THOMAS CEGAL GOUTHIER DE VILHENA
ROGÉRIO HONG HUI CHUNG

SISTEMA RTOOL
FERRAMENTA DE PROTOTIPAGEM RÁPIDA DE DESCRIÇÕES RTL PARA FINS DE DEPURAÇÃO

São Paulo
2011

THOMAS CEGAL GOUTHIER DE VILHENA
ROGÉRIO HONG HUI CHUNG

SISTEMA RTOOL
FERRAMENTA DE PROTOTIPAGEM RÁPIDA DE DESCRIÇÕES RTL PARA FINS DE DEPURAÇÃO

Trabalho de formatura apresentado ao Departamento de Engenharia de Sistemas Eletrônicos da Escola Politécnica da USP como parte dos requisitos necessários à obtenção do título de Engenheiro.

Área de concentração: Engenharia de Sistemas Eletrônicos

Orientador: Prof. Wang Jiang Chau

São Paulo
2011

AGRADECIMENTOS

Ao professor Wang Jiang Chau, pela sugestão do tema do projeto de formatura e pelo seu empenho no auxílio aos autores durante a etapa de desenvolvimento.

Aos familiares, que deram todo o suporte necessário aos autores durante suas trajetórias no curso de graduação, e à todos que colaboraram, direta ou indiretamente, na execução deste projeto.

Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

(Brian Kernighan)

RESUMO

Os autores desenvolveram como projeto de formatura do curso de Engenharia de Sistemas Eletrônicos uma ferramenta de prototipagem rápida de descrições RTL para fins de depuração. Enquanto o desempenho de ferramentas comerciais disponíveis no mercado (ex: ModelSim), baseadas em softwares que rodam em processadores de propósito geral, mostra-se satisfatório para projetos de pequeno porte, sistemas integrados complexos sofrem com seu baixo desempenho. A prototipagem em FPGA para conseguinte emulação de uma descrição RTL, com o objetivo de se efetuar sua verificação funcional, vem sendo utilizada como uma alternativa de melhor desempenho às ferramentas tradicionais. Contudo, observou-se na literatura a maciça utilização de soluções específicas (não reutilizáveis) para a prototipagem de projetos de sistemas integrados, de modo que os autores e o orientador vislumbram um espaço para a solução abrangente desenvolvida, a qual pode vir a ter um caráter comercial.

Palavras-chave: Verificação funcional, Prototipagem, FPGA.

ABSTRACT

The authors have developed a RTL description prototyping tool for functional verification purposes as their final graduation project in the course of Electronic Systems Engineering. Although the performance of commercially available tools (ex: ModelSim) is satisfactory for small projects, more complex digital systems requirements for performance can't be delivered by these same tools due to the fact that they are software based tools. The FPGA based prototyping and emulation of RTL descriptions for functional verification purposes is a well-known alternative method for designers pursuing higher performance during the verification stage of a digital system project. However, designers usually develop their prototyping environment from scratch in such a way that it becomes a specific solution, i.e., it is only compatible with one project and can't be reused in future projects. So, the authors perceived an opportunity for the development of a wider, non-specific, prototyping tool solution that could be used by several designers in their respective projects.

Key-words: Functional Verification, Prototyping, FPGA.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API ..... *Application Programming Interface*

CI..... Circuito Integrado

DUV..... *Device Under Verification*

EAP ..... Estrutura Analítica de Projeto

FIFO ..... *First In First Out*

FPGA ..... Field Programmable Gate Array

GPP..... *General Purpose Processor*

GUI..... *Graphical User Interface*

HDL ..... *Hardware Description Language*

JTAG ..... *Joint Test Action Group*

MEF..... Máquina de Estados Finitos

RTL ..... *Register Transfer Logic*

RTOOL..... Codinome escolhido para o projeto de formatura

SOPC ..... *System On Programmable Chip*

UART ..... *Universal Asynchronous Receiver/Transmitter*

USB..... *Universal Serial Bus*

UT ..... *Under Test*

VHDL..... VHSIC (*Very High Speed Integrated Circuits*) *Hardware Description Language*

SUMÁRIO

1. INTRODUÇÃO

As linguagens de descrição de hardware (VHDL, Verilog, etc) se estabeleceram na indústria de semicondutores como um padrão para o desenvolvimento de circuitos integrados digitais. Sua utilização permite que o projetista descreva de forma relativamente simples e rápida o funcionamento de um circuito digital, deixando a laboriosa tarefa de síntese do circuito digital para o compilador da linguagem utilizada. O processo de síntese é ilustrado na figura 1.

Figura 1. A sintetização do circuito digital a partir da descrição HDL é de responsabilidade do compilador

A facilidade proporcionada pelas linguagens de descrição de hardware (HDL) para o projeto de circuitos digitais permitiu um significativo aumento na complexidade dos projetos a serem desenvolvidos. A etapa de verificação da corretude funcional de uma descrição HDL torna-se, então, fundamental para garantir o sucesso do projeto.

Dado que maioria das soluções existentes para a simulação funcional de uma descrição RTL (*Register Transfer Logic*) em uma certa HDL são baseadas em aplicativos rodando em estações de trabalho, o tempo de simulação torna-se extremamente elevado (Serrestou, Beroulle, Robach, 2007). Em decorrência disso, este projeto propõe o desenvolvimento de um sistema de verificação de descrição RTL mais veloz que seja baseado na prototipagem da descrição RTL em um FPGA (*Field Programmable Gate Array*).

O codinome escolhido para este projeto foi **RTOOL**, uma referência à sigla RTL e à palavra do idioma inglês *tool*, que significa ferramenta.

1.1 Justificativa

Os circuitos digitais, por natureza, exibem elevado grau de paralelismo, ou seja, em um dado instante de sua operação diversas tarefas independentes estão ativas. Consequentemente, as HDLs exibem um paralelismo intrínseco em suas sintaxes, tornando-as bastante distintas das linguagens de programação convencionais.

Esta distinção entre as HDLs e as linguagens de programação convencionais resulta no fato de que a simulação de uma descrição RTL em um software simulador é extremamente lenta, justamente pelo fato de que todo o paralelismo inerente às HDLs é incompatível com um processador de propósito geral. Enquanto as HDLs permitem que o programador defina múltiplas operações a serem realizadas simultâneamente, a simulação deste mesmo comportamento em software requer que um GPP execute diversas instruções sequenciais (Tocci, Widmer, 2007).

Mais do que uma questão de compatibilidade, a simulação de descrições RTL é uma importante etapa do ciclo de projeto de um CI (circuito integrado) digital. É nesta etapa que erros desconhecidos são descobertos e corrigidos.

Na dinâmica da economia atual, o tempo tornou-se um recurso extremamente valioso, principalmente nos mercados de tecnologia. Um pequeno atraso de alguns dias, semanas ou meses pode determinar o fracasso de um novo produto, resultando em grandes perdas financeiras para uma empresa.

Rogers (Rogers, 2003), em seu estudo sobre a difusão de inovações tecnológicas em seus respectivos mercados, mostra que um produto inovador é inserido no mercado através de uma curva normal, como apresentado na figura 2. Inicialmente poucos consumidores adotam a inovação (A: Introdução). Conforme estes consumidores, através de suas redes de contato, demonstram sua satisfação com o novo produto, mais consumidores passam a adotar e difundir a inovação (B: Crescimento). Eventualmente, a difusão da tecnologia chegará a um patamar no qual cerca da metade dos consumidores já adotou a tecnologia, de modo que o ritmo de difusão irá desacelerar (C: Saturação). Finalmente, com a aproximação da plena adoção da inovação, o ritmo da difusão se aproximará de zero (D: Declínio), já que, no caso de bens duráveis, não ocorre substituição, e praticamente todos os consumidores já adotaram a inovação.

