos resultados da síntese.

7.2 Trabalhos futuros

No âmbito do projeto Brazil-IP, este trabalho será continuado, percorrendo o fluxo de projeto VLSI (*Very Large Scale Integration*) chegando até a prototipação do IP DCT 2-D em silício.

ANEXO A – Algoritmo DCT 1-D

Este anexo apresenta o algoritmo desenvolvido por Kovac e Ranganathan (1995) para o cálculo da DCT em uma dimensão, incluindo as correções feitas por Agostini (2002). Este algoritmo é implementado pelo módulo DCT 1-D desenvolvido nesta monografia.

Para o algoritmo apresentado a seguir, tem-se que:

- $m1 = \cos(4\pi/16)$;
- $m2 = \cos(6\pi/16)$;
- $m3 = \cos(2\pi/16) - \cos(6\pi/16)$;
- $m4 = \cos(2\pi/16) + \cos(6\pi/16)$;

Passo 1

$$\begin{array}{lll}
 b_0 = a_0 + a_7 & b_1 = a_1 + a_6 & b_2 = a_3 - a_4 \\
 b_3 = a_1 - a_6 & b_4 = a_2 + a_5 & b_5 = a_3 + a_4 \\
 b_6 = a_2 - a_5 & b_7 = a_0 - a_7 &
 \end{array}$$

Passo 2

$$\begin{array}{lll}
 c_0 = b_0 + b_5 & c_1 = b_1 - b_4 & c_2 = b_2 + b_6 \\
 c_3 = b_1 + b_4 & c_4 = b_0 - b_5 & c_5 = b_3 + b_7 \\
 c_6 = b_3 + b_6 & c_7 = b_7 &
 \end{array}$$

Passo 3

$$\begin{array}{lll}
 d_0 = c_0 + c_3 & d_1 = c_0 - c_3 & d_2 = c_2 \\
 d_3 = c_1 + c_4 & d_4 = c_2 - c_5 & d_5 = c_4 \\
 d_6 = c_5 & d_7 = c_6 & d_8 = c_7
 \end{array}$$

Passo 4

$$\begin{array}{lll}
 e_0 = d_0 & e_1 = d_1 & e_2 = m_3 \times d_2 \\
 e_3 = m_1 \times d_7 & e_4 = m_4 \times d_6 & e_5 = d_5 \\
 e_6 = m_1 \times d_3 & e_7 = m_2 \times d_4 & e_8 = d_8
 \end{array}$$

Passo 5

$$\begin{array}{lll}
 f_0 = e_0 & f_1 = e_1 & f_2 = e_5 + e_6 \\
 f_3 = e_5 - e_6 & f_4 = e_3 + e_8 & f_5 = e_8 - e_3 \\
 f_6 = e_2 + e_7 & f_7 = e_4 + e_7 &
 \end{array}$$

Passo 6

$$\begin{array}{lll}
 S_0 = f_0 & S_1 = f_4 + f_7 & S_2 = f_2 \\
 S_3 = f_5 - f_6 & S_4 = f_1 & S_5 = f_5 + f_6 \\
 S_6 = f_3 & S_7 = f_4 - f_7 &
 \end{array}$$

ANEXO B – Imagem utilizada para estimativa de consumo de energia

As imagens B.1 e B.2 foram utilizadas como vetores de estímulos para a estimativa do consumo de energia do IP e de cada um dos seus sub-módulos.



Figura B.1: Imagem “mibench_small” (contida no benchmark MiBench)

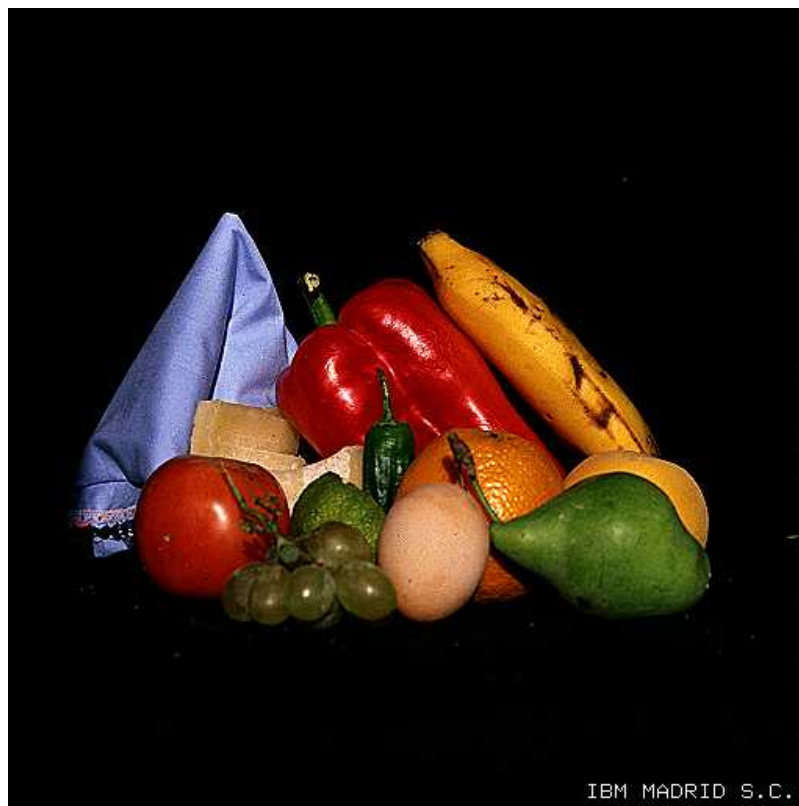


Figura B.2: Imagem “mibench_large” (contida no benchmark MiBench)

