UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Bruno Carvalho de Farias

CORREÇÃO DE REFERÊNCIA DE RELÓGIO PARA FLUXO DE TRANSPORTE MPEG-2 EM FPGA

Florianópolis

2014

Bruno Carvalho de Farias

CORREÇÃO DE REFERÊNCIA DE RELÓGIO PARA FLUXO DE TRANSPORTE MPEG-2 EM FPGA

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Engenharia Elétrica", e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica.

Florianópolis, 22 de Agosto 2014.

Prof. Dr. Carlos Galup Montoro
Coordenador do Curso

Banca Examinadora:

Prof. Dr. Eduardo Augusto Bezerra
Presidente

Prof. Dr. Héctor Pettenghi Roldán

Prof. Dr. Raimes Moraes

Prof. Dr. Eddie Batista de Lima Filho

Dedico este trabalho aos meus pais e esposa, meus incentivadores.

AGRADECIMENTOS

À Deus por me dar sabedoria, saúde e forças nos momentos difíceis.

À minha amada esposa Fernanda por suportar a saudade e todas as demais dificuldades vividas no meses em que ficamos distantes.

À minha linda filha Rebeca por alegrar meus dias a cada sorriso, e me motivar a buscar crescimento em prol de sua felicidade.

Aos meus pais Tibério e Kátia, meu irmão Renan, e minha avó Suzana que tanto investiram na minha formação, e por estarem sempre na torcida pelo meu sucesso, devo o que sou a vocês.

Aos colegas da Fundação CERTI, em especial ao Bruno Herrera e Andrei Biscaro pelo suporte, flexibilidade, profissionalismo, e demais contribuições importantíssimos para o meu crescimento técnico e profissional no tempo em que permaneci em Florianópolis.

Aos colegas do laboratório de TV Digital do CTPIM.

Ao orientador Eduardo Bezerra pela orientação, disponibilidade e paciência.

Ao co-orientador e amigo Eddie Filho pela motivação, conselhos, e dedicação fundamentais para a realização deste trabalho e para minha formação acadêmica.

Aos colegas do GSE, em particular Frederico Ferlini, Paulo Villa, e Felipe Silva pelo suporte técnico durante as implementações.

À todos aqueles que de alguma maneira contribuíram para que eu obtivesse este título.

"Você não sabe o quanto eu caminhei pra chegar até aqui. Percorri milhas e milhas antes de dormir, eu nem cochilei..."

Toni Garrido

RESUMO

O Fluxo de Transporte (*Transport Stream* - TS) MPEG-2 é um formato amplamente utilizado em sistemas de TV Digital para a transmissão de áudio, vídeo e informações relacionadas a programa. Entre outras informações, um fluxo de transporte carrega uma referência de tempo, conhecida como Referência de Relógio de Programa (Program Clock Reference - PCR), a qual é um retrato do relógio de 27 MHz do sistema. Esta informação permite a recuperação do relógio nos receptores, o qual garante a correta apresentação do conteúdo e até mesmo controla interfaces de saída. Porém, se o tempo de chegada dos pacotes de transporte variar durante a transmissão ou o processamento, tal cenário pode levar a erros no relógio do sistema, o que é conhecido como *jitter*.

Os métodos tradicionais para a correção da informação do relógio de programa normalmente são baseadas em contadores/acumuladores de 27 MHz com ponto flutuante, porém, não mitigam o *jitter* de PCR completamente. Métodos mais recentes usam contador/acumulador controlado por semáforo, e até mesmo propõem um esquema de adaptação de taxa integrada à correção da referência de relógio, a qual resulta em baixos níveis de *jitter* na saída. Além disto, não há estudos a respeito da implementação dos métodos controlados por semáforo em processadores de fluxo de transporte.

Com isto, o presente trabalho propõe uma metodologia para implementação em FPGA, utilizando linguagem de descrição de hardware. Os resultados obtidos validam o uso dos métodos controlados por semáforo e mostram que a estrutura proposta é efetiva, com o benefício de eliminar efeitos de metaestabilidade.

Palavras-chave: MPEG-2 TS, Correção de PCR, FPGA.

ABSTRACT

The MPEG-2 Transport Stream is a standard format commonly used in digital TV systems for conveying audio, video and program-related information. Among other data, a transport stream also encapsulates a timing reference, known as program clock reference, which is a snapshot of the 27 MHz system clock. This information allows clock recovery in receivers, which provides correct content presentation and even drives output interfaces. However, if the arrival time of transport packets change during transmission or processing, such a scenario may lead to system clock errors, which is known as jitter.

Traditional methods for correcting the program clock reference information are normally based on 27 MHz counters/floating-point accumulators and do not completely mitigate jitter. More recent methods use a semaphore-controlled counter/accumulator and even propose a joint adaptation and clock reference correction, which result in low jitter levels at the output. Nevertheless, there is no study concerning the implementation of semaphore-controlled methods in transport stream processors.

Given that, the present work proposes an implementation methodology for FPGA, using hardware description language. The related results validate the use of semaphore-controlled methods and show that the proposed structure is effective, with the benefit of avoiding metastability effects.

Keywords: MPEG-2 TS, PCR Correction, FPGA.

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS E SIGLAS

AAC *Advanced Audio Coding* - Compressão de Áudio Avançada

ASI *Asynchronous Serial Interface* - Interface Serial Assíncrona

ATSC *Advanced Television System Comitee* – Comitê do Sistema Avançado de Televisão

CAT *Conditional Access Table* - Tabela de Acesso Condicional

CLB *Configurable Logic Blocks* - Blocos Lógicos Configuráveis

DVB *Digital Video Broadcasting* - Radiodifusão de Vídeo Digital

DTMB *Digital Terrestrial Multimedia Broadcasting* - Radiodifusão de Multimídia Digital Terrestre

DTS *Decoding Time Stamp* - Referência de Tempo de Decodificação

EPG *Eletronic Programming Guide* - Guia de Programação Eletrônico

ES *Elementary Stream* - Fluxo Elementar

FPGA *Field-Programmable Gate Array* - Arranjo de Portas Programável em Campo

HD *High Definition* - Alta Resolução

ISDB-T *Integrated System Digital Broadcasting – Terrestrial*

ISDB-Tb *Integrated Services Digital Broadcasting – Terrestrial Brazil* - Serviços Integrados de Radiodifusão Digital – Terrestre Brasil

LB *Logical Blocks* - Blocos Lógicos

MPEG *Moving Pictures Experts Group* - Grupo de Especialistas em Imagens com Movimento

MPTS *Multiple Program Transport Stream* - Fluxo de Transporte com Múltiplos Programas

NIT *Network Information Table* - Tabela de Informação de Rede

OFDM *Orthogonal Frequency Division Modulation* - Múltiplas Portadoras Ortogonais entre Si

PAT *Program Association Table* - Tabela de Associação de Programa

PCR *Program Clock Reference* - Referência de Relógio de Programa

PES *Packetized Elementary Stream* - Fluxo Elementar Empacotado

PID *Program Identifier* - Identificador de Programa

PLL *Phase Locked Loop* - Malha de Detecção de Fase

PMT *Program Map Table* - Tabela de Mapa de Programa

PS Program Stream - Fluxo de Programa

PSI *Program Specific Information* - Informação Específica de Programa

PTS *Presentation Time Stamp* - Referência de Tempo de Apresentação

SBTVD Sistema Brasileiro de Televisão Digital

SD *Standard Definition* - Resolução Padrão

SI *Service Information* - Informação de Serviço

SPI *Synchronous Parallel Interface* - Interface Paralela Síncrona

STC *System Time Clock* - Relógio de Sistema

TP *Transport Packet* - Pacote de Transporte

TS *Transport Stream* - Fluxo de Transporte

VHDL *VHSIC Hardware Description Language* - Linguagem de Descrição de Hardware VHSIC

SUMÁRIO

1 Introdução

Os avanços nos processos de digitalização, compressão e codificação de sinais permitiram a migração dos sistemas de TV analógicos para digitais, tendo proporcionado melhorias significativas na qualidade de áudio e vídeo. Esse conjunto de mudanças acabou dando origem a um novo modelo de transmissão e recepção denominado TV Digital. O sub-sistema de transmissão foi o mais afetado por esse avanço, uma vez que os sub-sistemas de captura e codificação já eram digitais [24]. Na transmissão analógica, os sinais de áudio e vídeo eram codificados e transmitidos em subportadoras diferentes. Com o advento da TV Digital, os sinais de áudio e vídeo passaram a ser combinados em único feixe de bits, após processo de multiplexação [13]. Neste contexto, a multiplexação encapsula os dados de áudio e vídeo, em pacotes devidamente identificados, conforme a especificação MPEG2.

O fluxo de transporte MPEG-2 (*Transport Stream* - TS) é amplamente utilizado em sistemas de radiodifusão para a transmissão de áudio, vídeo e dados, pois garante uma maior eficiência na recuperação de sinais sujeitos a erros e perdas [13]. O TS possui uma informação de tempo, denominada referência de relógio de programa (*Program Clock Reference* - PCR) , que consiste em uma amostra do relógio de 27 MHz do transmissor, sendo periodicamente transmitida. O PCR é uma informação adicionada ao cabeçalho dos pacotes de transporte, que visa garantir a sincronização entre áudio, vídeo e dados e o perfeito funcionamento do receptor, através da recuperação do relógio do transmissor [15]. O PCR é também utilizado para a geração da frequência da subportadora de cor, da taxa de amostragem de vídeo e das referências de tempo necessárias aos processos de recepção, decodificação e apresentação dos sinais multiplexados. Sendo assim, atrasos constantes são importantes para garantir o correto funcionamento do sistema de recepção [13].

Se o tempo que os pacotes de transporte levam para chegar ao receptor for variável, tal comportamento pode resultar em erros na recuperação do PCR, que são observados na forma de *jitter* [31], podendo também ser causado pelos processos de remultiplexação e adaptação de taxa de TS. Como resultado, o receptor não conseguirá reproduzir o relógio de 27 MHz do transmissor e a cadeia de decodificação ficará comprometida, ocasionando descontinuidades de vídeo, congelamento de tela e até mesmo imagens com cores distorcidas ou, em casos extremos, sem cor.

O PCR permite que a decodificação ocorra na mesma cadência que a codificação feita no transmissor. Se a decodificação no receptor for muito rápida, o *buffer* de memória do receptor pode ficar vazio; por outro lado,

se a decodificação for muito lenta, pode ocorrer transbordamento de *buffer*. Segundo o padrão MPEG-2, o *jitter* máximo presente no TS não pode exceder 500 ns, de modo que o correto funcionamento do receptor seja garantido.

Sendo assim, é extremamente importante que o valor do campo de PCR chegue de maneira correta ao receptor. Para que isso ocorra, algumas soluções foram propostas, para corrigir a informação de PCR na presença de *jitter*, através da modificação do valor deste antes de sua transmissão/retransmissão. Tal procedimento consiste na aplicação de um fator de correção, que visa refletir o atraso variável sofrido pelos pacotes de TS.

O *Field Programmable Gate Array* (FPGA), que em português significa Arranjo de Portas Programável em Campo, é um circuito integrado formado por blocos lógicos reconfiguráveis, que permite alterar seu funcionamento, através da modificação do código que o descreve [22]. Esta característica é interessante, pois permite que um algoritmo de adaptação de taxa seja implementado em hardware, para diferentes configurações, sem a necessidade de um novo componente ou sem grandes modificações do projeto em uso. Uma vez que a adaptação de taxa leva à necessidade de um módulo de correção de PCR, este também pode ser implementado no mesmo FPGA, levando-se em conta que as diferentes técnicas de correção podem se apresentar mais ou menos vantajosas, dependendo das características do TS que esta sendo processado. A flexibilidade dos FPGAs também permite que as técnicas de correção de PCR sejam substituídas e avaliadas de maneira mais simples, quando comparado a um sistema prototipado em placa, onde um novo circuito teria que ser confeccionado para cada sistema.

Alguns dos métodos tradicionais de correção de PCR já foram implementados no domínio de arquiteturas reconfiguráveis [20, 19, 23, 21]. Porém, se os problemas decorridos pelo assincronismo entre os *clocks* forem considerados, tais soluções acabam se configurando como sub-ótimas. Se tal assincronismo não for tratado corretamente, os problemas decorrentes podem levar à um fenômeno denominado metaestabilidade, o que faz com que o sistema tenha um comportamento não-determinístico. Outra consequência é a inserção de *jitter* residual no TS de saída, ocasionado por escorregamento de bordas.

Novos métodos foram propostos por Savino e Filho [25, 26], os quais são baseados em uma estrutura de semáforos, cuja principal característica consiste em unir as principais vantagens apontadas por métodos tradicionais diferentes [25], ou seja, utilizar poucos recursos computacionais e realizar poucas operações matemáticas, podendo até mesmo estimar o melhor momento para o envio de pacotes, de modo a se reduzir o *jitter* de PCR do TS. Neste métodos, semáforos são utilizados para gerenciar o acesso ao contador, o qual funciona como um recurso compartilhado entre a entrada e a saída do

sistema.

1.1 Objetivo

O objetivo deste trabalho é apresentar uma metodologia para a implementação dos métodos baseados em semáforos [26, 25], em VHDL, e compará-los aos métodos clássicos de correção de PCR, no que diz respeito à quantidade de unidades lógicas ocupadas, *jitter* de PCR e estruturação em hardware.

1.2 Justificativa

A presente dissertação investiga métodos de correção de PCR, visando identificar aqueles com características que resultem em alto desempenho, ao se considerar uma implementação em hardware. Os métodos baseado em semáforos apresentam características e vantagens que o tornam interessante em uma implementação em FPGA, principalmente pelo fato de utilizar somente um contador e reduzir a quantidade de operações matemáticas. O Método de Correção Integrada à Adaptação de Taxa [26] vai um pouco além, pois consegue empregar a estrutura de semáforos e ainda agrega funcionalidades com o objetivo de estimar o *jitter*. O fato de alguns métodos já terem sido implementados em FPGA justifica a proposta de uma metodologia para a implementação dos métodos de Adaptação de Taxa Integrada à Correção de PCR e Contadores Controlados por Semáforos, uma vez que isso viabilizaria uma comparação entre todos.

1.3 Organização do Trabalho

O trabalho está organizado como segue. O capítulo 2 introduz a fundamentação teórica necessária para a compreensão do trabalho. No capítulo 3, os trabalhos correlacionados são descritos. No capitulo 4, a arquitetura e implementação utilizada neste trabalho são apresentadas. O capítulo 5, por sua vez, mostra os resultados obtidos. Finalmente, o capítulo 6 expõe as considerações finais.

2 Fundamentação Teórica

Neste capítulo, alguns conceitos teóricos relacionados à TV Digital e FPGA serão apresentados, de forma mais detalhada. O formato do TS é estabelecido pelo padrão MPEG-2, que descreve a composição de cada um dos campos do TS e como este tipo de sinal é utilizado para transmitir pacotes de áudio e vídeo, de maneira robusta, entre transmissora e receptores. A compreensão dessas informações é importante para o esclarecimento dos algoritmos de correção aplicados neste trabalho. Os tópicos a seguir descreverão como os programas são multiplexados em um TS, quais as principais estruturas de dados responsáveis por descrever suas características e como essas informações devem ser processadas para se extrair a informação de PCR. Após isso, abordar-se-á como a informação de PCR é utilizada para gerenciar o sincronismo e a temporização, em sistemas de TV Digital. Em seguida, uma visão geral sobre tecnologias reconfiguráveis, FPGA e Linguagens de Descrição de Hardware (HDL) será apresentada. Por último, tópicos sobre a linguagem VHDL, o FPGA Altera DE-2 115 e as ferramentas de síntese e simulação, utilizadas durante o desenvolvimento do projeto, serão introduzidos.

2.1 TV Digital

A migração do sistema de TV Analógico para o Digital ocorreu através do avanço e consequente uso de técnicas de digitalização, compressão de sinais e códigos corretores de erros. A aplicação destas técnicas viabilizou a transmissão de sinais em alta definição em um canal de 6 MHz, que é a mesma largura de banda utilizada em sistemas de TV Analógica. O melhor uso desta banda se caracteriza principalmente pelo ganho significativo na qualidade de vídeo e áudio, na transmissão de arquivos digitais, no aumento do número de opções de programação, na inclusão de aplicações interativas e Vídeo Sob Demanda (*Video on Demand* - VoD) e também devido à incorporação de funcionalidades de segurança, que evitam que usuários recebam sinais e serviços indevidamente [29].

A TV Digital terrestre é empregada através de diferentes padrões, ao redor do mundo. Atualmente, existem quatro padrões: o americano (ATSC), o europeu (DVB), o japonês (ISDB-T) e o chines (DTMB). O padrão brasileiro de TV Digital ISDB-Tb foi criado recentemente, após um período de estudos, que auxiliaram na escolha do padrão que melhor se adequasse às exigências técnicas dos órgãos reguladores brasileiros. Após esta análise, optou-se pelo padrão japonês ISDB-T, dotado de modificações nas camadas de *middleware*

e codificação de sinal fonte (compressão), que deram origem ao padrão nipo-brasileiro. O ISDB-Tb optou pela codificação de vídeo em MPEG-4 H.264 e propôs a criação de um novo *middleware* para execução de aplicações, denominado Ginga, que é resultado da parceria entre as universidades PUC-Rio e UFPB. O *Middleware* Ginga permite a execução de aplicações interativas transmitidas juntamente com áudio e vídeo, com o intuito de complementar a programação de TV e permitir a interação entre telespectador e transmissora.

Todos estes sistemas são divididos em um mesmo conjunto de camadas: compressão, multiplexação e transmissão, com agregação de *middleware* e canal de retorno [1]. Logo, um sinal de TV é o resultado de uma longa cadeia de processamento, que vai desde a captação de áudio e vídeo, através de microfones e câmeras, até a sua transmissão via RF, no canal de 6 MHz. É interessante ter essa noção do "todo", para que se compreenda em que ponto do sistema o presente trabalho está inserido, e qual sua influência no funcionamento geral do sistema de TV. De maneira geral, um sistema de TV pode ser dividido em blocos, conforme pode ser visto na Figura 1.

No padrão ISDB-Tb, os sinais de áudio e vídeo digitalizados são submetidos a seus respectivos codificadores, que geram, como saída, vídeo comprimido no formato MPEG4 H.264 e áudio no formato MPEG4 HE-AAC. Estas técnicas de compressão fazem um melhor uso do canal, através da exploração das propriedades da percepção áudio visual humana, para transmitir uma menor quantidade de dados, porém, preservando a qualidade necessária para sua visualização.

No H.264, o sinal de vídeo é constituído por um sinal de luminância e dois sinais de crominância [17]. Ele possui um desempenho excelente, chegando a ter uma taxa de bits 60% menor que seu antecessor MPEG-2, além de apresentar vantagens no que diz respeito às etapas de predição, transformação e codificação. O sinal de vídeo comprimido pode ser transmitido em alta definição (*High Definition TV* - HDTV), em definição padrão (*Standard Definition TV* - SDTV) ou baixa definição (*Low Definition TV* - LDTV), codificado a 30 *frames* por segundo.

O áudio, por sua vez, é transmitido no padrão 5.1 (multicanal), dando ao usuário uma maior sensação de imersão em cena [29]. A codificação de áudio HE-AAC é uma extensão do AAC e aprimora a alta qualidade de áudio para taxa de bits menores, com uma taxa de amostragem de geralmente 48 kHz [9].

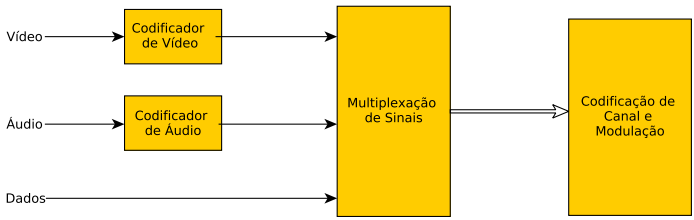


Figura 1 – Fluxo de processamento e transmissão dos Sistemas de TV Digital.

Os sinais de áudio e vídeo comprimidos, juntamente com sinais de dados, são combinados em um fluxo de transporte, através de um multiplexador, que gera o fluxo de transporte (*Transport Stream* - TS) MPEG-2. Na etapa seguinte, o sinal multiplexado é enviado a um modulador, o qual converte o sinal para um canal de frequência, para então ser transmitido pelo ar, através de uma antena. Na recepção, o processamento segue o fluxo contrário, onde o fluxo multiplexado é recebido por um demodulador, cuja saída serve como entrada em um demultiplexador. O processo de demultiplexação distingue três tipos de sinais (áudio, vídeo e dados), os quais são decodificados e exibidos na tela e nos alto-falantes respectivamente.

O MPEG-2 TS foi adotado por todos os padrões de TV do mundo, como camada de transporte. Este define um protocolo para encapsulamento de áudio, vídeo e informações de sistema, em contêineres na forma de pacotes de transporte (*Transport Packets* – TP). Dentre outras coisas, o MPEG2-TS agrega funcionalidades de correção de erro, sincronização de fluxos (*streams*), informações da transmissora, acesso condicional, Guia Eletrônico de Programação (*Eletronic Programming Guide* – EPG) e informações de data e hora, entre outros.

O ar é utilizado como meio de transmissão em sistemas terrestres, o que exige a aplicação de alguma técnica de modulação, para permitir a transmissão dos dados via radiofrequência. No padrão brasileiro, optou-se pela modulação COFDM (*Coded Ortogonal Frequency Division Multiplex*), a qual transporta sinais através de subportadoras ortogonais e divide a banda de 6 MHz em 14 segmentos de $428,571$ kHz , permitindo transmissão hierárquica, combinação de camadas, e maior robustez do sistema. No sistema ISDB-Tb, um dos segmentos é dedicado à transmissão de sinal com parâmetros ajustados para recepção móvel (celular) e portátil.

Outra característica importante da TV Digital é a aplicação de códigos corretores de erros, que permitem a validação, e correção (quando necessário) dos dados recebidos, reduzindo-se a degradação da qualidade do sinal.

Costuma-se dizer que, na TV Digital, o comportamento é binário, ou seja, ou o vídeo e o áudio são mostrados, em alta qualidade, ou nada é exibido. Nos sistemas analógicos, mesmo sinais deteriorados e com erros ainda podem ser visualizados.

Através desta seção, foi possível obter uma visão geral sobre as camadas que compõem um sistema de TV Digital. Maiores detalhes sobre a camada de multiplexação serão apresentados nas seções seguintes, pois é nesta camada que é aplicada a correção de PCR.

2.1.1 Sistema MPEG-2

O comitê MPEG-2 define uma subcamada denominada MPEG-2 *Systems*, que contempla a camada de transporte do MPEG-2 (MPEG-2 TS). Esta é responsável por encapsular dados de programa em um contêiner, que além de garantir maior eficiência na recepção, também adiciona informações importantes tanto para a correta recepção dos dados, quanto para a interação do usuário com o sistema de TV. O padrão MPEG-2 define como os TSs são formados e qual a informação que cada campo deve conter.

O TS é dividido e transmitido em pacotes denominados Pacotes de Transporte (*Transport Packet* - TP). Entre outras informações, o TP carrega consigo os dados de PCR. É importante entender como os TSs são formados e qual a posição dos *bytes* de PCR dentro dos pacotes. Esta informação ajudará a compreender como o PCR é utilizado para a reconstituição do relógio do codificador, no receptor, e sobre como a implementação realizada varre os pacotes para identificar *bytes* de PCR.

Conforme dito anteriormente, o *jitter* de PCR não pode ser maior que 500 ns, logo, é importante esclarecer como este é mensurado, o que será descrito ao final deste capítulo.

2.1.2 Multiplexação de Sinais

O multiplexador combina as informações de duas ou mais fontes de dados, em um único canal. Em sistemas de TV Digital, ele é utilizado para combinar sinais de áudio, vídeo e dados, gerados a partir de seus respectivos codificadores, resultando na geração do TS [2, 15], conforme a Figura 2.

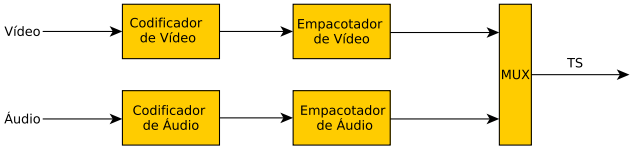


Figura 2 – Multiplexagem de Transport Stream.

Os codificadores de áudio, vídeo e dados geram sequências de bits comprimidas, em um formato denominado Fluxo Elementar (*Elementary Streams* - ES). Estes sinais são codificados de maneira independente, mas utilizam uma base de tempo comum denominada *System Time Clock - STC*. Algumas amostras deste *clock* são empacotadas junto aos ESs, de maneira a permitir a sincronização destes sinais no receptor.

Com este intuito, a recomendação H222.0 [16] definiu estruturas denominadas fluxos elementares empacotados (*Packetized Elementary Streams* - PES), que são ESs empacotados e dotados de informações de tempo necessárias ao correto funcionamento dos processos de decodificação e apresentação de áudio e vídeo. As informações de Tempo de Decodificação (*Decoding Time Stamp* – DTS) e de Apresentação (*Presentation Time Stamp* – PTS) são responsáveis pelo gerenciamento destes processos, no receptor [15]. A Figura 3 mostra a formação dos pacotes PES.

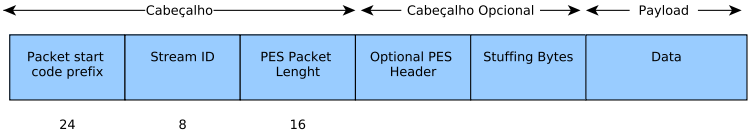


Figura 3 – Formação do PES.

Cada pacote PES é dividido em cabeçalho e corpo, sendo este último com tamanho variável. O cabeçalho do PES utiliza três *bytes* para sinalizar início de pacotes, cujo valor é fixo e igual a 000001h, 1 *byte* de Stream ID para a identificação do ES contido no pacote e 2 *bytes* que indicam a quantidade de *bytes* no pacote, após o campo *PES Packet Lenght*. Posteriormente, estão contidos bits de informações opcionais, *bytes* de preenchimento com valor FFh, e, por fim, os dados de áudio/vídeo que estão encapsulados no pacote.

O PES não possui a estrutura adequada para a transmissão pelo ar, uma vez que este ambiente de transmissão é muito suscetível a erros. A trans-

missão de pacotes com tamanho variável pode agravar essa suscetibilidade a erros, pelo fato de que quanto maior for o tamanho dos dados, maior é a probabilidade de inserção de erros até a recepção. Sendo assim, o TS foi criado com o objetivo de minimizar estes erros e torná-lo mais apropriado para a transmissão do sinal de TV.

2.1.3 A formação do MPEG-2 TS

O MPEG-2 TS é um mecanismo de transporte similar ao protocolo de redes IP (*Internet Protocol*), o qual transmite informações de áudio, vídeo e dados em pacotes divididos em cabeçalho e carga [29]. A sua unidade básica é o pacote de transporte (*Transport Packet* - TP), que carrega apenas um tipo de informação e apresenta tamanho fixo de 188 *bytes* (na prática, costuma-se dizer que também há pacotes de 204 e 207 *bytes*, se a informação de paridade Reed-Solomon for considerada), dos quais 4 ou mais *bytes* formam um cabeçalho e os *bytes* restantes constituem o corpo do pacote. Um TS pode combinar vários programas (*Multiple Program Transport Stream* - MPTS) em um único feixe de bits, com uma base de tempo específica para cada um deles. Nesta seção serão expostos os campos do TS que são relevantes para a identificação e extração do PCR.

O cabeçalho de um pacote de transporte é divido em campos conforme a Figura 4, onde cada pacote inicia com um *byte* de sincronismo (*sync byte*), cujo valor é $47h$. Este *byte* de sincronismo permite ao decodificador identificar o início de cada pacote, ou seja, ele deve ser recebido a cada intervalo de 188 *bytes*. Neste trabalho, considera-se que decodificador está no estado sincronizado após a identificação de 5 pacotes com valor $47h$, distantes 188 *bytes* entre si, o que mostrou-se adequado para identificação de *byte* de sincronismo entre os TSs comerciais comumente transmitidos.