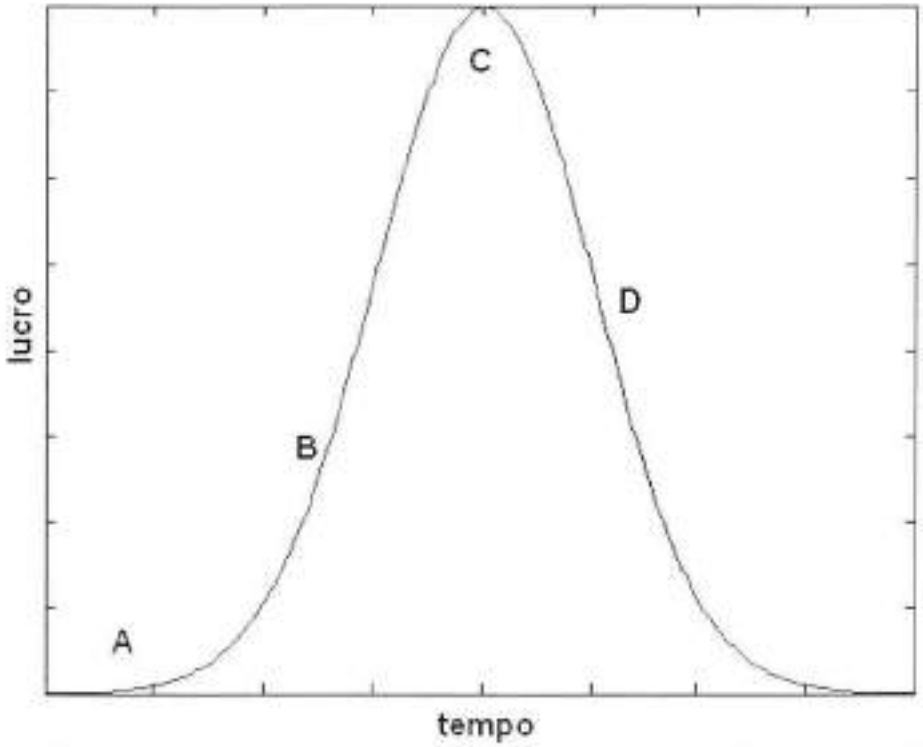

Figura 2. Difusão de uma inovação (Rogers, 2003). A: Introdução. B: Crescimento. C: Saturação. D: Declínio

O caráter inovador deste projeto está na proposição da utilização de um FPGA como o núcleo do sistema de verificação da descrição RTL. A partir da síntese da descrição RTL, sua gravação em um FPGA (juntamente com um sistema digital de suporte e comunicação) e posterior execução controlada, espera-se reduzir drasticamente o tempo gasto na etapa de depuração do projeto de um CI digital, o que significa um menor *time to market* para o produto sendo desenvolvido.

1.2 Objetivos do Projeto

Em média, 32% do tempo de projeto de um SOC é gasto em simulação top-level (Wilson research group, 2010). Portanto, espera-se com a ferramenta RTOOL uma considerável redução desta fatia de tempo gasta com a validação do sistema digital integrado.

A meta deste projeto, estimada a partir de resultados observados em Serrestou, Beroulle, Robach, 2007, será uma melhora de desempenho no tempo de verificação da descrição RTL de dez vezes. A metodologia a ser usada para a análise dos resultados será a comparação do desempenho do RTOOL com o software ModelSim, da Mentor Graphics, utilizando-se descrições RTL em linguagem VHDL sintetizável como *benchmarks* (Squillero, Reorda, Corno, 2000).

Dado que o tempo de execução da simulação de uma descrição RTL em uma ferramenta comercial torna-se um empecilho apenas para circuitos complexos, ou seja, circuitos de grande porte com números elevados de portas lógicas e de registradores, a avaliação de desempenho do sistema RTOOL será realizada a partir de *benchmarks* extensas.

Considerando-se o possível viés comercial deste projeto, mais um objetivo pode ser proposto. O sistema RTOOL deve ser de fácil implantação e de fácil utilização, ou seja, o esforço do usuário nas tarefas de integração da sua descrição RTL ao hardware do sistema RTOOL, de compilação do hardware integrado, de programação do FPGA núcleo do sistema e de utilização da API do sistema RTOOL deve ser mínimo.

2. REVISÃO DA LITERATURA

A verificação funcional de uma descrição RTL segue o modelo proposto por Bergeron, 2003, o qual é apresentado na figura 3. Neste modelo, um gerador de estímulos excita as entradas de uma descrição RTL em processo de simulação funcional. Em seguida, os sinais de saída da descrição RTL em simulação são comparados com valores admitidos corretos (calculados previamente). A descrição RTL em análise estará correta se não for detectada nenhuma diferença entre as saídas simuladas e as saídas esperadas para um conjunto abrangente de vetores de entrada.

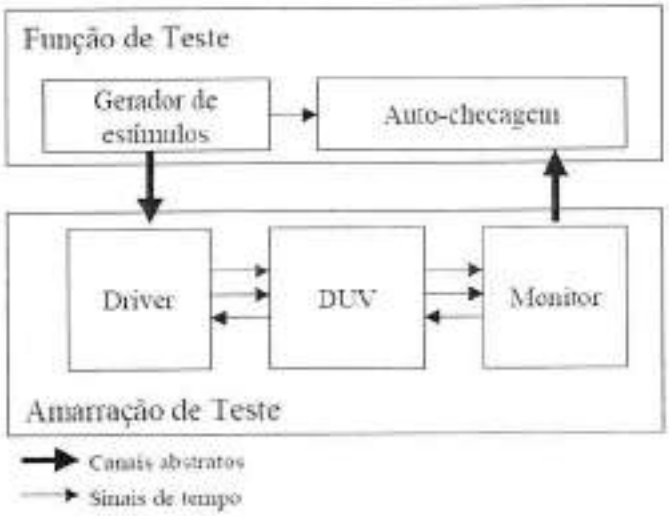

Figura 3. Modelo geral de testbench proposto por Bergeron, 2003

Note que o termo "simulação" torna-se incorreto no contexto deste projeto, porque a descrição RTL cuja funcionalidade se deseja depurar será sintetizada e em seguida prototipada sobre um FPGA. Módulos operacionais adicionais também serão introduzidos no FPGA juntamente com a sintetização da descrição RTL, de modo a permitir a interação do usuário com o protótipo em operação.

Uma visão do sistema RTOOL em camadas é apresentada na figura 4. Cada camada utiliza funcionalidades da camada imediatamente inferior, e provê funcionalidades à camada imediadamente superior. A interface de programação do sistema (camada superior do sistema RTOOL) fornecerá ao usuário todas as funcionalidades necessárias à depuração da descrição RTL sintetizada.

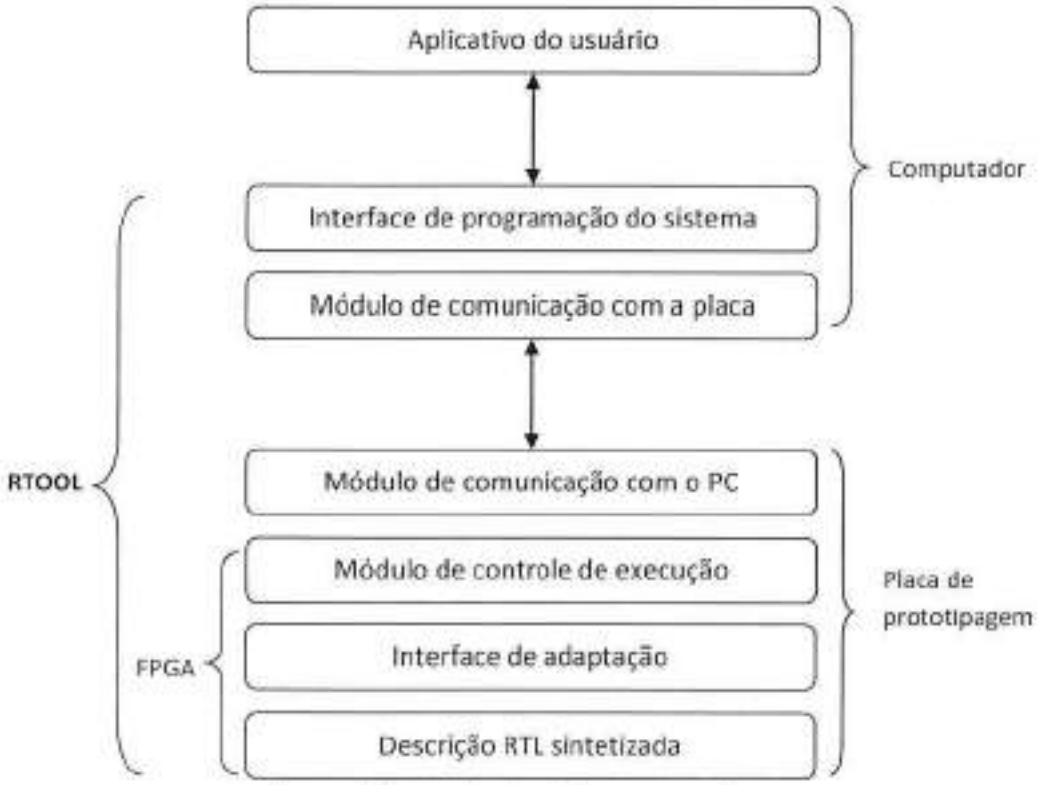

Figura 4. Camadas do sistema proposto.

