

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

Eberle Andrey Rambo

Projeto de um IP para aplicação em telefonia móvel

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

Prof. Luiz Cláudio Villar dos Santos, Dr. (Orientador)

Florianópolis, Novembro de 2008

Projeto de um IP para aplicação em telefonia móvel

Eberle Andrey Rambo

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Ciência da Computação e aprovada em 04 de Novembro de 2008 em Florianópolis, Santa Catarina.

Banca Examinadora

Prof. Luiz Cláudio Villar dos Santos, Dr. (Orientador)

Prof. José Luís Almada Güntzel, Dr.

Prof. Olinto José Varela Furtado, Dr.

Sumário

Sumário	iii
Lista de figuras	v
Lista de tabelas	vi
Lista de acrônimos	vii
Resumo	viii
Abstract	ix
1 Introdução	1
1.1 Contexto tecnológico	1
1.2 Aplicações contemporâneas amparadas por SoCs	3
1.3 Escopo e contribuição técnica	4
1.4 Organização da monografia	5
2 Paradigmas contemporâneos de projeto	6
2.1 Projeto baseado em plataforma	6
2.2 Modelagem baseada em transações (TLM)	7
2.3 Linguagens de descrição de sistemas	8
3 Fundamentos da aplicação alvo	9
3.1 Breve histórico e futuro do padrão GSM	9
3.2 Breve descrição do processo de codificação de áudio	9
4 Metodologia e infra-estrutura	11
4.1 Metodologia de projeto e validação	11

4.2	Infra-estrutura de modelagem da plataforma	11
4.2.1	Mapeamento da aplicação para a plataforma	12
5	Particionamento hardware-software e modelagem funcional do IP	14
5.1	Seleção de funcionalidade para o IP	14
5.1.1	“Profiling” e escolha	14
5.2	Modelagem Funcional do IP na Plataforma TLM	15
5.2.1	Interface de Entrada/Saída do IP	16
5.2.2	Acesso ao IP por E/S Mapeada em Memória	16
5.2.3	Utilização como Modelo de Validação	17
5.3	Resultados experimentais	17
5.3.1	Configuração	17
5.3.2	Validação do IP Funcional na Plataforma	18
5.3.3	Resultados	18
6	Modelagem RTL do IP	20
6.1	Proposta de uma micro-arquitetura	20
6.2	Codificação RTL do IP	22
6.3	Validação e resultados experimentais	23
7	Conclusão	25
7.1	Trabalhos Futuros	25
A	Código da Análise LPC	26
	Referências bibliográficas	37

Lista de figuras

1.1	Aumento de produtividade com o uso de IPs	2
3.1	Algoritmo de compressão GSM	10
4.1	Fluxo de validação	12
5.1	Árvore de chamadas simplificada	15
5.2	Visão geral da plataforma	16
5.3	Módulo <i>Gsm_LPC_Analysis</i>	17
6.1	Arquitetura RTL do IP	20
6.2	Diagrama de blocos	21
6.3	Módulo <i>autocorrelation</i>	22
6.4	Módulo <i>reflection_coefficients</i>	22
6.5	Fluxo de validação do modelo RTL	24

Lista de tabelas

5.1	Resultado dos <i>profilings</i>	15
5.2	Registradores mapeados em memória e seus endereços relativos	17
5.3	Resultados das execuções da plataforma com e sem IP	19
6.1	Sumário de componentes do IP para o StratixII EP2S15F484C3	23

Lista de acrônimos

APCM	<i>Adaptative PCM</i>
ARP	<i>ArchC Reference Platform</i>
ASIC	<i>Application-Specific Integrated Circuit</i>
CI	<i>Circuito integrado</i>
CMOS	<i>Complementary Metal Oxide Semiconductor</i>
DUV	<i>Device Under Verification</i>
ESL	<i>Electronic System Level</i>
EDA	<i>Electronic Design Automation</i>
FDMA	<i>Frequency-Division Multiple Access</i>
FPGA	<i>Field-Programmable Gate Array</i>
GSM	<i>Global System for Mobile Communication</i>
HDL	<i>Hardware Description Language</i>
HDS	<i>Hardware Dependent Software</i>
IP	<i>Intellectual Property Blocks</i>
LAR	<i>Log Area Ratios</i>
LPC	<i>Linear Predictive Coding</i>
LSP	<i>Line Spectral Pairs</i>
PBD	<i>Platform-based Design</i>
PCM	<i>Pulse-Code Modulation</i>
RGM	<i>Reference Golden Model</i>
RTL	<i>Register Transfer Level</i>
SoC	<i>System-on-Chip</i>
TDMA	<i>Time-Division Multiple Access</i>
TLM	<i>Transaction-Level Modeling</i>
VHDL	<i>Very High Speed Integrated Circuit Hardware Description Language</i>
VLSI	<i>Very Large Scale Integration</i>

Resumo

A crescente densidade de transistores que podem ser fabricados em um chip de silício e a demanda imposta por aplicações cada vez mais complexas dentro da indústria de sistemas embarcados resultaram no surgimento de sistemas inteiros em um único chip, chamados *System-on-chip* (SoCs), que agregam tanto o hardware quanto o software necessários para garantir a funcionalidade do sistema.

Para reduzir o custo de desenvolvimento e acomodar o *time-to-market*, o projeto de SoCs tende a adotar uma plataforma-alvo, ou seja, uma arquitetura de referência com a finalidade de viabilizar o reuso de componentes, chamados de blocos de propriedade intelectual (IPs). Tal plataforma pode então ser derivada para uma aplicação específica, através da inclusão e/ou remoção de IPs desnecessários.

Este trabalho apresenta a modelagem de um IP para aplicação em telefonia móvel. O IP implementa a Análise LPC (*linear predictive coding*), extensamente utilizada na codificação de áudio em geral, selecionada através do “profiling” do codificador GSM. São apresentados dois modelos, um puramente funcional em SystemC e outro com precisão de ciclos em VHDL, e seus resultados experimentais. O desenvolvimento destes modelos visam a posterior prototipação em FPGA e em silício.

Abstract

The growing number of transistors available in a silicon chip and the demand imposed by embedded systems applications becoming more and more complex resulted in the Systems-on-chip (SoCs), a whole system inside a chip, holding both software and hardware required to guarantee the system's functionality.

And to reduce the non-recurrent engineering costs and the time-to-market, begun the Platform-based Design, relying on reference architectures and component reuse, called Intellectual Property Blocks (IPs). The platform then can be derived to a specific application by including and/or removing unnecessary IPs.

This work presents an IP modeling for mobile communications. The IP implements the LPC Analysis (Linear Predictive Coding), widely used in audio coding, selected after a profiling of the GSM coder. Two models are presented, one purely functional in SystemC and the other cycle accurate in VHDL, and their experimental results. These models design is aimed at FPGA and silicon prototyping.

Capítulo 1

Introdução

Este capítulo situa a contribuição técnica reportada nesta monografia, justificando sua relevância no contexto tecnológico de computação embarcada e no âmbito de aplicações contemporâneas.

1.1 Contexto tecnológico

Tipicamente, um sistema embarcado inclui pelo menos um processador, permitindo que grande parte de sua funcionalidade seja realizada através de software embarcado. Entretanto, algumas funções críticas do sistema costumam requerer implementação direta em hardware, por razões de desempenho ou redução de consumo de energia ou potência.

Até cerca de 15 anos atrás, a implementação de funções críticas em hardware era feita através de *circuitos integrados dedicados* ou ASICs (*Application-Specific Integrated Circuit*). Nessa época, um sistema embarcado típico compunha-se de vários circuitos integrados (CIs) distintos: um microprocessador, CIs de memória e periféricos e um ou mais ASICs.

O projeto de um ASIC consistia basicamente em se obter uma descrição do circuito dedicado no *nível de transferência entre registradores* ou RTL (*Register Transfer Level*), obtida através de uma linguagem de descrição de hardware ou HDL (*Hardware Description Language*), tal como VHDL [ASH 02] ou Verilog [THO 02]. A partir de uma descrição RTL, a implementação em VLSI (*Very Large Scale Integration*) era obtida através de ferramentas de síntese automática.

Na primeira metade da década de 1990, quando as dimensões dos transistores CMOS atingiram a faixa de 350 a 250 nm, tornou-se possível agrupar a maior parte dos blocos de processamento (e alguns de armazenamento) em uma única pastilha de silício [MAR 03], dando origem aos *siste-*

mas integrados ou SoCs (*Systems-on-Chip*).