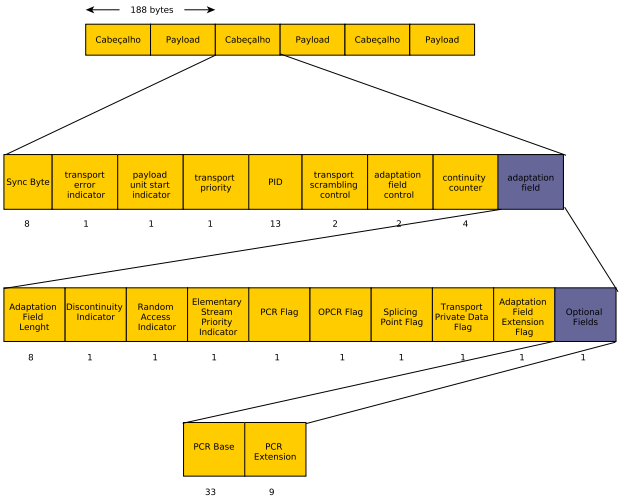


Figura 4 – Campos obrigatórios e opcionais do pacote de transporte.

Em seguida há um bit denominado indicador de erro de transporte (*Transport Error Indicator*). O mesmo tem valor 1 sempre que o pacote contiver erros, que não possam ser recuperados pelo demodulador na recepção.

Os dois bits que seguem não têm importância para a análise em questão, então não serão detalhados. A sequência de 13 bits seguintes formam o identificador de pacote (*Packet Identifier* - PID), que tem como principal função diferenciar os tipos de pacotes encapsulados no TS. Este campo possibilita a filtragem de pacotes no demultiplexador, onde este pode selecionar determinados pacotes a partir de seu respectivo PID [13].

Outro campo importante do cabeçalho do TP é o Contador de Continuidade (*Continuity Counter*), o qual é incrementado pelo codificador cada vez que o este envia um novo pacote com um mesmo PID. Esta informação auxilia o decodificador a identificar pacotes perdidos, fora de sequência ou repetidos.

Um TP tipicamente contém 4 *bytes* de cabeçalho, porém, o padrão MPEG-2 Systems define que este cabeçalho pode ser maior, em algumas circunstâncias. Este espaço extra é denominado campo de adaptação (*Adaptation Field*). O campo Controle de Campo de Adaptação (*Adaptation Field Control*) é responsável por sinalizar que o TP contém um Campo de Adaptação no cabeçalho, fazendo isto quando seu valor for ”11” ou ”10”. O PCR está contido no campo de adaptação do TP e, sempre que o pacote possuir tal

informação, haverá a possibilidade do mesmo conter PCR.

O campo *PCR Flag* indica a existência do PCR no campo de adaptação. Caso seu valor seja igual a 1, então o PCR está presente; caso contrário, o campo de adaptação não possui PCR. O PCR está presente nos 42 bits localizados após o bit *PCR Flag*.

No TS MPEG-2, um programa refere-se a um conjunto de dados formado por áudio, vídeo e dados associados a uma mesma base de tempo. Assim, é possível que um TS contenha, por exemplo, um programa formado por áudio e vídeo em HD de uma partida de futebol, e um outro programa que contém áudio e vídeo de baixa definição, de uma novela. Essa característica faz com que seja necessária a identificação de pacotes, uma vez que *bytes* de diferentes programas podem ser encapsulados em um mesmo TS. Para este fim, o padrão MPEG-2 definiu um conjunto de tabelas de informação de programa (*Program Specific Information* - PSI), com o objetivo de permitir que o decodificador consiga identificar e filtrar dados.

2.1.4 Tabelas transmitidas no feixe de TV Digital

Um receptor de TV deve ser capaz de distinguir os feixes de áudio e vídeo, de diferentes programas, e apresentar informações complementares para o usuário como guia eletrônico de programação, data e hora e classificação indicativa, entre outras informações. A recomendação H.222.0 define que estas informações devem ser enviadas no corpo de alguns pacotes de TS, no formato de tabelas, organizadas em duas categorias: Tabelas de Informação Específica de Programa (*Program-Specific Information* - PSI) e Tabelas de Informação de Serviço (*Service Information* - SI), as quais são inseridas no fluxo de transporte a partir de PES de dados, durante o processo de multiplexação de sinais. A Tabela 1 lista as tabelas PSI.

Tabela 1 – Tabelas PSI.

Nome	Descrição
PAT	Identifica programas do TS
PMT	Identifica o PID de áudio, vídeo e dados de cada programa
CAT	Restrição de acesso a conteúdo
NIT	Informações do TS e da Rede de transmissão

- Program Association Table (PAT) - A PAT é a tabela mais importante e a primeira a ser consultada pelos receptores. A principal finalidade desta tabela é apontar os programas que compõem o TS que está sendo

transmitido, de maneira análoga ao índice de um livro. A Tabela PAT tem PID fixo de valor 0 e, através desta, é possível recuperar o PID de todos os programas que compõem o TS. Assim, cada PID identificado na PAT está associado a uma tabela denominada *Program Map Table* (PMT), ou seja, cada programa de um TS corresponderá a uma tabela PMT no fluxo de transporte. Através da PAT, também é possível identificar o PID da Tabela NIT.

- Program Map Table (PMT) - As tabelas PMT indicam o PID dos pacotes de transporte que contém dados de áudio, vídeo e dados de cada programa. Com esta informação, o decodificador consegue filtrar os pacotes referentes a um programa específico e mostrar o conteúdo escolhido pelo usuário, através da seleção de um canal. Na PMT, existe um campo de 13 bits denominado PCR_PID, o qual informa o PID dos pacotes que carregam informação de PCR de um determinado programa. Sendo assim, o receptor pode filtrar pacotes com esse PID para tratar pacotes de PCR de um determinado programa, porém, esta não foi a abordagem adotada. Neste trabalho, optou-se por verificar a presença de PCR checando a flag PCR_flag presente no campo de adaptação do TS conforme explanado na seção 2.1.3. Adicionalmente, a PMT pode conter informações relacionadas a direitos de cópia (copyright).

- Conditional Access Table (CAT) - A tabela CAT é transportada em pacotes com PID 0x0001 e tem como principal função restringir o acesso a programações de TV. Assim, uma transmissora pode garantir que somente os usuários que pagam por um pacote específico receberão uma determinada programação. Além disto, esta tabela permite a associação com *streams* do tipo *Entitlement Management Message* (EMM), os quais permitem que o decodificador consiga desembaralhar e desencriptografar dados.

- Network Information Table (NIT) - A tabela NIT é definida pelo DVB e permite a transmissão de parâmetros de configuração da rede e de organização física dos TSs. Entre outras coisas, a NIT contém informações como frequência do canal, tipo de transmissão e tipo de modulação [13].

O comitê do padrão DVB propõe sete tabelas adicionais, com o objetivo de simplificar a operação dos receptores ao prover informações relacionadas aos serviços (ou programas) contidos em um TS [11]. Por esse motivo, são denominadas Tabelas *Service Information* (SI). A Tabela 2 lista as principais tabelas que compõem a SI.

Tabela 2 – Tabelas SI.

Nome	Descrição
BAT	Informações sobre buquês e respectivas informações
SDT	Informações sobre serviços
EIT	Mapeamento entre serviços e eventos
RST	Gerencia o estado dos eventos
TDT	Referência de data e hora
TOT	Sinaliza diferença de horário
ST	Invalida outras tabelas

- Bouquet Association Table (BAT) - A Tabela BAT provê informações relacionados ao buquê de programas fornecidos pelos transmissores. Por exemplo, quando um mosaico com o conteúdo de vários canais é exibido pelo provedor de tv por assinatura, o mesmo está fazendo uso desta funcionalidade de buquê de serviços e, consequentemente, consultando a BAT.

- Service Descriptor Table (SDT) - A SDT é responsável por transportar a descrição de cada serviço do TS. Assim, ela contém informações como: o nome do serviço, nome da emissora e informações descritivas do serviço.

- Event Information Table (EIT) - A Tabela EIT gerencia eventos e contém dados como nome, início e duração de evento.

- Running Status Table (RST) - A RST é uma tabela complementar à EIT e tem como objetivo informar o estado de um evento (rodando, parado, pausado, reiniciado).

- Time Date Table (TDT) - A Tabela TDT serve para encapsular data e horário (em UTC) no TS. Esta base de tempo é utilizada para controlar funcionalidades como o Guia de Programação Eletrônico (EPG).

- Time Offset Table (TOT) - A TOT é utilizada para fazer ajustes de horário baseado em localidade. O Brasil, por exemplo, que adota horário de verão durante um período do ano, deve informar aos receptores essa diferença de horário através desta tabela.

- Stuffing Table (ST) - A Tabela ST é utilizada para invalidar ou substituir tabelas.

Os dados das tabelas são divididos e transmitidos em seções, a exemplo do que é feito no ESs que são divididos em pacotes PES. As seções tem tamanho variável, porém, limitam-se a 1 kB para tabelas PSI e podem chegar a 4 kB para tabelas EIT [11]. O conteúdo de uma tabela pode ser transmitido em mais de uma seção, caso o conteúdo da mesma extrapole o tamanho de uma seção [13].

A descrição destas tabelas é importante para a compreensão dos tipos de dados empacotados num TS, e como um processador de TS deve manipular essas informações para decodificá-las e exibi-las no receptor.

2.1.5 A Referência de Relógio de Programa – PCR

O sincronismo entre os elementos de áudio, vídeo e dados de um programa, denominados ESs, é baseado em um relógio de sistema de 27 MHz denominado Relógio de Sistema (*System Time Clock* - STC). O sistema de transporte MPEG-2 provê um mecanismo que permite a geração do STC no receptor. Para isso, os TPs carregam no *Adaptation Field* uma amostra desse relógio no campo PCR, de modo a garantir a correta reprodução deste relógio no receptor, o que é muito importante para os processos de decodificação e sincronismo entre os elementos de áudio e vídeo. Uma vez que cada programa tem sua própria base de tempo, existirá somente um PCR associado a cada programa, apesar de existir a possibilidade de diferentes programas compartilharem um mesmo PCR.

O TS carrega cópias do PCR com o objetivo de permitir que o decodificador reproduza um espelho do STC na recepção. No receptor, é empregado um oscilador local de 27 MHz, o qual incrementa um contador de 42 bits, de modo que a cada recepção de pacotes com PCR, o valor do contador recebido é comparado com o valor do contador local. Se os valores forem diferentes, então uma Malha de Captura de Fase (*Phase-Locked Loop* - PLL) é utilizada para ajustar o *clock* local; caso contrário, o sistema está em perfeito sincronismo e nenhum ajuste é aplicado, conforme representado na Figura 5.

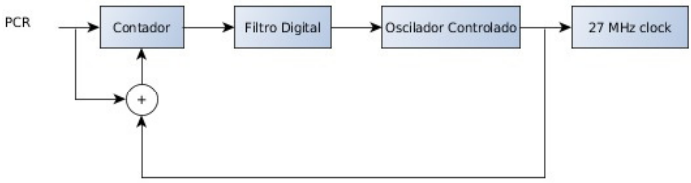


Figura 5 – Formação do PCR.

O PCR pode apresentar alguns problemas, principalmente quando o intervalo de tempo entre dois pacotes com PCR excede 40 ms [12], ou quando a diferença entre o PCR recebido e o PCR ideal é maior que 500 ns, o que corresponde a aproximadamente 14 ciclos do relógio STC.

Algumas técnicas de processamento de sinais, comumente aplicadas em sistema de TV, podem levar a erros de PCR. Por exemplo, a remultiplexação de pacotes permite que fluxos de bits de TS diferentes sejam combinados em um único TS, o que faz com que os *bytes* de PCR de um programa sejam reposicionados no TS, levando inexoravelmente à necessidade de correção dos *bytes* de PCR. Outra técnica comumente aplicada é a adaptação de taxa, que permite a modificação da taxa de dados de um TS, sem que informações anteriores sejam perdidas ou corrompidas. Entretanto, ao se alterar a taxa de dados para um valor maior, é necessária a inserção de pacotes nulos, de modo a se manter a cadência e o sincronismo de dados entre a entrada e a saída. A inserção destes pacotes nulos também faz com que os pacotes sejam reposicionados, o que acaba novamente gerando a necessidade de correção da informação de PCR.

O PCR é um número de 42 bits composto por 2 campos denominados *PCR_Base*, o qual corresponde aos 33 bits mais significativos do número, e o *PCR_Extension* que corresponde aos 9 bits restantes conforme representado na Figura 6.

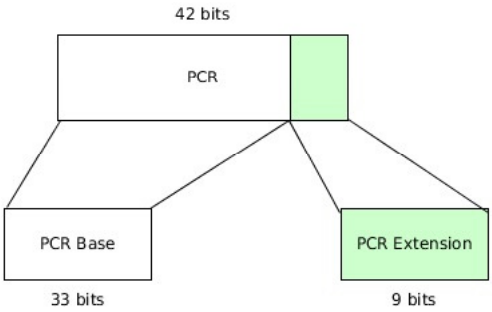


Figura 6 – Formação do PCR.

O *PCRExtension* é incrementado a 27 MHz, e cada vez que seu valor é igual a 300, este é reinicializado, e o valor do PCR Base é incrementado em 1. Isto faz com que o PCR Base seja modificado a uma frequência de 90 kHz, que é a mesma frequência utilizada pelo PTS no receptor. O cálculo do valor de PCR, para um determinado pacote *i*, pode ser obtido pelas equações a seguir.

$$PCR(i) = PCR\_base(i) \times 300 + PCR\_ext(i) \tag{2.1}$$

$$PCR\_base(i) = (valor\_STC \div 300) \bmod 2^{33} \tag{2.2}$$

$$PCR\_ext(i) = valor\_STC \bmod 300. \tag{2.3}$$

O MPEG-2 aconselha o envio de um pacote contendo PCR a cada 100 ms no máximo, enquanto o padrão DVB especifica um período máximo de 40 ms entre envios de pacotes com PCR.

Sempre que o transmissor necessita trocar sua fonte de dados, o campo indicador de descontinuidade (*Discontinuity Indicator*) do pacote recebe o valor '1', o que força a reinicialização do contador no receptor. Assim, este passa a aguardar uma nova referência de PCR, o que garante assim maior robustez no processo de decodificação.

Problemas na geração do PCR podem fazer com que o usuário tenha uma experiência ruim, como perda de quadros e erros de decodificação, o que evidencia a importância dos algoritmos de correção de PCR, em sistemas que reposicionem pacotes de TS.

2.1.6 Jitter de PCR

O tempo que os pacotes de um TS levam para chegar no receptor pode sofrer variações, causadas principalmente pelo reposicionamento de pacotes por processo de remultiplexação, ou pelo fato dos dados serem enviados pelo ar, o que os torna susceptíveis à erros e multipercursos. Estas variações são enxergadas pelo receptor na forma de *jitter*, causando inconsistências e variações no STC. Alguns tipos de decodificadores são mais sensíveis à variações no *jitter*. Por exemplo, em receptores que contêm saídas de vídeo composto analógico, o STC é utilizado para gerar o relógio de quadros e a sub-portadora de cor. Este é um tipo de sistema que não permite uma variação no *clock* de amostras, o que pode afetar todo o sistema de cor e o torna pouco tolerante ao *jitter* de PCR.

A remultiplexação reposiciona os pacotes de um TS. Com isso, as localizações temporais relativas ficam incorretas. Por exemplo, se um pacote com PCR for reposicionado para uma posição muito posterior à original, tal fato fará com que, na recepção, ele seja processado muito depois do que deveria. Assim, quando o PCR for recebido e comparado ao contador local, haverá uma diferença considerável, levando o PLL a ajustar o *clock* e gerando, consequentemente, problemas na decodificação e apresentação do conteúdo.

A literatura propõe algumas técnicas para corrigir o PCR fazendo com que os níveis de *jitter* permaneçam abaixo dos 500 ns especificados pela norma MPEG *Systems*. Essas técnicas serão detalhadas nas próximas seções deste trabalho.

2.2 Tecnologia de Hardware Reconfigurável

Os sistemas complexos implementados em hardware, de maneira geral, costumam empregar circuitos para fins específicos (*Application Specific Integrated Circuits* - ASIC) [14]. A adoção deste tipo de tecnologia faz com que qualquer modificação de projeto de hardware implique na produção de novos CIs, o que leva a um maior custo de projeto, devido ao tempo gasto com atividades deste tipo.

Os FPGAs surgiram como uma alternativa aos ASICs, uma vez que este tipo de tecnologia permite a reconfiguração do CI e a utilização deste para diferentes fins. O FPGA pode ser descrito como um circuito integrado formado por blocos de lógica configurável e interconexões entre estes blocos [22]. A maneira como essas portas são interconectadas depende de uma matriz de trilhas condutoras e comutadores (*switches*) programáveis. A configuração destes componentes se dá por meio de um arquivo binário, o qual é gerado a partir de ferramentas de síntese, que por sua vez utilizam como entrada, circuitos descritos em arquivos texto, máquinas de estado e esquemáticos. Os circuitos deste trabalho foram escritos a partir de uma Linguagem de Descrição de Hardware (*Hardware Description Language* - HDL).

FPGAs são amplamente utilizados no processamento de informações digitais, explorando-se o fato de que uma tarefa executada em hardware, na maioria dos cenários, leva um tempo menor para ser executada quando comparada a implementações em software. A principal vantagem no desenvolvimento de projetos em software é o fato de que o comportamento do mesmo pode ser alterado. Apenas com uma nova compilação de código, o sistema passa a conter um comportamento diferenciado. Com o surgimento do hardware reconfigurável, isto também passou a ser uma realidade na área de circuitos digitais.

Diante dessas características, os FPGAs tornam-se muito úteis no escopo deste trabalho. Existem diferentes opções de implementação de correção de PCR, podendo ser menos ou mais vantajosas, dependendo do cenário onde são aplicadas. Esta flexibilidade torna interessante a aplicação desta tecnologia de FPGA, uma vez que basta trocar o código VHDL para então substituir o comportamento de correção de PCR, ou seja, o sistema passa a conter um novo circuito, sem a necessidade de uma nova placa.

As arquiteturas reconfiguráveis possuem algumas vantagens quando comparadas à tecnologia ASIC. Dentre elas, é possível citar: Rápida Prototipagem, *Time-To-Market* menor e possibilidade de reprogramação para depuração, entre outras. Porém, apresentam como desvantagens: maior preço, menor desempenho, maior consumo de energia, dificuldades de depuração e limitação de memória.

2.2.1 FPGA

O FPGA é um dispositivo semicondutor, que tem como principal característica a reconfiguração do chip pós-manufatura [4], o que o torna extremamente interessante uma vez que permite o desenvolvimento incremental de funcionalidades, correção de erros e melhorias de um sistema em hardware. Outra característica interessante do FPGA é que este oferece uma boa relação custo/benefício e maior flexibilidade, quando comparado a sistemas baseados em ASIC.

Um FPGA é composto por três elementos principais: Blocos Lógicos (*Logical Blocks* - LBs), Blocos de Entrada e Saída (I/O Blocks - IOBs) e Chaves de Interconexão [22], conforme pode ser visto na Figura 7.

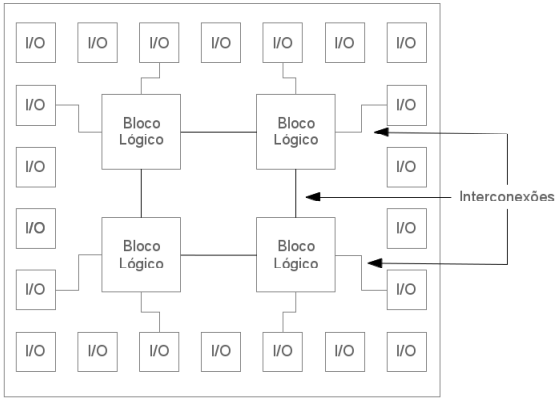


Figura 7 – Arquitetura de um FPGA.

Os Blocos Lógicos são os componentes principais do FPGA e são responsáveis pela lógica do circuito sintetizado, através do uso de combinações de *Look-Up Tables* (LUTs). As diferentes conexões destes blocos permitem a mudança de comportamento dos circuitos. Nos FPGAs da Altera, os blocos lógicos são denominados ALM (*Adaptive Logic Module*); já na Xilinx, usa-

se o termo CLB (*Configurable Logic Blocks*). Cada fabricante implementa o bloco lógico a sua maneira e não existe uma padronização para isto [28].

Os IOBs são responsáveis pelo interfaceamento das saídas provenientes das combinações de CLBs, com as interfaces de entrada e saída do sistema, sendo comumente encontrados junto aos pinos do FPGA. Os mesmos podem ser configurados como porta de entrada, saída ou entrada e saída.

As Chaves de Interconexão são responsáveis por conectar os elementos de roteamento e CLBs ou conectar CLBs e IOBs. Fica a cargo das ferramentas de síntese gerar essas conexões.

De maneira geral, os arquivos que descrevem um projeto são processados por uma ferramenta de síntese, a qual gera um arquivo (*bitstream*) que contém dados de configuração do FPGA. Este arquivo é utilizado para gravação do circuito sintetizado no FPGA, com a finalidade de informar como cada um dos componentes se comportarão e como serão interligados. Nos dias atuais, uma boa parte das FPGAs contém alguns outros componentes especiais como blocos de memória, blocos de DSP e interfaces PCIe, com o objetivo de melhorar o desempenho dos FPGAs e consumir menos energia.

Atualmente, FPGAs chegam a conter milhões de LUTs com poder computacional suficiente para operar a 600 MHz, o que os tornam capazes de processar TS em altas taxas.

Existem duas linguagens principais no domínio de FPGAs: Verilog e VHDL. Neste trabalho, optou-se pela utilização da linguagem VHDL para geração de *bitstream* para um FPGA da Altera, pela maior vivência e experiência com esta linguagem.

2.2.2 Linguagem de Descrição de Hardware - HDL

A Linguagem de Descrição de Hardware (*Hardware Description Language* - HDL) permite ao projetista de sistemas digitais descrever o comportamento de um circuito lógico, utilizando um conjunto de instruções, ao invés de um esquema tradicional baseado em diagramas esquemáticos [18]. Estas instruções estão em um alto nível de abstração e, assim, permitem ao projetista descrever o comportamento do sistema. Algumas outras linguagens de programação como C++ já são utilizadas por alguns projetistas para desenvolver sistemas para FPGA, porém, neste tipo de linguagem não há tanta flexibilidade no controle de características temporais e concorrência [18].

O projetista descreve o comportamento de seu *design* em HDL através de um conjunto de instruções, o qual é submetido a um compilador de linguagem que tem como função verificar a corretude da sintaxe do código HDL [18]. Como o estágio de desenvolvimento é independente de plataforma, o

mesmo código HDL pode ser submetido a compiladores e ferramentas de simulação de diferentes fabricantes ou fornecedores, uma vez que nesta fase ainda não existem artefatos específicos de plataforma [18].

As duas linguagens de descrição de hardware mais utilizadas são VHDL e Verilog. A VHDL é a mais antiga das duas e foi criada pelo Departamento de Defesa dos EUA (DARPA), inspirada em linguagens como Ada e Pascal. A sua principal característica está no fato de ser fortemente tipada, o que garante erros de compilação quando restrições de tipo não são obedecidas. Já a Verilog é mais moderna, baseada em C e fracamente tipada. No presente trabalho, optou-se pela linguagem VHDL.

O código VHDL pode ser utilizado tanto para propósito de síntese quanto para simulação, sendo que estes cenários apresentam algumas diferenças. Um VHDL compilado para síntese de circuito é utilizado para implementar a funcionalidade de um sistema digital, em um hardware real. Já o código compilado para a simulação apresenta algumas diferenças sintáticas e é utilizado para validar o comportamento de um circuito sintetizado.

A aprendizagem da linguagem VHDL não é muito simples, principalmente para aqueles que já desenvolvem software, uma vez que exige-se um tempo para se habituar com as particularidades da linguagem, no que diz respeito à execução paralela de instruções. Suas funcionalidades permitiram, entre coisas, a modularização dos componentes, o que facilita o desenvolvimento de testes dedicados.

2.2.3 Ferramentas de Síntese

Os fabricantes de FPGA fornecem uma ferramenta responsável por transformar o código HDL em uma lista de conexões (*netlist*) compreensível pela plataforma, a qual é denominada ferramenta de Síntese de Circuitos. A *netlist* armazena uma lista de elementos, juntamente com uma lista de conexões aplicadas aos elementos. As duas principais ferramentas são: Quartus (Altera) e ISE (Xilinx). A ferramenta de síntese gera uma lista de erros quando a sintaxe ou semântica do código HDL não estão de acordo com a especificação. Ela pode também gerar uma lista de "*warnings*", que servem para alertar sobre trechos de código que podem gerar circuito com comportamento indesejado, ou até mesmo *latches* inapropriados. Além de sintetizar, esse tipo de ferramenta também agrega outros tipos de funcionalidade, tais como: visualização do RTL gerado, visualização de *netlist*, definição de pinagem e gravação no FPGA.

Durante o desenvolvimento deste trabalho, adotou-se a ferramenta Altera Quartus II v. 12.0, para compilação e síntese do código VHDL para a

plataforma-alvo Altera DE2-115.

2.2.4 Ferramentas de Simulação

Projetos em FPGA geralmente são simulados antes de se realizar a gravação do *bitstream* na placa. A simulação permite que o projetista consiga antecipar eventuais erros e validar o comportamento do sistema, o que facilita o processo de depuração, caso erros sejam identificados. Existem dois tipos de simulação: a simulação funcional (comumente utilizado como validação inicial do sistema, não considerando o atraso na comunicação entre as portas lógicas) e a simulação temporal (simula o sistema, levando em conta os atrasos de propagação dos sinais no dispositivo físico). Nesta última, qualquer mudança no estado lógico de um sinal vai levar o mesmo tempo que levaria em um dispositivo real.

Na simulação temporal, faz-se uso de arquivos .sdf (formato de atraso padrão), os quais carregam informações específicas do FPGA, permitindo que a ferramenta de simulação consiga simular o sistema sob diversos cenários, como diferentes temperaturas e voltagens.

As ferramentas de simulação geram formas de ondas que representam os valores das entradas e saídas do sistema, ao longo do tempo. A simulação é baseada em resposta a estímulos, que são as entradas geradas pelo desenvolvedor de maneira automática ou manual. As ferramentas de simulação mais utilizadas são a ModelSim (Mentor Graphics), empregada neste trabalho, ISE Simulator (Xilinx), Synopsys VCS e NC-Sim (Cadence). Neste trabalho, optou-se pelo uso da ferramenta ModelSim, a qual pode ser visualizada na Figura 8 e onde é possível observar as formas de onda geradas a partir de uma das simulações realizadas durante este trabalho.

Os estímulos (sinais), que servem como entrada para os sistemas a serem testados, podem ser gerados graficamente, a partir da modificações dos níveis do sinais de entrada diretamente na janela dos gráficos, ou utilizando a própria linguagem VHDL para gerar os sinais e estímulos de simulação.

2.2.5 Propriedade Intelectual - IP

Durante o desenvolvimento deste trabalho, alguns blocos lógicos implementados pela Altera em forma de núcleos IP (Intelectual Property) foram utilizados. O uso de IP vem crescendo na área de ferramentas para a automação do projeto de sistemas eletrônicos (*Electronic Design Automation* - EDA), uma vez que permite o reuso de circuitos em projetos de FPGAs e

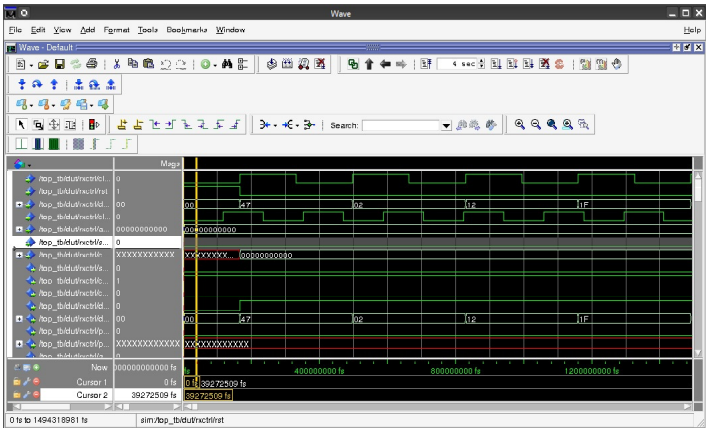


Figura 8 – Visualização da ferramenta ModelSim da Mentor Graphics.

ASICs. A maioria dos circuitos comumente encontrados em sistemas digitais estão disponíveis em forma de IP abertos, como processadores, memórias e UART.