Não foram encontradas soluções estruturadas no mercado de sistemas de depuração que utilizam a metodologia proposta acima, ou seja, a utilização de um FPGA como núcleo do sistema. Contudo, observou-se que tal metodologia é aplicada em casos específicos para se obter maior desempenho (Serrestou, Beroulle, Robach, 2007).

Uma arquitetura específica para a prototipagem de uma descrição RTL em um FPGA pode ser vislumbrada na figura 5. Esta arquitetura foi proposta dentro de um contexto de quantificação da qualidade de métodos de verificação funcional, ou seja, da avaliação se estes métodos são realmente eficazes na detecção de erros de hardware.

A abordagem usada para se avaliar a eficácia de métodos de verificação funcional adotada por Serrestou, Beroulle e Robach consiste, inicialmente, na introdução de erros propositais numa descrição RTL, gerando-se descrições "mutantes" (*meta-mutante*). Em seguida, é verificado se a metodologia de depuração de interesse é capaz de acusar o erro introduzido propositalmente. Idealmente, uma metodologia de depuração eficaz não deixará passar branco nenhum erro.

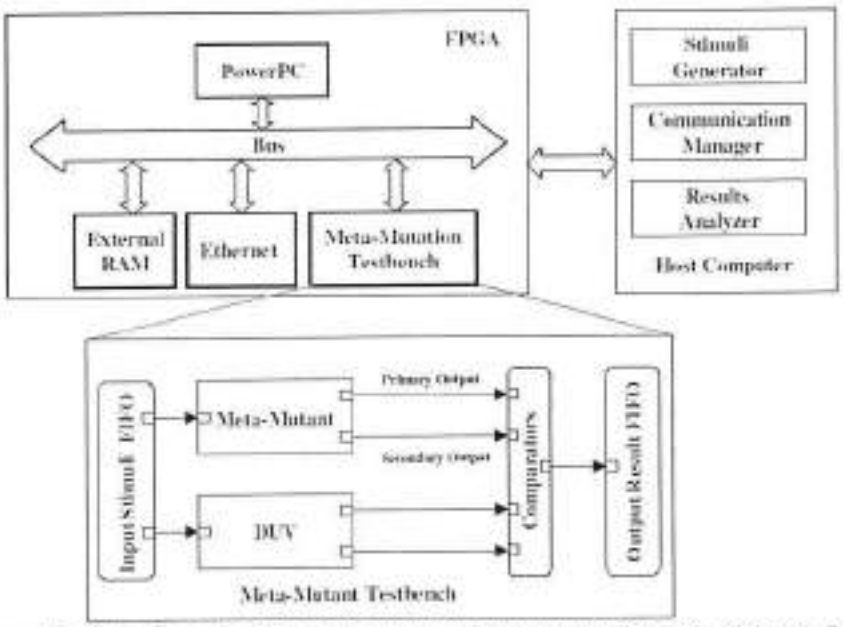

Figura 5. Solução específica proposta por Serrestou, Beroulle, Robach, 2007

A arquitetura intra-FPGA da figura 5 contém, além da sintetização da descrição RTL do usuário (com o nome de *Meta-Mutation Testbench*), um módulo de comunicação ethernet, uma memória RAM externa e um processador para gerenciamento da emulação. Todos estes elementos estão interligados por um barramento interno de comunicação (*Bus*).

Apesar da solução específica da figura 5 conter todos os elementos de hardware necessários à prototipagem de uma descrição RTL, alguns pontos dificultam sua reutilização em outros projetos:

- A integração de uma nova descrição RTL com entradas/saídas distintas da original requer o desenvolvimento manual de um hardware de adaptação desta nova descrição RTL ao resto do sistema.

- O software embarcado do processador intra-FPGA necessitaria ser modificado para fornecer suporte à nova descrição RTL, a qual possui portos de entrada/saída distintos daqueles do projeto original.

- Uma modificação deveria ser feita no driver gerenciador da comunicação, o qual é executado no computador, para torná-lo compatível com o novo software embarcado.

Vale lembrar que, para se efetuar todas estas modificações, é necessário estudar a fundo a arquitetura específica em questão. Todos estes empecilhos tornam o reúso de uma solução específica inviável, dado o tamanho do esforço que seria empregado na adaptação da solução específica a um novo projeto.

Algumas propostas de sistemas de depuração com hardware dedicado encontradas na literatura (Tsai, 1994) propõem o desenvolvimento de placas multiprocessadoras para acelerar a simulação HDL. Contudo, tais soluções demandam o desenvolvimento de complicados softwares de interpretação e execução HDL que busquem tirar proveito de sistemas multiprocessados, e, mesmo assim, conclui-se que não há garantias sólidas de maior desempenho.

3. MATERIAIS E MÉTODOS

A implementação do hardware do sistema RTOOL foi feita a partir da utilização de um kit de desenvolvimento DE-2 da ALTERA, o qual possui os seguintes recursos de interesse para este projeto:

- FPGA Cyclone II EP2C35F672C6 com 35 mil elementos lógicos
- Memórias externas SDRAM com capacidade de 8 MB de dados
- USB-Blaster embutido para Entrada/Saída de dados do FPGA

A figura 6 mostra uma foto do kit de desenvolvimento DE-2 da ALTERA, destacando o conector USB no canto superior esquerdo, a memória SDRAM com capacidade de 8 MB ao centro e o FPGA Cyclone II mais à direita.