SoC é um sistema que integra distintos componentes eletrônicos em um único chip, formando um sistema eletrônico completo que inclui não somente o hardware mas também o software nele embarcado [GHE 05]. Em um SoC, as funções críticas do sistema embarcado são implementadas na forma de blocos dedicados. Como podem ser desenvolvidos por terceiros e adquiridos sob licença, os blocos dedicados são conhecidos como *blocos de propriedade intelectual* ou IPs (*Intellectual Property Blocks*).

O alto custo das máscaras de circuitos integrados em tecnologias avançadas e a necessidade de garantir um *time-to-market* adequado mesmo diante do aumento da complexidade dos SoCs, levou à necessidade de reuso de hardware e de software, o que culminou num novo paradigma de projeto amparado na noção de plataforma.

Uma plataforma é uma abstração no fluxo de projeto que facilita um número de possíveis refinamentos em uma camada de abstração subsequente no fluxo de desenvolvimento. Ou seja, uma biblioteca de componentes junto com as suas regras de composição, contendo blocos que realizam a computação e blocos de interconexão.

O paradigma de projeto orientado a plataforma [SV 01] deu ao projeto do hardware do SoC um nível de produtividade adequado, viabilizando sistemas complexos através do reuso da maior parte de seus componentes, reduzindo assim o projeto do sistema ao projeto de uns poucos IPs. O projeto do hardware tornou-se assim a "montagem" dos componentes de acordo com uma arquitetura de referência [BER 02]. Isto pode ser notado no gráfico da Figura 1.1, onde uma maior produtividade e a complexidade desejada são alcançadas através do uso de IPs.

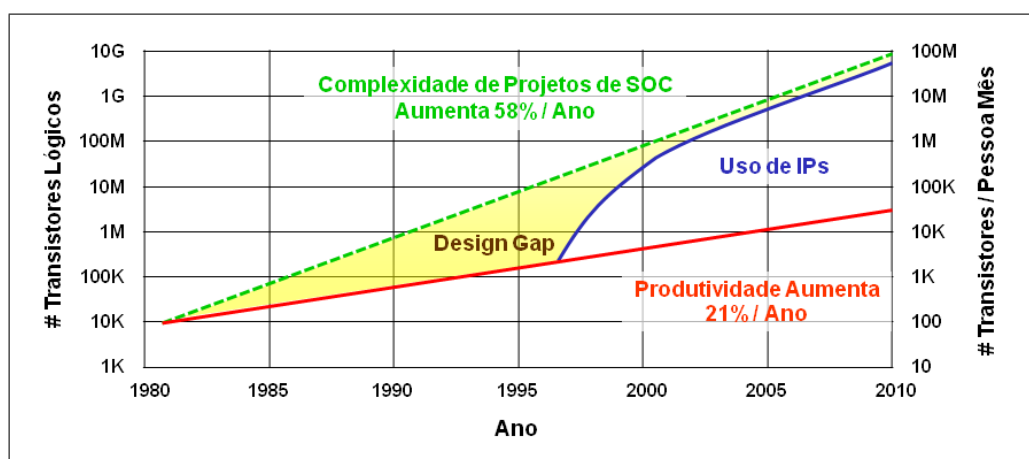


Figura 1.1: Aumento de produtividade com o uso de IPs

Com a maior complexidade dos SoCs (quando comparada aos ASICs), surgiu a necessidade

de se elevar o nível de abstração inicial do fluxo de projeto. Ora, não somente uma descrição RTL de todo o sistema integrado seria inviável para garantir a otimização e verificação do hardware, mas também não capturaria o fato de que o SoC executa software embarcado (um aspecto ausente nos ASICs).

Os diferentes níveis e estilos de descrição adequados ao projeto de sistemas integrados são genericamente denominados de *nível de sistema eletrônico* ou ESL (*Electronic System Level*) [MAR 02]. Uma das alternativas mais promissoras para elaborar uma descrição ESL de um SoC é o uso da linguagem SystemC [SYS 08].

Em uma descrição ESL, os IPs podem ser descritos em diferentes níveis e estilos de descrição, obtidos através de refinamentos sucessivos.

A descrição RTL de cada um dos IPs representa a saída da vertente de hardware do fluxo de projeto ESL de um SoC. A partir da descrição RTL de um IP, pode-se obter sua implementação VLSI (ou um protótipo em FPGA), através de ferramentas de síntese automática. Os *layouts* dos circuitos VLSI dos IPs e dos demais componentes do SoC são então usados para obter o *layout* de todo o circuito integrado, através de ferramentas de posicionamento e roteamento.

Embora a noção de plataforma inclua também o reuso de componentes de software, a necessidade de antecipar o desenvolvimento de *software dependente de hardware* ou HDS (*hardware-dependent software*), levou à proposta de um estilo de descrição denominado de *modelagem baseada em transações* ou TLM (*Transaction Level Modeling*).

TLM é um estilo de descrição no nível de sistema (ESL), caracterizado por abstrair os detalhes de comunicação em transações. Os componentes modelados possuem interface *bit-true*, porém sua simulação é rápida e a modelagem causa baixo impacto no custo do projeto [GHE 05].

As principais aplicações contemporâneas de sistemas embarcados que demandam SoCs serão discutidas na próxima seção.

1.2 Aplicações contemporâneas amparadas por SoCs

A computação embarcada tornou-se a base de suporte da sociedade moderna. Na indústria automotiva os custos com sistemas embarcados estão chegando à proporção de um terço, algo que já havia acontecido há tempos na indústria aeronáutica [WOL 07]. Segundo [SV 07], um automóvel moderno possui entre 12 e aproximadamente 100 unidades eletrônicas de controle.

Nesse contexto de extensa utilização de SoCs, o GSM tem se solidificado como padrão de telefonia móvel. Os milhões de dispositivos móveis existentes e também em fabricação, têm como

base o GSM, desenvolvido nos anos 80 com o propósito de padronizar os serviços móveis na Europa. Porém com um êxito muito maior do que o previsto na idéia inicial, o GSM tornou-se um padrão global.

Com seus 2,5 bilhões de usuários [CHI 08], o GSM tem várias extensões como o EGSM (*Extended GSM*), GPRS (*General Packet Radio Service*), EDGE (*Enhanced Data Rates for GSM Evolution* ou EGPRS), e HSDPA (*High-Speed Downlink Packet Access*) entre vários outros. Esse grande número de tecnologias baseadas em GSM demonstra a sua grande importância e participação na área, onde, embora a comunicação de dados tem evoluído rapidamente, a codificação de voz tem mudado muito pouco.

1.3 Escopo e contribuição técnica

Esta monografia relata o projeto de um IP partindo de uma descrição TLM de um sistema integrado, onde a especificação executável do IP é um modelo funcional atemporal descrito em SystemC.

A aplicação-alvo é GSM e a função crítica escolhida para ser implementada no IP é a de Análise LPC (*Linear Predictive Coding*). LPC é um algoritmo de processamento de áudio que utiliza um modelo de predição linear para compressão/descompressão (análise/síntese). A análise procura no sinal de áudio padrões que se repetem e calcula coeficientes para representar a forma de onda, que serão utilizados posteriormente para a síntese (ou descompressão) do áudio.

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

Através da prototipação em FPGA e síntese na forma de circuito VLSI, o IP poderá ser caracterizado em termos de seu desempenho (frequência de operação e *throughput*) e consumo de energia/potência, utilizando-se ferramentas comerciais de automação de projeto eletrônico ou EDA (*Electronic Design Automation*).

São as seguintes as contribuições técnicas desta monografia: A descrição funcional e a descrição RTL sintetizável do IP, com extensa área de aplicação em telefonia celular entre tantas outras possibilidades. E, dado o número reduzido de IPs disponíveis em repositórios públicos, pretende-se disponibilizá-lo ao domínio público.

1.4 Organização da monografia

Este trabalho está organizado da seguinte maneira: o Capítulo 2 traz uma breve revisão bibliográfica dos paradigmas contemporâneos de projeto e, o Capítulo 3, dos fundamentos da aplicação alvo GSM. No Capítulo 4 descrevemos a metodologia e a infra-estrutura utilizadas no desenvolvimento deste trabalho. No Capítulo 5 aborda-se o projeto do IP: sua seleção, modelagem funcional, integração com a plataforma TLM e resultados experimentais. No Capítulo 6 são apresentados a modelagem RTL e seus resultados experimentais. Finalmente, o Capítulo 7 resume as conclusões deste trabalho e aponta caminhos para se continuar o trabalho aqui iniciado.