Uma das partes mais críticas do sistema implementado reside no assincronismo entre os sistemas de entrada e saída. Os algoritmos são implementados no contexto de um adaptador de taxa, onde a taxa de um TS é convertida para valores maiores. Desta maneira, o *clock* de saída é sempre maior que o *clock* de entrada. Para gerenciar este assincronismo, um IP da Altera foi empregado para desempenhar o papel de um *buffer* assíncrono, o qual é denominado *DCFIFO* (Dual-Clock FiFo). Como o próprio nome diz, esse IP trabalha com dois *clocks* e permite que o projetista faça operações de escrita e leitura sincronizadas por *clocks* diferentes, o que se enquadra perfeitamente no contexto deste trabalho. O AltPLL, um IP da Altera, é utilizado no sistema para converter o *clock* interno da placa (50 MHz) para os *clocks* necessários (*e.g.* 4 MHz e 8 MHz).

O uso destes IPs permitiu uma redução significativa no tempo de desenvolvimento e na qualidade do código, uma vez que reutilizaram-se soluções desenvolvidas pelo fabricante do FPGA e o esforço foi concentrado nas especifidades do projeto.

3 Trabalhos correlatos de correção de PCR em Hardware Reconfigurável

Neste capítulo, serão descritos os algoritmos de correção de PCR existentes na literatura, com um foco em implementações em hardware reconfigurável. Através disto, será possível entender melhor quais as vantagens da metodologia apresentada neste trabalho, frente às abordagens clássicas.

3.1 Métodos Tradicionais de Correção de PCR

De maneira geral, as técnicas de correção de PCR seguem a seguinte lógica. No momento em que um pacote com PCR é recebido, o valor deste é carregado em um *buffer*, e algum modo de contagem é inicializado. Em um momento posterior, este mesmo pacote com PCR é escolhido para deixar o *buffer*, e então os *bytes* de PCR são substituídos pelo valor contido no componente de contagem.

O conceito básico da correção de PCR é adicionar um termo ao valor original de PCR, de modo a compensar o *jitter* ocasionado por reposicionamento de pacotes contendo PCR. A correção de PCR dá-se por

$$PCR_{corr} = PCR + \Delta PCR_{comp}, \tag{3.1}$$

onde PCR é o valor de PCR original, Δ PCR$_{comp}$ é o termo de compensação e PCR$_{corr}$ é o valor de PCR corrigido. Esta mesma lógica de correção é aplicada em todos os métodos, porém, de formas diferentes.

O Método dos Contadores Dedicados desenvolvido por Lee [19], em 1997, é o precursor em correção de PCR. O desenvolvimento deste algoritmo surgiu a partir da necessidade da implementação de um remultiplexador de TS em FPGA. A função do remultiplexador em sistemas de TV Digital é combinar vários programas em um único feixe, o pode fazer com que os *bytes* de PCR sejam reposicionados e necessitem de correção. Lee propõe que se utilize um contador de 42 bits, por TS, e que, através de um Multiplexador, selecione-se qual TS e respectivo valor de contador devem ser enviados para a saída do sistema. A correção de PCR é feita em FPGA, no bloco de remultiplexação do sistema, mas não são dados maiores detalhes a respeito da solução adotada, ou seja, não se apresentam informações importantes como FPGA utilizado durante os testes, quantidade de unidades lógicas e linguagem HDL empregada. Além de ter sido o primeiro método, outro ponto positivo deste trabalho foi que este apresentou o gerenciamento de *buffers* utilizados no projeto, o que serviu como base para a presente implementação, porém,

não há detalhes de arquitetura do sistema desenvolvido. O ponto crítico do trabalho é não mostrar o *jitter* do TS de saída, pois essa informação é muito importante para a validação do algoritmo. Este trabalho apresenta uma arquitetura para implementação deste método, que pode servir como referência para outros trabalhos.

No trabalho de Longfei [20], apresenta-se o Método da Compensação, onde é feita uma comparação com o Método do Conjunto de Contadores Dedicados. Neste trabalho, realiza-se uma análise de *jitter* do TS processado e se apresenta um comparativo da quantidade de unidades lógicas utilizadas, em diferentes cenários de teste. A principal contribuição deste trabalho foi mostrar que este algoritmo é capaz de corrigir PCR com uma quantidade fixa de unidades lógicas, independentemente do número de programas contidos no TS. Entretanto, nenhum diagrama de blocos do circuito sintetizado é apresentado, nem trechos de código em HDL que mostrem a lógica utilizada para implementar o corretor de PCR, problema recorrente do Método dos Contadores Dedicados.

Peng [23] propôs um algoritmo baseado em um conjunto fixo de contadores, ao contrário do Método dos Contadores Dedicados [19], que previa um número variável de contadores, baseado na quantidade de programas. Neste trabalho, Peng compara seu algoritmo ao Método da Compensação [20], mostrando que conseguiu uma redução na quantidade de portas lógicas de 63.7%, com relação ao Método da Compensação. Além disso, há análise de *delay* de escrita e leitura de PCR no contador, o que é uma medida importante para análise de desempenho.

Em 2009, Machmerth e Stoerte [21] apresentaram o Método dos Acumuladores, o qual não utiliza o conceito de contadores incrementados a 27 MHz. Ao invés disto, um acumulador é atualizado na mesma da taxa de saída do sistema [21]. A desvantagem desta técnica é a necessidade de operações em ponto flutuente, porém, dispensa a necessidade de um *clock* de 27 MHz. No domínio de FPGAs, isso é muito importante, uma vez que minimiza os efeitos de metaestabilidade.

Uma nova classe de métodos baseada em uma estrutura de semáforos foi proposto por Savino e Filho: Método dos Contadores Controlados por Semáforo [25] e Método de Correção de PCR Integrada à Adaptação de Taxa [26]. O primeiro tem como principal vantagem unir os pontos positivos de cada um dos métodos tradicionais, ou seja, utiliza poucos recursos e realiza o mínimo de operações matemáticas. No método de correção integrada, os autores propuseram um esquema que, além de agregar as funcionalidades de correção de PCR com semáforo, consegue predizer o pacote mais apropriada para o envio de PCRs de modo a se atingir o menor nível de *jitter*.

Cada um destes métodos têm suas vantagens, dependendo das carac-

terísticas do TS a ser transmitido e da aplicação-alvo. Se o TS possui poucos programas, o método dos contadores dedicados [19] pode ser mais interessante, uma vez que uma menor quantidade de operações aritméticas é necessária. A maioria dos TSs comerciais contém 2 programas, onde geralmente emprega-se um programa em baixa definição, no segmento 0, e um segundo programa em alta definição, ocupando os 12 segmentos restantes. Se o TS possuir muitos programas, por exemplo, mais do que dois, o método da compensação [20] pode se mostrar mais vantajoso, uma vez que utilizaria uma menor quantidade de recursos (unidades lógicas) em um FPGA.

Os trabalhos de correção de PCR em FPGA, em sua grande maioria, descrevem a quantidade de unidades lógicas utilizadas, porém, apresentam números que diferem muito entre si, uma vez que as arquiteturas utilizadas não são baseadas em uma mesma solução.

Todas as técnicas anteriores aos métodos com semáforos foram implementadas em FPGA. Diante disto, julgou-se interessante uma implementação em FPGA, para fins de comparação com os métodos tradicionais.

No contexto do presente trabalho, aborda-se uma solução para a implementação do Método de Correção de PCR Integrada à Adaptação de Taxa [26]. Neste novo algoritmo, é possível utilizar somente uma estrutura para cálculo de PCR (independente da quantidade de programas), onde os valores de PCR podem ser calculados sem a necessidade de um *clock* de 27 MHz, o que minimiza problemas de múltiplos de *clocks* e os efeitos de metaestabilidade, em uma implementação em FPGA. Este algoritmo foi implementado em VHDL e comparado a outros métodos.

As principais referências da literatura em correção de PCR [20, 19, 23], com implementação em FPGA, apresentam o seguinte tipo de análise: quantidade de unidades lógicas consumidas e desempenho temporal para a correção dos *bytes* de PCR. Este trabalho apresentará os dados demonstrados nestas referências e fará um comparativo com os trabalhos já implementados.

Nas seções seguintes, cada uma das técnicas citadas será descrita em maiores detalhes. Além disso, apresentar-se-á a proposta do Método do Contador de Bytes Controlado por Semáforos [25], frente a estes métodos.

3.2 Método dos Contadores Dedicados

O Método dos Contadores Dedicados é a principal referência sobre correção de PCR em feixes de transporte MPEG-2. Durante a sua criação, implementou-se um remultiplexador de TS com correção de PCR, em FPGA, e uma Unidade de Transporte de Dados *Data Transport Unit* - DTU foi desenvolvida em um microcontrolador de alto desempenho. A Figura 9 mostra

o diagrama de blocos do componente de correção de PCR implementado no remultiplexador o qual foi retirado do trabalho de Lee [19].

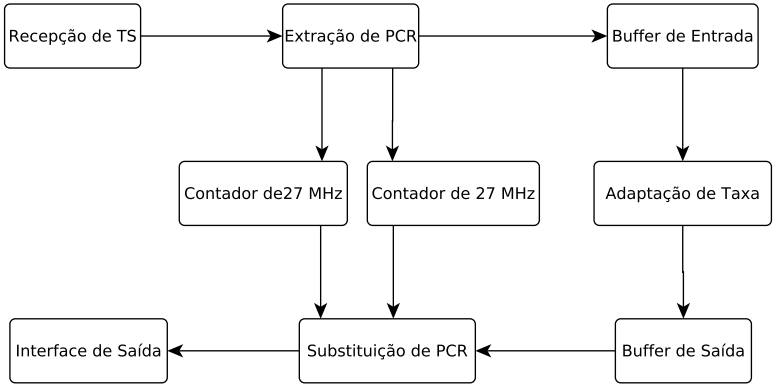


Figura 9 – Diagrama em blocos do Método dos Contadores Dedicados [19].

Como cada programa de um TS é sincronizado por um *clock* independente, a solução mais intuitiva seria a utilização de um contador de 27 MHz para cada programa, e é justamente isto que este método faz.

O gerenciador de entrada do sistema busca pelo *byte* de sincronismo denominado *sync_byte* ($47H$), para identificar o início de um pacote. Caso o pacote contenha PCR, os 42 bits deste campo são carregados em um contador livre, que é incrementado a uma taxa de 27 MHz. No gerenciador de saída, utiliza-se a seguinte lógica de agendamento:

1) Se somente um *buffer* contém um pacote completo, este é o escolhido pelo multiplexador, para enviar dados para a saída.
2) Se mais de um *buffer* tem um pacote completo, o multiplexador seguirá a ordem na qual foram recebidos.
3) Se nenhum *buffer* tem pacote completo, então um pacote nulo é enviado para a saída pelo multiplexador.

No momento em que o pacote de PCR é recuperado do *buffer*, após o processo de adaptação de taxa, este é substituído pelo valor atual do contador.

Para se estimar corretamente o tamanho dos *buffers*, uma simulação computacional do algoritmo foi realizada, em PC. A simulação mostrou que um *buffer* de 2 pacotes (376 *bytes*) seria o suficiente para evitar estouro de memória (*overflow*).

O principal problema deste método é que quanto maior for a quanti-

dade de programas no fluxo de transporte, maior será a quantidade de contadores utilizados, o que pode se constituir como um problema em sistemas onde recursos de memória sejam críticos. Outro fator a ser levado em conta, e que também influencia a complexidade do hardware, é que cada contador ficará alocado todo o tempo que os pacotes estiverem armazenados em *buffer*.

Na verdade não seria necessário um contador para cada programa, mas sim um para cada base de tempo, pois programas diferentes podem compartilhar a mesma base de tempo.

3.3 Método da Compensação

O método da compensação surgiu durante a implantação do sistema de transmissão de TV Digital, na China. Neste período, foi desenvolvido um processador de TS que incluía um módulo de adaptação de taxa, geração de PSI e a correção de PCR, onde foi empregado o método da compensação [20].

A idéia central deste algoritmo é compensar o atraso inserido no caminho percorrido pelos pacotes, através da adição de um fator de compensação, calculado conforme

$$PCR' = PCR + \Delta PCR \tag{3.2}$$

e

$$\Delta PCR = \Delta PCR_{act} - \Delta PCR_{const}, \tag{3.3}$$

em que PCR' é o valor corrigido, PCR é o valor de entrada, Δ PCR$_{act}$ é o fator de compensação e Δ PCR$_{const}$ representa uma constante utilizada para balancear o valor. Por conveniência e para simplificar a equação, o termo Δ PCR$_{const}$ é ignorado, uma vez que, nesta implementação, ele substitui a parte constante por uma média dos atrasos de pacote.

É possível assumir que *Counter_I* representa o valor do contador no momento em que um PCR é identificado, na entrada no sistema, e *Counter_O* representa o valor do contador na saída do sistema. Isto então leva às seguintes equações:

$$\Delta PCR_{act} = Counter\_O - Counter\_I \tag{3.4}$$

e

$$\begin{aligned} PCR' &= PCR + (Counter\_O - Counter\_I) \\ &= (PCR - Counter\_I) + Counter\_O \\ &= PCR'' + Counter\_O. \end{aligned} \tag{3.5}$$

No momento em que o PCR é extraído do TS, o seu valor é subtraído do valor contido no contador, representado na equação como $PCR - Counter\_I$, o qual pode ser representado como PCR''. O resultado desta subtração é armazenado no *buffer*, ao invés do valor de PCR original. No momento em que o PCR é escalonado para sair, o mesmo tem seu valor substituído pela soma entre o seu valor atual e o valor contido no contador de 27 MHz, representado como $PCR'' + Counter\_O$. A Figura 10 retirada do trabalho de Longfei [20] mostra como este algoritmo funciona.

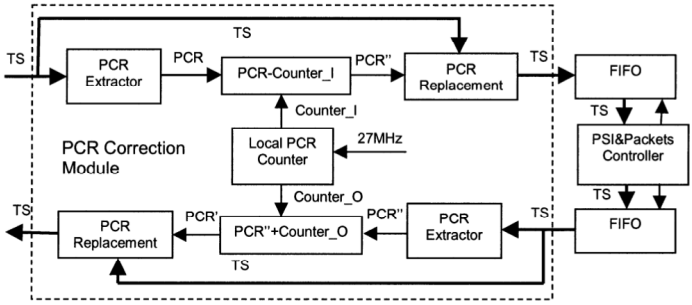


Figura 10 – Diagrama em Bloco do Método da Compensação [20].

Experimentos mostraram que o *jitter* do TS de saída se manteve praticamente inalterado, em relação ao *jitter* do TS original, e que também se manteve abaixo dos 500 ns especificados pela norma. Os autores também apresentam um comparativo com o método dos contadores, no que diz respeito à quantidade de unidades lógicas empregadas. Além disso, vale a pena ressaltar que somente a partir de 4 programas o circuito sintetizado conteve menos células lógicas que o método convencional. Entretanto, no principal cenário existente no Brasil, a transmissora só envia 2 programas remultiplexados em um mesmo TS. Logo, isso mostra que, em cenários reais, o algoritmo pode não ser tão vantajoso.

A grande vantagem deste método é restringir o uso de somente um contador de 42 bits, independentemente da quantidade de programas contidas no programa, porém, também exige operações de soma e subtração [27]. A desvantagem é que na recepção de pacotes sempre serão necessários opera-

ções de subtração, e, na saída, sempre haverá somas quando o PCR estiver presente no pacote. Se essas operações não forem rápidas o suficiente (por exemplo, se não forem feita em somente um *clock*), isso pode comprometer o processo de correção e a cadeia de processamento [27].

3.4 Método dos Contadores com Janelas Deslizantes

A abordagem proposta por Peng, Song e Wang [23] consiste na utilização de um conjunto fixo de contadores, que incrementam os 42 bits de PCR a uma taxa de 27 MHz, de maneira similar ao que ocorre no método dos contadores dedicados. Entretanto, neste algoritmo, utiliza-se um conjunto fixo de contadores, ao invés de um número variável, em função da quantidade de programas. O PCR extraído do TS é carregado, em todos os contadores existentes, e um esquema de janelas deslizantes funciona como um mecanismo de escolha de ordem dos contadores (Figura 11).

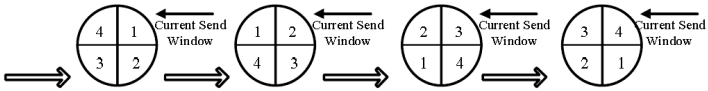


Figura 11 – Mecanismo de Janelas Deslizantes [23].

O mecanismo de janelas deslizantes é utilizado para informar, ao Multiplexador, qual contador deve ser utilizado para substituir os *bytes* de PCR. Assim, inicialmente, o contador 1 é utilizado para escrita e leitura; no pacote com PCR seguinte, o contador 2 é alocado, e assim sucessivamente. No exemplo dado, quatro contadores controlados por dois bits são utilizados, os quais são incrementados a cada operação de leitura ou escrita. O *clock* de sistema que controla o processador de TS é assíncrono ao *clock* de 27 MHz. Para evitar problemas decorrentes disso, os autores apresentam um máquina de estados, cujo intuito é garantir que o valor de PCR, extraído do TS, seja escrito no contador numa borda de subida do *clock* de 27 MHz. Esse trabalho menciona que a escrita no contador causa um pequeno desvio no valor do contador, o qual é incrementado com o tempo, causando um escorregamento de relógio (*clock skew*). Para evitar este tipo de problema, o relógio local é ajustado em intervalos de tempo fixos.

Os autores ainda argumentam que o método da compensação é melhor porque evita o escorregamento de borda, mencionado no parágrafo anterior, uma vez que utiliza uma calibragem mais fina. Porém, este método pode não ser tão vantajoso em FPGA, pois exige o instanciamento de vários somadores e subtratores, os quais consomem muitos recursos. Outro ponto ressaltado,

que é fundamental, é que em frequências altas as somas e subtrações podem não ocorrer em somente um ciclo de relógio.

O trabalho de Peng [23] apresenta o diagrama de blocos do método dos contadores, com janelas deslizantes, conforme pode ser visto na Figura 12.

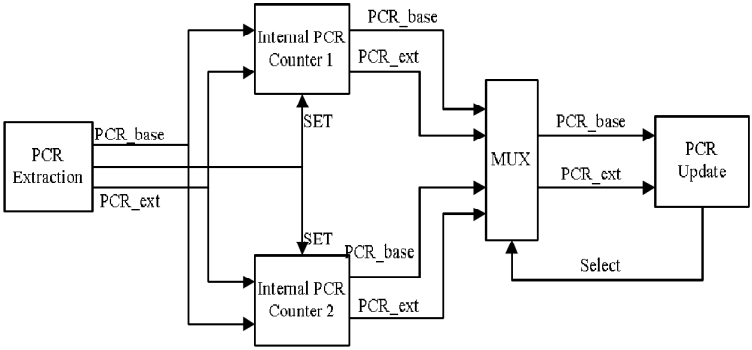


Figura 12 – Diagrama do Método dos Contadores com Janela Deslizante [23].

Os circuitos desenvolvidos neste trabalho foram sintetizados para o FPGA Cyclone EP1C12240PQFP, onde mostrou-se que, com este método, conseguiu-se uma redução de unidades lógicas de $63,7\%$, com relação ao método da compensação. Entretanto, os autores limitaram-se a fazer somente a verificação funcional do algoritmo, em alguma ferramenta de simulação. Com isso, é possível concluir que, apesar do circuito ter sido sintetizado e o mapa de alocação apresentado, este não chegou a ser executado na placa, em um cenário de teste real.

A vantagem desta técnica é a dispensa do uso de somadores e subtratores, o que causa uma grande redução no número de unidades lógicas utilizadas. Além disso, há também garantia de acurácia de temporização, no processo de correção.

3.5 Método dos Acumuladores

O método dos acumuladores foi proposto por Machmert e Stoerte [21], durante a implementação de um adaptador de transmissão A-VSB. Essa técnica corrige PCR de maneira não-convencional, através de acumuladores. A principal diferença e vantagem dessa técnica está na maneira como o cálculo de PCRs é feito, a qual dispensa a necessidade de um oscilador de 27 MHz. Ao invés disso, um pseudo-clock é gerado a partir da taxa de saída do sistema

[21], de acordo com

$$accu\_rate = \frac{sys\_clk\_freq}{transport\_rate}, \tag{3.6}$$

onde sys_clk_freq representa o relógio de programa de 27 MHz e $transport\_rate$ é a taxa de saída de pacotes. O valor de $accu\_rate$ dita a taxa de incremento do valor de PCR.

O gerenciamento do acumulador apresentado no trabalho é mostrado na Figura 13. Nesse esquema, o acumulador também pode ser inicializado sempre que descontinuidades de PCR são detectadas, o que não acontece com muita frequência, durante a transmissão. A Figura 14, retirada do trabalho de Machmerth [21], mostra um diagrama de blocos do funcionamento do Método dos Acumuladores.

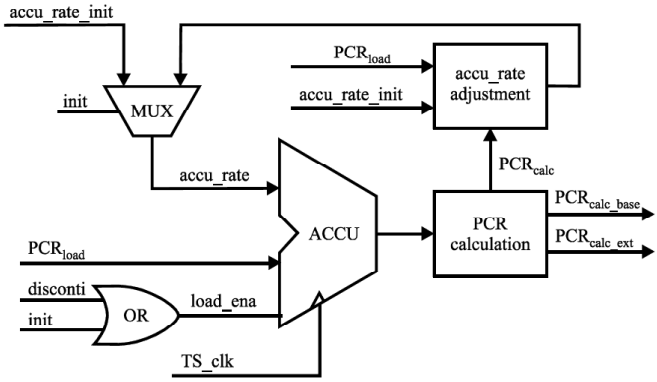


Figura 13 – Acumulador de PCR [21].

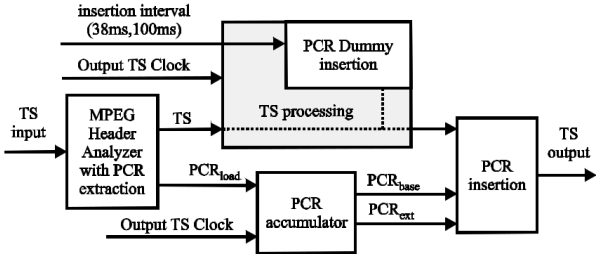


Figura 14 – Diagrama de blocos do Método dos Acumuladores [21].

Em um TS com múltiplos programas, é necessário o uso de um acu-

mulador para cada programa, porém, um esquema com um único acumulador também pode ser utilizado, adotando-se uma estratégia similar ao Método da Compensação. Entretanto, nesse caso, o valor do acumulador é utilizado como referência para soma e subtração, ao invés de um contador.

3.6 Método do Contador Controlado por Semáforos

O Método do Contador Controlado por Semáforos veio com a proposta de unir as principais vantagens dos dois métodos tradicionais, ou seja, utilizar somente um contador de 27 MHz, como no método da compensação, e realizar poucas (até mesmo nenhuma) operações aritméticas de correção [25], de maneira similar ao método dos contadores dedicados. A Figura 15 mostra o diagrama de blocos apresentado no trabaho de Savino [25].

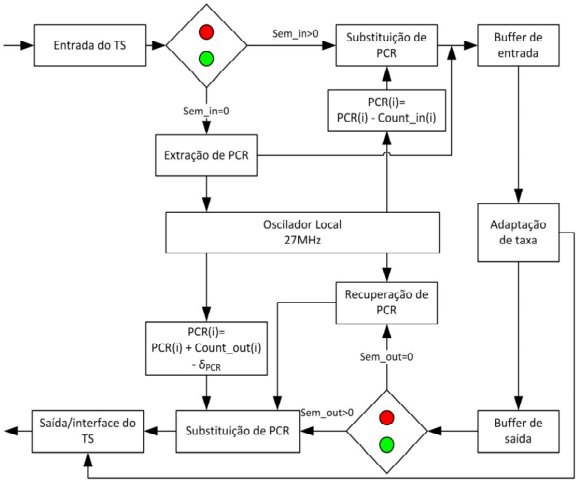


Figura 15 – Diagrama de blocos do Método do Contador Controlado por Semáforo [25].

A ideia básica dessa técnica consiste no uso de dois semáforos, um na entrada e outro na saída, cujo objetivo é gerenciar o acesso de escrita e leitura do contador de 27 MHz. Os semáforos servem para indicar em qual estado o contador se encontra: ocupado ou livre. Se o valor do semáforo é maior que zero, significa que o contador está ocupado com uma referencia de PCR, que ainda não saiu do *buffer*; caso contrário, o contador se encontra no estado livre.

Na chegada de um valor de PCR, o semáforo de entrada é lido. Se o

Tabela 3 – Tabela de comparação de jitter entre os métodos [25].

Método	Jitter de Saída
Semáforos	< 50 ns
Compensação	108 ns
Chen el al.	74 ns
He et al.	> 100 ns

seu valor for igual a zero, o PCR pode ser escrito no contador de 27 MHz; caso contrário, uma referência intermediária é armazenada no *buffer* de entrada. Já na saída do sistema, um outro semáforo é consultado. Se o valor deste for igual a zero, significa que o valor do contador pode ser usado para sobrescrever os *bytes* de PCR; caso contrário, uma operação de soma deve ser feita, com o intuito de compensar a diferença causada pelo armazenamento no *buffer*.

Na maioria das aplicações comerciais, os pacotes com PCR estão afastados entre si, de modo que dificilmente ocorrem concorrências no acesso ao contador de 27 MHz. Diante dessa característica, uma lógica baseada em semáforos, que gerencie o acesso ao contador, permite a utilização de somente um elemento. E um TS com poucos programas (até 4 programas), ou com PCRs afastados entre si, esse algoritmo realizará somente operações de atribuição, que são bem mais simples do ponto de vista computacional, aumentando a acurácia do sistema.

Savino e Filho [25] apresentaram o desempenho deste método, aplicado em um conversor de taxa, através de simulações realizadas para duas configurações de TS diferentes. Em um destes cenários, foi utilizado um TS de 30 segundos, composto por 14 programas com 8 bases de tempo, cuja a taxa foi convertida de $24,1$ Mbps para 43 Mbps. Neste caso, o algoritmo executou apenas 330 operações de 42 bits, frente a 15836 operações realizadas no método da compensação, ou seja, uma redução de operações aritméticas, de aproximadamente 98%, foi obtida. Além disso, foi mostrado que o contador se manteve no estado livre em 97,9% dos acessos, o que explicita a alta disponibilidade do contador. Isso mostra que manter vários contadores ativos é um alto desperdício de recursos.

Em comparação aos demais métodos, o contador controlado por semáforos apresenta um melhor desempenho, uma vez que alivia a quantidade de recursos e consequentemente os custos. Vale ressaltar que o método mencionado também gera TSs com menor *jitter*, sob as mesmas condições que os métodos anteriores, conforme a Tabela 3.

Esse algoritmo foi implementando em software e ainda não existe uma

implementação em hardware. Logo, uma implementação em FPGA permitiria uma comparação deste método com os demais apresentados na literatura. Neste trabalho, essa técnica é implementada em VHDL, para fins de comparação com o método dos contadores dedicados e compensação.

3.7 Método da Adaptação de Taxa Integrada à Correção de PCR

Os autores da técnica apresentada na seção anterior notaram que, apesar dos outros métodos de correção conseguirem corrigir PCR corretamente, esses acabam não considerando o descasamento de relógio, que ocorre entre os *clocks* de entrada, de PCR (27 MHz) e de saída, o que inerentemente acarreta aumento de *jitter*, no TS processado. Ao perceberem isso, os autores adicionaram elementos para minimizar ou compensar o assincronismo entre os relógios, de modo a reduzir o *jitter* do TS processado.

O principal diferencial desse método é que o mesmo provê uma metodologia, que prediz em qual pacote o PCR pode ser enviado, de modo a se obter o menor *jitter* de saída. Savino e Filho verificaram que o melhor momento para o envio do PCR é quando ocorre uma borda de subida do *clock* de 27 MHz, o que faz com que a parte fracionária do PCR seja igual a zero; caso contrário, quanto menor for a parte fracionária do número, menor será o *jitter*.