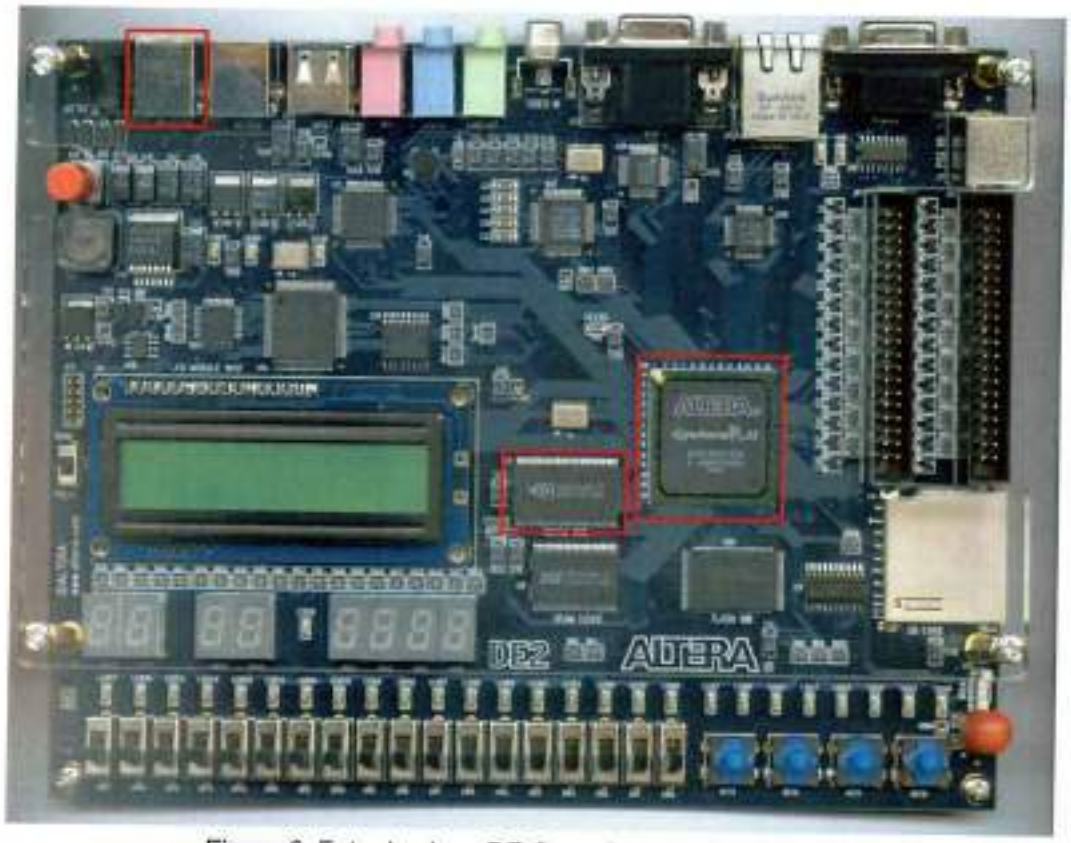

Figura 6. Foto da placa DE-2 usada no projeto RTOOL

A implementação do software do sistema RTOOL será feita em estações de trabalho compatíveis com a tecnologia atual, i.e, estações de trabalho com processadores de frequência de operação em torno de 2 GHz e utilizando-se o sistema operacional Microsoft Windows XP ou Microsoft Windows 7. Sobre estas estações de trabalho serão instalados os seguintes softwares:

- Quartus II, da Altera, para o desenvolvimento de projetos em VHDL
- Nios II Software Build Tools for Eclipse, para o desenvolvimento de software embarcado
- Compilador GNU GCC, para o desenvolvimento das ferramentas de usuário
- Microsoft Visual Studio, para o desenvolvimento de interfaces gráficas para as ferramentas de usuário

Quando finalizado, a forma de utilização do sistema RTOOL pelo usuário ocorrerá como apresentado na figura 7. Inicialmente tem-se um gerador de estímulos responsável por estimular as entradas do sistema RTOOL e as entradas do modelo de referência (tido como livre de erros). Após o processamento dos dados, as saídas do sistema RTOOL e as saídas do modelo de referência são comparadas em um bloco entitulado *Checker*. Após comparar ambas as saídas, o bloco *Checker* realimentará o gerador de estímulos com os resultados, de modo a se efetuar uma verificação abrangente.

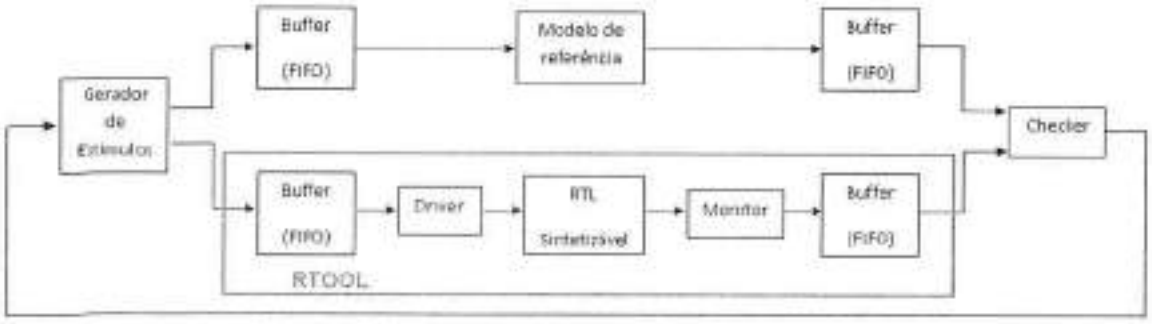

Figura 7. Diagrama de blocos do sistema de verificação

A seguir tem-se um maior detalhamento da função de cada componente do diagrama:

Gerador de estímulos: O gerador de estímulos gera inicialmente entradas aleatórias utilizadas na verificação da correção do RTL. Entretanto, devido à realimentação do *Checker*, o gerador de estímulos poderá gerar sinais de entrada para determinadas condições de teste que ainda não foram verificadas. Apenas dados ou sinais relevantes representativos do comportamento do sistema são gerados, sendo sinais mais específicos de protocolo, como clock, enable, request, etc, desconsiderados.

Buffer (FIFO): O buffer será do tipo FIFO (*First In First Out*) e será implementado tanto na entrada dos blocos do modelo de referência e do RTL, quanto na saída destes. A principal função destes blocos é de casar os sinais síncronos, provenientes do RTL, com os assíncronos provenientes do modelo de referência.

Modelo de referência: O modelo de referência é um programa previamente compilado e testado que executa as mesmas tarefas das que são pretendidas pelo projeto do RTL, servindo, portanto, como base de comparação para a realização de testes. Assume-se que o modelo de referência é uma descrição sem erros do comportamento do circuito. Um fato importante é que o modelo de referência, por ser uma descrição funcional em linguagens de programação de alto nível (C, C++, Java, por exemplo), utiliza-se de algoritmos e modelos matemáticos altamente eficientes para calcular suas saídas, de modo a requerer poucos recursos de processamento.

Driver: O driver é o componente responsável pela conversão do sinal de entrada em vetores de testes que serão utilizados pelo RTL. Além disso, é responsável pela geração dos sinais de controle da interface de entrada do RTL, sendo, por isso, bastante específico e variando de acordo com cada RTL que seja testado.

Monitor: Faz o papel inverso do driver, convertendo os vetores de teste da saída do RTL em sinais compatíveis com a entrada do *Checker*.

Checker: É o bloco que fará a comparação dos sinais de saída gerados pelo modelo de referência com o RTL e que fornece uma realimentação ao gerador de estímulos para que este possa excitar o sistema de uma forma abrangente.

Para que um projetista digital não tenha de se adequar a uma nova metodologia de depuração, o sistema RTOOL será desenvolvido de modo a atender aos requisitos impostos pelo ambiente de verificação de corretude funcional de uma descrição RTL apresentado acima.

3.1 Estrutura analítica de projeto

O projeto RTOOL, no que diz respeito à sua implementação, foi dividido em cinco grupos principais de tarefas, como é apresentado na EAP (Estrutura Analítica de Projeto) da figura 8.

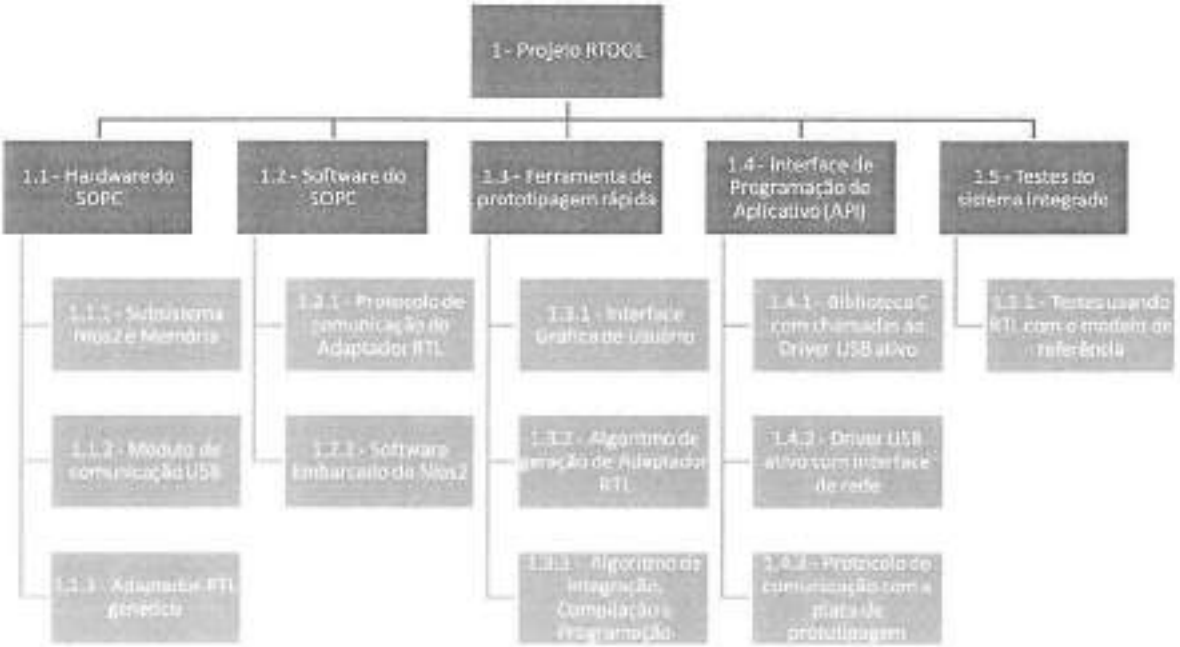

Figura 8. EAP do projeto RTOOL

É possível observar que, dentre as tarefas listadas na EAP, cerca de 25%, apenas, correspondem ao desenvolvimento de hardware, e o restante ao desenvolvimento de software. Contudo, a maior complexidade observada no desenvolvimento de hardware acarreta em um maior gasto de tempo nestas tarefas. Logo, é justo considerar uma divisão de trabalho no grupo de dois integrantes do projeto RTOOL na qual um focará mais atentamente no hardware do sistema, enquanto o outro focará mais atentamente no software do sistema.