Capítulo 2

Paradigmas contemporâneos de projeto

Este capítulo faz um panorama dos paradigmas de projeto contemporâneos utilizados no desenvolvimento deste trabalho.

2.1 Projeto baseado em plataforma

O Projeto Baseado em Plataforma ou *Platform-Based Design* (PBD), tem como motivação a redução dos crescentes custos de máscaras e preparo de produção. Também problemas como a necessidade de redução do *time-to-market* juntamente com o aumento da complexidade têm levado indústrias de sistemas e circuitos integrados para longe de métodos de projetos totalmente personalizados (*full-custom design*) e em direção a projetos que possam ser montados rapidamente a partir de componentes pré-projetados e pré-caracterizados, priorizando reuso, montagem correta de componentes e compilação rápida e confiável de especificações a implementações [SV 01].

Uma plataforma, conceitualmente, é uma camada de abstração no fluxo de projeto que facilita um número de possíveis refinamentos em uma camada de abstração subsequente. Ou seja, em termos práticos, uma plataforma é uma biblioteca de componentes junto com suas regras de composição, contendo tanto blocos que realizam a computação quanto blocos de interconexão. Uma instância da arquitetura de referência (plataforma) é obtida através da escolha de componentes ou do ajuste de parâmetros de componentes reconfiguráveis [SV 01].

A validação de refinamentos de blocos é facilitada em uma plataforma, dado que os mesmos podem estar descritos em níveis diferentes. Tendo em mãos uma instância pré-validada de um módulo da plataforma (modelo de referência), o refinamento é realizado substituindo o bloco descrito em um nível mais alto (de abstração) pelo bloco refinado. Os resultados observados após a

substituição devem ser os mesmos que no caso anterior, ou seja, o comportamento dos dois blocos, do ponto de vista externo, deve ser o mesmo. Caso a plataforma apresente erro após o refinamento de um bloco, o erro se encontrará no módulo refinado, uma vez que os outros blocos estavam corretos e não foram modificados. Este processo, ao ser aplicado a sucessivos blocos, resulta no refinamento de toda a plataforma em direção a uma representação que pode ser implementada em silício.

2.2 Modelagem baseada em transações (TLM)

Para se iniciar um projeto em um nível de abstração acima do RTL, há a necessidade de uma modelagem que contemple suporte para desenvolvimento tanto de hardware quanto de software [GHE 05]. No final da década de 90, chegou-se à descrição com precisão de ciclos (*cycle accurate*). Porém, como ainda era muito detalhado e próximo ao RTL, acabou aumentando custos e deixando a simulação na faixa de kHz, somente em torno de dez vezes mais rápida (em comparação com as centenas de Hz obtidos na simulação RTL).

A modelagem baseada em transações surgiu devido à necessidade de uma abstração de mais alto nível que permitisse uma modelagem muito mais rápida e que ainda assim fosse suficientemente precisa para capturar a funcionalidade e suficientemente eficiente para permitir a execução de software, através de simulação amparada no modelo do hardware.

TLM é um estilo de descrição no nível de sistema (ESL) caracterizado por abstrair os detalhes de comunicação em transações de forma a permitir o desenvolvimento e teste de software. Os componentes modelados, como na descrição *cycle accurate*, possuem interface *bit-true* mas a simulação é rápida e a modelagem com baixo impacto no custo do projeto. Essa maior velocidade na simulação é obtida ao se abstrair os muitos detalhes presentes na descrição com precisão de ciclos, evitando um *overhead* desnecessário neste ponto do projeto.

O TLM tem se mostrado confiável e possibilita o co-projeto do desenvolvimento de software e de hardware, análise arquitetural e verificação funcional por diferentes equipes utilizando um mesmo modelo de referência [GHE 05, p. 15]. Desse modo permite antecipar o desenvolvimento do software para as etapas iniciais do projeto, habilitando o co-projeto e co-validação de hardware/software, reduzindo assim custos de desenvolvimento e *time-to-market*.

2.3 Linguagens de descrição de sistemas

Para o desenvolvimento ESL, é necessária uma linguagem de modelagem que suporte os requisitos presentes neste nível [MAR 02]. Em busca deste ambiente de suporte, em 1999, um conjunto de empresas líderes na indústria de Automação de Projetos Eletrônicos, semicondutores e IPs anunciou a linguagem SystemC. Por amparar-se em código aberto que não requer pagamento da licença de uso, SystemC tem sido uma atrativa opção com aceitação crescente no meio acadêmico e na indústria.

SystemC [SYS 08] é, na realidade, uma biblioteca de classes C++. O *kernel* de simulação fornece a noção de tempo e o suporte à concorrência. Como é uma extensão de uma linguagem de programação, a modelagem em SystemC é realizada servindo-se do mesmo conjunto de ferramentas e ambientes de desenvolvimento para a linguagem C++. Os modelos são descritos em C++ fazendo referência às bibliotecas de classes e, ao compilar, linkar com o *kernel* e com as classes pré-compiladas do SystemC.

Capítulo 3

Fundamentos da aplicação alvo

3.1 Breve histórico e futuro do padrão GSM

O Padrão Global para Comunicações Móveis, ou *Global Standard for Mobile communications* (GSM), foi projetado para ser introduzido com um sistema móvel digital na Europa em 1991 [VAR 88]. Atualmente, é um dos padrões mais disseminados mundialmente [GUT 01] e serve de base para várias extensões e avanços como o EDGE (*Enhanced Data Rates for GSM Evolution* ou EGPRS) e o HSDPA (*High-Speed Downlink Packet Access*), que pertence à família de protocolos 3G (*third generation* ou terceira geração).

3.2 Breve descrição do processo de codificação de áudio

O codificador GSM *full-rate* utiliza uma combinação de múltiplo acesso por divisão de frequência e por divisão de tempo (FDMA/TDMA) para codificar e decodificar fluxos de áudio.

O processo de codificação pode ser sub-dividido em 5 partes, como ilustra a Figura 3.1 [VAR 88]:

- **Pré-processamento:** O sinal de voz é dividido em segmentos de 20ms (160 amostras) e aplicada compensação de *offset*, para prevenir a formação de ruído pelo processo de decodificação, posteriormente.
- **Análise LPC:** São calculados coeficientes de filtragem com estimação paramétrica (linear) e coeficientes de reflexão. Os coeficientes são quantificados logaritmicamente e são aplicados como um filtro ao sinal.

- **Análise de filtragem *short-term*:** São interpolados linearmente o primeiro e o último coeficientes (produzidos pela análise LPC anterior) por um período de transição de 5ms, resultando em coeficientes $r'(i)$ deste estágio.
- **Looping preditor *long-term*:** Divide o sinal de 160 amostras em 4 pedaços de 40 amostras cada e calcula a diferença do bloco de amostras atual com o resultado do RPE do passo anterior, realizando uma predição de longo prazo (*long-term*).
- **Codificação RPE:** Finalmente, um algoritmo de filtragem de bloco FIR é aplicado a cada sub-segmento de 40 amostras do sinal residual. E a sequência RPE é quantificada pelo PCM bloco adaptativo (APCM).

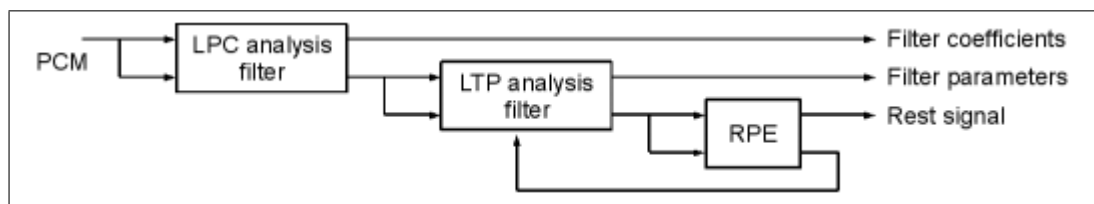


Figura 3.1: Algoritmo de compressão GSM

Capítulo 4

Metodologia e infra-estrutura

Neste capítulo será apresentada a metodologia utilizada para o desenvolvimento e validação de IPs, bem como a infraestrutura utilizada para a modelagem TLM da plataforma.

4.1 Metodologia de projeto e validação

Depois de obtida a descrição do componente de hardware, será utilizado o conceito de *reference golden model* (RGM). “Golden model” é um modelo de referência que garantidamente contém as funcionalidades e cumpre os requisitos previamente especificados. Uma vez com o modelo de referência, o dispositivo sob verificação, ou *device under verification* (DUV), pode ser testado.