Referências Bibliográficas

- AGOSTINI, L. V. *Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG*. Dissertação (Mestrado) — UFRGS, mar. 2002.
- ALTERA CORPORATION. *Cyclone II Device Family Data Sheet*. [S.l.], fev. 2008.
- ARM. *AMBA AXI Protocol Specification*. [S.l.], jun. 2003.
- ARM. *AMBA 3 Specification*. 2009. Acessado em: 05 maio 2009. Disponível em: <<http://www.arm.com/products/solutions/AMBA3AXI.html>>.
- BAILEY, B.; MARTIN, G.; ANDERSON, T. *Taxonomies for the Development and Verification of Digital Systems*. [S.l.]: Springer, 2005. ISBN 978-0-387-24019-0.
- BELANOVIC, P. et al. Design methodology of signal processing algorithms in wireless systems. In: *International Conference on Computing, Communications and Control Technologies CCCT'03*. [S.l.: s.n.], 2003. p. 288–291.
- BHASKARAN, V.; KONSTANTINIDES, K. *Image and Video Compression Standards: Algorithms and architectures*. [S.l.]: Kluwer Academic Publishers, 1999.
- BRAZIL-IP. 2009. Acessado em: 14 maio 2009. Disponível em: <<http://www.brazilip.org.br/>>.
- CAI, L.; GAJSKI, D. Transaction level modeling: an overview. In: *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2003. p. 19–24. ISBN 1-58113-742-7.
- ETBC. 2008. Acessado em: 3 jul. 2008. Disponível em: <<http://lad.dsc.ufcg.edu.br/pmwiki.php?n=Lad.ETBc>>.
- GUTHAUS, M. et al. Mibench: A free, commercially representative embedded benchmark suite. In: . [S.l.: s.n.], 2001. p. 3–14.
- IEEE. Standard verilog hardware description language. *IEEE Std 1364-2001*, 2001.
- INCORPORATED, C. *GIF – Graphics Interchange Format (tm) – A standard defining a mechanism for the storage and transmission of raster-based graphics information*. jun. 1987. Acessado em: 25 mar. 2009. Disponível em: <<http://www.w3.org/Graphics/GIF/spec-gif87.txt>>.
- JPEG. *Joint Photographic Experts Group*. 2009. Acessado em: 22 jan. 2009. Disponível em: <<http://www.jpeg.org/>>.

- KOVAC, M.; RANGANATHAN, N. Jaguar: a fully pipelined vlsi architecture for jpeg image compression standard. *Proceedings of the IEEE*, v. 83, n. 2, p. 247–258, fev. 1995. ISSN 0018-9219.
- MARTIN, G.; CHANG, H. *Winning the SoC Revolution: Experiences in Real Design*. [S.l.]: Springer, 2003. ISBN 1402074956, 9781402074950.
- MARWEDEL, P. *Embedded System Design*. [S.l.]: Springer, 2006.
- PASRICHA, S.; DUTT, N.; BEN-ROMDHANE, M. Extending the transaction level modeling approach for fast communication architecture exploration. In: *DAC '04: Proceedings of the 41st annual conference on Design automation*. New York, NY, USA: ACM, 2004. p. 113–118. ISBN 1-58113-828-8.
- RAMACHER, U. Software-defined radio prospects for multistandard mobile phones. *IEEE Computer*, v. 40, n. 10, p. 62–69, out. 2007.
- ROELOFS, G. *PNG - Portable Network Graphics - An Open, Extensible Image Format with Lossless Compression*. 2009. Acessado em: 25 mar. 2009. Disponível em: <<http://www.libpng.org/pub/png/>>.
- SANGIOVANNI-VICENTELLI, A.; NATALE, M. D. Embedded system design for automotive applications. *IEEE Computer*, v. 40, n. 10, p. 42–51, out. 2007.
- SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. *Design & Test of Computers, IEEE*, v. 18, n. 6, p. 23–33, nov. 2001. ISSN 0740-7475.
- SEMATECH. 2009. Disponível em: <<http://www.sematech.org/>>.
- SYSTEMC. *Open SystemC Initiative*. 2008. Acessado em: 14 maio 2009. Disponível em: <<http://www.systemc.org/>>.
- SYSTEMC. *SystemC Verification Library*. 2008. Acessado em: 14 maio 2009. Disponível em: <<http://www.systemc.org/>>.
- SYSTEMVERILOG. *SystemVerilog*. 2009. Acessado em: 02 maio 2009. Disponível em: <<http://www.systemverilog.org/>>.
- TALLA, D.; GOLSTON, J. Using davinci technology for video devices. *IEEE Computer*, v. 40, n. 10, p. 53–61, out. 2007.
- THOMAS, D. E.; MOORBY, P. *The Verilog Hardware Description Language*. [S.l.]: Kluwer Academic Publishers, 1991.
- VERISC. 2009. Acessado em: 14 maio 2009. Disponível em: <<http://lad.dsc.ufcg.edu.br/pmwiki.php?n=Lad.VeriSC>>.
- VOLPATO, D. P.; ECCO, L. L. Projeto e validação de um ip para o padrão jpeg e sua integração a uma plataforma descrita no estilo tlm. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação) - Centro Tecnológico, Universidade Federal de Santa Catarina. out. 2007.
- WOLF, W. The embedded systems landscape. *IEEE Computer*, v. 40, n. 10, p. 29–31, out. 2007.