3.1.1 Detalhamento da EAP

Para que o leitor tenha um melhor aproveitamento em sua leitura da seção "3.2 – Implementação do sistema RTOOL", a seguir é fornecido um detalhamento das tarefas da EAP da figura 8.

- **Hardware do SOPC:**

  Este grupo de tarefas corresponde ao desenvolvimento da arquitetura de hardware interna do FPGA. Nela, haverá um processador NIOS II responsável pelo gerenciamento dos demais periféricos, um módulo de comunicação USB, uma memória SRAM interna, uma memória SDRAM externa e um Adaptador RTL (descrito abaixo). Todas estas entidades estarão conectadas entre si pelo barramento AVALON.

- Subsistema NIOS II e memória: O processador NIOS II da ALTERA deverá estar conectado a uma memória SDRAM para que seja possível armazenar localmente os vetores de dados provenientes do circuito RTL do usuário em processo de emulação. A memória SRAM interna será usada apenas para guardar o software embarcado.

- Módulo de comunicação USB: A comunicação entre o computador, no qual é fornecida a API do sistema RTOOL, e a placa de prototipagem será feita através da interface USB, já que a interface serial RS-232 vem se tornando cada vez mais obsoleta.

- Adaptador RTL: Esta entidade tem por objetivo interligar a descrição RTL do usuário ao sistema RTOOL, funcionando como um "soquete" no qual o hardware do usuário é "encaixado". O Adaptador RTL fornece uma interface de comunicação assíncrona padronizada de modo a permitir uma diferenciação entre o hardware do RTOOL e o hardware do usuário no que diz respeito à domínios de *clock*.

➢ Software do SOPC:

Este grupo de tarefas corresponde à todos os aspectos do software interno ao FPGA, desde a definição de protocolos de comunicação até o desenvolvimento do firmware do processador NIOS II.

- Protocolo de comunicação do Adaptador RTL: A estrutura de hardware do Adaptador RTL foi desenvolvida de modo a fornecer ao processador NIOS II (mestre do barramento AVALON) funcionalidades de leitura/escrita de dados, gerenciamento de interrupções e configuração da frequência do sinal de *clock* que alimenta o dispositivo RTL do usuário. A comunicação entre estas duas entidades é feita através de um mapa de registradores que controlam a operação do Adaptador RTL.

- Software Embarcado do NIOS II: O software embarcado do processador NIOS II tem por objetivo gerenciar os diversos processos que ocorrem na placa. São eles: processo de leitura/escrita do PC, processo de leitura do DRIVER e processo de escrita do MONITOR.

➢ Ferramenta de prototipagem rápida:

Um dos aspectos mais importantes no desenvolvimento de um software é o nível de dificuldade que o usuário irá enfrentar ao usar o mesmo. Quanto mais fácil e intuitiva for a utilização de um software, maiores serão suas chances de ter sucesso no mercado. A ferramenta de prototipagem rápida visa facilitar ao

máximo a tarefa de implantação do sistema RTOOL para sua posterior utilização. Tarefas repetitivas como a integração da descrição RTL do usuário ao sistema, compilação do sistema e programação do FPGA foram automatizadas por esta ferramenta, a qual possui uma interface gráfica de usuário amigável.

- Interface Gráfica de Usuário: Consiste de uma interface gráfica de usuário amigável para conduzir o usuário durante o processo de implantação do sistema RTOOL.

- Algoritmo de geração de Adaptador RTL: A versão corrente do sistema RTOOL possui uma estrutura fixa para o Adaptador RTL, a qual fornece uma interface de comunicação assíncrona para o hardware do usuário. Logo, esta tarefa prevista no cronograma do projeto tornou-se desnecessária de acordo com a nova metodologia adotada.

- Algoritmo de Integração, Compilação e Programação: Tarefas repetitivas como a integração da descrição RTL do usuário ao sistema, compilação do sistema e programação do FPGA e do software embarcado foram automatizadas a partir deste algoritmo.

➢ Interface de Programação de Aplicativo:

Após a implantação do sistema RTOOL pelo usuário, i.e., após a prototipagem de uma descrição RTL cuja corretude se deseja verificar, a interação do usuário com o sistema RTOOL será feita a partir de uma biblioteca de linguagem C que fornecerá todas as funcionalidades necessárias à depuração do circuito digital do usuário. Além disso, as funções da biblioteca em linguagem C podem ser facilmente integradas a uma descrição de hardware em linguagem SystemC, permitindo a co-simulação do um modelo de referência em linguagem SystemC e do código RTL sintetizável em processo de emulação no FPGA.

- Biblioteca C com chamadas ao Driver USB ativo: Definição das funções que serão disponibilizadas para o usuário do sistema RTOOL e implementação destas funções através da comunicação com o Driver USB Ativo (o qual se comunica diretamente com a placa de prototipagem).

- Driver USB ativo com interface de rede: Optou-se pelo desenvolvimento de um Driver USB ativo em modo usuário, em detrimento ao modo *kernel*, pela maior facilidade de implementação. Este Driver USB Ativo irá possuir a forma de um processo num sistema operacional, e a sua comunicação com a biblioteca C que será integrada a um aplicativo do usuário será feita via interface de rede TCP/IP. A comunicação entre processos via interface

de rede é uma alternativa de mais fácil implementação à forma tradicional que usa "*pipes*" do sistema operacional.

- Protocolo de comunicação com a placa de prototipagem: Este protocolo de comunicação foi definido na tarefa "1.2", e sua implementação é compatível com as especificações ali determinadas. Em resumo, este protocolo estabelece quais serviços são oferecidos ao usuário pela placa de prototipagem e como estes serviços devem ser requisitados através da comunicação via interface USB.

➢ **Testes do Sistema Integrado:**

Esta é a etapa final e uma das mais importantes do projeto. Requer controle de tempo rigoroso e exaustivos testes para se ter certeza da corretude do sistema. Será utilizado como modelo de teste um modelo de referência seguido do seu RTL equivalente.

- Testes utilizando o RTL com o modelo de referência: Os testes realizados nessa etapa serão feitos utilizando-se como base um modelo de referência, supostamente correto, que será comparada ao RTL equivalente que também será admitido como correto. Para isso a equipe pretende elaborar modelos e RTL simples para efetuar os testes iniciais. Após validado o sistema RTOOL, será feita a medida de seu desempenho para uma comparação com as ferramentas de simulação mais usadas na indústria.

3.2 Implementação do sistema RTOOL

A figura 9 mostra um diagrama de blocos detalhado do objetivo, do ponto de vista da implementação, do sistema RTOOL, com todas as entidades previamente apresentadas.