Para se testar o DUV (Figura 4.1), são fornecidos os mesmos estímulos de entrada que os fornecidos ao “golden model”. E se, para um mesmo estímulo, os resultados da execução dos dois forem diferentes, é sinal de que o dispositivo na plataforma difere do modelo de referência, isto é, o “golden model” e o dispositivo sob verificação não são compatíveis.

Esta verificação é realizada até que não sejam encontrados mais erros, ou seja, as saídas resultantes dos dois processamentos são iguais. Os resultados sendo os mesmos significa que a plataforma implementa as mesmas funcionalidades que o modelo de referência para um dado estímulo.

4.2 Infra-estrutura de modelagem da plataforma

Neste trabalho, será utilizada como infra-estrutura a *ArchC Reference Platform* (ARP) para organizar e gerenciar os diferentes tipos de componentes.

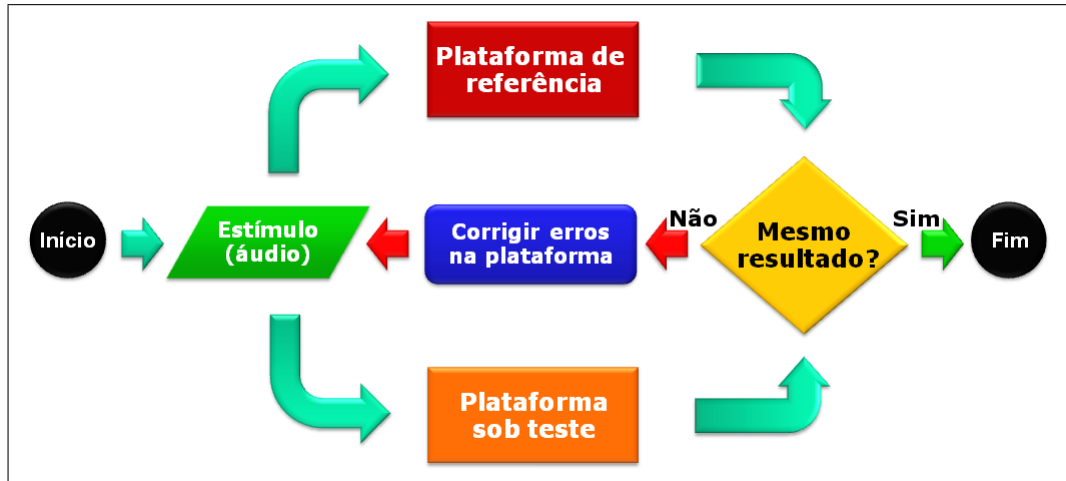


Figura 4.1: Fluxo de validação

ARP é um framework básico para se construir modelos de plataforma utilizando simuladores ArchC [AZE 05]. Ela provê uma estrutura de diretórios para os principais componentes presentes em modelos de plataformas heterogêneas e *scripts* para construir e simular essas plataformas.

A estrutura interna da ARP é composta por 9 diretórios:

- *bin*: contém *scripts* da ARP
- *doc*: contém arquivos de documentação
- *IP*: contém os núcleos de hardware
- *IS*: contém IPs de estruturas de interconexão
- *lib*: contém bibliotecas para extensão de funcionalidades
- *Platforms*: contém as plataformas incluindo seus arquivos principais onde ocorre a instanciação de componentes
- *Processors*: contém modelos escritos na ADL ArchC de processadores.
- *SW*: contém o software embarcado alvo a ser executado
- *Wrappers*: contém os adaptadores necessários para a conexão de ISs e IPs

4.2.1 Mapeamento da aplicação para a plataforma

Foi desenvolvida uma plataforma, descrita em TLM, que realiza a compressão de áudio para o padrão GSM. A plataforma é composta de um processador PowerPC (descrito em ArchC), um

bloco de memória e um IP, interconectados por um barramento.

O IP desenvolvido implementa em hardware parte da funcionalidade do padrão GSM. A transação entre processador e IP é invocada por meio de registradores mapeados em memória.

O software, compilado para PowerPC e carregado na memória da plataforma, foi modificado para, ao invés de chamar uma função em software, invocar uma transação com o modelo funcional do IP e utilizar a função crítica nele implementada.

Capítulo 5

Particionamento hardware-software e modelagem funcional do IP

Este capítulo inicialmente relata o processo de seleção de uma função para implementação em hardware e sua modelagem funcional. Em seguida, o capítulo descreve a modelagem do IP na plataforma TLM.

5.1 Seleção de funcionalidade para o IP

Para ser definida a funcionalidade a ser implementada no IP, foi realizado um “profiling” a partir de um programa que captura as funcionalidades do padrão GSM. A partir dos resultados obtidos, pôde-se observar os pontos críticos do sistema e propôr um particionamento de funcionalidades em hardware e software.

5.1.1 “Profiling” e escolha

O “profiling” foi realizado com a ferramenta *gprof*, que é distribuída e mantida pela GNU em seu pacote de utilitários binários (*GNU Binary Utilities*). Esta análise fornece uma espécie de perfil do software para uma determinada entrada, mostrando o número de vezes que as funções são chamadas na execução, o tempo gasto em cada uma e a sua porcentagem com relação ao tempo total daquela execução do software.

Os resultados obtidos na análise do comportamento do software com diferentes entradas está descrito na Tabela 5.1 a seguir e as chamadas exemplificadas na Figura 5.1, onde as funções mais chamadas (no processo de codificação) foram *Gsm_Long_Term_Predictor* com 45,2% e a

Gsm_LPC_Analysis com 16,1% de todas as chamadas.

Tabela 5.1: Resultado dos *profilings*

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
45.16	0.14	0.14	28976	0.00	0.00	Gsm_Long_Term_Predictor
16.13	0.19	0.05	7244	0.01	0.01	Gsm_Short_Term_Analysis_Filter
9.68	0.22	0.03	28976	0.00	0.00	Gsm_RPE_Encoding
9.68	0.25	0.03	7244	0.00	0.01	Gsm_LPC_Analysis
9.68	0.28	0.03	7244	0.00	0.00	Gsm_Preprocess

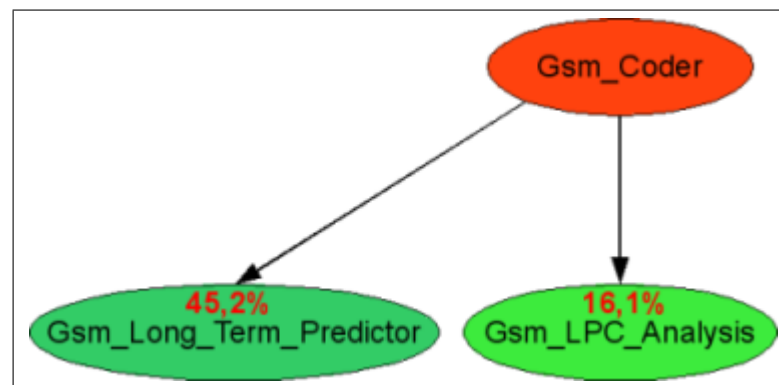


Figura 5.1: Árvore de chamadas simplificada

Após a análise dos “profilings”, foi constatado que a função LTP é a mais chamada, seguida pela *Gsm_Short_Term_Analysis_Filter* e pela *RPE_Encoding*. Porém, a Análise LPC (*Linear Predictive Coding*) é mais genérica e, portanto, mais propícia à reutilização do que as outras. Porque, além do GSM, ela também é encontrada em outras aplicações como na construção de *vocoders* (da mistura das palavras do inglês, *voice* e *coder* – codificador de voz), onde instrumentos musicais são utilizados como sinal de excitação de um filtro estimado pela voz do cantor (técnica muito popular na música eletrônica). Também é utilizada no codificador de áudio FLAC [FLA 08] e em *text-to-speech*.

5.2 Modelagem Funcional do IP na Plataforma TLM

Conforme resultados da análise e seleção, a função a ser implementada é a *Gsm_LPC_Analysis*. Ela calcula coeficientes de filtragem com estimação paramétrica (linear) e calcula coeficientes de reflexão utilizando o algoritmo de recursão de Schur. E então, finalmente, quantifica os coeficientes logaritmicamente (*Log Area Ratios*).