A arquitetura do sistema é composta por dois acumuladores, um localizado na entrada, que é utilizado com o intuito de compensar o momento no qual os PCRs são carregados, e outro localizado na saída, que é controlado pela taxa de saída do TS. A taxa do acumulador de saída é dada por

$$accu\_rate = \frac{sys\_clk\_freq}{transport\_rate}. \tag{3.7}$$

A idéia central do algoritmo consiste em se calcular os possíveis PCRs, para um conjunto de pacotes, e armazená-los em um vetor de atrasos, onde cada elemento é um número fracionário resultante da leitura do acumulador de PCR, na saída do sistema. A posição que contiver o PCR com a menor parte fracionária será aquele que conterá o menor *jitter* embutido. Outro conceito importante, nesse algoritmo, é que o mesmo consegue prever em qual pacote de saída o PCR seria transmitido, o qual é denominado $expected\_pkt$, para fins de simplificação. O Vetor de atrasos então armazena PCRs para pacotes variando entre $expected\_pkt$ e um valor máximo, a ser apresentado mais a frente. O pacote que contém o menor *jitter* será referenciado como $nice\_pkt$, o qual sempre será maior ou igual a $expected\_pkt$. Com isso, o sistema de saída enviará pacotes nulos, até que o valor do contador de pacotes

da saída seja igual ao valor de *nice_pkt*. Nesse período de envios de pacotes nulos, o pacote que deveria ser transmitido em *expected_pkt* continua sendo armazenado no *buffer* de saída, e a escrita continua ativa, o que faz com que o buffer aumente a quantidade de elementos armazenados. Sendo assim, é necessário alguma técnica que obtenha a quantidade de pacotes, durante a qual o sistema pode permanecer neste estado, sem que o *buffer* de saída estoure. Para isso, os autores apresentaram uma formula para se encontrar tal valor, baseada na capacidade do *buffer* e na taxa de saída do sistema:

$$max\_hold = \frac{(buff_{size} - pack\_n)}{PCR\_n} \times \frac{output\_TS\_rate}{input\_TS\_rate}, \tag{3.8}$$

onde *max_hold* representa a quantidade de elementos que o Vetor de Atrasos pode armazenar, $buff_{size}$ é o tamanho do *buffer*, *pack_n* é o número de pacotes armazenados no buffer e *PCR_n* é o número de pacotes, dentre os armazenados no *buffer*, que contêm PCR no campo de adaptação.

Os resultados de simulação desse método mostraram que o *jitter* de saída pode ser reduzido consideravelmente, quando comparado a os outros esquemas existentes. A Figura 16 mostra um diagrama de blocos, que representa o funcionamento desse algoritmo retirado do trabalho de Savino [26].

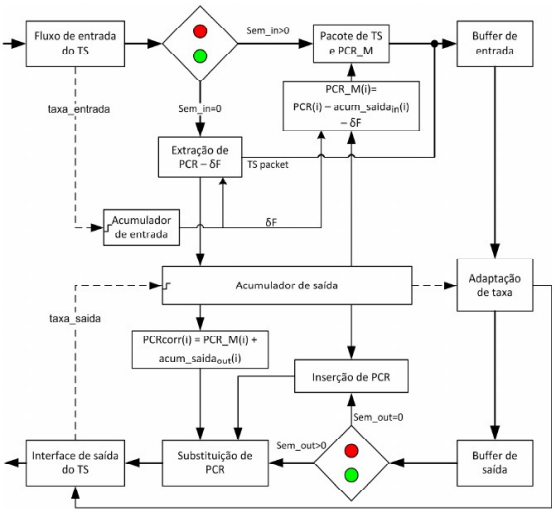


Figura 16 – Arquitetura do método da adaptação de taxa integrada à correção de PCR. [26]

Este algoritmo apresenta, como vantagem, a redução dos níveis de *jitter* na saída, o que justifica seu uso principalmente em aplicações fortemente dependentes de bases de tempo estáveis. Porém, apresenta como desvantagem uma demanda maior por número de unidades lógicas, uma vez que o sistema é mais complexo e também muito dependente de operações sobre ponto flutuante, o que pode se apresentar como impedimento quando considerado o domínio de FPGAs, uma vez que as implementações de ponto flutuante são escassas e demandam mais recursos.

3.8 Remultiplexador MPEG-2 TS em FPGA

Um dos objetivos do presente trabalho é apresentar implementações dos métodos de correção de PCR existentes e compará-los. Nessa linha, Chen e Chien [8] implementaram um remultiplexador de MPEG-2 TS, em FPGA, no qual empregaram as técnicas dos Contadores Dedicados [20] e de Compensação [20]. O objetivo principal deste sistema era prover uma interface para comunicação entre equipamentos com interface SPI e ASI. Essa conversão de interfaces faz com que seja necessário utilizar-se correção de PCR, para compensar os reposicionamentos de *bytes*. Foram implementados em FPGA: um adaptador de taxa, um corretor de PCR e uma interface de usuário. Esses módulos, sintetizados em FPGA, comunicam-se com uma CPU, um CPLD e um modulador QAM, implementado em um circuito integrado (Figura 17).

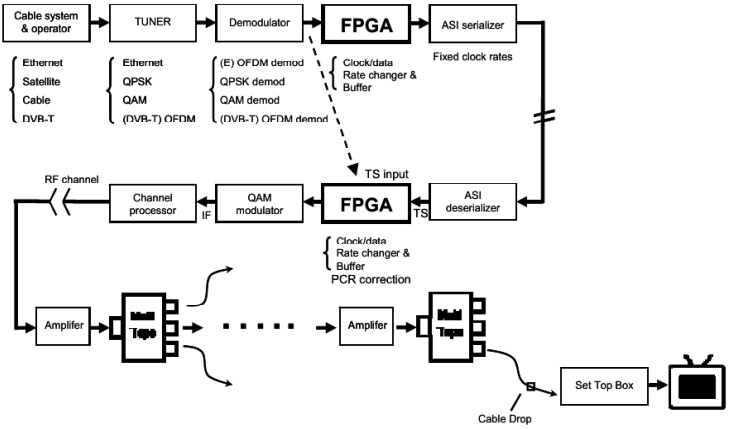


Figura 17 – Arquitetura do Remultiplexador em FPGA [8].

Uma lógica responsável por identificar o tamanho dos pacotes (188 ou 204 *bytes*) foi implementada, com o intuito de garantir o sincronismo do sistema, após o recebimento de um número seguidos de *bytes* de sincronismo ($47H$), separados por 188 ou 204 *bytes* [8]. Após esse passo, o sistema identifica se o pacote contém PCR, analisando o cabeçalho deste. Caso o pacote contenha PCR, os 42 bits que o compõem são carregados no(s) contador(es). O sistema também emprega uma FIFO, para gerenciar o assincronismo entre domínios, acrescido de algumas *flags* para informar o estado do FIFO. Esta infra-estrutura é comum aos 3 métodos implementados.

No método dos contadores dedicados, os desenvolvedores mantiveram um *pool* de contadores de 42 bits, alimentados quando o módulo *Packet Classification* identifica um pacote com PCR. Uma característica interessante é que informações adicionais relevantes são armazenadas no *buffer*: o número serial do contador, um *Tag* de PCR, para indicar que o pacote contém um PCR, e uma *Tag* de *byte* de sincronismo. Os autores referenciam o método dos contadores como Corretor de PCR Convencional (*Conventional PCR Corrector* - CPC) e desenvolvem o algoritmo conforme descrito na literatura. No artigo, cita-se que a vantagem do algoritmo é que o mesmo é fácil e simples de ser implementado, porém, tem como desvantagem a dificuldade de se definir a quantidade de contadores que devem ser alocados.

O método da compensação, denominado pelo autores como Corretor de PCR Aperfeiçoado (*Improved PCR Corrector* - IPC), segue a descrição de Longfei[20], ao fazer uma operação de subtração, na entrada do sistema, e uma adição na saída. Antes da subtração, uma comparação é feita, para verificar se o valor contido no TS é menor do que o valor do contador. Em caso afirmativo, o valor 1 é atribuído à *flag* Minus Tag; caso contrário, o valor é 0. Nesse algoritmo, os seguintes campos são armazenados na FIFO: Minus Tag (1 bit), PCR Tag (1 bit), Synchronous Byte Tag (1 bit) e TS Data (8 bits) [8]. Chen aponta que esse algoritmo reduz consideravelmente a quantidade de recursos no FPGA, além da complexidade do projeto.

Por fim, é apresentado um terceiro método, denominado Corretor de PCR Ótimo (Optimum PCR Corrector - OPC), o qual consegue unificar o gerenciamento das interfaces SPI e ASI, em um único módulo baseado no método da compensação, porém, não é descrito quais as modificações executadas para se chegar a este novo algoritmo.

O trabalho de Chen [8] é o que contém a maior quantidade de informações referentes às implementações dos métodos tradicionais, ou seja, apresentam maiores detalhes sobre como o TS é processado no circuito. As contribuições importantes, no escopo desse trabalho, foram o comparativo da quantidade de unidades lógicas, utilizadas em cada um dos métodos (Tabela 4), e a descrição da lógica empregada para identificação de PCR e controle

Tabela 4 – Comparação de quantidade de unidades lógicas utilizas pelos corretores de PCR [8].

Algoritmo de correção de PCR	Número de Células Lógicas
CPC com 66 PCRs	4825
CPC com 55 PCRs	4788
IPC	1585
OPC	1094

do fluxo de dados.

3.9 A proposta deste trabalho frente aos trabalhos correlatos

A principal proposta do presente trabalho é apresentar a implementação, síntese e resultado de simulações de uma arquitetura de correção de PCR em VHDL, utilizando o Método de Adaptação de Taxa Integrada à Correção de PCR. Nesta implementação, utiliza-se um conjunto de operações matemáticas simples sob números fracionários de ponto fixo, onde a necessidade de um oscilador local de 27 MHz é eliminada, simplificando a implementação do algoritmo em FPGA. Outros métodos tradicionais também são implementados, utilizando-se a mesma arquitetura base, o que proporciona a avaliação de eficiência de cada método, no que diz respeito a alocação de recursos e velocidade de processamento.

O algoritmo de correção integrada [26], apesar de ótimo, não leva em conta certas limitações encontradas no domínio de FPGAs. Por exemplo, este algoritmo é muito dependente de operações com números em ponto flutuante, o que é um fator limitante para FPGAs, onde o suporte a ponto flutuante é limitado e implementado pelo próprio desenvolvedor, na maioria das soluções. Os autores se limitaram a implementar o algoritmo em MatLab e ANSI C. No presente trabalho, propõe-se uma arquitetura em VHDL, que implementa o algoritmo proposto por Savino e Filho. Como principal contribuição, apresenta-se uma metodologia para simplificar o cálculo do PCR, através do uso de algumas operações aritméticas, baseadas na taxa de saída do sistema, o que simplifica a implementação e o circuito sintetizado para o sistema.

Outra contribuição importante é apresentar uma estrutura de adaptação de taxa genérica, que permite a substituição do algoritmo de correção de PCR, de maneira simples. O Algoritmo de Correção Integrada emprega as funcionalidades do Acumulador, contendo um semáforo para controlar o

acesso a este Acumulador. Desta maneira, para se obter o comportamento do Método da Compensação, basta substituir o acumulador por um contador e manter o semáforo sempre bloqueado. Desta maneira, as operações de subtração serão realizadas na recepção dos pacotes, e adições no envio destes, comportamento esperado no Método da Compensação. Já para se substituir a correção pelo Método do Acumulador Controlado por Semáforo, basta remover-se os componentes relacionados ao cálculo do *nice_pkt*, permitindo que os pacotes saiam tão logo sejam disponibilizados pelo *buffer*. Logo, a solução proposta se mostra bastante adaptável e as comparações se tornam mais justas e fáceis de serem obtidas, uma vez que uma mesma base arquitetural é utilizada.

Os algoritmos controlados por Semáforo [26, 25] nunca foram implementados em FPGA. Sendo assim, esta será a primeira implementação em hardware, o que pode apresentar ganhos de desempenhos quando comparado com as implementações em software [25], e ainda mostrará sua viabilidade técnica.

Neste trabalho foi explorado ao máximo o uso de IPs para fins de ganho de tempo, reuso de código, o que permitiu um foco maior nos componentes necessários para obter-se o conversor de taxas e o corretor de PCR.

3.10 Considerações sobre os algoritmos existentes sob a perspectiva de arquiteturas reconfiguráveis

Em geral, os algoritmos de correção de PCR são dependentes de um *clock* de 27 MHz, ou, no caso do Método dos Acumuladores [21], de um *clock* em função da taxa de saída do sistema, o que traz alguns problemas, quando a sua implementação é realizada em FPGA. Um deles é chamado de Metaestabilidade, que, em resumo, acontece quando há transferência de dados entre circuitos com *clocks* assíncronos [30]. No caso de um sistema de adaptação de taxa que empregam os Métodos clássicos (Compensação e Contadores Dedicados), há três *clocks* envolvidos: um *clock* na entrada do sistema, outro *clock* para o PCR e um terceiro *clock* na saída.

A solução de cálculo de PCR apresentada neste trabalho elimina definitivamente a necessidade de um *clock* extra, com o intuito de controlar o valor do contador de 42 bits. Ao invés disso, utiliza-se uma função que consegue calcular o novo valor de PCR, com base na quantidade de *bytes* que saem do sistema, durante o intervalo de tempo em que o respectivo pacote permanece no *buffer*. Tal técnica simplifica a correção de PCR e elimina os problemas de metaestabilidade decorrentes de um *clock* dedicado, uma vez que as operações de escrita e acesso são sincronizadas pelo *clock* de saída.

Neste trabalho, os detalhes técnicos de implementação de cada um dos componentes da arquitetura proposta são apresentados, o que pode servir como base para futuras implementações de corretores de PCR, em FPGA.

4 Implementação da solução proposta em VHDL

Neste capítulo, a implementação da arquitetura proposta será descrita, com objetivo de explicar cada componente e qual o papel que desempenham individualmente. No esquema proposto, utiliza-se um mesmo algoritmo de adaptação de taxa e somente os algoritmos de correção são substituídos, para realizar avaliações e comparações entre estes.

4.1 Visão Geral

Este trabalho apresenta a implementação de uma metodologia de adaptação de taxa de TS associada à correção de PCR. A adaptação de taxa consiste na transmissão de uma quantidade maior de dados, através da adoção de uma taxa de bits superior a utilizada nos TSs originais. Assim, por exemplo, um TS com uma taxa de 32 Mbps pode ser convertido para 48 Mbps. Entretanto, os dados do TS continuam inalterados e, para que uma maior quantidade de dados seja transmitida, é necessário a inserção de pacotes de preenchimento, denominados pacotes nulos.

A inserção de pacotes nulos faz com que os pacotes com PCR sejam reposicionados e consequentemente passem a conter referências de tempo incorretas. Logo, faz-se necessária a utilização de alguma técnica de correção de PCR, para que estes pacotes reposicionados passem a conter valores de PCR válidos. Para este fim, algumas técnicas de PCR foram implementadas em VHDL, neste trabalho, sendo então comparadas quanto a quantidade de unidades lógicas ocupadas e *jitter* de saída.

Outro ponto importante, nos processos de adaptação de taxa, é que os relógios de entrada e saída do sistema são diferentes, onde geralmente a frequência de saída é maior que a frequência de entrada, levando à necessidade do uso de *buffers* para gerenciar esse assincronismo. Nesse cenário, é comum utilizar-se uma fila (FIFO - *First In First Out*), onde a ordem de armazenamento dos *bytes* é igual à ordem de leitura. A escrita no *buffer* é feita na mesma taxa do TS original, sendo que e a leitura é feita utilizando-se a taxa de saída da adaptação de taxa. No presente trabalho, adotou-se o seguinte esquema de gerenciamento de *buffers*: sempre que o *buffer* contiver menos que 2 pacotes (376 ou 408 bytes), pacotes nulos são transmitidos; caso contrário, os *bytes* do TS armazenados no *buffer* são transmitidos.

De maneira geral, a arquitetura implementada neste trabalho é divida em 4 sub-sistemas: recepção, adaptação de taxa (*buffer*), correção de PCR e transmissão. O sistema de recepção é responsável por ler os pacotes de TS,

identificar aqueles que contém PCR no campo de adaptação e fazer a escrita no *buffer*.

O *Buffer* é gerenciado por 2 *clocks*, conforme citado anteriormente, e é responsável pelo interfaceamento entre os dois domínios de *clock*. Para este fim, foi empregado um núcleo IP da Altera, denominado $DCFIFO$, o qual implementa uma fila que, por sua vez, permite a sincronização de dados manipulados por relógios assíncronos.

A correção de PCR segue um fluxo similar para todos os algoritmos. Extrai-se o PCR no bloco de recepção e, posteriormente, o valor de PCR é escrito em um componente que armazena o valor lido. Na saída, o valor de PCR é substituído pelo valor calculado no algoritmo.

No sub-sistema de transmissão, a leitura dos pacotes é feita a partir do $DCFIFO$ (*buffer*). Quando estes possuem PCR no campo de adaptação, a correção ocorre conforme a técnica escolhida. Por último, o sub-sistema de transmissão é responsável por enviar os pacotes TS para a interface de saída do sistema.

Além destes componentes principais, o sistema também contém módulos responsáveis por prever o pacote de saída do PCR. Após o recebimento do PCR, há o módulo responsável por operações com números fracionários e outros responsáveis pelo cálculo do pacote de saída, que gera PCR com *jitter* reduzido.

A Figura 18 apresenta um diagrama resumido dos módulos que fazem parte do sistema, como descrito nos parágrafos anteriores.

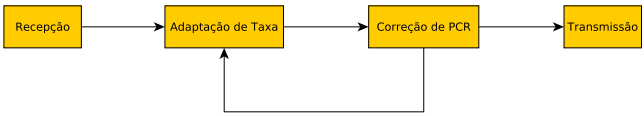


Figura 18 – Principais módulos dos algoritmos de correção de PCR.

A arquitetura desenvolvida neste trabalho permite que o bloco de Correção de PCR seja facilmente substituído, de modo que os demais blocos permaneçam inalterados. Nas seções seguintes, serão dados maiores detalhes sobre a implementação de cada um destes componentes, assim como estes se enquadram na Figura 18. Os seguintes métodos foram implementados: Método da Compensação [20], Método do Acumulador Controlado por Semáforo e, finalmente, o Método da Correção de PCR Integrada à Adaptação de Taxa [26]. Entretanto, apenas o último algoritmo será detalhado, pois é neste que se localizam as contribuições do presente trabalho. Todos os algoritmos foram implementados no kit de desenvolvimento Altera DE-2 115 e

simulados na ferramenta Mentor Graphics ModelSim.

4.2 Arquitetura do Sistema

Nesta seção, os componentes utilizados para a implementação do Método de Correção de PCR Integrada à Adaptação de Taxa serão descritos. A Figura 19 detalha os componentes implementados neste trabalho.

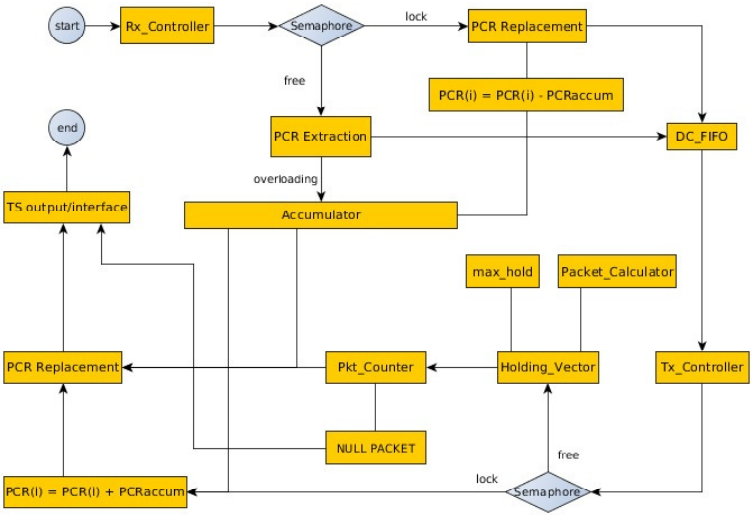


Figura 19 – Arquitetura do Sistema.

Os sistemas de correção de PCR implementados empregam um módulo de recepção e análise denominado *RxController*, um núcleo IP da Altera nominado *DCFIFO*, que implementa um *buffer* do tipo FIFO responsável pelo gerenciamento do assincronismo entre os *clocks* de entrada e saída, e um módulo de transmissão na saída do sistema, denominado *TxController*. Nos métodos controlados por Semáforo [26, 25], um par de semáforos controla o acesso ao contador compartilhado entre as bases de tempo do TS. Nas próxima seções, maiores detalhes sobre estes e os demais componentes serão disponibilizados.

4.2.1 A correção de PCR otimizada para FPGAs

A ideia principal do método proposto consiste no cálculo de um novo valor de PCR com base na quantidade de *bytes* transmitidos, através da saída do sistema, no intervalo de tempo em que o PCR permanece no $DCFIFO$. O sistema é composto por 5 componentes principais: $RxController$, $DCFIFO$, Acumulador, Semáforo, $HoldingVector$ e $TxController$

O $RxController$ extrai o PCR do TS e checa o estado do semáforo. Se o estado deste é "livre", então o PCR é utilizado para sobrescrever o Acumulador; caso contrário, uma referência intermediária é armazenada no $DCFIFO$ e o Acumulador não é alterado. Após isto, um módulo estima o pacote no qual o PCR deveria originalmente ser transmitido, denominado $packet\_expected$.

O $TxController$ é responsável pela leitura dos *bytes* de TS, a partir do $DCFIFO$, e controla o processo de substituição de PCR na saída do sistema. Quando o valor do contador de pacotes localizado na saída do sistema alcança o valor $packet\_expected$ - 1, o $Tx\_Controller$ habilita um componente responsável pelo cálculo do melhor pacote para o envio do PCR, de maneira que o menor *jitter* seja obtido, o qual é denominado $nice\_pkt$. Com esta informação, a saída pode enviar pacotes com PCR quando o *jitter* é baixo, através de uma técnica de estimação de *jitter* descrita no trabalho de Savino e Filho [26].

O $nice\_pkt$ é recuperado do $HoldingVector$, o qual armazena um conjunto de PCRs calculados como a seguir. Primeiramente, o sistema computa

$$pcr\_ticks\_byte = \frac{27 \times 10^6 \times 8}{freq \times 10^6} \tag{4.1}$$

$$pcr\_ticks\_pkt = pcr\_ticks\_byte \times pack\_size, \tag{4.2}$$

onde $freq$ é a frequência de saída do sistema, em MHz, $pack\_size$ é o tamanho do pacote (188 ou 204 bytes), pcr_ticks_pkt representa a quantidade de ciclos de 27 MHz, ocorridos durante a transmissão de um pacote, e pcr_ticks_byte mostra a quantidade de ciclos de 27 MHz, ocorridos a cada *byte* transmitido pela saída.

Considerando que $packet\_in$ corresponde ao valor do contador de pacotes quando o PCR é recebido, e $packet\_out$ corresponde ao valor deste quando o mesmo PCR é agendado para ser transmitido, então é possível obter

$$\Delta_{packet} = packet\_out - packet\_in, \tag{4.3}$$

onde $packet\_out$ varia entre $packet\_expected$ e max_hold, que condiz com o

intervalo de pacotes que o $HoldingVector$ armazena internamente, e a partir do qual o *nice_pkt* é escolhido.

A quantidade de ciclos de 27 MHz, ocorridos até um pacote de saída especifico, pode ser calculada como

$$pcr\_ticks = \Delta_{packet} \times pcr\_ticks\_pkt. \tag{4.4}$$

O *nice_pkt* corresponde à posição do $HoldingVector$ onde obteve-se o *pcr_ticks* com a menor fração. Com isso, a correção pode ser feita simplesmente através da adição de *pcr_ticks* ao valor de PCR recebido (armazenado em DCFIFO).

O PCR é codificado (e consequentemente processado) em duas partes: *PCRBase*, o qual representa os 33 bits mais significativos, e *PCRExt*, o qual é composto pelos 9 bits menos significativos do PCR. Cada vez que *PCRExt* alcança 300, então *PCRBase* é incrementado em 1. Este comportamento é representado por

$$PCR_{BaseTicks} = pcr\_ticks \div 300 \tag{4.5}$$

e

$$PCR_{ExtTicks} = pcr\_ticks\%300, \tag{4.6}$$

onde *pcr_ticks* e $PCR_{BaseTicks}$ são truncados para valores inteiros.

Finalmente, o PCR corrigido é obtido a partir da soma entre o valor de PCR original e os ciclos (*ticks*) calculados com as equações (4.5) e (4.6). Assim,

$$PCR_{BaseNew} = PCR_{Base} + PCR_{BaseTicks} \tag{4.7}$$

e

$$PCR_{ExtNew} = PCR_{Ext} + PCR_{ExtTicks}. \tag{4.8}$$

Com estas equações, é possível calcular o PCR corrigido, estimando-se a quantidade de ciclos de 27 MHz a partir da taxa de saída, o que é posteriormente armazenado em $HoldingVector$.

4.2.2 Controlador de Recepção (RxController)

O *RxController* é responsável pela recepção e análise do fluxo de entrada e também pelo interfaceamento com $DCFIFO$, semáforos de entrada e

acumulador. O seu funcionamento é sincronizado pelo relógio de entrada, denominado *clock_in*. Especificamente, as principais funções do *RxController* são: verificar se o processador de TS está sincronizado, através da identificação do *byte* de sincronismo, extrair o PCR dos pacotes, copiá-lo para o Acumulador e enviar os *bytes* recebidos, na entrada, para o *buffer* do sistema. Para desempenhar esses papéis, o componente utiliza as entradas e gera as saídas demonstradas na Figura 20.

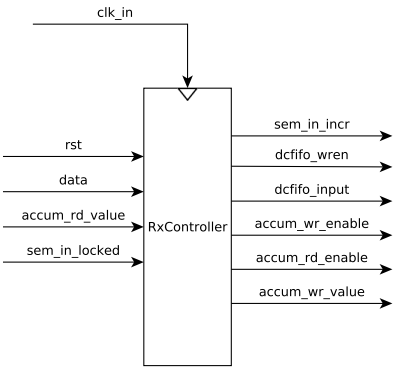


Figura 20 – Diagrama do RxController.

clk representa o *clock* de entrada do sistema, *rst* é utilizado para reiniciar o componente, *data* é o sinal de dados que contém os *bytes* do TS, *accum_rd_value* contém o valor do acumulador, para operações de subtração na entrada (quando o semáforo esta bloqueado, conforme descrito nas seções anteriores), e *sem_in_locked* é um bit que informa que o semáforo de entrada está bloqueado, o que ocorre quando seu valor é igual a 1. Já na saída, *sem_in_incr* é utilizado para alterar o semáforo de entrada, cada vez que um pacote com PCR é identificado, $dcfifo\_wren$ é utilizado para habilitar a escrita no *buffer DCFIFO*, colocando seu nível lógico em 1, $dcfifo\_input$ contém os *bytes* enviados para o *DCFIFO*, $accum\_wr\_enable$ é o sinal utilizado para habilitar a escrita dos 42 bits do PCR, no acumulador, *accum_rd_enable* habilita a leitura do acumulador, o que faz com que o seu valor seja enviado para a entrada *accum_rd_value*, e, finalmente, $accum\_wr\_enable$ habilita a escrita do PCR no Acumulador.

A lógica de controle e manipulação de dados do *RxController* é feita por uma Máquina de Estados Finitos, conforme mostrado na Figura 21.

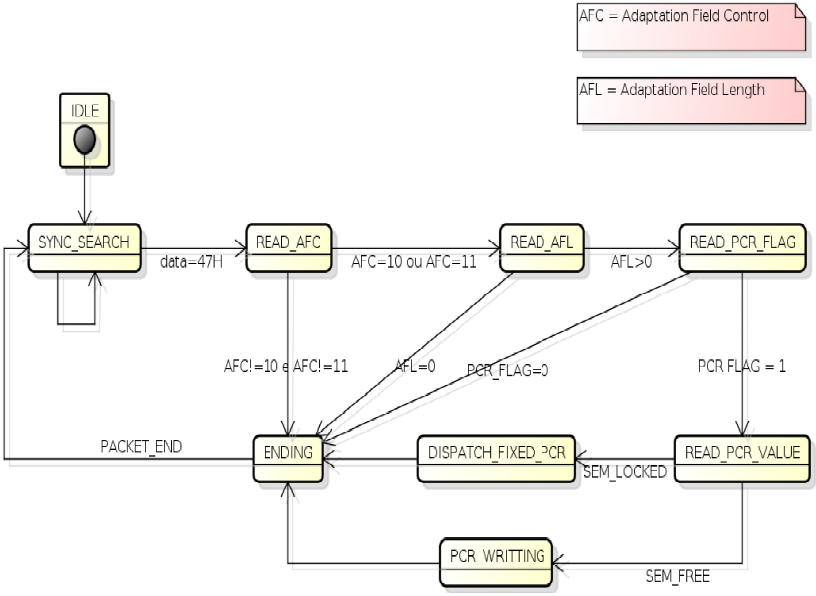


Figura 21 – Máquinas de Estados do *RxController*.