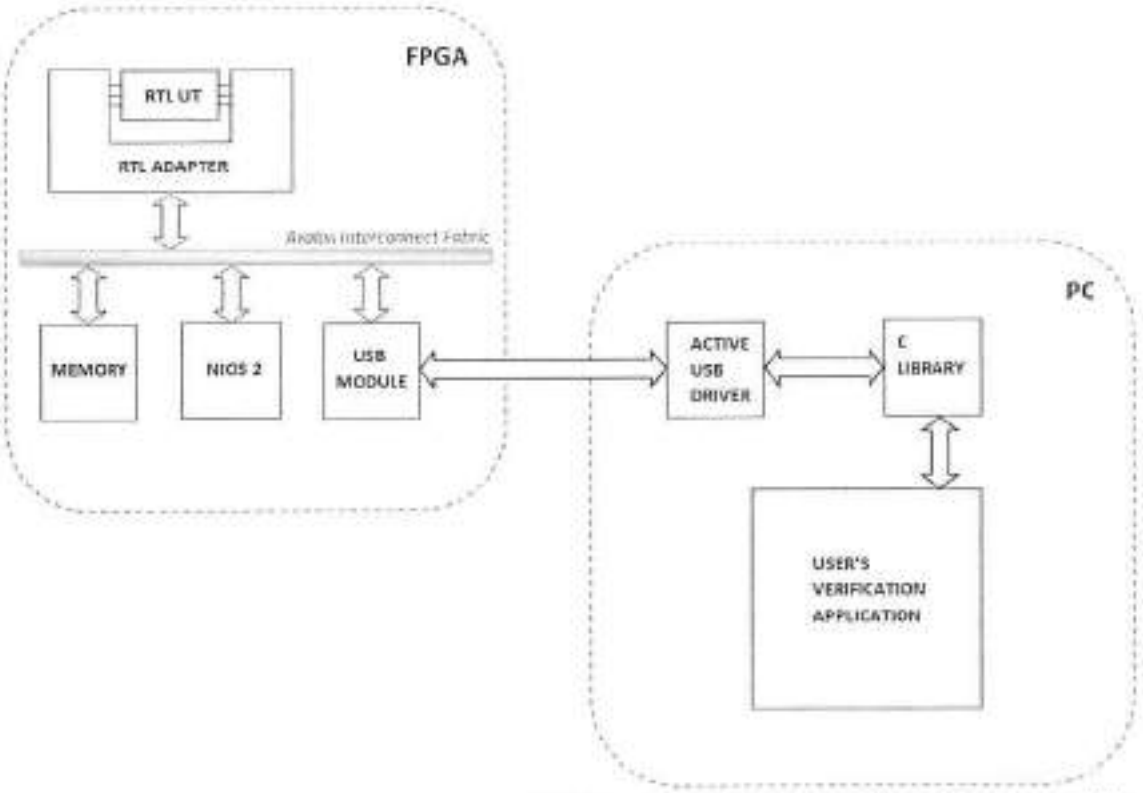

Figura 9. Diagrama de blocos do sistema RTOOL. Os blocos em vermelho são responsabilidade do usuário

Os detalhes pertinentes à forma pela qual cada submódulo do sistema RTOOL foi implementado são fornecidos nas subseções que seguem.

3.2.1 Hardware do SOPC

Este capítulo descreve a implementação dos três módulos de hardware constituintes do SOPC do sistema RTOOL. Serão abordados: O subsistema Nios II e memória SDRAM externa; O módulo de comunicação USB; O adaptador RTL.

3.2.1.1 Subsistema Nios II e memória SDRAM externa

Para se implementar o subsistema Nios II e memória SDRAM externa utilizou-se o software QUARTUS II da ALTERA com o auxilio de seu módulo interno QSYS, o qual permite a construção de um SOPC através da interligação de componentes de hardware sobre um barramento de dados.

Os principais componentes deste subsistema são:

- Processador NIOS II *Standard* com 20 KB de *Onchip Memory*

- Controlador de memória SDRAM compatível com o barramento AVALON
- Memória SDRAM externa ao FPGA com 8 MB de dados
- PLL (*Phase Locked Loop*)

O processador NIOS II, categorizado como "*softcore*" pelo fato de ter sido desenvolvido completamente numa HDL, corresponde a uma arquitetura de segunda geração de processadores embarcados de 32 bits voltados especificamente à família de FPGAs da ALTERA.

Segundo a Gartner Research, o processador NIOS II é o *soft processor* mais amplamente usado na indústria de FPGAs. Ele também é um processador muito versátil, podendo ser configurado minunciosamente para atender aos requisitos de cada aplicação, sejam eles requisitos de desempenho (Nios II *fast core*), de confiabilidade (Nios II *safe critical core*), de uso de recursos (Nios II *economy core*), entre outros.

A configuração adotada para o sistema RTOOL foi a "*standard core*", a qual busca otimizar tanto o desempenho do processador quanto a sua utilização de recursos do FPGA. A figura 10 mostra um diagrama simplificado do subsistema construído e os periféricos selecionados.

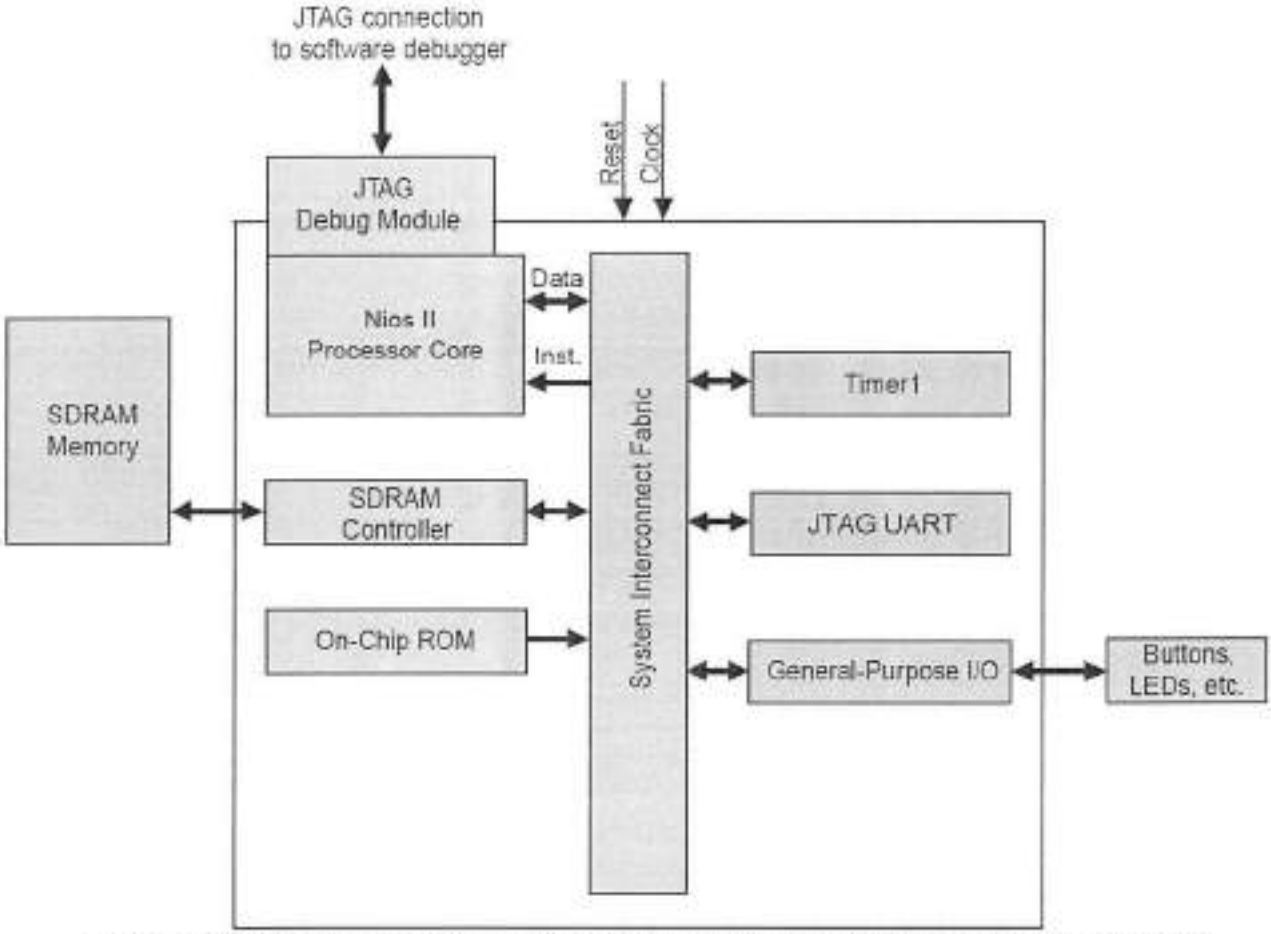

Figura 10. Sistema Nios II construído para o projeto RTOOL (Modificado de: *Nios II Processor Reference Handbook*, pág. 11)

Pode-se observar que a comunicação entre o processador NIOS II e seus periféricos é feita através de um barramento de dados entitulado "*System Interconnect Fabric*". Este barramente, de 32 bits de comprimento, também chamado de barramento AVALON, é especificado pela ALTERA para que projetistas de hardware possam desenvolver dispositivos compatíveis com o processador NIOS II.