A assinatura da função selecionada é a seguinte:

```
void Gsm_LPC_Analysis (word * s, word * LARc);
```

onde `word` é um inteiro de 16 bits com sinal, `s` é um ponteiro para um array de 160 words, utilizados tanto para entrada quanto para a saída, e `LARc` é um ponteiro para um array de 8 posições para a saída de dados. O código desta função é mostrado no Apêndice A.

A plataforma TLM é composta de um processador PowerPC, um adaptador (*wrapper*) para servir de interface entre o processador e o barramento, uma memória de acesso aleatório (RAM), o IP sob projeto e um barramento que interliga todos esses componentes. Na Figura 5.2 tem-se uma visão geral da plataforma.

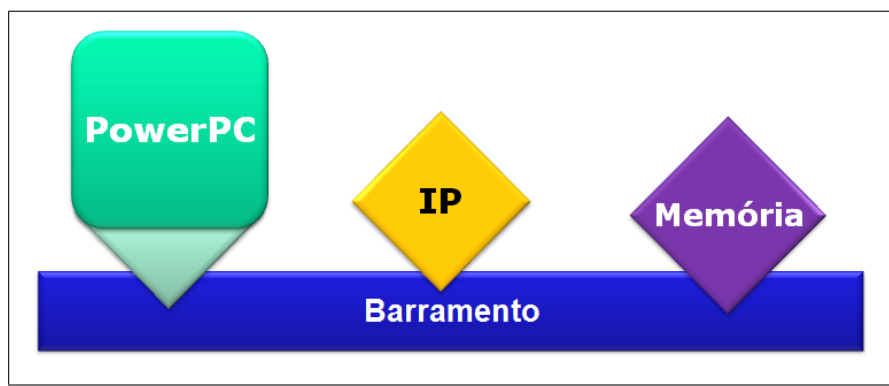


Figura 5.2: Visão geral da plataforma

5.2.1 Interface de Entrada/Saída do IP

O IP contém dois sinais de 32 pinos: um para dados e um para endereços. No total são 64 pinos, utilizados tanto para entrada quanto para saída. Esta interface está ilustrada na Figura 5.3.

5.2.2 Acesso ao IP por E/S Mapeada em Memória

Os dois parâmetros para a função foram colocados em registradores de 32 bits mapeados em memória. No total foram três registradores, um a mais para controle de execução, conforme descrito na Tabela 5.2. Os endereços apresentados são relativos ao endereço base atribuído ao IP na sua instanciação e a faixa de endereço a ser reservada é de 0x0C (doze bytes dos três registradores).

Para invocar uma transação com o IP, o processador carrega os parâmetros nos registradores 's' e 'LARc'. Para disparar a transação, o processador carrega o valor "1" no registrador 'con-

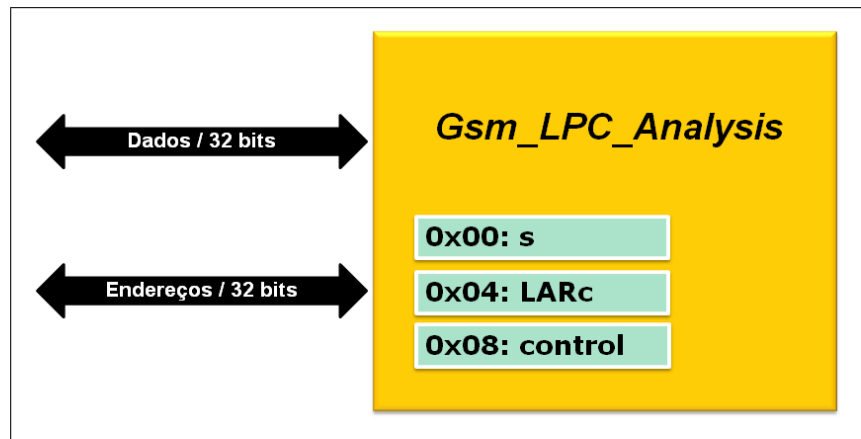


Figura 5.3: Módulo Gsm_LPC_Analysis

Tabela 5.2: Registradores mapeados em memória e seus endereços relativos

Endereço relativo	Registrador
0x00	s
0x04	LARc
0x08	control

trol'. Quando a tarefa é concluída, o IP zera o registrador 'control'. Este evento é detectado pelo processador através de consulta periódica (*polling*).

5.2.3 Utilização como Modelo de Validação

Após a sua validação, este modelo funcional será utilizado como o modelo de referência (*golden model*) para a validação da descrição RTL do IP.

5.3 Resultados experimentais

Nesta seção, estão descritos os resultados experimentais e a configuração utilizada para a sua realização.

5.3.1 Configuração

Os experimentos foram realizados em um computador com a seguinte configuração: processador Intel Core 2 Duo T5250, 1.5GHz de frequência, cache de 2 MB; memória principal de 2 GB; rodando sistema operacional GNU/Linux Ubuntu 7.04, kernel 2.6.20 SMP.

As versões das ferramentas utilizadas seguem abaixo:

- **ArchC** versão 2.0
- **SystemC** versão 2.1.1
- **SystemC TLM** versão 1.0 (2005-04-08)
- **Compilador GCC** versão 4.1.2
- **Cross-compiler GCC para PowerPC** versão 3.1
- **GSM *speech compression*** versão 06.10 13 kbit/s RPE/LTP, do pacote de *benchmarks* MiBench¹ versão 1.0

O modelo funcional de PowerPC utilizado é a versão 0.7.3, para ArchC 2.0. O software GSM rodando neste modelo processador foi compilado com o *cross-compiler*. Tanto o modelo do processador PowerPC quanto o *cross-compiler* estão disponíveis na página do projeto ArchC.

5.3.2 Validação do IP Funcional na Plataforma

A validação do IP na plataforma se deu utilizando o método previamente descrito na Seção 4.1. Como comparador de igualdade entre as saídas, foi utilizado o programa *diff*, que realiza uma comparação bit a bit entre dois arquivos. O modelo de referência (“golden model”) utilizado na validação é a implementação encontrada no *benchmark* MiBench, assim como os estímulos utilizados, que são arquivos de áudio no formato AU².

Para cada arquivo de áudio, foram executadas a plataforma e o *golden model*. Os arquivos de saída resultantes das execuções – áudio comprimido em GSM – foram comparados com o utilitário *diff*, seguramente garantindo a igualdade das saídas.

5.3.3 Resultados

Na Tabela 5.3 estão listados os resultados das execuções da plataforma. A tabela mostra o nome do arquivo de áudio de entrada, o tempo de execução da plataforma, o número de instruções executadas pelo processador, e se a plataforma executou com ou sem o IP em projeto.

Da análise destes resultados, podemos salientar, por exemplo, a redução do número de instruções executadas pelo processador. Como podemos observar na Tabela 5.3, a execução da plataforma com o IP nos dá uma economia de aproximadamente 18% nas instruções executadas,

¹[GUT 01]

²developed by Sun Microsystems

em comparação com uma plataforma rodando a compressão somente em software. Esta redução também indica uma possível economia de energia.

Tabela 5.3: Resultados das execuções da plataforma com e sem IP

Arquivo	IP	Tempo(s)	Instruções
input_small.au	com	35,24	21.663.557
input_small.au	sem	30,42	22.942.384
input_large.au	com	3124,13	1.170.169.053
input_large.au	sem	3453,61	1.235.633.811

Capítulo 6

Modelagem RTL do IP

Este capítulo trata da modelagem RTL do IP sob projeto: proposta de uma micro-arquitetura, relata o processo de codificação e os resultados experimentais obtidos.

6.1 Proposta de uma micro-arquitetura

Após a análise mais detalhada do IP funcional e o aumento do conhecimento sobre a compressão LPC, percebeu-se que a representação dos coeficientes do LPC utilizada tornava o IP muito específico. As maneiras mais difundidas de representação (derivação) dos coeficientes do LPC são: LAR (*Log Area Ratios*) e LSP (*Line Spectral Pairs*), sendo que o software tratado neste trabalho somente implementa LAR. Portanto, visando à reutilização, optou-se por um reparticionamento hardware/software, onde permanece em hardware a Análise LPC e em software a derivação dos coeficientes. Isto resultou na arquitetura apresentada na Figura 6.1, que realiza a computação até antes da derivação dos coeficientes.