Para facilitar a compreensão do componente e dos estados, algumas figuras de simulação, que mostram o valor das entradas e saídas, serão apresentadas, onde será possível visualizar os sinais em formas de onda.

A máquina de estados é iniciada no estado *SYNCH_SEARCH*, onde a busca pelo *byte* de sincronismo (47H) é realizada. Cada pacote deve iniciar com o *byte* de sincronismo, porém, não há qualquer restrição nas normas MPEG-2 que impeça a presença do valor 47H nos demais bytes do TS. Logo, para a confirmação de que o byte 47H lido corresponde de fato ao *byte* de sincronismo, adota-se a seguinte lógica: lê-se o primeiro byte com 47h e, 188 *bytes* depois, verifica-se novamente se o valor do *byte* é igual a 47H. Após cinco verificações como esta, considera-se que o sistema está em sincronismo, levando a Máquina ao estado *READ_AFC*. A Figura 22 mostra os valor das entradas e saídas quando o componente está no estado *SYNC_SEARCH*.

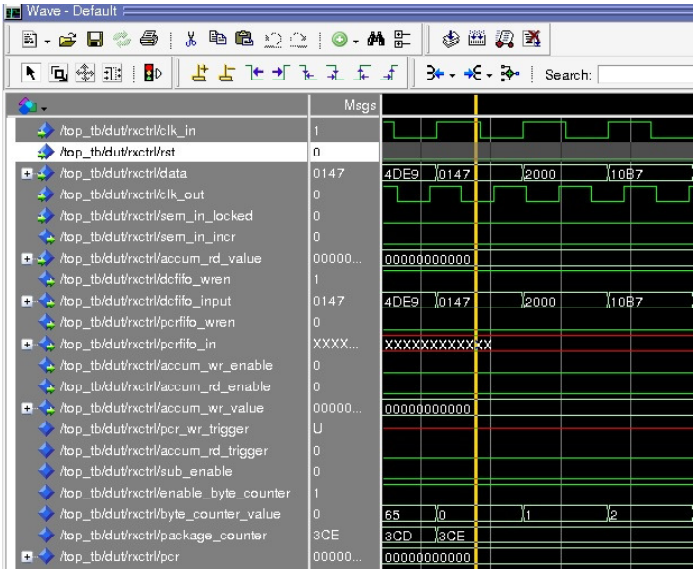


Figura 22 – *RxController* no Estado *SYNCSEARCH*.

A barra amarela, localizada na Figura 22, mostra o momento em que o componente se encontra no estado *SYNC_SEARCH*. É possível visualizar os sinais *clk_in* e *clk_out*, em formato de ondas quadradas característico dos sinais de relógio. O sinal *data* recebe 2 *bytes* (16 bits) do TS, a cada borda de subida do *clock* de entrada. Vale ressaltar que os 8 bits menos significativos correspondem ao primeiro *byte* recebido pelo TS, ao passo que os 8 bits mais significativos pertencem ao *byte* seguinte. Esta mesma lógica vale para todos os demais *bytes* recebidos em sequência, logo, o valor "0147"lido em destaque, na Figura 22, corresponde ao valor "4701"no TS. Neste mesmo momento, os sinais *sem_in_locked* e *sem_in_incr* estão em 0, o que significa que o semáforo de entrada está livre e que o mesmo não está sendo incrementado. Além disso, $dcfifo\_wren$ está habilitado, pois seu valor é 1 e o valor da entrada *data* está sendo copiado para a saída $dcfifo\_input$, conforme pode ser verificado na imagem. Como o *byte* de sincronismo foi identificado, então a máquina transita para o estado *READ_AFC*.

No estado *READ_AFC*, o algoritmo verifica se o TS contém campo de adaptação. Para isso, o campo *Adaptation Field Control* do TS é analisado. Caso o valor deste campo seja igual a ”10” ou ”11”[15], então o pacote contém campo de adaptação e a máquina transita para o estado *READ_AFL*; caso contrário, o pacote não contém campo de adaptação e, obviamente, não

possui PCR. Então, a busca pelo PCR se encerra, o que leva a máquina para o estado *ENDING*. Na Figura 23, é possível verificar as formas de onda da simulação, quando o *RxController* está no estado *READ_AFC*. Nesse caso, é possível verificar que o componente recebe, como entrada de dados, os bytes "0020"no sinal *data*. Sendo assim, o campo *Adaptation Field Control* tem valor ”10”, ou seja, o TS está sinalizando que o pacote contém campo de adaptação; com isso, a máquina transita para o estado *READ_AFL*. As saídas do componente permanecem inalteradas, com relação ao estado anterior *SYNC_SEARCH*.

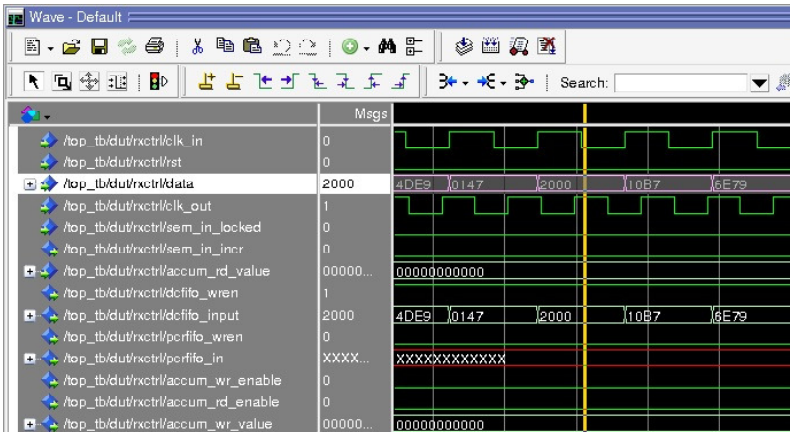


Figura 23 – *RxController* no Estado *READAFC*.

No estado *READ_AFL*, o RxController verifica as duas últimas condições para que o pacote contenha PCR. Primeiramente, o sistema checa se o comprimento do campo de adaptação é maior que zero e, para isto, o *RxController* verifica o valor do campo *Adaptation Field Length* do TS. Caso seu valor seja maior que zero, então isso significa que o pacote tem um campo de adaptação válido. A segunda condição é que o campo *PCR_flag* esteja em 1, sinalizando a existência de PCR no campo de adaptação. Se essas duas condições forem satisfeitas, então a máquina muda para o estado *READ_PCR*; caso contrário, então não há PCR no pacote e novamente a máquina muda para o estado *ENDING*. A Figura 24 mostra as formas de onda geradas durante a simulação funcional do componente, onde é possível verificar o valor dos sinais de entrada e saída do módulo.

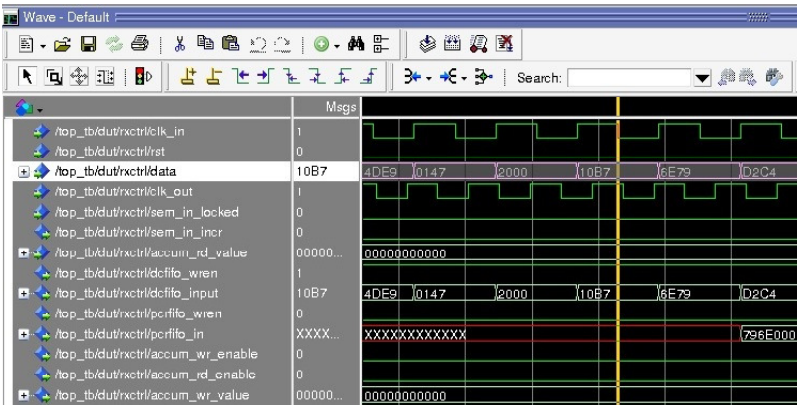


Figura 24 – *RxController* no Estado *READAFL*.

Na Figura 24, pode-se observar que a entrada *data* recebe o valor $B710h$. Logo, o campo Adaptation Field Length tem valor $B7h$, o que satisfaz a primeira condição descrita anteriormente. O byte seguinte contém o valor $10h$, o que, segundo a norma MPEG-2, corresponde justamente o campo PCR_flag, que neste caso tem valor $'1'$. Assim, o *RxController* pode constatar que o pacote contém PCR, o que acaba fazendo com que a máquina de estados transite para o estado *READ_PCR*.

No estado *READ_PCR*, os 42 bits de PCR são lidos e guardados em um registro. A máquina permanece nesse estado por três ciclos, uma vez que a entrada tem 16 bits. Assim, três ciclos são necessários para preencher estes 42 bits. A Figura 25 destaca, com duas barras amarelas, o intervalo de tempo em que a simulação temporal esteve no estado *READ_PCR*.

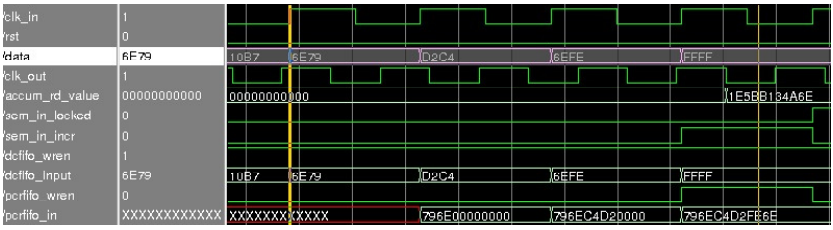


Figura 25 – *RxController* no Estado *READPCR*.

Na Figura 25, é possível verificar que, durante os três ciclos em que permanece no estado *READ_PCR*, os bytes lidos do TS foram $796EC4D2FE6Eh$, do qual foi extraído o PCR, cujo valor é $1E5BB134A6Eh$. Após a extração

desse valor, o *RxController* o copia para a saída *accum_wr_value* e habilita a escrita no acumulador, colocando a saída *accum_wr_enable* em 1. Após a leitura do PCR, a máquina transita para o estado *ENDING*.

No estado *ENDING*, o *RxController* envia os *bytes* restantes do TS para a saída, sem alterá-los, até que o contador de bytes seja igual a 203 ou 188 (o contador inicia em 0), quando, então, a máquina volta ao estado *SYNC_SEARCH* e todo o ciclo é reiniciado.

4.2.3 DCFIFO

O *DCFIFO* (*dual-clock FIFO*) é um núcleo IP fornecido pela Altera, que é responsável por gerenciar o assincronismo entre entrada e saída, operando como uma fila do tipo First-In-First-Out [6]. A Figura 26 mostra as entradas e saídas que compõem o circuito do *DCFIFO*, onde *data*[7..0] é a entrada de dados (16 bits), *wrreq* e *rdreq* habilitam a escrita e leitura no *buffer*, respectivamente, *wrclk* e *rdclk* são os clocks de escrita e leitura, respectivamente, *aclr* limpa/reinicializa o *buffer*, *wrfull* e *rdfull* informam quando o *buffer* está cheio, *wrempty* e *rdempty* sinalizam quando o *buffer* está vazio, *wrusedw* e *rdusedw* mostram a quantidade de palavras armazenadas no *buffer* e *q* é a saída de dados do sistema. Todos os sinais com prefixo *wr* são sincronizados pelo *clock* de entrada *wrclk*, ao passo que todos com prefixo *rd* são sincronizados pelo *clock* de saída *rdclk*.

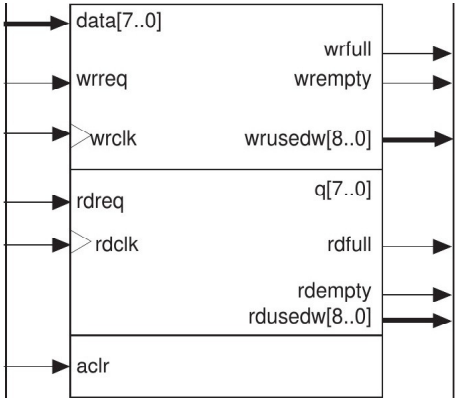


Figura 26 – Portas de Entrada e Saída do DCFIFO. [6]

Esse IP tem algumas características interessantes, que tornam seu uso mais seguro, como, por exemplo, previne que o usuário estoure o *buffer* ou

tente recuperar *bytes*, quando ele estiver vazio. Assim, sempre que o buffer estiver cheio, a sinalização em *wrreq* é ignorada; por outro lado, quando o *buffer* fica vazio, qualquer tentativa de leitura por *rdreq* também é ignorada.

É importante entender a latência de saída das sinalizações de escrita e leitura (*wrreq* e *rdreq*), para que o uso do IP seja feito da maneira esperada. Por exemplo, uma vez que *wrreq* esteja em nível alto, somente dois *clocks* depois *wrusedw* é incrementado, e a palavra só será disponibilizada, na saída, depois de um *wrclk*, seguido por um *rdclk*, com ambos sinalizados por bordas de subida. Todos as restrições relacionadas a latência, entre os sinais de entrada e saída, podem ser obtidas no guia de usuário do IP.

O IP também permite que o usuário configure outros parâmetros como: latência, número de estágios de sincronização, nível de proteção contra metaestabilidade, área ocupada e frequência máxima (fmax). Diferentes configurações desses parâmetros estão organizadas em três grupos. No presente caso, optou-se pelo grupo "*Minimal Setting for unsynchronized clocks*", no qual se obtêm dois estágios de sincronização, boa proteção contra metaestabilidade, área ocupada com tamanho médio e alta frequência máxima (fmax).

Como trata-se de um sistema assíncrono, o TS é processado com um *clock* de entrada referente à taxa de entrada e convertido, na saída, para uma outra taxa, aplicando-se um *clock* de saída diferente. Isso faz com que eventuais problemas relacionados a metaestabilidade possam ocorrer. Este IP é robusto o suficiente para tratar este tipo de cenário, o que faz desta uma implementação que se encaixa perfeitamente nos requisitos deste trabalho, reduzindo consideravelmente esforço e tempo de desenvolvimento.

4.2.4 Semáforos

O Semáforo é o componente que controla o acesso de leitura e escrita ao acumulador (ou contador), apresentando uma construção relativamente simples. A lógica central está em um contador de 4 bits, onde cada vez que este apresenta o valor zero, o semáforo está no estado 'livre'; caso contrário, para qualquer outro valor, o semáforo está no estado 'bloqueado'. O semáforo é composto por quatro entradas (*clk*, *rst*, *clear*, *increment*) e duas saídas (*count* e *locked*), conforme representado na Figura 27.

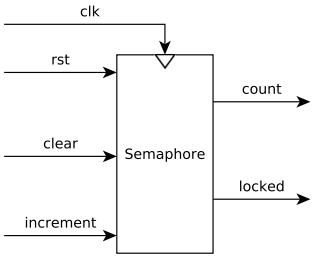


Figura 27 – Semáforo.

clk representa o relógio que sincroniza o componente, *rst* é utilizado para reiniciar o semáforo e *clear* zera o contador. Na saída, *count* mostra o valor do contador interno e *locked* informa se o semáforo está bloqueado (*locked* = '1') ou livre (*locked* = '0'). Dois semáforos são empregados no sistema: um deles está localizado na entrada e é incrementado pelo *RxController*, cada vez que um novo pacote com PCR é identificado; outro semáforo está localizado na saída do sistema, é incrementado pelo $TxController$, cada vez que um pacote com PCR deixa o *buffer* ($DCFIFO$), e então é enviado à interface de saída. Cada vez que o $TxController$ envia um pacote com PCR, ele também incrementa o semáforo de saída e, em seguida, verifica se o valor do contador dos dois semáforos são iguais. Quando isso ocorre (ex: counter_in = 2 e counter_out = 2), significa que todos os pacotes com PCR recebidos pelo *RxController* já foram recuperados do *buffer* e enviados para a saída pelo $TxController$. Logo, os semáforos podem voltar ao estado 'livre', ou seja, ambos podem ser reinicializados. Esta operação de reinicialização é feita também pelo $TxController$, colocando a entrada *rst* de ambos em nível lógico alto (*HIGH*). Se os valores dos semáforos forem diferentes, então não são reinicializados.

A Figura 28 mostra o comportamento do semáforo de entrada, durante uma das simulações, onde é possível verificar que, inicialmente, o valor do contador *count* está em zero e o semáforo está livre, o que está indicado através do sinal *locked*, cujo valor está em zero. Após um tempo, o sinal *increment* vai para 1, fazendo com que o contador *count* mude para 1 e, por consequência, a saída *locked* mude para nível alto.

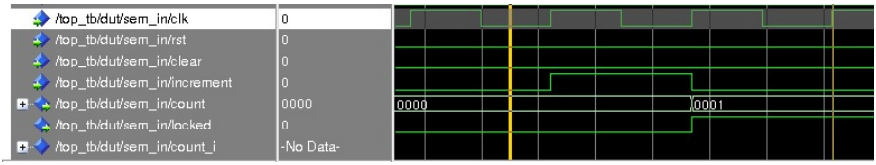


Figura 28 – Simulação Funcional do Semáforo.

O sinal *locked* é utilizado pelo *RxController*, na entrada. Quando seu valor é zero, o valor extraído do PCR é escrito diretamente no Acumulado; caso contrário, o semáforo sinaliza que o Acumulador está com acesso bloqueado, o que faz com que o *RxController* envie, para o *DCFIFO*, uma referência intermediária e mantenha o Acumulador inalterado. Já no *TxController*, o *locked* é consultado para verificar se o valor do Acumulador pode ser utilizado diretamente na substituição dos *bytes* de PCR, ou se o resultado da soma entre o acumulador e o valor do *buffer* deve ser empregado, conforme descrito no capítulo anterior.

4.2.5 Estimador de Pacote

O Estimador de Pacote tem o papel de predizer em qual pacote de saída um PCR será enviado. Por exemplo, um pacote com PCR é recebido no *RxController*, no momento em que o contador de pacotes da saída é igual a 1288. Dada esta informação, mais a quantidade de pacotes armazenados no *DCFIFO* e a taxa de leitura do *DCFIFO*, este componente consegue prever em qual pacote de saída o PCR deixará o *buffer*, para ser enviado pelo *TxController*. A equação 4.9 nos dá esta informação, a qual foi obtida no trabalho de Savino [27], conforme

$$pkt\_out = pkt\_in + (dcfifo\_wrusedw / pkt\_size), \tag{4.9}$$

onde *pkt_out* é o pacote de saída calculado, *pkt_in* representa o pacote de saída no qual o PCR foi recebido, $dcfifo\_wrusedw$ é a quantidade de palavras armazenadas no *DCFIFO* e *pkt_size* é o tamanho do pacote em *bytes* (em nosso caso 204 bytes).

Para prover a informação do pacote de saída, o componente precisa das entradas indicadas na Figura 29, onde *rst* é utilizada para reiniciar o componente, *enable* habilita o cálculo do estimador, quando seu valor está em nível lógico alto (HIGH), *pkt_in* é o valor do contador do pacotes de saída, no momento em que o PCR é extraído na entrada, $dcfifo\_wrusedw$ é a quan-

tidade de pacote armazenados, no momento em que *enable* é habilitado, e a saída *pkt_out* informa o valor calculado para as entradas informadas.

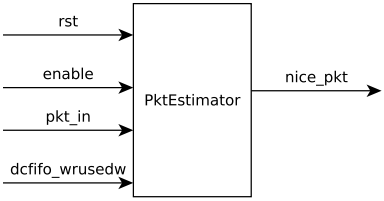


Figura 29 – Estimador de Pacote.

4.2.6 Acumulador

O *Acumulador* é um dos principais componentes do sistema, pois é responsável por armazenar o PCR quando este é recebido pelo *RxController* e por disponibilizar o PCR corrigido ao *TxController*, quando o pacote for escolhido para ser enviado.

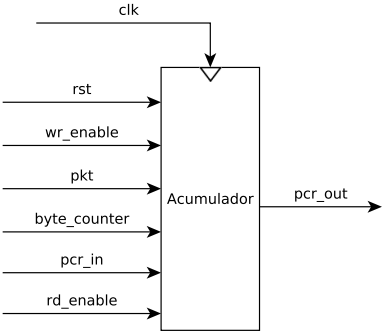


Figura 30 – Diagrama do Acumulador.

Quando um pacote com PCR é recebido pelo *RxController*, este extrai os 42 bits que compõem o valor de PCR e habilita a sua escrita no acumulador. Para isso, o *RxController* escreve os 42 bits na entrada *pcr_in* e habilita a escrita, modificando o valor da entrada *wr_enable* para '1'. O cálculo do

PCR é feito a partir da quantidade de *bytes* transmitidos pela saída do sistema, no intervalo de tempo em que o pacote com PCR permaneceu no *buffer*. Para este fim, o *TxController* disponibiliza um contador de *bytes* e um contador de pacotes, os quais são conectadas às entradas *pkt* e *byte_counter*, respectivamente. O valor destas entradas são armazenados em sinais internos, no momento em que *wr_enable* é habilitado. A Figura 31 mostra a simulação funcional do Acumulador, no momento em que um novo PCR é carregado. É possível verificar que a escrita é habilitada através do sinal *wr_enable*, a entrada *pcr_in* é igual a 1*E*5*BB*134*A*6*Eh*, o contador de pacotes de saída (*pkt*) é igual a 1288 e o contador de bytes da saída (*byte_counter*) é igual a 44.

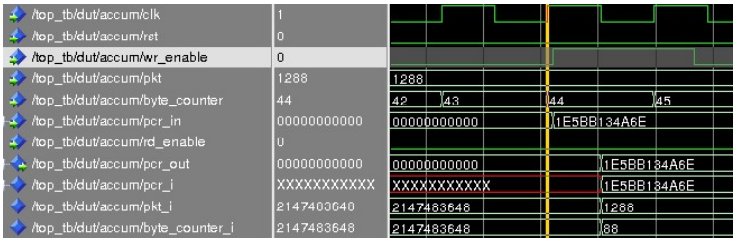


Figura 31 – Simulação Funcional do Acumulador.

Posteriormente, o pacote com PCR é escolhido para deixar o sistema, pelo *TxController*, momento no qual este requisita a recuperação do PCR corrigido ao *Acumulador*, colocando o sinal *rd_enable* em '1'. Para obter o valor corrigido, o *Acumulador* faz uma nova leitura do contador de pacote e do contador de bytes, para então calcular a quantidade de ciclos de 27 MHz, decorridos no intervalo de tempo entre a entrada do PCR e a requisição de leitura de PCR, denominado *counter_ticks* (a ser descrito na próxima seção). Na Figura 32, é possível verificar, em destaque, o momento em que um novo PCR é requisitado ao *Acumulador*. *rd_enable* é igual a 1 e o contador de pacotes da saída é 1294, o que fez com o componente colocasse na saída *pcr_out* o valor 1*E*5*BB*13708*Dh*, que representa o valor de PCR corrigido para o pacote 1294.

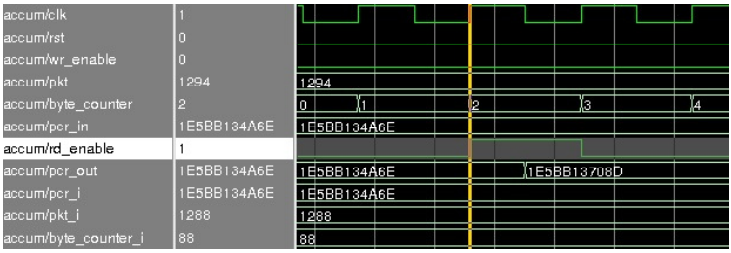


Figura 32 – Simulação Funcional do Acumulador.

De posse da quantidade de *ticks* de 27 MHz disparados, é possível calcular o PCR, conforme descrito anteriormente, na seção 4.2.1. O valor original do PCR está armazenado em uma variável *pcr_in*, onde os 33 bits mais significativos representam o campo PCRBase e os 9 bits menos significativos armazenam o PCRExtension, conforme descrito na norma MPEG-2 sistemas [15]. Sendo assim, os dois campos são separados em duas variáveis, com os mesmos nomes (PCRBase e PCRExtension). Vale ressaltar que o *PCRBase* é incrementado cada vez que *PCRExtension* alcança o valor 300. Sendo assim, na presente implementação, uma variável calcula a quantidade de vezes que *PCRBase* foi incrementado, conforme

$$PCRBaseTicks = counter\_ticks/300, \tag{4.10}$$

e uma outra armazena a quantidade de incrementos para o *PCRExtension*, de acordo com

$$PCRExtTicks = counter\_ticks\%300. \tag{4.11}$$

Com isso, é possível obter o valor de PCR corrigido, bastando somar o valor original do PCR à quantidade de incrementos, calculado de acordo com

$$PCRNewBase = PCRBase + PCRBaseTicks \tag{4.12}$$

e

$$PCRNewExt = PCRExt + PCRExtTicks. \tag{4.13}$$

O valor do PCR corrigido, disponibilizado na saída *pcr_out*, é composto pelos 33 bits do *PCRNewBase* e os 9 bits do $PCRNewExt$. O valor de *pcr_out* é disponibilizado como entrada no $TxController$, que utiliza os *bytes* dessa entrada para substituir o PCR, na interface de saída. O *counter_ticks* é fornecido pelo Contador de Ciclos, descrito na próxima seção.

4.2.7 Contador de Ciclos (TicksCounter)

O Contador de Ciclos é um dos principais componentes do trabalho, pois este é responsável pelo cálculo da quantidade de ciclos de 27 MHz ocorridos durante um intervalo de pacotes/bytes, na saída do sistema. Para calcular a quantidade total de *ticks*, primeiramente é preciso saber quantos *clocks* de 27 MHz ocorrem a cada ciclo do *clock* de saída, o que é obtido dividindo-se e a frequência do *clock* de PCR pela frequência do *clock* de saída (equação 4.1). Para exemplificar, as equações serão aplicadas a um sistema que converte a taxa do TS para 43 Mbps. Sendo assim, para este caso, $pcr\_ticks\_byte = 5.02325581395$. Outra variável importante é *pcr_ticks_pkt*, a qual mostra a quantidade de *ticks* de 27MHz a cada pacote (204 *bytes*). Para este exemplo, $pcr\_ticks\_pkt = 1024.74418605$. Assim, primeiramente calcula-se a quantidade de *ticks*, baseado na variação de pacotes, conforme

$$pkt\_ticks = pcr\_ticks\_pkt \times (pkt\_out - pkt\_in). \tag{4.14}$$

Entretanto, é importante observar que, por exemplo, se o PCR foi recebido no pacote 1000 e *byte* 15 e foi escolhido para envio no pacote 1005 *byte* 5, é necessário considerar que foram enviados 4 pacotes e 194 *bytes*, ao invés de 5 pacotes completos. Logo, para melhorar a precisão do contador de ciclos, deve-se considerar esse detalhe no cálculo do valor. Para isso, utiliza-se uma variável responsável por considerar a quantidade de *clocks* por *byte*, de acordo com

$$byte\_ticks = pcr\_ticks\_byte \times (byte\_out - byte\_in). \tag{4.15}$$

Se *byte_out* for maior que *byte_in*, então o valor de *byte_ticks* deve ser somado ao valor de *pkt_ticks*; caso contrário, esse valor será subtraído. O resultado desta última operação matemática é enviado para o acumulador, para a devida correção de PCR.

De maneira resumida, nesta solução, utilizou-se um contador de *bytes* de saída para estimar a quantidade de *clocks* de 27 MHz e, assim, corrigir o PCR. Os PCRs corrigidos, para diferentes pacotes de saída, são armazenados no Vetor de Atrasos (*HoldingVector*).

4.2.8 Vetor de Atrasos (*Holding Vector*)

O Vetor de Atrasos é um componente que armazena a quantidade de *clocks* para alguns intervalos de pacote e, com isso, disponibiliza para o sistema o pacote que possui o *counter_ticks* com menor parte fracionária, dentre aqueles calculados e armazenados em um vetor interno. O melhor pacote é disponibilizado na saída *nice_pkt*, a qual está representada na Figura 33, juntamente com as demais entradas e saídas do componente.

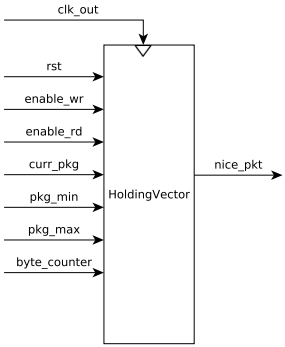


Figura 33 – Vetor de Atrasos.