O controlador de memória SDRAM usado tem duas funcionalidades principais. Primeiro, como todos os controladores de memória SDRAM disponíveis no mercado, ele executa ciclos periódicos de atualização (*refresh*) da memória SDRAM para que os dados guardados na mesma sejam preservados. Segundo, ele compatibiliza a interface de leitura/escrita da memória SDRAM à interface de leitura/escrita do barramento AVALON. A figura 11 mostra os detalhes do controlador.

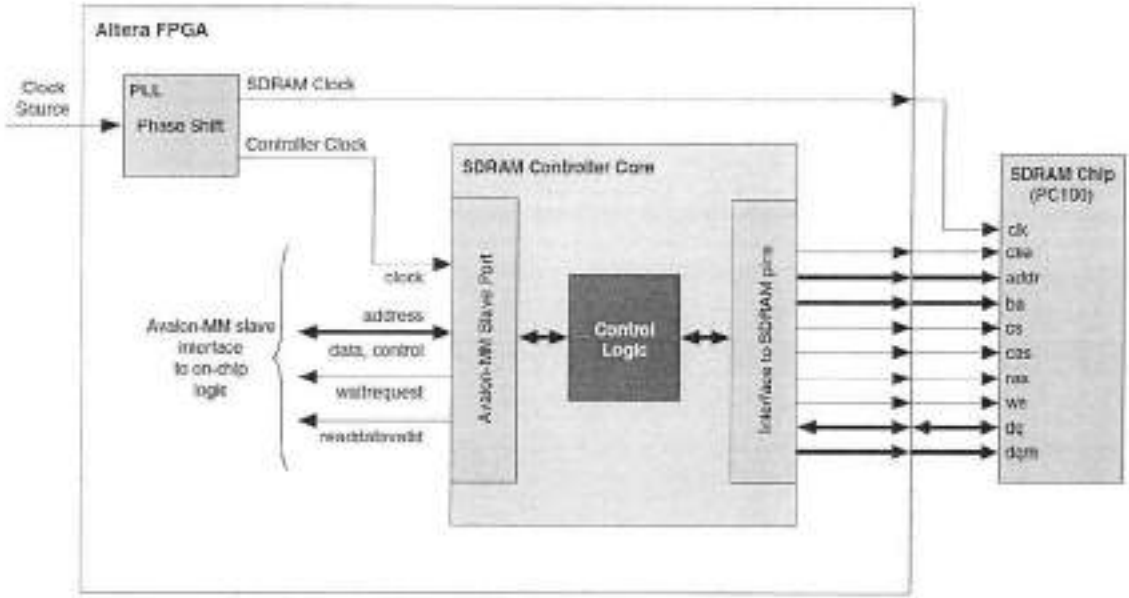

Figura 11. Controlador de memória SDRAM (Fonte: *Embedded Peripherals IP User Guide*, pág. 24)

Na figura 11 também está presente o PLL, que tem o intuito de ajustar a defasagem entre o sinal de clock que alimenta o controlador de memória SDRAM e o sinal de clock que alimenta a própria memória SDRAM. Este ajuste depende do layout da placa de circuito impressa na qual estão soldados o FPGA e a memória SDRAM, e ele visa compensar o atraso imposto pelo comprimento das trilhas que interligam estes dois componentes.

Optou-se pelo uso da memória SDRAM como meio de armazenamento maciço de dados levando-se em conta a sua capacidade de memória (8 MB). Isto permitiu reduzir a utilização de recursos do FPGA, tornando o hardware do RTOOL mais enxuto.

Para projetos mais simples, seria possível utilizar também a memória interna do FPGA para o propósito de armazenagem em detrimento de uma maior utilização de área do FPGA. No caso deste projeto a memória interna será utilizada somente com o propósito de armazenar o software embarcado do NIOS II.

LEDs de indicação de status também foram adicionados ao hardware do sistema para facilitar o processo de depuração do software embarcado.

3.2.1.1.1 Software de gerenciamento da memória SDRAM

O projeto RTOOL prevê que a memória SDRAM externa será utilizada para armazenar vetores de dados de entrada/saída do dispositivo RTL que será emulado. Logo, o software de gerenciamento da memória divide a mesma em duas regiões, de acordo com a figura 12.

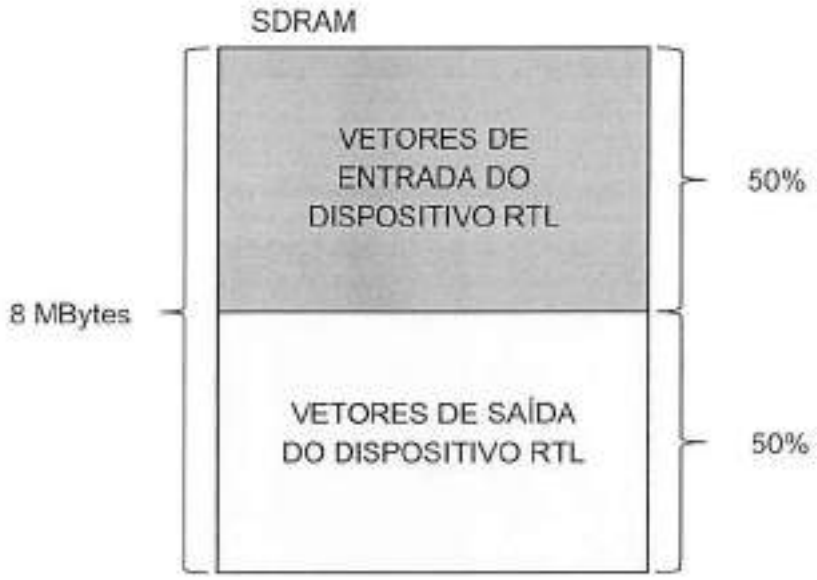

Figura 12. Divisão da Memória SDRAM

Cerca de 4 MB de dados foram reservados para o armazenamento de vetores de entrada do dispositivo RTL (enviados pelo PC e requisitados pelo DRIVER), e cerca de outros 4 MB foram reservados para o armazenamento de vetores de saída do dispositivo RTL (enviados pelo MONITOR e requisitados pelo PC).

Cada uma das regiões de memória é tratada idependentemente da outra como um *buffer* FIFO circular, ou seja, sempre que, em uma operação de escrita/leitura, se atingir a última posição de uma região de memória, o endereço de escrita/leitura seguinte irá retornar à posição inicial desta região de memória.

Este tipo de tratamento da memória é adequado para o sistema RTOOL porque os vetores de dados são acessados sequencialmente.