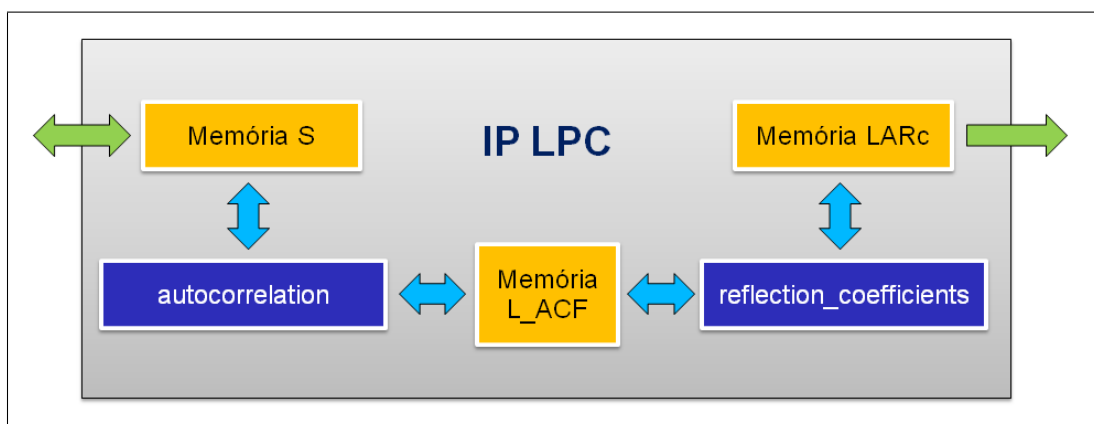


Figura 6.1: Arquitetura RTL do IP

O IP RTL proposto é composto por dois sub-módulos síncronos, *autocorrelation* e *reflection_coefficients*. Os dados utilizados no IP são armazenados em três memórias RAM de duas portas de leitura/escrita, conforme Figura 6.2. Essas memórias se fazem necessárias porque os dados são utilizados em paralelo, ou seja, o primeiro módulo opera sobre todos os valores de *s* e de *L_ACF* antes de terminar para o segundo módulo poder entrar em funcionamento, o que impede o fluxo serial dos dados.

Esta entidade principal, que agrega os demais componentes, possui como sinais de entrada: *clock*, *reset* assíncrono, *start* e também as interfaces com as memórias, porém estas serão detalhadas mais à frente, e *ready* como sinal de saída. Possui uma máquina de estados pequena (3 estados) que somente espera o sinal de *start* para iniciar a execução dos módulos, em sequência, esperar pelo processamento e sinalizar que o IP está pronto. O processamento é realizado iniciando-se a execução do *autocorrelation* e quando este está pronto, inicia-se a execução do *reflection_coefficients*. Quando o último módulo concluir, o IP terminou a sua tarefa e este sinaliza colocando o sinal *ready* em '1'.

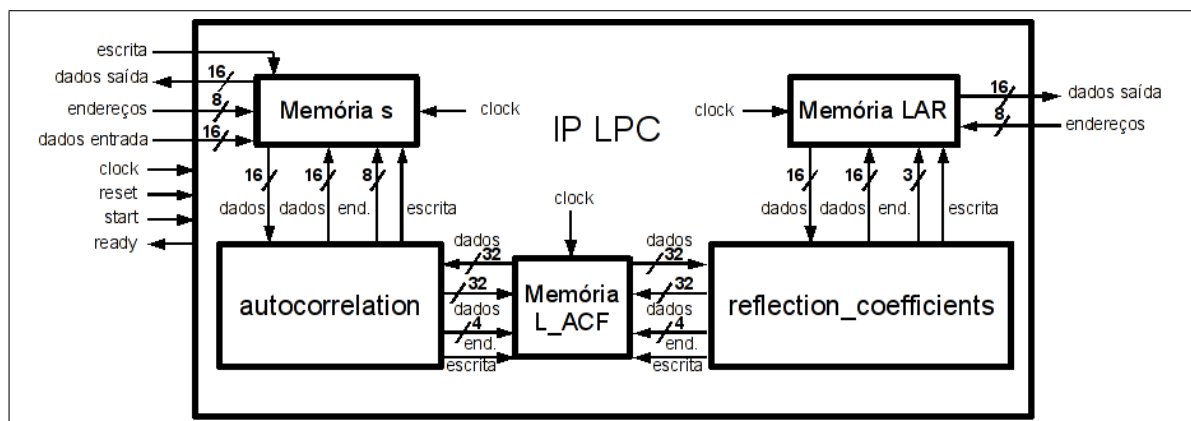


Figura 6.2: Diagrama de blocos

A comunicação com as memórias é realizada por um sinal, que habilita modo de escrita, e três barramentos: um de endereço, um de dados de leitura e um de dados para escrita. A memória *s* armazena o sinal de áudio de entrada em 512 bytes (256 *words* de 16 bits), *L_ACF* armazena os coeficientes de autocorrelação intermediários (16 *words* de 32 bits, ou 64 bytes) e *LAR* armazena os coeficientes de reflexão em 16 bytes (8 *words* de 16 bits).

A entrada e saída de dados no IP é realizada através das interfaces externas com as memórias *s* e *LAR*. Os dados de entrada são colocados em *s* e, quando o processamento estiver concluído, os resultados poderão ser lidos em *s* e em *LAR*.

O módulo *autocorrelation* (Figura 6.3) contém uma máquina de estados com 28 estados. Ele

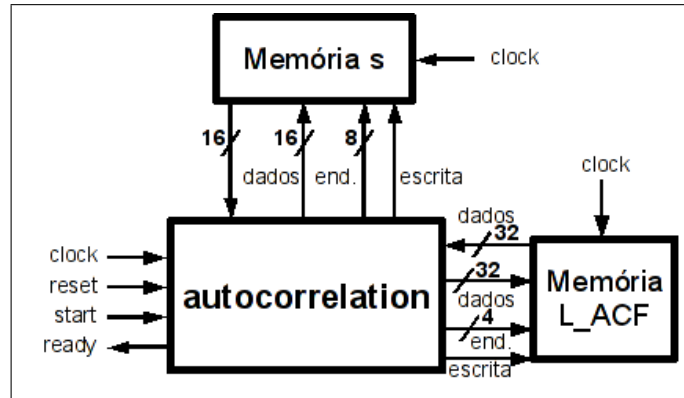


Figura 6.3: Módulo *autocorrelation*

encontra padrões que se repetem no áudio de entrada *s*. O resultado é colocado em *s* e em *L_ACF*. Possui sinais *clock*, *reset* assíncrono, *start*, e dois barramentos de dados de entrada. Como saída, possui os barramentos de dados e endereços, sinais para habilitar escrita nas memórias, e um sinal *ready*.

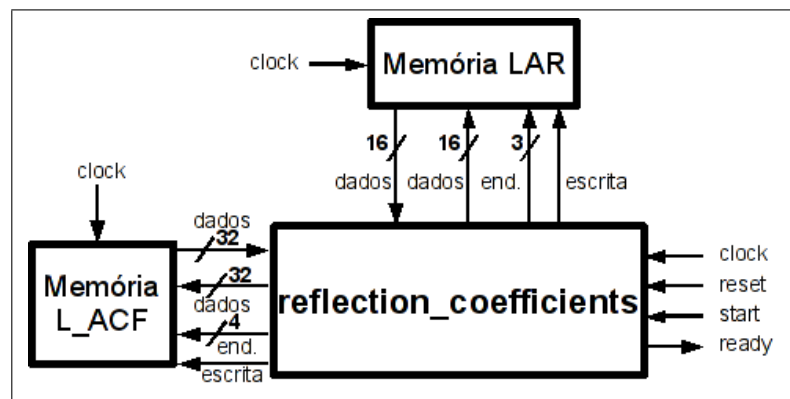


Figura 6.4: Módulo *reflection_coefficients*

O segundo módulo, *reflection_coefficients* (Figura 6.4), possui uma máquina de estados com 33 estados que calcula os coeficientes de reflexão da onda de som. Conta com dois barramentos de dados de entrada e sinais de *clock*, *reset* assíncrono, *start* e *ready*. Os resultados deste módulo são escritos em *LAR* através dos barramentos de endereços e dados de saída.

6.2 Codificação RTL do IP

A codificação sintetizável do IP foi feita na linguagem de descrição de hardware VHDL, utilizando o software Quartus II, da Altera, versão 8.0sp1, com licença acadêmica sem custo. Foram codificadas aproximadamente 1500 linhas de código e criadas 3 memórias com a ferramenta

MegaFunction, presente no Quartus II e cujo resultado não foi incluído na contagem das linhas de código.