As entradas necessárias ao cálculo dos PCRs e à determinação do *nice_pkt* são: *clk*, que representa o relógio de sincronismo (relógio de saída do sistema), *rst*, que é utilizado para reinicialização, *enable_wr*, que habilita a escrita no componente, quando um PCR é recebido pelo *RxController*, e *enable_rd*, que habilita a leitura do *nice_pkt*, pelo sistema de saída (TxController). A faixa de pacotes que o Vetor deve tratar é limitado por *pkt_min* e *pkt_max*, *byte_counter* contém o valor do contador de *bytes* na saída e, finalmente, a saída *nice_pkt* contém o valor do pacote com menor *jitter*, para a faixa de pacotes informada.

Um detalhe importante do Vetor de Atrasos é que sua capacidade de armazenamento é limitada pelo *buffer*, uma vez que quando *nice_pkt* é maior que o pacote originalmente planejado para envio, a leitura do *buffer* é interrompida, ou seja, seu tamanho começa a crescer, enquanto a saída envia pacotes de preenchimento, até que o contador de pacotes atinga o valor de *nice_pkt*, o que reativa a leitura de pacotes no *buffer*. Logo, é necessário que o sistema limite o tempo, ou quantidade de pacotes, em que o sistema pode interromper a leitura do *buffer*, sem que este estoure. Isso é feito através de um componente denominado *Max Hold*.

4.2.9 Atraso Máximo (*Max Hold*)

O *Max Hold* é o componente responsável por calcular a capacidade de armazenamento do Vetor de Atrasos, o qual é obtido levando-se em consideração a capacidade e o estado do *buffer*, no momento do calculo do *nice_pkt* no *Holding Vector*. Essa limitação é necessária, pois no período em que o contador de pacotes for maior que *packet_expected* e menor que *nice_pkt*, o sistema bloqueará a leitura do *buffer* e enviará pacotes nulos, fazendo com que a quantidade de palavras armazenadas (*wrused*) no *buffer* aumente a cada ciclo do *clock* de saída. Para evitar que o *buffer* estoure, é necessário algum mecanismo limite o período em que o sistema pode permanecer nesse estado, sem que o *buffer* estoure. Para isso, aplica-se

$$max\_hold = \frac{(buff_{size} - pack\_n)}{PCR\_n} \times \frac{output\_TS\_rate}{input\_TS\_rate}, \tag{4.16}$$

onde $buff_{size}$ é o tamanho do buffer de adaptação de taxa (DCFIFO), *pack_n* é o número de pacotes armazenados no *buffer* e *PCR_n* é o número de pacotes com PCR, dentre aqueles armazenados no *buffer*. O Vetor de Atrasos armazena PCRs para um número de pacotes, variando de 0 a *n*, onde a posição 0 referencia o pacote de envio calculado no Estimador de Pacote, e *n* referencia o pacote apontado pelo *max_hold*. A Figura 34 mostra o diagrama em blocos, onde é possível visualizar as entradas e saídas do componente.

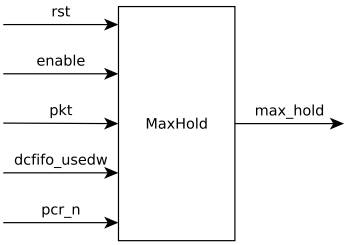


Figura 34 – MaxHold.

4.2.10 Malha de Captura de Fase (PLL)

A Malha de Captura de Fase (*Phase-Locked Loop* - PLL) é um circuito empregado para se comparar a diferença de fase entre sinais de entrada e saída, através de um sistema retroalimentado, visando-se modificar a frequência o sinal de referência. No presente sistema, um PLL é empregado para converter o clock do oscilador interno do FPGA (50 MHz) para as frequências utilizadas no sistema de adaptação de taxa, dentre elas: *clock* de entrada, *clock* de 27 MHz e *clock* de saída. A Altera disponibiliza um PLL em forma de IP, dentro da ferramenta Quartus, denominado ALTPLL. Nesta seção, as configurações disponíveis no IP e as opções escolhidas para a presente implementação serão abordadas.

O ALTPLL está disponível para os FPGAs Stratix, Cyclone, Arria e Hardcopy, através do qual é possível configurar o tipo de PLL e o modo de operação, entre outros. O IP permite que o desenvolvedor selecione entre dois tipos de PLL, porém, no presente trabalho, utilizou-se um FPGA da família Cyclone, o qual dispõe somente um tipo de PLL. De modo mais específico, o modo de seleção automático foi escolhido, onde o PLL é configurado com base em um conjunto de opções, escolhidas através de um agente de configurações, que possibilita a personalização do PLL. Durante a configuração do PLL, quase todas as opções padrões foram mantidas, alterando-se somente o *clock* de entrada do IP, para 50 MHz, e o número de *clocks* de saída, que ficou em dois, sendo um para o sistema de entrada (*clock* de entrada) e outro para o *clock* do TS convertido (*clock* de saída).

O IP apresentado nesta seção conseguiu gerar os *clocks* necessários sem maiores problemas, durante as simulações do sistema e até mesmo nos testes realizados na placa FPGA.

4.2.11 Suporte a números fracionários

Na presente implementação, optou-se por utilizar a aritmética de ponto fixo na representação de números fracionários, ao invés de representá-los por ponto flutuante. Esta escolha foi feita devido ao fato de que o suporte a ponto fixo apresenta algumas vantagens frente a ponto flutuante, dentre elas: hardware mais simples, melhor desempenho, precisão mais controlável e, principalmente, pela disponibilidade de um pacote VHDL pronto para uso.

Um pacote de autoria de David Bishop [7] foi utilizado, o qual oferece suporte às principais operações aritméticas com números fracionários em ponto fixo, além de dispor de algumas funções auxiliares para a manipulação dos números, tais como arredondamento e conversão de números em

ponto fixo para tipos nativos da VHDL, entre outros.

O pacote disponibiliza dois tipos: *ufixed*, para ponto fixo sem sinal, e *sfixed*, para ponto fixo com sinal. No código implementado neste trabalho, apenas o tipo ufixed foi utilizado. Através desse tipo, é possível definir a quantidade de bits para as partes inteira e fracionária, do número considerado. Por exemplo, um sinal pode ser declarado como na Figura 35, onde 4 bits são utilizados para a parte inteira (3 downto 0) e 4 para a parte decimal (-1 downto -4). Assim, por exemplo, o número $3,5$ pode ser representado como "0011.1000".

```
library ieee ;
use ieee_proposed.fixed_pkg.all;
signal counter: ufixed(3 downto -4);
```

Figura 35 – Código VHDL para uso de número em ponto fixo.

A precisão de ponto fixo, fornecida pelo pacote, foi suficiente para garantir o correto funcionamento do algoritmo de correção, sem maiores problemas. Nas simulações realizadas, foi possível manipular números fracionários com precisão de até 6 casas decimais, o que se mostrou suficiente para as operações aritméticas utilizadas e para a determinação do *nice_pkt*, a partir de comparações na parte fracionária dos números, em *ufixed*.

4.2.12 Controlador de Transmissão (TxController)

O *TxController* é responsável por ler *bytes* do *buffer*, gerenciar os contadores de *bytes* e de pacotes, substituir os *bytes* de PCR pelo resultado da correção e enviar os *bytes* do TS para a saída do sistema. Este componente foi concebido com nove entradas e oito saídas, conforme diagrama mostrado na Figura 36. *clk* é o *clock* de saída do sistema, *rst* é utilizado para reiniciar o componente, *data_in* contém os 16 bits lidos do *DCFIFO*, a cada borda de subida do clock, *dcfifo_usedw* contém a quantidade de palavras escritas no buffer *DCFIFO*, *accum_value* contém o valor de PCR recuperado do acumulador, *sem_in* contém o valor do semáforo de entrada, *sem_out* contém o valor do semáforo de saída, *pkt_expected* contêm o pacote originalmente escolhido para envio de PCR, calculado por *packet_calculator*, e *nice_pkt* é proveniente do *HoldingVector*. Na saída, *sem_out_incr* é utilizado para incrementar o valor do semáforo de saída, *sem_reset* é o sinal utilizado para reiniciar os dois semáforos do sistema, localizados na entrada, para controlar o acesso de escrita no acumulador, e na saída, para controlar o acesso de leitura do acumulador, *dcfifo_rden* habilita a leitura do *DCFIFO*, *accum_rd* habilita

a leitura do acumulador, *max_hold_enable* habilita a leitura do *MaxHold*, *pkt_out* sinaliza, para os demais componentes, quantos pacotes de TS já deixaram o sistema, através do *TxController*, *byte_counter_out* mostra o valor do contador de *bytes* do pacote atual e, finalmente, *data_out* contém os 16 *bytes* que compõem o TS de saída, enviados para a interface de saída.

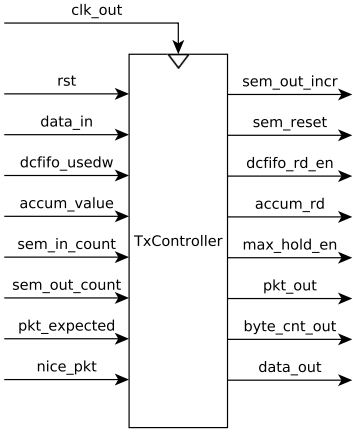


Figura 36 – Diagrama do TxController.

A exemplo do *RxController*, o *TxController* também é controlado por uma Máquina de Estados Finitos, permitindo o gerenciamento de algumas operações, como a habilitação de leitura do *buffer* (*DCFIFO*), identificação de pacotes com PCR, substituição de PCR, habilitação do *MaxHold* e *HoldingVector* e envio de pacotes nulos. Os estados da máquina estão representados na Figura 37

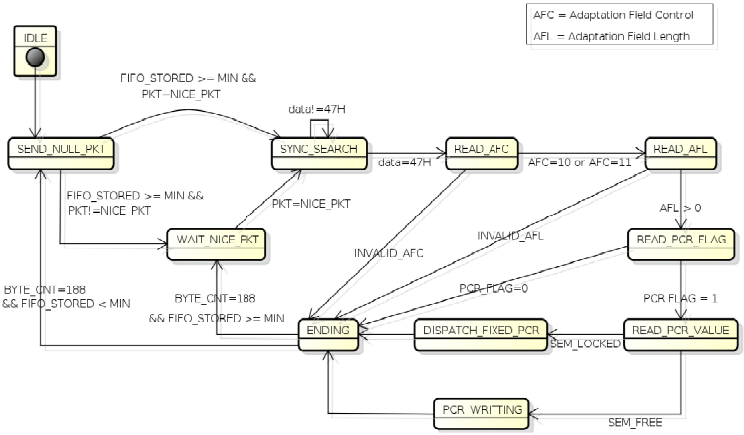


Figura 37 – Máquina de Estados do TxController.

A Máquina de Estados é iniciada no estado *IDLE*, quando a placa é alimentada, ou colocando-se o sinal $'rst'$ em nível lógico alto. Após o primeiro ciclo de *clock*, a máquina transita para o estado *SEND_NULL_PKT*, onde o *TxController* envia pacotes nulos para a interface de saída. Se o *nice_pkt* já tiver sido definido pelo *HoldingVector*, então a máquina transita para estado *WAIT_NICE_PKT*, na transição de *clock* seguinte; caso contrário, a máquina permanece no estado *SEND_NULL_PKT*, até que o *DCFIFO* alcance a marca de 2 pacotes armazenados, quando então transita para o estado *SYNC_SEARCH*.

Quando a máquina está no estado *WAIT_NICE_PKT*, tudo o que o *TxController* faz é enviar pacotes nulos para a saída, até que o valor do contador de pacotes se iguale ao *nice_pkt*.

No estado *SYNC_SEARCH*, o *TxController* verifica se o primeiro *byte* é igual ao *byte* de sincronismo ($47H$) e então a máquina transita para o estado *READ_AFC*.

No estado *READ_AFC*, o valor do campo *Adaptation Field Control* é verificado. Caso o seu valor seja igual a "10"ou "11", então o pacote tem campo de adaptação e a máquina transita para o estado *READ_AFL*; caso contrário, a transição é para o estado *ENDING*.

Em *READ_AFL*, o *TxController* verifica o valor do campo *Adaptation Field Length*, para obter o tamanho do campo de adaptação. Caso o valor do AFL seja maior que zero, então a transição é feita para o estado *SEND_PCR*_1; caso contrário, a máquina salta para o estado *ENDING*.

Nos estados *SEND_PCR*_1, *SEND_PCR*_2 e *SEND_PCR*_3, os 42

bytes que compõem o PCR são preenchidos com os 16 *bytes* recebidos pela entrada *data_in*, durante três ciclos de *clock*, respectivamente, totalizando 48 *bytes*, dos quais 6 *bytes* são ignorados por serem campos reservados, conforme determinado pela norma MPEG.

No estado *SEMAPHORE_CHECK*, o algoritmo checa se os semáforos possuem o mesmo valor, ou seja, se a saída *sem_count*, dos semáforos de entrada e saída, são iguais. Caso sejam, então a saída *sem_reset* do *Tx-Controller* é habilitada, o que faz com que os semáforos sejam reiniciados e, consequentemente, seus contadores internos sejam zerados.

Finalmente, a máquina transita para o estado *ENDING*, onde simplesmente se copiam os *bytes* do *DCFIFO* para a saída do sistema. A máquina permanece nesse estado até que o contador de *bytes* atinga o valor 203, quando então algumas entradas do componente são avaliadas, com o objetivo de se escolher o novo estado da máquina.

Se o envio do último pacote fez com que o *buffer* ficasse com menos do que 2 pacotes armazenados, então a máquina transita para o estado *SEND_NULL_PKT*, onde a leitura do *buffer* é interrompida e o sistema passa a enviar pacotes nulos para a saída. Caso isso não ocorra, outro caso verificado é se o nice_pkt e o pkt_expected já foram calculados. Se isso for verdade e o próximo pacote já é o *pkt_expected*, então a máquina transita para o estado *SYNC_SEARCH*. Finalmente, caso nenhuma dessas condições sejam satisfeitas, então a máquina pode transitar para o estado *WAIT_NICE_PKT*.

4.3 Execução no FPGA

A proposta inicial do presente trabalho era realizar testes funcionais dos algoritmos de correção de PCR, na placa de FPGA, juntamente a outros equipamentos de TV Digital comerciais, provando a aplicabilidade real do sistema. Com isso, seria possível capturar o TS em algum equipamento de TV Digital (por exemplo a partir de um gerador de TS), converter a taxa no processador gravado no FPGA e então disponibilizar o TS convertido para um outro equipamento, responsável pela transmissão do sinal pelo ar. Os equipamentos de TV Digital comumente possuem dois tipos de interfaces de comunicação: SPI (*Synchronous Parallel Interface*) e ASI (*Asynchronous Serial Interface*) [10]. Optou-se pelo uso da interface SPI, uma vez que, nos equipamentos de TV Digital, ela está geralmente disponível e apresenta menor complexidade de implementação, quando comparada à interface ASI. A SPI utiliza a sinalização LVDS (*Low-voltage Differential Signaling*) para a transmissão de dados, através de conectores DB-25. Na placa FPGA DE2-115, a única interface que provê sinalização LVDS é a Altera *High Speed*

Mezzanine Card (HSMC) [5], que utiliza um conector não compatível com o DB-25. Logo, seria necessário um cabo ou placa para converter a interface SPI para HSMC. Após um período de busca junto ao fabricante da placa e outros fornecedores, constatou-se que não se dispunha de um produto pronto que conseguisse converter as interfaces, permitindo a comunicação entre o FPGA e os equipamentos de TV. Assim, seria necessário um esforço considerável para a criação de um cabo próprio para realizar esta conversão, o que foi um impedimento para a concretização de testes junto a outros equipamentos de TV. Assim, uma outra solução foi investigada para a realização de testes na placa.

A segunda alternativa considerada foi processar o TS fazendo interfaceamento com as memórias RAM da placa. A placa DE-2 115 dispõe de 2 MB de Memória SRAM e 128 MB de Memória SDRAM, conforme pode ser visto no diagrama em blocos da Figura 38. Logo, poder-se-ia carregar o conteúdo do TS original na memória SRAM, processá-lo no FPGA e então escrever o TS convertido na Memória SDRAM.

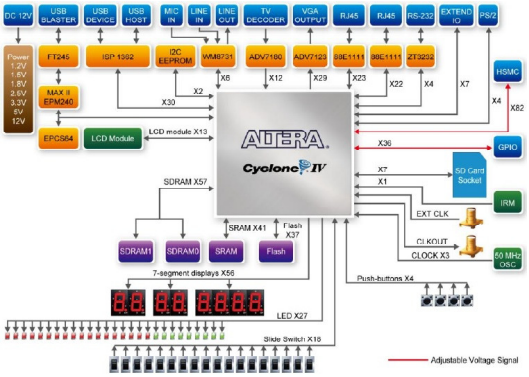


Figura 38 – Diagrama em blocos do FPGA Altera DE-2 115 [3]

O kit fornece uma ferramenta de controle denominada *DE2-115 Control Panel*, que permite o usuário se conectar à placa via USB e utilizar este programa para controlar algumas interfaces e periféricos [3]. Dentre outras funcionalidades, esta ferramenta permite que o usuário carregue o conteúdo de um arquivo, localizado no PC, nas memórias RAM (SRAM e SDRAM) da placa, e também que o conteúdo das memórias seja gravado em um arquivo, no PC. Com isto, é possível carregar o conteúdo do TS na memória SRAM, a partir da ferramenta, gravar o sistema no FPGA para o processamento, lendo-se os *bytes* a partir da SRAM, e escrever os *bytes* do TS processado na SDRAM, a partir do $TxController$. Assim, fez-se necessária a

implementação de um controlador SRAM, para fazer leituras na SRAM e disponibilizar os *bytes* lidos para o processador de TS, através do componente de entrada *RxController*.

A SRAM da DE-2 115 tem 2 MB de capacidade, com um barramento de dados de 16 bits, e opera com *clocks* de até 125 MHz [3]. A arquitetura das memórias SRAM é relativamente simples, possuindo um barramento de endereço, dados, e algumas portas para habilitação de leitura e escrita. O desenvolvimento de um controlador para a SRAM foi bem simples, pois exigiu somente o mapeamento de interface com os sinais disponibilizados pelo chip. Como sua taxa de operação é alta, não houve problemas para realizar leituras de dados neste chip, controlado por *clocks* na ordem 10 MHz. Assim, foi possível realizar testes na placa envolvendo, o envio de dados da SRAM para o *RxController*. Nestes testes, o conteúdo de um TS foi carregado em memória e o *RxController* conseguiu identificar os *bytes* de sincronismo, extraiu o PCR, e bloqueou o semáforo de entrada, após a identificação do PCR. Durante os testes, o conteúdo dos PCRs identificados foram escritos nos *displays* de 7 segmentos da placa e um LED vermelho foi utilizado para sinalizar o estado do semáforo de entrada, o que funcionou conforme desejado.

Já a SDRAM possui uma arquitetura bem mais complexa e que, entre outras coisas, exige a implementação de um circuito que garanta a recarga dos seus capacitores, além de possuir algumas restrições temporais, que tornam sua implementação bastante complexa. Durante a realização do trabalho, implementou-se um controlador SDRAM para o chip IS42S16320B, presente na DE2-115. Ao se realizarem testes na placa, verificou-se que as operações de escrita funcionavam corretamente durante um intervalo de tempo, mas, após um certo endereço o conteúdo da memória, não casava com os valores contidos no TS, devido à alguma violação na logica de controle de atualização e temporização implementada em VHDL.

Como o intuito principal do trabalho não era implementar um controlador de memoria SDRAM, então resolveu-se focar os esforços na implementação dos algoritmos de correção de PCR e nos detalhes de arquitetura, provando sua corretude através de simulações e permitindo que os ajustes necessários para execução na placa fossem realizados em outro momento. O foco do trabalho ficou então em propor uma arquitetura flexível e que processasse o TS, convertendo-o para diferentes taxas e sob diferentes configurações, além de garantir que os níveis de *jitter* se mantivessem o mais baixo possíveis. No próximo capítulo, serão apresentados os resultados, a partir das simulações funcionais realizadas com os algoritmos implementados em VHDL.

5 Resultados Experimentais

Este capítulo apresenta o desempenho do adaptador de taxa, descrito em VHDL sob diferentes configurações de taxa, e algoritmos de correção de PCR. Os *Transport Streams* avaliados são obtidos a partir das simulações, que geram arquivos no formato TS com a taxa convertida, e a partir destes arquivos a análise de *jitter* de PCR é feita, para se verificar o desempenho e compará-los a outros trabalhos da literatura. Além do *jitter*, o intervalo de tempo entre PCRs e o desvio de frequência também são calculados, pois eles são importantes para a verificação da acurácia do *clock* gerado pelo PCR. As configurações de TS utilizadas nas simulação permitem que o algoritmo seja submetido a cenários diferentes e, assim, provam a sua corretude e robustez.

As simulações funcionais dos circuitos implementados foram realizadas na ferramenta ModelSim da Mentor Graphics, utilizando-se arquivos no formato TS como entrada, e permitem que os componentes sejam submetidos à mesma sequência de *bytes* que um aparelho de TV recebe das transmissoras. Todos os circuitos descritos em VHDL foram sintetizados para o FPGA Altera DE2-115.

O *jitter* de PCR e o número de unidades lógicas ocupadas pelo circuito sintetizados são comumente avaliadas nos trabalhos que implementaram corretores de PCR em FPGA [20, 23], os quais serão apresentados para cada uma das técnica analisadas. Para a análise de jitter de PCR, foi empregado um *script* em MatLab, e o número de unidades lógicas foi obtido a partir da ferramenta de síntese Quartus II, da Altera.

5.1 Medição do jitter de PCR

Os feixes elementares multiplexados no TS devem ser decodificados e apresentados a uma taxa fixa, para garantir que os mesmos sejam visualizados no momento adequado, o que leva à necessidade de um *clock* estável na recepção. O *jitter* é um termo comumente empregado em processamento de sinais e está relacionado a variações ou desvios temporais, na recepção de dados. O *jitter* de PCR, mais especificamente, é um erro percebido no decodificador, quando existe diferença entre o valor de PCR recebido e o valor ideal para um dado pacote de TS. A cadeia de processamento dos sinais de TV (codificação, multiplexação e transmissão) pode fazer com que os pacotes cheguem em intervalos de tempo variáveis, o que causa o *jitter* de PCR. As fontes de *jitter* de PCR mais comuns são erros em algoritmos de codificação e reposicionamento de pacotes com PCR.

Neste capítulo, as medições apresentadas avaliam a acurácia dos PCRs gerados, onde é verificada a diferença entre o valor do PCR contido no TS e o valor ideal para um determinado *byte* ou posição no TS, de acordo com a recomendação ETSI TR 101 290 [12]. Assim, o *jitter* de PCR para um determinado pacote é dado por

$$jitter(i) = \frac{PCR(i) - PCR\_ideal}{27MHz}, \tag{5.1}$$

onde *i* representa um pacote, $PCR(i)$ é o PCR lido, para o pacote *i*, e *PCR_ideal* é o PCR estimado.

Para se verificar o erro inserido pela adaptação de taxa e correspondente correção de PCR, primeiramente é preciso checar o tempo que o pacote permanece no adaptador, o que é dado por

$$T_{pkt} = \frac{8 \times STC_{freq}}{taxa\_saida} \times tam\_pacote, \tag{5.2}$$

onde T_{pkt} representa a quantidade de períodos de clock de saída, por pacote, STC_freq é a frequência de PCR (27 MHz), *taxa_saida* representa a taxa para a qual TS está sendo convertido e *tam_pacote* representa o tamanho do pacote de transporte (188 ou 204 *bytes*) [27]. Assim, em uma conversão para 48 Mbps, o T_{pkt} obtido seria

$$T_{pkt} = \frac{8 \times 27MHz}{48Mbps} \times 204 = 918, \tag{5.3}$$

mostrando que, durante o processo de adaptação, o pacote permanece por um número inteiro de *clocks* de 27 MHz. Desta maneira, o envio do PCR para a saída fica sincronizado com o relógio de 27 MHz, fazendo com que o nível de *jitter*, no TS de saída, permaneça no mesmo patamar do TS de entrada. Porém, se a conversão for feita para uma taxa de 30 Mbps, obtém-se

$$T_{pkt} = \frac{8 \times 27MHz}{30Mbps} \times 204 = 1468,8. \tag{5.4}$$

Assim, o clock de saída terá um erro de $0,8$ período de *clock*, denominado $\delta clock\_saida$, o que evidencia o descasamento entre o *clock* de saída e o *clock* de 27 MHz. O erro de correção de PCR é influenciado pela quantidade de pacotes, durante a qual o PCR permanece no módulo de adaptação de taxa. O erro acumulado, em um intervalo de pacotes, pode ser calculado como

$$erro_{\text{correção}} = \frac{(\delta clock\_saida \times pacotes) mod 1}{STC_{freq}}. \tag{5.5}$$

Por exemplo, se um pacote com PCR permanecer no *buffer* durante 18 paco-

tes, então o erro de correção, ou *jitter*, será

$$erro_{\text{correção}} = \frac{(0,8 \times 18) mod 1}{27MHz} \approx 14,81ns. \quad (5.6)$$

Entretanto, se o mesmo pacote permanecer no módulo de adaptação por 20 pacotes, então não haverá *jitter* no pacote de saída, pois

$$erro_{\text{correção}} = \frac{(0,8 \times 20) mod 1}{27MHz} = 0ns. \quad (5.7)$$

Esta base matemática é utilizada para se estimar o *jitter* para diferentes conversões, o que servirá como referência para a análise do *jitter* obtido, a partir das simulações realizadas neste trabalho.

Um dos principais requisitos para a validação dos algoritmos de correção de PCR é garantir que o nível de *jitter* de PCR, do TS de saída, permaneça no mesmo patamar do *jitter* do TS de entrada, e sem exceder os 500 ns especificados pelo MPEG. A partir disto, surge a necessidade de se ter alguma ferramenta que permita medir o *jitter* de um TS. Para este fim, um *script* Matlab (.m) foi implementado, durante o trabalho de Heitor e Filho [27], com o objetivo de analisar, validar e medir o *jitter* de cada programa contido no TS de saída, de acordo com a recomendação TR 101 290 [12]. A execução deste *script* gera um gráfico que mostra o nível de *jitter* de PCR, para cada pacote do TS. Neste gráfico, cada ponto no eixo das abscissas representa um PCR identificado no TS, sendo que o eixo das ordenadas mostra o *jitter* medido para o dado PCR. A diferença entre o PCR ideal e o PCR medido é multiplicado por 37 ns (período de um *clock* de PCR) e plotado nos gráficos gerados pelo Matlab.

5.2 Medição do intervalo entre PCRs

O intervalo de PCR mede o tempo decorrido entre a chegada de dois PCRs consecutivos, o qual é calculado levando-se em conta a quantidade de bits transmitidos, entre 2 PCRs, e a taxa de bits de saída do TS. O valor do intervalo entre PCR é calculado como

$$intervalo = \frac{N\_bits}{Taxa\_TS}, \quad (5.8)$$

onde N_bits representa a quantidade de bits, entre 2 PCRs, e $Taxa\_TS$ contem a taxa de bits de saída, em *bits* por segundo. A norma DVB recomenda que o intervalo entre PCRs deve ser inferior a $40ms$, para o correto funcionamento dos receptores [12].

5.3 Medição do desvio de frequência

O desvio de frequência (*Frequency Offset* - FO) mostra a diferença entre a frequência nominal do sistema (27 MHz) e a frequência recuperada a partir do PCR. A norma DVB especifica que este erro deve ser menor que 810 Hz, ou 30 ppm [12].

A frequência é obtida a partir de um dado PCR, com

$$freq(i) = \frac{(PCR(i') - PCR(i'')) \times Taxa\_TS}{N\_bytes}. \tag{5.9}$$

Com isso, é possível calcular o erro com relação à frequência nominal (27 MHz), utilizando-se

$$desvio\_freq(i) = \frac{freq(i) - freq\_nom}{freq\_nom} \times 10^6, \tag{5.10}$$

onde $freq\_nom$ representa a frequência nominal de 27 MHz e $desvio\_freq$ contem o desvio de frequência, em partes por milhão (ppm).