A Tabela 6.1 mostra o relato de componentes utilizados pelo IP ao ser compilado para a família de dispositivos StratixII, da Altera, como o número total de registradores e ALUTs utilizados. O dispositivo escolhido automaticamente pelo Quartus II foi o EP2S15F484C3, e o pior caso de tempo de *setup* apontado pela análise de “timing” é de 5,689 ns. O clock indicado pelo analisador padrão é de 114,27 MHz (período de 8,751 ns). Simulado para este dispositivo e tecnologia, o IP operou com sucesso a uma frequência de clock de 111 MHz (ou período de 9 ns).

Tabela 6.1: Sumário de componentes do IP para o StratixII EP2S15F484C3

Componente	Quantidade	% do total
Registradores	408	–
ALUT combinacionais	539	4%
Block memory bits	7.216	2%
Pinos	64	19%
Pinos virtuais	0	–
PLLs	0	0%
DLLs	0	0%

6.3 Validação e resultados experimentais

Para a realização dos experimentos, foram utilizados os softwares ModelSim, da Mentor Graphics, e Quartus II, da Altera, executando em um computador com as mesmas configurações descritas na Seção 5.3.1 mas com o sistema operacional Windows Vista Home Premium *service pack 1*.

A validação da descrição RTL foi realizada aplicando-se o método da Seção 4.1 com algumas adaptações (Figura 6.5). Os estímulos são extraídos do modelo funcional e carregados na memória do IP RTL e os resultados da execução são comparados com os resultados do modelo funcional previamente validado. Os estímulos reais, reutilizados do modelo funcional, garantem robustez e correteude ao IP ao ser simulado em um ambiente real de aplicação.

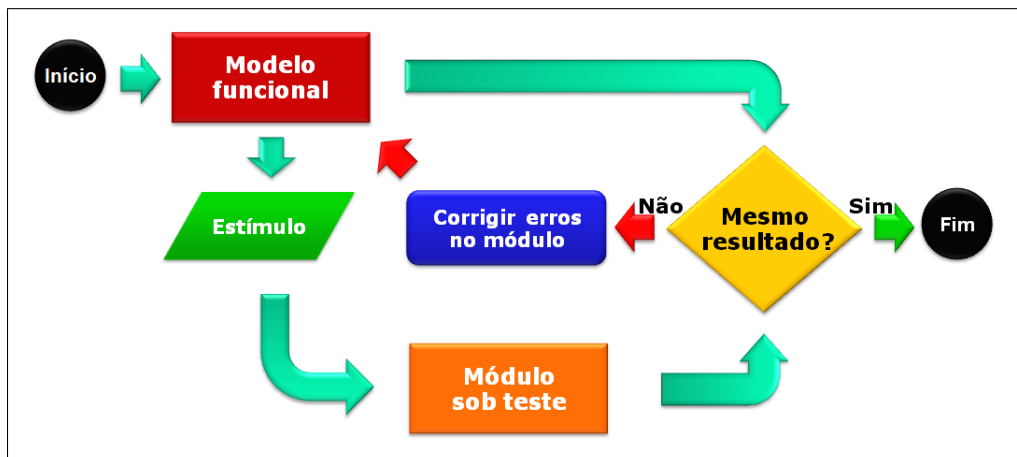


Figura 6.5: Fluxo de validação do modelo RTL

Capítulo 7

Conclusão

Os resultados experimentais mostram a corretude e robustez do IP projetado, sendo validado com estímulos extraídos de aplicação real possibilitada pela simulação do modelo funcional em uma plataforma TLM executando o software alvo. Ainda, é possível observar um ganho de produtividade ao se iniciar o projeto em um nível mais alto de abstração.

A utilização de modelagem em nível transacional mostra-se essencial para o projeto de sistemas que estão cada vez mais complexos, requerindo pouco tempo para se modelar (em relação ao tempo total do projeto) e reduzindo o *time-to-market* ao antecipar o desenvolvimento de software dependente de hardware.

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

7.1 Trabalhos Futuros

Como trabalhos futuros, ficam a prototipação do IP em FPGA e em silício. Que poderão ser realizadas mais facilmente tendo em vista que a descrição RTL resultante deste trabalho é sintetizável. As prototipações serão realizadas no âmbito do projeto BrazilIP, fomentado pelo CNPQ.

Apêndice A

Código da Análise LPC

O código a seguir é o que realiza a filtragem LPC (*linear predictive coding*), encontrada no software de compressão GSM do MiBench.

```
/*
 * Copyright 1992 by Jutta Degener and Carsten Bormann, Technische
 * Universitaet Berlin. See the accompanying file "COPYRIGHT" for
 * details. THERE IS ABSOLUTELY NO WARRANTY FOR THIS SOFTWARE.
 */

/* $Header: /home/mguthaus/.cvsrc/mibench/telecomm/gsm/src/lpc.c,v 1.
2000/11/06 19:54:26 mguthaus Exp $ */

#include <stdio.h>
#include <assert.h>

#include "private.h"

#include "gsm.h"
#include "proto.h"

#undef P

/*
```

```

* 4.2.4 .. 4.2.7 LPC ANALYSIS SECTION
*/

/* 4.2.4 */

static void Autocorrelation P2((s, L_ACF),
word      * s, /* [0..159] IN/OUT */
longword * L_ACF) /* [0..8] OUT */
/*
* The goal is to compute the array L_ACF[k]. The signal s[i] must
* be scaled in order to avoid an overflow situation.
*/
{
register int k, i;

word temp, smax, scalauto;

#ifdef USE_FLOAT_MUL
float float_s[160];
#endif

/* Dynamic scaling of the array s[0..159]
*/

/* Search for the maximum.
*/
smax = 0;
for (k = 0; k <= 159; k++) {
temp = GSM_ABS( s[k] );
if (temp > smax) smax = temp;
}

```

```

/* Computation of the scaling factor.
 */
if (smax == 0) scalauto = 0;
else {
assert(smax > 0);
scalauto = 4 - gsm_norm( (longword)smax << 16 );/* sub(4,..) */
}

/* Scaling of the array s[0...159]
 */

if (scalauto > 0) {

# ifdef USE_FLOAT_MUL
#   define SCALE(n) \
case n: for (k = 0; k <= 159; k++) \
float_s[k] = (float) \
(s[k] = GSM_MULT_R(s[k], 16384 >> (n-1))); \
break;
# else
#   define SCALE(n) \
case n: for (k = 0; k <= 159; k++) \
s[k] = GSM_MULT_R( s[k], 16384 >> (n-1) ); \
break;
# endif /* USE_FLOAT_MUL */

switch (scalauto) {
SCALE(1)
SCALE(2)
SCALE(3)
SCALE(4)
}
# undef SCALE

```

```

}
# ifdef USE_FLOAT_MUL
else for (k = 0; k <= 159; k++) float_s[k] = (float) s[k];
# endif

/* Compute the L_ACF[...].
*/
{
# ifdef USE_FLOAT_MUL
register float * sp = float_s;
register float  sl = *sp;

# define STEP(k)  L_ACF[k] += (longword)(sl * sp[ -(k) ]);
# else
word * sp = s;
word  sl = *sp;

# define STEP(k)  L_ACF[k] += ((longword)sl * sp[ -(k) ]);
# endif

# define NEXTII  sl = *++sp

for (k = 9; k--; L_ACF[k] = 0) ;

STEP (0);
NEXTII;
STEP(0); STEP(1);
NEXTII;
STEP(0); STEP(1); STEP(2);
NEXTII;
STEP(0); STEP(1); STEP(2); STEP(3);
NEXTII;

```

```

STEP (0); STEP (1); STEP (2); STEP (3); STEP (4);
NEXTI;
STEP (0); STEP (1); STEP (2); STEP (3); STEP (4); STEP (5);
NEXTI;
STEP (0); STEP (1); STEP (2); STEP (3); STEP (4); STEP (5); STEP (6);
NEXTI;
STEP (0); STEP (1); STEP (2); STEP (3); STEP (4); STEP (5); STEP (6); STEP (7);

for (i = 8; i <= 159; i++) {

NEXTI;

STEP (0);
STEP (1); STEP (2); STEP (3); STEP (4);
STEP (5); STEP (6); STEP (7); STEP (8);
}

for (k = 9; k--; L_ACF[k] <<= 1) ;

}

/* Rescaling of the array s[0..159]
*/
if (scalauto > 0) {
assert (scalauto <= 4);
for (k = 160; k--; *s++ <<= scalauto) ;
}
}

#if defined(USE_FLOAT_MUL) && defined(FAST)

static void Fast_Autocorrelation P2((s, L_ACF),
word * s, /* [0..159] IN/OUT */
longword * L_ACF) /* [0..8] OUT */

```

```

{
register int k, i;
float f_L_ACF[9];
float scale;

float          s_f[160];
register float *sf = s_f;

for (i = 0; i < 160; ++i) sf[i] = s[i];
for (k = 0; k <= 8; k++) {
register float L_temp2 = 0;
register float *sfl = sf - k;
for (i = k; i < 160; ++i) L_temp2 += sf[i] * sfl[i];
f_L_ACF[k] = L_temp2;
}
scale = MAX_LONGWORD / f_L_ACF[0];

for (k = 0; k <= 8; k++) {
L_ACF[k] = f_L_ACF[k] * scale;
}
}
#endif /* defined (USE_FLOAT_MUL) && defined (FAST) */

/* 4.2.5 */

static void Reflection_coefficients P2( (L_ACF, r),
longword * L_ACF, /* 0...8 IN */
register word * r /* 0...7 OUT */
)
{
register int i, m, n;
register word temp;
register longword ltmp;

```



```

word ACF[9]; /* 0..8 */
word P[ 9]; /* 0..8 */
word K[ 9]; /* 2..8 */

/* Schur recursion with 16 bits arithmetic.
*/

if (L_ACF[0] == 0) {
for (i = 8; i--; *r++ = 0) ;
return;
}

assert( L_ACF[0] != 0 );
temp = gsm_norm( L_ACF[0] );

assert(temp >= 0 && temp < 32);

/* ? overflow ? */
for (i = 0; i <= 8; i++) ACF[i] = SASR( L_ACF[i] << temp, 16 );

/* Initialize array P[..] and K[..] for the recursion.
*/

for (i = 1; i <= 7; i++) K[ i ] = ACF[ i ];
for (i = 0; i <= 8; i++) P[ i ] = ACF[ i ];

/* Compute reflection coefficients
*/
for (n = 1; n <= 8; n++, r++) {

temp = P[1];
temp = GSM_ABS(temp);
if (P[0] < temp) {

```

```

for (i = n; i <= 8; i++) *r++ = 0;
return;
}

*r = gsm_div( temp, P[0] );

assert(*r >= 0);
if (P[1] > 0) *r = -*r; /* r[n] = sub(0, r[n]) */
assert (*r != MIN_WORD);
if (n == 8) return;

/* Schur recursion
*/
temp = GSM_MULT_R( P[1], *r );
P[0] = GSM_ADD( P[0], temp );

for (m = 1; m <= 8 - n; m++) {
temp      = GSM_MULT_R( K[ m ], *r );
P[m]      = GSM_ADD( P[ m+1 ], temp );

temp      = GSM_MULT_R( P[ m+1 ], *r );
K[m]      = GSM_ADD( K[ m ], temp );
}
}
}

/* 4.2.6 */

static void Transformation_to_Log_Area_Ratios Pl((r),
register word * r /* 0..7 IN/OUT */)
)
/*
* The following scaling for r[..] and LAR[..] has been used:

```

```

*
*  r[..]    = integer( real_r[..]*32768. ); -1 <= real_r < 1.
*  LAR[..] = integer( real_LAR[..] * 16384 );
*  with -1.625 <= real_LAR <= 1.625
*/
{
register word temp;
register int i;

printf("\n");
/* Computation of the LAR[0..7] from the r[0..7]
*/
for (i = 1; i <= 8; i++, r++) {

temp = *r;

printf("read%x_%d\n", r, temp);

temp = GSM_ABS(temp);
assert(temp >= 0);

if (temp < 22118) {
temp >>= 1;
} else if (temp < 31130) {
assert( temp >= 11059 );
temp -= 11059;
} else {
assert( temp >= 26112 );
temp -= 26112;
temp <<= 2;
}

*r = *r < 0 ? -temp : temp;

```

```

printf("written%x_%d\n", r, *r);

assert( *r != MIN_WORD );
}
}

/* 4.2.7 */

static void Quantization_and_coding P1((LAR),
register word * LAR      /* [0..7] IN/OUT */
)
{
register word temp;
longword ltmp;

/* This procedure needs four tables; the following equations
 * give the optimum scaling for the constants:
 *
 * A[0..7] = integer( real_A[0..7] * 1024 )
 * B[0..7] = integer( real_B[0..7] * 512 )
 * MAC[0..7] = maximum of the LARc[0..7]
 * MIC[0..7] = minimum of the LARc[0..7]
 */

# undef STEP
# define STEP( A, B, MAC, MIC ) \
temp = GSM_MULT( A, *LAR ); \
temp = GSM_ADD( temp, B ); \
temp = GSM_ADD( temp, 256 ); \
temp = SASR( temp, 9 ); \
*LAR = temp>MAC ? MAC - MIC : (temp<MIC ? 0 : temp - MIC); \

```

```

LAR++;

STEP( 20480, 0, 31, -32 );
STEP( 20480, 0, 31, -32 );
STEP( 20480, 2048, 15, -16 );
STEP( 20480, -2560, 15, -16 );

STEP( 13964, 94, 7, -8 );
STEP( 15360, -1792, 7, -8 );
STEP( 8534, -341, 3, -4 );
STEP( 9036, -1144, 3, -4 );

# undef STEP
}

void Gsm_LPC_Analysis P3((S, s,LARc),
struct gsm_state *S,
word * s,/* 0..159 signals IN/OUT */
word * LARc) /* 0..7 LARc's OUT */
{
longword L_ACF[9];

#if defined(USE_FLOAT_MUL) && defined(FAST)
if (S->fast) Fast_Autocorrelation (s, L_ACF );
else
#endif
Autocorrelation (s, L_ACF );
Reflection_coefficients (L_ACF, LARc );
Transformation_to_Log_Area_Ratios (LARc);
Quantization_and_coding (LARc);
}

```

Referências bibliográficas

- [ASH 02] Ashenden, P. J., editor. **The Designer's Guide to VHDL**. 2nd. ed. Morgan Kaufmann, 2002.
- [AZE 05] AZEVEDO, R. et al. The archc architecture description language and tools. **Int. J. Parallel Program.**, Norwell, MA, USA, v.33, n.5, p.453–484, 2005.
- [BER 02] BERGAMASCHI, R. A.; COHN, J. The a to z of socs. In: ICCAD '02: PROCEEDINGS OF THE 2002 IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2002. **Proceedings...** New York, NY, USA: ACM, 2002. p.790–798.
- [CHI 08] CHIA, S. et al. 3g evolution. **Microwave Magazine, IEEE**, [S.l.], v.9, n.4, p.52–63, Aug., 2008.
- [FLA 08] FLAC. **Free Lossless Audio Codec**.
- [GHE 05] Ghenassia, F., editor. **Transaction Level Modeling with SystemC**. Springer, 2005.
- [GUT 01] GUTHAUS, M. et al. Mibench: A free, commercially representative embedded benchmark suite. In: 4TH ANNUAL WORKSHOP ON WORKLOAD CHARACTERIZATION, 2001. **Proceedings...** Austin, TX: [s.n.], 2001.
- [MAR 02] Martin, G. et al., editors. **System Design with SystemC**. Kluwer Academic Publisher, 2002.
- [MAR 03] Martin, G.; Chang, H., editors. **Winning the SoC Revolution**. Kluwer Academic Publisher, 2003.
- [SV 01] SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. **IEEE Design & Test of Computers**, [S.l.], v.18, p.23–33, Nov/Dez, 2001.

- [SV 07] SANGIOVANNI-VINCENTELLI, A.; NATALE, M. D. Embedded system design for automotive applications. **Computer**, Los Alamitos, CA, USA, v.40, n.10, p.42–51, 2007.
- [SYS 08] SYSTEMC. **Open SystemC Initiative**.
- [THO 02] Thomas, D. E.; Moorby, P. R., editors. **The Verilog hardware description language**. 5th. ed. Springer, 2002.
- [VAR 88] VARY, P. et al. Speech codec for the european mobile radio system. In: INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING ICASSP-88, 1988. **Proceedings...** New York, NY, USA: [s.n.], 1988. v.1, p.227–230.
- [WOL 07] WOLF, W. Guest editor's introduction: The embedded systems landscape. **Computer**, Los Alamitos, CA, USA, v.40, n.10, p.29–31, 2007.