5.4 Simulações

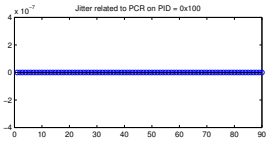
Alguns cenários de simulação foram verificados neste trabalho, com o intuito de se avaliar o nível de *jitter*, o intervalo entre PCRs e o desvio de frequência dos TSs gerados pelo adaptador de taxa, sob diferentes configurações de multiplexação. Em resumo, duas situações-chave foram avaliadas. Na primeira delas, utilizou-se um TS com *jitter* zero para se verificar o *jitter* inserido inerente a cada método. Em um segundo momento, utilizou-se um TS com 8 bases de tempo, com o objetivo de avaliar o comportamento dos algoritmos, quando há concorrência no acesso ao contador de 27 MHz, o que faz com que os semáforos sejam utilizados efetivamente, uma vez que um TS com poucas bases de tempo não é influenciado pelos semáforos e não há concorrência no acesso ao contador. No total, 4 segundos de TS foram simulados, para os Métodos de Compensação, Acumulador Controlado por Semáforo e Correção de PCR Integrada à Adaptação de Taxa, destacando-se que o último método nunca foi modelado em linguagem de descrição de hardware. Para cada cenário, apresentam-se três gráficos por programa contido no TS. O primeiro gráfico mostra o jitter de PCR, onde o eixo x representa um dado PCR e o eixo y mostra o *jitter* medido para este, onde o nível varia entre -400 e +400 ns. O segundo gráfico mostra o intervalo entre os PCRs, onde o eixo y varia entre -100 e 100 ms. O terceiro gráfico, por sua vez, mostra

PCRs no eixo x, sendo que o eixo y representa o desvio de frequência, numa escala que varia entre -30 e +30 ppm.

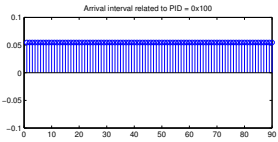
5.4.1 Cenário 1: Conversão para 48 Mbps com jitter de entrada em zero

Nesta seção, são apresentados os resultados de simulações funcionais, onde um TS com uma taxa de $32,5079365079$ Mbps e tamanho de pacote igual a 204 bytes (188 bytes de dados + 16 bytes de Reed-Solomon), com dois programas, tem sua taxa convertida para 48 Mbps. O primeiro programa está codificado em alta definição, com uma taxa de $15,524$ Mbps, e o segundo programa está em baixa definição, com uma taxa de 600 kbps.

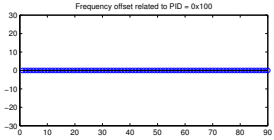
A Figura 39 apresenta o *jitter* de PCR, o intervalo entre PCRs e o desvio de frequência medidos no TS de entrada a partir do script MatLab, onde verifica-se que o *jitter* se mantem em zero durante toda a simulação, o intervalo entre PCRs é um pouco maior que 50 ms, e o desvio de frequência também se mantem em zero.



(a) *Jitter* de PCR



(b) Intervalo entre PCRs



(c) Desvio de Frequência

Figura 39 – Análise de *clock* do TS de entrada empregado no cenário 1.

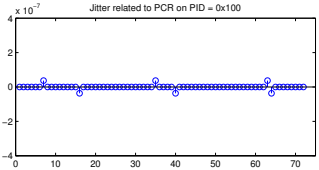
Conforme visto anteriormente, o TP_{saida} pode ser calculado conforme

$$TP_{saida} = \frac{8 \times 27MHz}{48Mbps} \times 204bytes = 918. \tag{5.11}$$

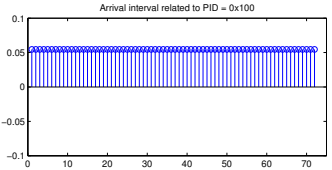
Como a parte fracionária no número é igual a zero, espera-se que, após a conversão para 48 Mbps, o *jitter* apresente valores bem baixos, pois o final de cada pacote deve coincidir com a borda de subida do *clock* de 27 MHz. Logo, o envio de pacotes e o acúmulo de PCR são síncronos a cada fim de pacote.

Este cenário é importante para provar que a conversão funciona corretamente e que os métodos não inserem *jitter* durante a conversão, quando o *clock* de 27 MHz é síncrono com o *clock* de saída do sistema.

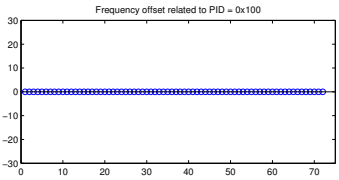
A Figura 40 mostra o *jitter* de PCR, intervalo entre PCRs, e desvio de frequência medido no TS, após a conversão, utilizando-se o Método da Compensação. É possível notar que o *jitter* se manteve, na maior parte do tempo, em zero, com alguns erros de 37 ns (o que equivale a um período de *clock*). O intervalo entre PCRs se manteve na faixa dos 55 ms e o desvio de frequência em zero.



(a) *Jitter* de PCR



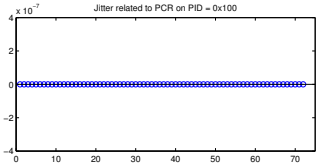
(b) Intervalo entre PCRs



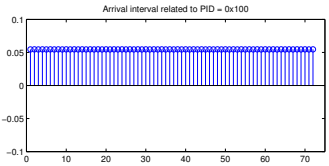
(c) Desvio de Frequência

Figura 40 – Análise de *clock* após conversão para 48 Mbps utilizando o Método da Compensação.

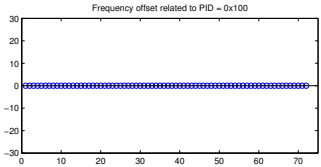
A Figura 41 mostra os mesmos dados após a conversão com o Método dos Acumuladores. Neste caso, o intervalo entre PCRs e o desvio de frequência foram similares, porém, o *jitter* se manteve fixo em zero.



(a) *Jitter* de PCR

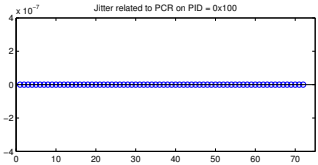


(b) Intervalo entre PCRs

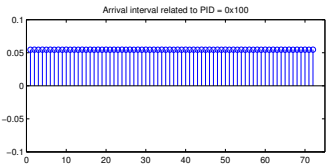


(c) Desvio de Frequência

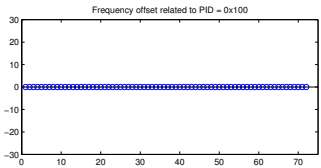
Figura 41 – Análise de *clock* após conversão para 48 Mbps utilizando o Método dos Acumuladores.



(a) *Jitter* de PCR



(b) Intervalo entre PCRs



(c) Desvio de Frequência

Figura 42 – Análise de *clock* após conversão para 48 Mbps utilizando o Método de Correção Integrada.

A Figura 42 apresenta os dados relativos à conversão utilizando o Método de Correção Integrada, onde é possível constatar que o jitter medido no TS, após a conversão de taxa, manteve-se nulo. O mesmo resultado foi obtido no Método dos Acumuladores (figura 41), porém, houve erros no Método da Compensação conforme mostrado na Figura 40.

Os erros identificados no Método da Compensação são justificados pela dependência do clock de 27 MHz, que insere erros de PCR se houver um mínimo escorregamento de borda. Isso acontece principalmente por conta de atrasos variáveis conhecidos como *delta delay*, ocasionados em ambientes de simulação, que se apresentam quando eventos diferentes ocorrem no mesmo tempo de simulação, o que permite a propagação de sinais atualizados num mesmo *clock*. No caso do método da compensação, que utiliza um terceiro *clock* de 27 MHz incrementado pelo contador de 42 bits, a cada borda de subida, se um novo pacote PCR for recebido no meio de um *clock* de 27 MHz, então ocorrerá um erro de 37 ns, o que justifica o comportamento identificado na Figura 40. Nos demais métodos, a cada nova recepção de PCR, o valor do contador é atualizado, que é sincronizado pelo *clock* de saída, juntamente com o envio do PCR. Com isso, os erros ocorridos no Método da Compensação são contornados.

O intervalo entre PCRs se manteve similar aos valores lidos no TS de entrada, ou seja, a cada 50 ms (aproximadamente) um novo PCR foi inserido no TS de saída.

Já o desvio de frequência se manteve em zero para todos os métodos, o que era esperado, levando-se em conta que o *jitter* também se manteve em zero nos três métodos avaliados neste cenário.

5.4.2 Cenário 2: Conversão para 43 Mbps com jitter de entrada em zero

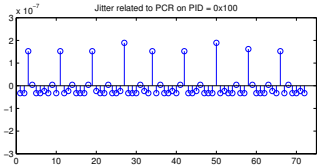
Nesta simulação, foi utilizado o mesmo TS do Cenário 1 (Figura 39), ou seja, um TS com taxa original de $32,5079365079$ Mbps. Entretanto, agora a conversão é feita para 43 Mbps. Considerando-se que o tamanho dos pacotes do TS é de 204, obtêm-se o TP_saida

$$TP_{saida} = \frac{8 \times 27MHz}{43Mbps} \times 204bytes = 1024,744. \tag{5.12}$$

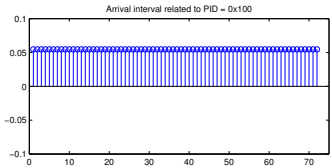
Como a parte fracionária de TP_{saida} é diferente de zero, é possível concluir que há inserção de *jitter* inerente ao processo de adaptação de taxa, conforme explanado anteriormente.

A Figura 43 apresenta o *jitter* para o Método da Compensação, onde é possível notar que os níveis se mantiveram abaixo dos 150 ns, com alguns

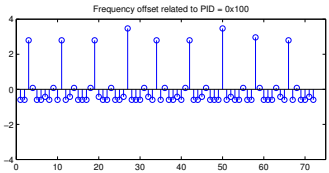
picos de aproximadamente 190 ns, o que está conforme as recomendações MPEG. Além disto, o intervalo entre PCRs se manteve constante e em aproximadamente 55 ms, valor aproximado ao medido no TS de entrada. O desvio de frequência se manteve abaixo de $3,5$ ppm.



(a) *Jitter* de PCR



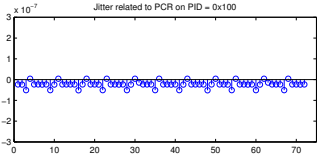
(b) Intervalo entre PCRs



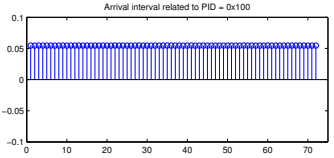
(c) Desvio de Frequência

Figura 43 – Análise de *clock* após conversão para 43 Mbps utilizando o Método da Compensação.

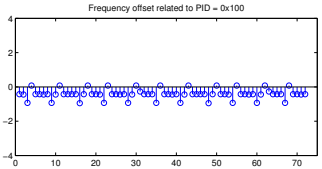
Na Figura 44, o *jitter* para o Método dos Acumuladores é mostrado, onde se verifica que os níveis se mantiveram em torno de 23 ns, ou seja, menos que um *clock* de erro. Porém, é possível visualizar que o TS gerado contém picos de PCR na ordem de -50 ns e, além disso, os erros são equidistantes, o que mostra que o erro é sistemático e inerente ao descasamento entre as bordas. Entretanto, esta solução também é adequada à utilização em sistemas de conversão de taxa. O intervalo entre PCRs foi de aproximadamente 50 ms e o desvio de frequência foi menor que 1 ppm.



(a) *Jitter* de PCR

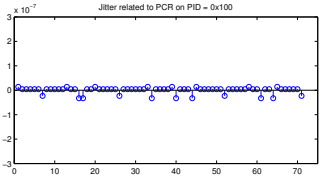


(b) Intervalo entre PCRs

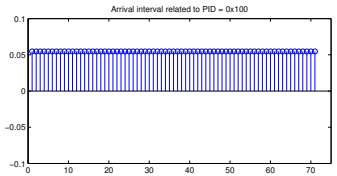


(c) Desvio de Frequência

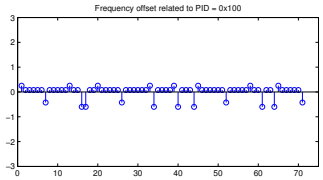
Figura 44 – Análise de *clock* após conversão para 43 Mbps utilizando o Método dos Acumuladores.



(a) *Jitter* de PCR



(b) Intervalo entre PCRs



(c) Desvio de Frequência

Figura 45 – Análise de *clock* após conversão para 43 Mbps utilizando o Método de Correção Integrada.

No Método de Correção Integrada (Figura 45), o *jitter* se manteve regular (sem picos) e abaixo de 33 ns, o que representa um erro inferior ao período de um *clock* (1 clock = 37 ns) e bem abaixo dos 500 ns especificados pela norma MPEG-2 [15]. Vale ressaltar também que tal resultado é melhor que os níveis apresentados nos trabalhos de Xingdong [32], He [33] e Chen [8]

Xingdong [32] implementa a correção de PCR em um remultiplexador e emprega o método da compensação para a correção de PCR, onde os detalhes do TS não são apresentados, porém o mesmo cita que foi utilizado um TS de entrada com *jitter* igual a zero e que o *jitter* de saída foi de aproximadamente 148 ns. He [33] utiliza correção de PCR em um demultiplexador, onde relata-se que foi utilizado um MPTS de entrada com *jitter* inferior a 37 ns, e obteve-se na saída programas com *jitter* de até 160 ns. Chen [8] apresenta informações relacionadas à implementação de um remultiplexador em FPGA, onde utiliza-se um corretor de PCR. O TS de entrada empregado tinha uma taxa de 26,97 Mbps com *jitter* variando entre 74 e -148 ns, e um segundo TS com taxa de 38,81 Mbps com jitter entre 111 e -148 ns. O TS gerado na saída do remultiplexador conteve erros de até dois clocks (74 ns).

Logo, é possível observar que o nível de *jitter* é consideravelmente menor que os valores informados em outros trabalhos, conforme pode ser verificado na Tabela 5.

O intervalo entre PCRs foi de aproximadamente 50 ms, valor similar ao medido no TS de entrada, e inferior aos 100 ms especificados pela norma [12].

O desvio de frequência foi menor que 1 ppm nos métodos dos acumuladores e correção integrada, o que representa uma melhor desempenho com relação ao método da compensação que teve picos de aproximadamente 4 ppm.

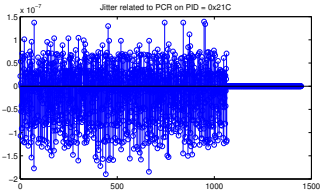
Tabela 5 – Níveis de Jitter após conversão para 43 Mbps.

Método	Jitter(Min:Max)
Compensação	-32,73ns : 189,49ns
Acumuladores	-51,68ns : 4,3ns
Correção Integrada	-32,73ns : 13,78ns

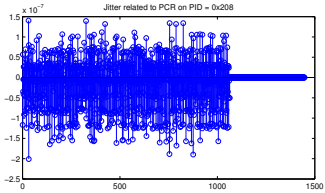
5.4.3 Cenário 3: Conversão para 37 Mbps com 8 bases de tempo

Este cenário tem como objetivo apresentar o desempenho dos métodos diante de um TS com múltiplas bases de tempo. Para isto, utilizou-se um TS com 14 programas e 8 bases de tempo, multiplexado a uma taxa de aproximadamente 24,1 Mbps e com um tamanho de pacote de 188 *bytes*.

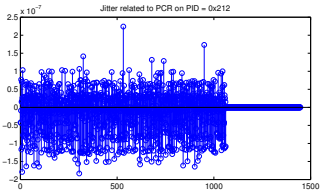
O *jitter* de PCR de cada uma das bases de tempo é exibido nas Figuras 46 e 47.



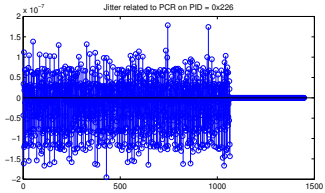
(a) *Jitter* de PCR para o PID 21Ch



(b) *Jitter* de PCR para o PID 208h

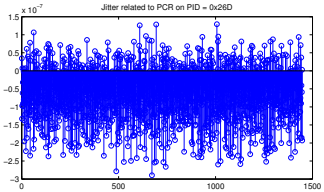


(c) *Jitter* de PCR para o PID 212h

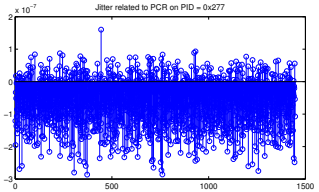


(d) *Jitter* de PCR para o PID 226h

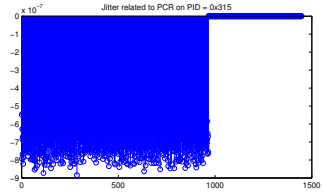
Figura 46 – Jitter de PCR para os programas 21Ch, 208h, 212h e 226h.



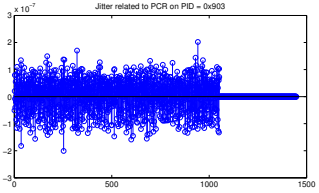
(a) *Jitter* de PCR para o PID 26Dh



(b) *Jitter* de PCR para o PID 277h



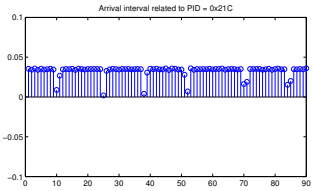
(c) *Jitter* de PCR para o PID 315h



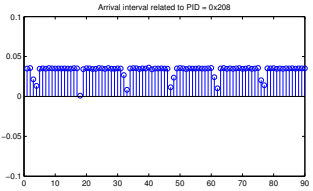
(d) *Jitter* de PCR para o PID 903h

Figura 47 – Jitter de PCR para os programas 26*D*h, 277h, 315h e 903h.

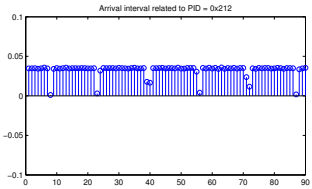
As Figuras 48 e 49 apresentam o intervalo entre PCRs medido para cada programa contido no TS de entrada.



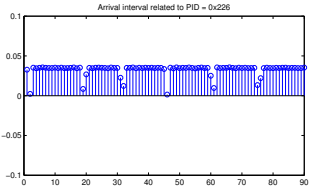
(a) Intervalo de PCR para o PID 21Ch



(b) Intervalo de PCR para o PID 208h

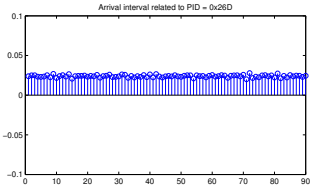


(c) Intervalo de PCR para o PID 212h

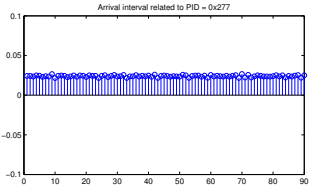


(d) Intervalo de PCR para o PID 226h

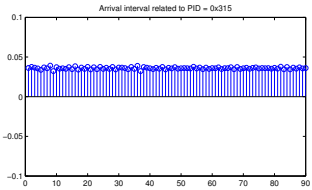
Figura 48 – Intervalo de PCR para os programas 21*C*h, 208h, 212h e 226h.



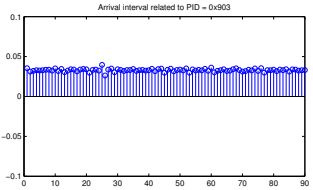
(a) Intervalo de PCR para o PID 26Dh



(b) Intervalo de PCR para o PID 277h



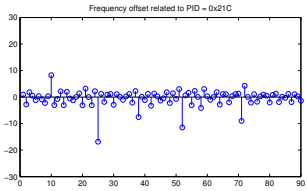
(c) Intervalo de PCR para o PID 315h



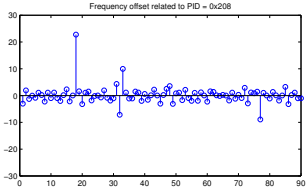
(d) Intervalo de PCR para o PID 903h

Figura 49 – Intervalo de PCR para os programas 26*D*h, 277h, 315h e 903h.

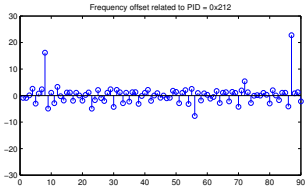
E as Figuras 50 e 51 apresentam o desvio de frequência medido no TS de entrada



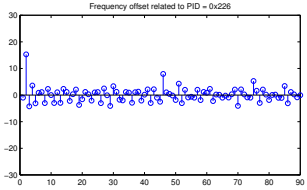
(a) Desvio de Frequência no PID 21Ch



(b) Desvio de Frequência no PID 208h

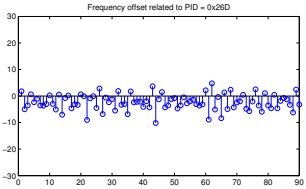


(c) Desvio de Frequência no PID 212h

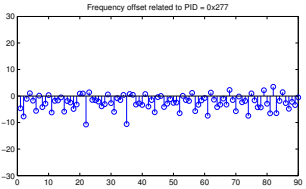


(d) Desvio de Frequência no PID 226h

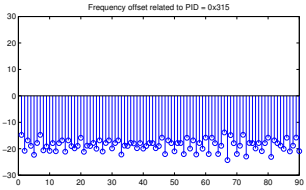
Figura 50 – Desvio de Frequência nos programas 21*C*h, 208h, 212h e 226h.



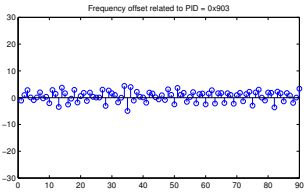
(a) Desvio de Frequência no PID 26Dh



(b) Desvio de Frequência no PID 277h



(c) Desvio de Frequência no PID 315h



(d) Desvio de Frequência no PID 903h

Figura 51 – Desvio de Frequência nos programas 26*D*h, 277h, 315h e 903h.

Pode-se observar que, no TS de entrada, todos os programas apresentaram *jitter* abaixo dos 500 ns especificados pela norma, exceto o programa cujo PID é 315h, o qual possui *jitter* na faixa de 900 ns. Isto permite verificar o comportamento dos métodos neste tipo de cenário. O intervalo entre PCRs se manteve pouco abaixo de 50 ms. E o desvio de frequência foi menor que 20 ppm na maior parte dos pontos.

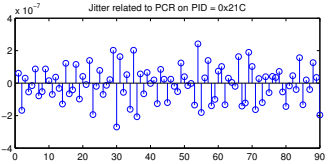
Neste cenário, a conversão é realizada para uma taxa de 34 Mbps, então obtêm-se o TP_{saida} como

$$TP_{saida} = \frac{8 \times 27MHz}{34Mbps} \times 188bytes = 1194,353. \tag{5.13}$$

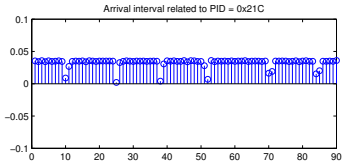
Mais uma vez, a parte fracionária do número obtido mostra que a conversão insere *jitter* no TS processado.

5.4.3.1 Desempenho do Método da Compensação

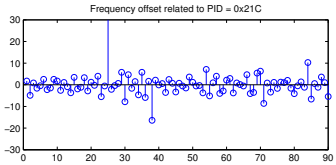
As Figuras apresentadas nesta seção mostram a análise de acurácia de *clock* para cada um dos programas contido no TS, após a conversão de taxa através do Método da Compensação.



(a) *Jitter* de PCR

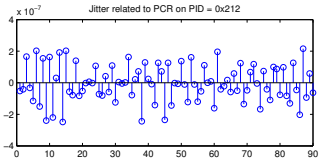


(b) Intervalo de PCR

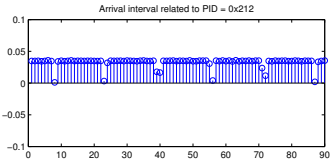


(c) Desvio de Frequência

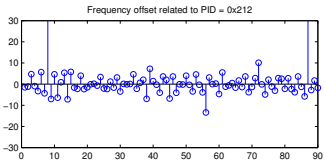
Figura 52 – Análise de Clock para o PID 21Ch



(a) *Jitter* de PCR

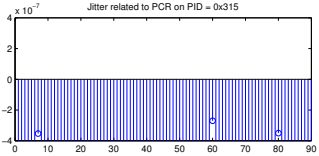


(b) Intervalo de PCR

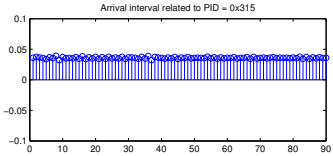


(c) Desvio de Frequência

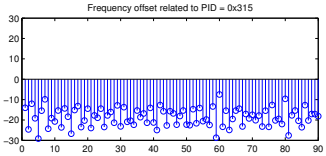
Figura 53 – Análise de Clock para o PID 212h



(a) *Jitter* de PCR

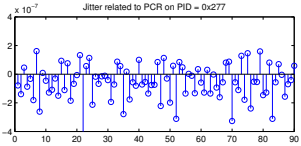


(b) Intervalo de PCR

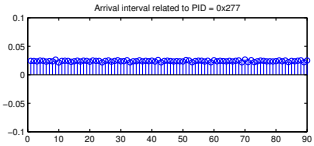


(c) Desvio de Frequência

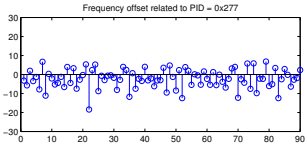
Figura 54 – Análise de Clock para o PID 315h



(a) *Jitter* de PCR

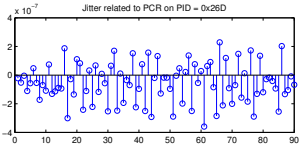


(b) Intervalo de PCR

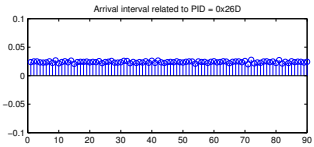


(c) Desvio de Frequência

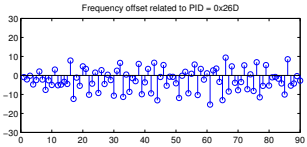
Figura 55 – Análise de Clock para o PID 277h



(a) *Jitter* de PCR

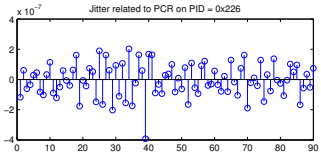


(b) Intervalo de PCR

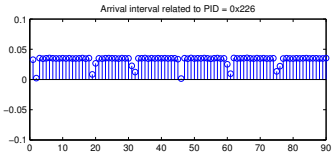


(c) Desvio de Frequência

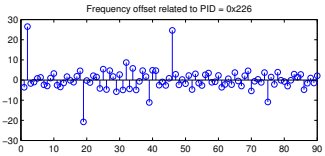
Figura 56 – Análise de Clock para o PID 26Dh



(a) *Jitter* de PCR

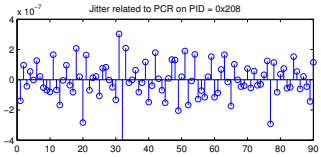


(b) Intervalo de PCR

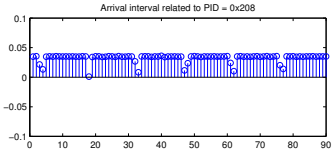


(c) Desvio de Frequência

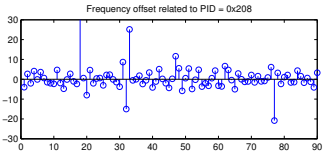
Figura 57 – Análise de Clock para o PID 226h



(a) *Jitter* de PCR

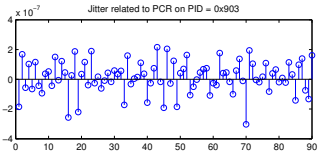


(b) Intervalo de PCR

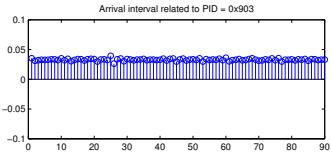


(c) Desvio de Frequência

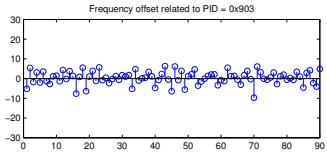
Figura 58 – Análise de Clock para o PID 208h



(a) *Jitter* de PCR



(b) Intervalo de PCR



(c) Desvio de Frequência

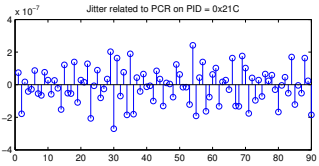
Figura 59 – Análise de Clock para o PID 903h

De maneira geral, os programas tiveram *jitter* inferior a 300 ns, intervalo entre PCRs menor que 50 ms, e desvio de frequência abaixo de 10 ppm.

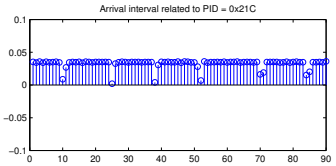
Na Figura 54 é possível notar que o programa com PID 315h teve *jitter* acima do limite especificado por norma por conta de erros no TS original. Porém, o intervalo entre PCRs se manteve abaixo dos 50 ms e o desvio de frequência esteve entre 0 e -30 ppm.

5.4.3.2 Desempenho do Método dos Acumuladores

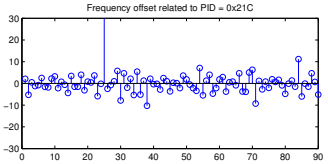
As Figuras apresentadas neste tópico mostram o resultado da conversão de taxa utilizando o Método dos Acumuladores.



(a) *Jitter* de PCR

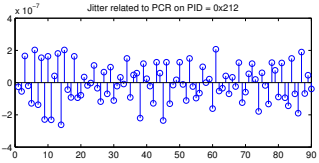


(b) Intervalo de PCR

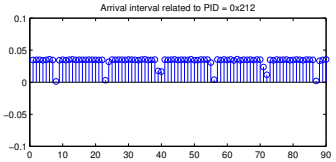


(c) Desvio de Frequência

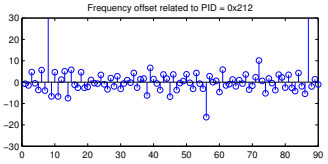
Figura 60 – Análise de Clock para o PID 21Ch



(a) *Jitter* de PCR

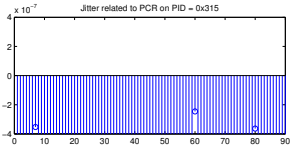


(b) Intervalo de PCR

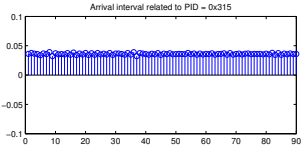


(c) Desvio de Frequência

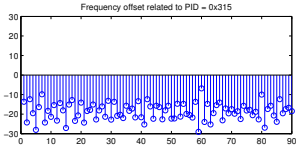
Figura 61 – Análise de Clock para o PID 212h



(a) *Jitter* de PCR

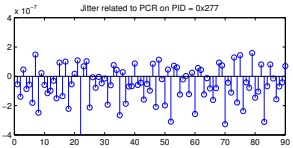


(b) Intervalo de PCR

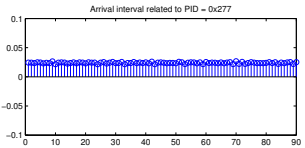


(c) Desvio de Frequência

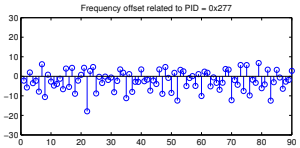
Figura 62 – Análise de Clock para o PID 315h



(a) *Jitter* de PCR

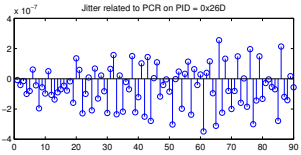


(b) Intervalo de PCR

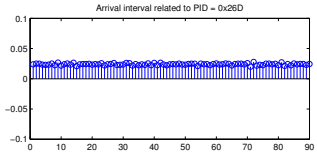


(c) Desvio de Frequência

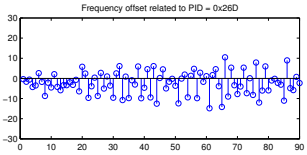
Figura 63 – Análise de Clock para o PID 277h



(a) *Jitter* de PCR

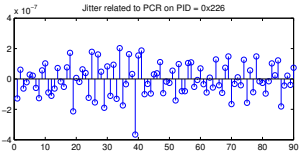


(b) Intervalo de PCR

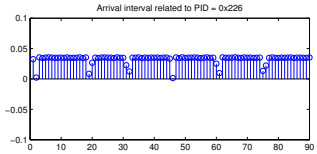


(c) Desvio de Frequência

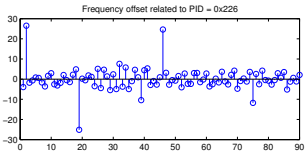
Figura 64 – Análise de Clock para o PID 26Dh



(a) *Jitter* de PCR

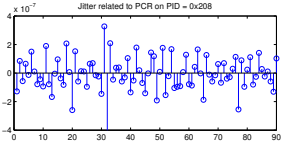


(b) Intervalo de PCR

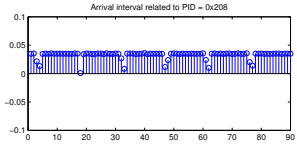


(c) Desvio de Frequência

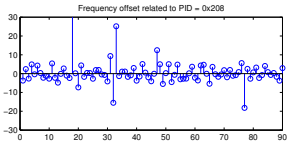
Figura 65 – Análise de Clock para o PID 226h



(a) *Jitter* de PCR

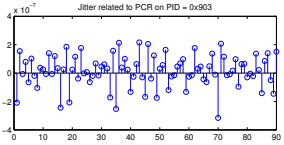


(b) Intervalo de PCR

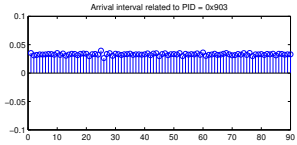


(c) Desvio de Frequência

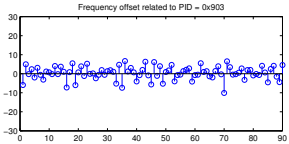
Figura 66 – Análise de Clock para o PID 208h



(a) *Jitter* de PCR



(b) Intervalo de PCR



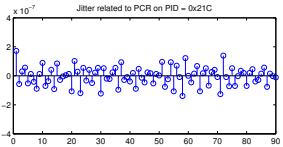
(c) Desvio de Frequência

Figura 67 – Análise de Clock para o PID 903h

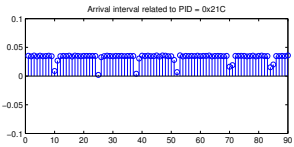
Na maioria dos programas o *jitter* se manteve menor que 200 ns, o intervalo entre PCRs foi inferior a 50 ms, e o desvio de frequência menor que 10 ppm. O desempenho do método dos acumuladores foi muito parecido com o obtido no método da compensação, com pequenos ganhos em parâmetros de alguns PIDs.

5.4.3.3 Desempenho do Método de Correção Integrada

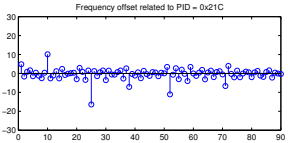
Nesta seção são apresentados os resultados de análise de *clock* do TS gerado após conversão de taxa utilizando o Método de Correção Integrada.



(a) *Jitter* de PCR

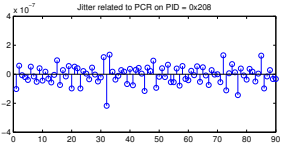


(b) Intervalo de PCR

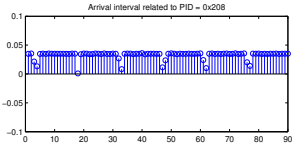


(c) Desvio de Frequência

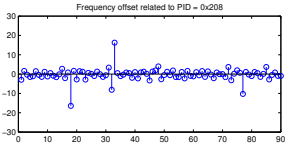
Figura 68 – Análise de Clock no PID 21Ch.



(a) *Jitter* de PCR

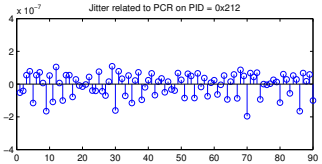


(b) Intervalo de PCR

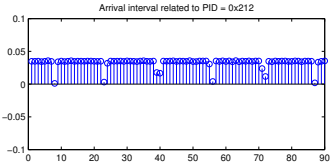


(c) Desvio de Frequência

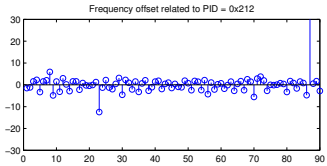
Figura 69 – Análise de Clock para o PID 208h.



(a) *Jitter* de PCR

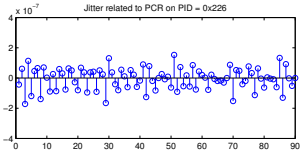


(b) Intervalo de PCR

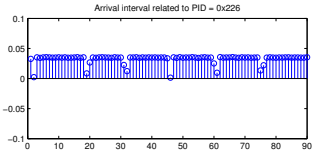


(c) Desvio de Frequência

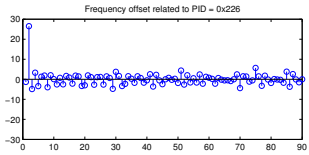
Figura 70 – Análise de Clock para o PID 212h.



(a) *Jitter* de PCR

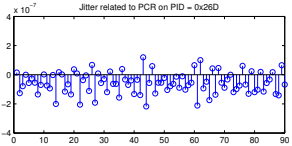


(b) Intervalo de PCR

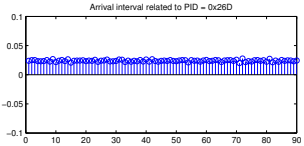


(c) Desvio de Frequência

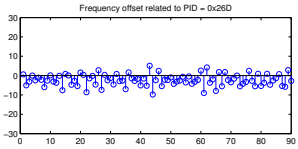
Figura 71 – Análise de Clock para o PID 226*h*.



(a) *Jitter* de PCR

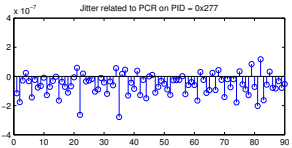


(b) Intervalo de PCR

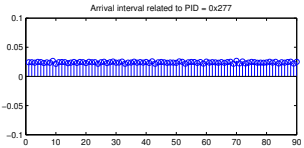


(c) Desvio de Frequência

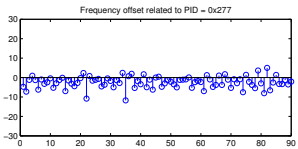
Figura 72 – Análise de Clock para o PID 26Dh.



(a) *Jitter* de PCR

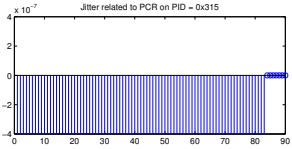


(b) Intervalo de PCR

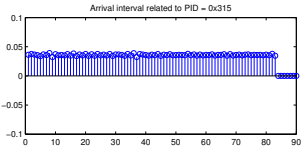


(c) Desvio de Frequência

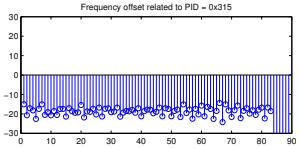
Figura 73 – Análise de Clock para o PID 277*h*.



(a) *Jitter* de PCR

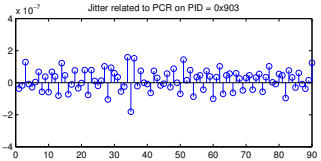


(b) Intervalo de PCR

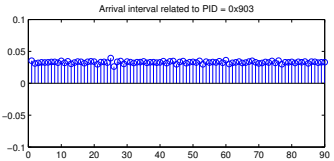


(c) Desvio de Frequência

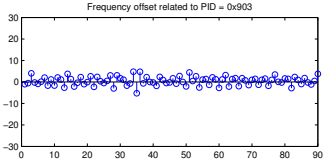
Figura 74 – Análise de Clock para o PID 315h.



(a) *Jitter* de PCR



(b) Intervalo de PCR



(c) Desvio de Frequência

Figura 75 – Análise de Clock para o PID 903h.

Na Figura 68 é possível notar que para o programa de PID 21*C*h o desvio de frequência esteve abaixo de 20 ppm e sem picos, diferentemente dos outros dois métodos onde houveram alguns picos para este programa.

No programa de PID 208h o *jitter* esteve abaixo de 200 ns sem a presença de picos encontrados nos demais métodos, o intervalo na ordem de 40 ms, e o desvio de frequência inferior a 5 ppm, onde picos também foram eliminados.

No programa de PID 226*h* todos os parâmetros mantiveram-se dentro dos limites especificados pela recomendação [12], e o mesmo ocorreu para o PID 26*Dh* conforme pode ser visualizado nas Figuras 71 e 72.

No PID 315*h* (Figura 74) observou-se que apesar do *jitter* se manter fora do intervalo válido na maior parte dos PCRs, o desvio de frequência se manteve abaixo de 30 ppm; e o intervalo entre PCRs foi inferior a 50 ms;

Os programas de PID 21*C*h e 903*h* se destacaram positivamente pois apresentaram os menores níveis de *jitter*, desvio de frequência e intervalo entre PCRs.

5.4.3.4 Análise de Desempenho dos Métodos na conversão para 34 Mbps

De maneira geral, o desempenho no que diz respeito a níveis de *jitter*, desvio de frequência e intervalo entre PCRs, nos Métodos por Compensação e Acumulador Controlado por Semáforo foram muito parecidos. Ou seja, o uso de semáforos eliminou a necessidade de operações aritméticas simplificando o processador, e ainda manteve um *clock* tão bom quanto um dos algoritmos clássicos da literatura, comumente encontrado em processadores de TS. A eliminação destas operações faz com que as restrições temporais sejam alcançados de maneira mais suave, atrasos gerados pela espera de resultados aritméticos são eliminados na entrada e na saída nos casos em que o semáforo estiver livre.

O Método de Correção integrada se mostrou mais eficaz conseguindo manter o *jitter* no mesmo nível da entrada, e mostrando uma performance melhor que os outros dois métodos implementados, conforme demonstrado na Tabela 6. Além disto os níveis de desvio de frequência também foram menores, e o intervalo entre PCRs manteve-se sempre abaixo dos 50 ms. Esta melhor performance se deve principalmente à inteligência agregada neste método que permite o envio dos pacotes com PCR no momento mais apropriados de modo a se manter o nível de *jitter* baixo, permitindo a correta geração dos *clocks* na recepção.

A Tabela 6 apresenta uma visão dos níveis de *jitter* em cada programada contido no MPTS empregado no cenário 3.

Tabela 6 – Níveis de Jitter após conversão para 34 Mbps.

Método	Jitter Máximo
PID 0x21C	
Compensação	-270,15
Acumulador	278,87
Correção Integrada	172,11
PID 0x208	
Compensação	-403,05
Acumulador	-413,94
Correção Integrada	-217,86
PID 0x212	
Compensação	263,62
Acumulador	-261,44
Correção Integrada	-196,08
PID 0x226	
Compensação	-392,16
Acumulador	-366,01
Correção Integrada	-172,11
PID 0x26D	
Compensação	-376,91
Acumulador	-348,58
Correção Integrada	-217,86
PID 0x277	
Compensação	-448,8
Acumulador	-435,73
Correção Integrada	-278,87
PID 0x315	
Compensação	-1008,7
Acumulador	-1021,8
Correção Integrada	-840,96
PID 0x903	
Compensação	-355,12
Acumulador	-344,23
Correção Integrada	-180,83

5.5 Alocação de Unidades Lógicas

Os trabalhos de correção de PCR em FPGA, em sua maioria, avaliam a quantidade de unidades lógicas alocadas para a síntese do algoritmo de correção.

A Tabela 7 mostra a quantidade de unidades lógicas alocadas por esquema, onde é possível verificar que o Quartus foi capaz de sintetizar os métodos com uma pequena quantidade de unidades lógicas, considerando que o plataforma-alvo FPGA (Cyclone IVE EP4CE115F29C7) tem 114.480 blocos disponíveis. O método de correção integrada (o mais complexo entre os métodos implementados), que apresentou o maior valor, ocupou somente 6176 unidades, o que equivale a menos que 6% do total de unidades disponíveis. Como consequência, pode-se dizer que o número de unidades lógicas deixaram de ser um problema na maioria dos FPGAs disponíveis, no que diz respeito a implementação de algoritmos de correção de PCR, uma vez que os recursos disponíveis são suficientes para o desenvolvimento dos esquemas existentes, sem maiores restrições de espaço.

Tabela 7 – Quantidade de unidades lógicas alocadas por método de correção.

Método	Número de Unidades Lógicas
Compensação [20]	406
Contador Controlado por Semáforo [25]	1209
Acumuladores [21]	5365
Correção Integrada [26]	6176

O método de correção integrada agrega características do método de compensação e do contador controlado por semáforos. Sendo assim, poder-se-ia questionar o ganho do uso desta técnica, uma vez que o bloco que implementa o método da compensação (a parte do circuito não-dependente de semáforos) já corrige o PCR de maneira satisfatória e uma possível eliminação do bloco dependente de semáforos reduziria a quantidade de unidades lógicas. Em resumo, o circuito de semáforo não ocasionaria ganho, dado que a economia de operações aritméticas não teria impacto na arquitetura de FPGA, dado que o circuito precisa estar presente (o que é diferente de uma implementação em software).

Entretanto, vale ressaltar que o uso de semáforos reduz a complexidade do processamento, pois a carga de um número de 42 bits consome menos energia do que repetidas operações de soma e subtração, com números de 42 bits. A redução do consumo de energia é um fator importante que deve ser levado em conta, uma vez que, os FPGAs estão cada vez mais pre-

sentes em sistemas embarcados, onde se emprega uma bateria como fonte de alimentação. De fato, a arquitetura proposta foi criada para ser utilizada em conversores de interface, que são muito empregados em aparelhos de TV digital, sendo estes muitas vezes energizados por baterias.

6 Conclusão

Este trabalho apresenta uma metodologia alternativa para a correção de erros de PCR, em adaptadores de taxa, baseada no Método de Correção de PCR Integrada à Adaptação de Taxa [26]. Um dos principais ganhos desta alternativa foi a eliminação da necessidade de um oscilador dedicado, o que mitigou os efeitos de metaestabilidade oriundos do assincronismo entre o clock de PCR e o clock de saída. O esquema apresentado foi capaz de combinar características dos Métodos de Adaptação Integrada à Correção de PCR e Contador Controlado por Semáforos, onde o cálculo de PCR foi simplificado, empregando-se algumas operações aritméticas em ponto fixo. A análise de *jitter* demonstrou que o algoritmo é válido, uma vez que os níveis de *jitter*, intervalo entre PCRs e desvios de frequência estão de acordo com os requisitos impostos pelo padrão MPEG-2. A vantagem da proposta é que esta é mais apropriada ao domínio de FPGAs, uma vez que o acesso ao acumulador é gerenciado e otimizado.

Apesar de já existirem alguns trabalhos na literatura que implementam algoritmos de correção de PCR em FPGA, somente um destes apresenta detalhes da arquitetura do sistema implementado. Neste trabalho, todos os módulos que compõem o sistema são detalhados, o que pode servir como base para outros trabalhos relacionados à adaptação de taxa e correção de PCR, em FPGA. A síntese destes sistemas permitiu a verificação e comparação da quantidade de unidades lógicas alocadas, para os dois métodos baseados em semáforos, além de compará-los às demais metodologias descritas neste trabalho.

Os trabalhos de Savino e Filho [25, 26], os quais são baseados em semáforos, nunca foram implementados em sistemas reais (software ou hardware). Neste trabalho, implementou-se a primeira solução em VHDL do Métodos de Correção Integrada, o que permitiu uma comparação com os demais sistemas.

Um dos principais gargalos na implementação de processadores de TS em hardware está no *buffer*. Neste trabalho, uma solução pronta, implementada em forma de IP, foi apresentada, da qual é possível aproveitar código VHDL e poupar tempo de desenvolvimento, através de uma solução pré-testada e fornecida pelo próprio fabricante do chip FPGA, a qual pôde atender aos requisitos do projeto de maneira satisfatória.

O Método de Correção Integrada conseguiu obter níveis de jitter até 46% menores que os encontrados nos Métodos dos Acumuladores e Compensação, o que o torna uma melhor opção principalmente em aplicações que exigem um *clock* de sistema extremamente estável. Apesar de apresentar

menores níveis de *jitter*, este novo método teve uma ocupação de unidades lógicas 15 vezes maior que o método da compensação, que apresentou a menor quantidade de unidades lógicas: uma ocupação inferior a 6% no FPGA Altera DE2-115. Isso pode ser um problema se o FPGA adotado contiver poucas unidades lógicas.

Outro ponto importante mostrado nas simulações foi que o uso de semáforos, no Método da Adaptação de Taxa Integrada à Correção de PCR, foi capaz de reduzir consideravelmente o número de operações aritméticas realizadas, quando comparado ao Método da Compensação. A eliminação dessas operações faz com que o processamento seja simplificado, uma vez que estas são substituídas por atribuições com números de 42 bits. Esta simplificação leva a uma redução no consumo de energia, o que é muito importante ao considerar-se que os FPGAs são comumente empregados em sistemas embarcados que utilizam bateria como alimentação.

Apesar do poder computacional dos FPGAs ter evoluído consideravelmente, nos últimos anos, ainda há dificuldades para se interfacear o FPGA com equipamentos de TV Digital. Este tipo de equipamento costumeiramente emprega interfaces do tipo SPI, o qual não é facilmente encontrada nas placas de desenvolvimento comercializadas pelos principais fabricantes de FPGA. Diante dessa barreira, abordou-se a execução de testes *offline*, através do uso de arquivo TS armazenados em memória RAM da placa. Então, o segundo obstáculo veio a tona, que foi a capacidade de armazenamento das memórias empregadas na placa. Um TS com poucos segundos de vídeo, em alta definição, facilmente ocupa mais do que 100 MB de tamanho. Além disso, após um processo de adaptação de taxa, o valor resultante pode ir além de 150 MB (dependendo da taxa de conversão), sendo que, o FPGA utilizado possui apenas 128 MB de espaço. Diante destes fatos, optou-se pela apresentação do resultado de simulações funcionais do sistema.

Mesmo que neste trabalho não se tenha atingido a execução na placa de FPGA, as contribuições foram significativas, pois detalhou-se uma arquitetura em VHDL que implementa o algoritmo de Correção Integrada, onde obteve-se níveis de *jitter* no mesmo patamar do trabalho de Savino [26, 25]. Além disso, a solução foi otimizada, de modo a se adequar ao domínio de FPGAs. O grande gargalo para execução no FPGA foi a falta de ferramentas de depuração e interfaces para se verificar o comportamento do processador, juntamente a outros equipamentos de aferição, ou até mesmo equipamentos de TV Digital.

Como sugestões para trabalhos futuros, propõe-se a adequação deste sistema para a execução em FPGAs com uma maior quantidade de memória, ou interfaces que possam se comunicar com equipamentos de radiodifusão, de modo a validar o sistema em um ambiente real. Para isso, pode-se verificar

uma solução baseada em software executado no processador *softcore* Nios II, que permitiria o envio de dados através da interface de rede ou armazenamento do TS processado, em cartões SD. Esta mesma solução de correção de PCR pode ser empregada em outras aplicações, como remultiplexadores, demultiplexadores, conversor de interface, etc.

APÊNDICE A

LISTA DE PUBLICAÇÕES

Artigo completo publicado em congresso internacional:

- Farias, B.; Filho, E. B. L. e Bezerra, E. A. A Proposal for Clock Reference Correction in MPEG-2 Transport Stream Processors. *ITS 2014 - International Telecommunications Symposium, São Paulo*, 2014

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ABNT/NBR 15601. *Televisão Digital Terrestre – Sistema de transmissão*. Brasil, 1a edição, 2007.

[2] ABNT/NBR 15602:3. *Televisão Digital Terrestre – Codificação de vídeo, áudio e multiplexação – Parte 2: Sistemas de multiplexação de sinais*. Brasil, 1a edição, 2007.

[3] Altera. *DE2-115 Manual*. Junho 2009.

[4] Altera. Fpgas. Website, Dezembro 2009. URL: `http://www.altera.com/products/fpga.html`.

[5] Altera. *High Speed Mezzanine Card (HSMC) Specification*. Junho 2009.

[6] Altera. *SCFIFO and DCFIFO Megafunctions*. Fevereiro 2012.

[7] David Bishop. *Fixed Point Package User's Guide*.

[8] Ying-Pei Chen e Tsan-Nan Chien. An Agile and Low Cost FPGA Implementation of MPEG-2 TS Remultiplexer for CATV Head-end Equipment. *International Symphosium on Pervasive Systems, Algorithms, and Networks*, October 2009.

[9] Andre Marcelo Coelho da Silva. Um Estudo Sobre o Padrao H.264/AVC de Compressao de Video.

[10] Digital Video Broadcasting (DVB). *Interfaces for CATV/SMATV Headends and Similar Professional Equipment*. 1997.

[11] ETSI EN 300 468. *Digital Video Broadcasting (DVB); Specification for Service Information(SI) in DVB Systems*, Abril 2010.

[12] ETSI TR 101 290. *Digital Video Broadcasting (DVB): Measurement guidelines for DVB Systems*. 2001.

[13] Walter Fischer. *Digital Video and Audio Broadcasting Technology*. Springer, 3a edição, Abril 2010.

[14] C. Harper H. Jones. *Handbook of Components for Electronics*. McGraw-Hill, 2a edição.

[15] ISO/IEC 13818-1, Switzerland. *Information Technology - Generic Coding of Moving Pictures and Associated Audio Information: Systems*, Outubro 2007.

[16] ITU-T. *Recomendação H.222 Information technology - Generic Coding of Moving Pictures and Associated Audio Information: Systems.* 2000.

[17] ITU-T. *Recomendação H.264 Advanced Video Coding for Generic Audiovisual Services.* 2012.

[18] Gaganpreet Kaur. *VHDL: Basics to Programming*. Pearson Education India, 1a edição, Fevereiro 2011.

[19] Soo In Lee, Sung Bae Cho, Jae Han Kim, Hyun Ho Jeon, e Deock Gil Oh. Implementation of MPEG-2 TS Remultiplexer and Data Transport Unit for HDTV Satellite Broadcasting. *IEEE Transactions on Broadcasting, vol 48, pp. 348-352*, Dezembro 2002.

[20] Liang Longfei, Yu Songyu, e Wang Xingdong. Implementation of a New MPEG-2 Transport Stream Processor for Digital Television Broadcasting. *IEEE Transactions on Broadcasting, vol 48, pp. 348-352*, Dezembro 2002.

[21] M. Machmerth e C. Stoerte. Accumulator Based PCR Restamping. *IEEE International Symposium on Consumer Electronics*, Maio 2009.

[22] Clive Maxfield. *FPGAs: Instant Access*. Newnes, 2a edição, Abril 2011.

[23] Xi Peng, Jingtao Song, e Kuang Wang. Counter-set based PCR Jitter Correction Method for DVB-T System. *Wireless Communications, Networking and Mobile Computing, Intern. Conf. 21-25, pp. 2940-2943*, Setembro 2007.

[24] Ulrich Reimers. *DVB - The Family of International Standards for Digital Video Broadcasting*. Springer, 2a edição, 2005.

[25] Heitor J. Savino e Eddie B. L. Filho. Um Contador Controlado por Semáforos Para a Correção de PCR. *XXX Simpósio Brasileiro de Telecomunicações - SBRT*, Setembro 2012.

[26] Heitor J. Savino e Eddie B. L. Filho. A Framework for Adaptative PCR Jitter Correction in MPEG-2 TS Processors. *XXXI Simpósio Brasileiro de Telecomunicações - SBrT2013*, Setembro 2013.

[27] Heitor Judiss Savino. Correção de PCR em Processadores de Fluxos de Transporte MPEG-2. Dissertação de Mestrado: Faculdade de Tecnologia, UFAM, Manaus, 2012.

[28] Gina Smith. *FPGAs 101*. Newnes, 12a edição, Fevereiro 2010.

[29] Luiz Fernando Gomes Soares e Simone Diniz Junqueira Barbosa. Programando em NCL 3.0. Desenvolvimento de Aplicaçoes para o Middleware Ginga. TV Digital e Web. PUC-Rio, 2a edição, 2012.

[30] Jennifer Stephenson, Doris Chen, Ryan Fung, e Jeffrey Chromczak. Understanding Metastability in FPGAs. Technical report, Altera Corporation, 2009.

[31] C. Tryfonas e A. Varma. Timestamping Schemes for MPEG-2 System Layer and Their Effect on Receiver Clock Recovery. *IEEE Transactions on Multimedia*, vol.1(num.3):251–263, Setembro 1999.

[32] Y. Songyu W. Xingdong e L. Longfei. Implementation of MPEG-2 Transport Stream Remultiplexer for DTV Broadcasting. *IEEE Trans. Consumer Electron., vol. 48, no. 2, pp. 329-334*, Maio 2002.

[33] J. Zhou Y. He e Y. Zhou. Implementation of TS De-multiplexer with FPGA in DVB_IP Gateway for Network TV. *IEEE Workshop on Multimedia Signal Processing*, pages 1–4, Out 2005.