Em pseudo-código, as funções de escrita/leitura de uma região memória tratada como um *buffer* FIFO circular seriam:

```
/* Definição das dimensões do buffer */
#define TOPO_DA_REGIAO    0x00600000
#define FUNDO_DA_REGIAO   0x00000000
#define TAMANHO_DA_REGIAO 0x00600000

/* Indica o número de dados que podem ser lidos do buffer */
int DadosDisponiveis()
{
    int dif;
    dif = ponteiro_de_escrita - ponteiro_de_leitura;
    if( ponteiro_de_escrita > ponteiro_de_leitura )
        return dif;
    else
        return TAMANHO_DA_REGIAO + dif;
```

```
}

/* Indica o número de dados que podem ser escritos no buffer */
int EspacosDisponiveis()
{
    return TAMANHO_DA_REGIAO - DadosDisponiveis();
}

/* Insere um dado no buffer */
void EscreveDado( unsigned int dado )
{
    *ponteiro_de_escrita = dado;
    *ponteiro_de_escrita = *ponteiro_de_escrita + 1;
    if( ponteiro_de_escrita > TOPO_DA_REGIAO )
        ponteiro_de_escrita = FUNDO_DA_REGIAO;
}

/* Lê um dado do buffer */
unsigned int LeDado()
{
    unsigned int dado;
    dado = *ponteiro_de_leitura;
    *ponteiro_de_leitura = *ponteiro_de_leitura + 1;
    if( ponteiro_de_leitura > TOPO_DA_REGIAO )
        ponteiro_de_leitura = FUNDO_DA_REGIAO;
    return dado;
}
```

A partir destas funções é possível implementar-se um *buffer* FIFO circular independente em cada uma das regiões de memória. Contudo, no pseudo-código apresentado faz-se necessária uma verificação de memória cheia ou memória vazia antes de se efetuar operações de escrita ou leitura, respectivamente, para que não haja *overflow* ou *underflow*.

Apesar da memória SDRAM externa ter comprimento de palavra de apenas 16 bits, o controlador da memória SDRAM, por causa de sua arquitetura de hardware, permite que operações de escrita/leitura com palavras de 32 bits de comprimento sejam efetuadas pelo processador NIOS II. Desta forma, o processador NIOS II consegue, em média, reduzir pela metade o uso do barramento AVALON para o acesso à memória.

O uso da memória SDRAM pelo sistema RTOOL será retomado mais adiante quando for tratado o software do SOPC e sua a máquina de estados.

3.2.1.2 MÓDULO DE COMUNICAÇÃO USB

A solução adotada para a comunicação USB entre a placa de prototipagem e o computador (PC) foi a utilização de um núcleo JTAG UART compatível com o barramento AVALON.

Assim como em uma linguagem convencional de programação de software existem bibliotecas com funções pré-definidas que podem ser aproveitadas por um programador durante o desenvolvimento de sua aplicação, em linguagens de programação de hardware também existem núcleos funcionais pré-definidos que podem ser integrados ao projeto de um hardware qualquer. O núcleo JTAG UART, desenolvido pela Altera, foi escolhido pelo grupo para ser integrado ao projeto do RTOOL devido à sua praticidade.

A figura 13, extraída do manual do fabricante, apresenta uma visão geral das funcionalidades que a utilização do núcleo JTAG UART permite introduzir ao sistema:

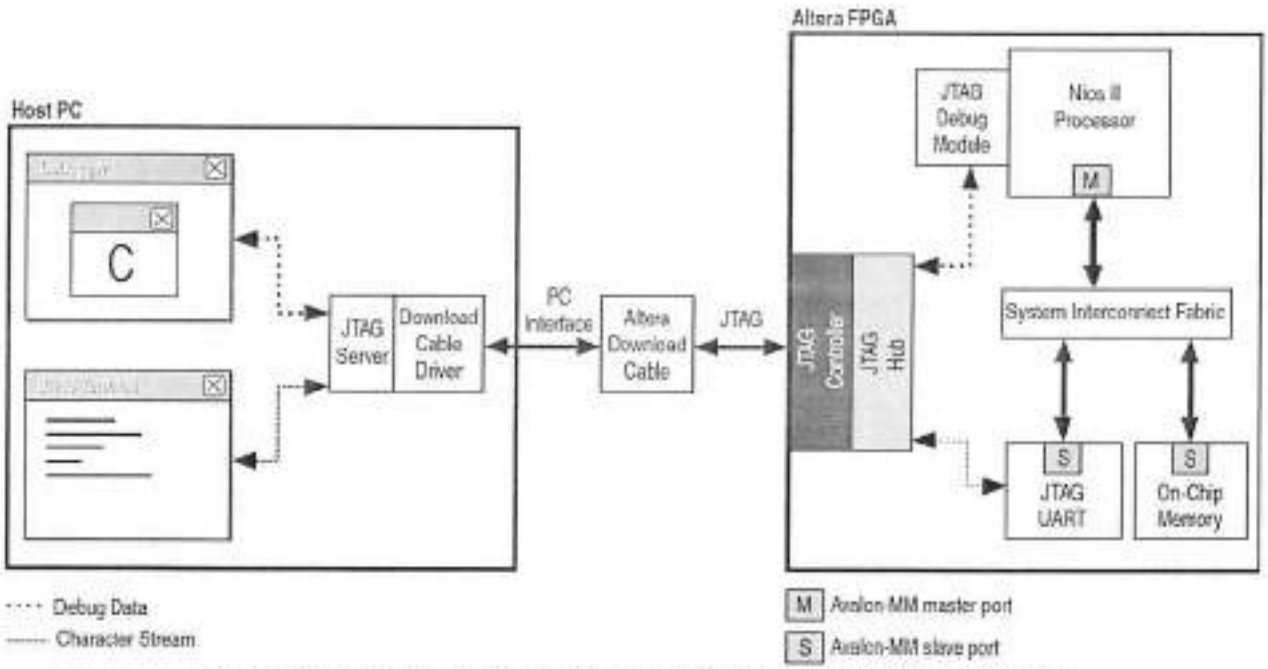

Figura 13. Diagrama de blocos (Fonte: Manual JTAG UART core, pág. 3)

O núcleo JTAG UART atuará como um escravo do barramento AVALON, e permitirá que o processador NIOS II envie um fluxo de dados à uma aplicação em execução no PC. O caminho que este fluxo de dados irá percorrer até atingir seu destino final é mostrado na figura 13, e é descrito sucintamente a seguir.

Logo após ter sido enviado ao núcleo JTAG UART, o fluxo de dados será encaminhado ao controlador JTAG (JTAG Controller). Em seguida, o controlador JTAG irá transmitir o fluxo de dados ao PC através de um cabo de download da

Altera (Altera Download cable), que consiste de um cabo USB. Ao chegar ao computador através do cabo USB, o fluxo de dados será armazenado em um buffer de entrada do driver do cabo de download (Download Cable Driver). Este driver atuará como um servidor (JTAG Server), fornecendo uma API para permitir que a aplicação do usuário receba o fluxo de dados armazenado no buffer temporário. Ao ser lido pela aplicação final, o fluxo de dados originado no processador NIOS II termina seu percurso.

Caso a aplicação no PC deseje enviar um fluxo de dados ao NIOS II, o percurso que este fluxo irá percorrer até seu destino final será o mesmo descrito para o caso do processador NIOS II enviar um fluxo de dados ao PC, porém na ordem contrária. A aplicação do PC irá inicialmente enviar o fluxo de dados ao servidor JTAG, o qual encaminhará os dados, através do cabo de download USB da Altera, ao controlador JTAG presente na placa de prototipagem. O controlador JTAG, por sua vez, enviará o fluxo de dados ao núcleo JTAG UART onde os dados serão armazenados em um buffer de entrada, do qual poderão ser recuperados pela aplicação em execução no processador NIOS II através de uma operação de leitura no barramento Avalon.

Todos os componentes presentes no subsistema JTAG, sendo eles o núcleo JTAG UART, o controlador JTAG, o cabo de download da Altera, o driver do cabo de download e o servidor JTAG não necessitam ser implementados, ou seja, eles são fornecidos pela ALTERA e estão prontos para serem usados. Consequentemente, pelo fato destes componentes não precisarem ser desenvolvidos neste projeto, um tempo valioso foi economizado. Além disso, a solução sendo adotada possui alta confiabilidade por ser amplamente utilizada por projetistas que utilizam FPGAs da ALTERA (esta solução só é disponibilizada para FPGAs da ALTERA).

No lado do processador NIOS II, a utilização do núcleo JTAG UART será feita através de macros de acesso ao barramento AVALON, como apresentado no código abaixo:

```
/* Macro para leitura do barramento Avalon */
unsigned int IORD( void* BASE_ADDRESS, void* OFFSET );

/* Macro para escrita no barramento Avalon */
void IOWR( void* BASE_ADDRESS, void* OFFSET, unsigned int VALUE );
```

Sendo BASE_ADRRESS o endereço do periférico do barramento Avalon que se deseja acessar, OFFSET uma variação de endereço relativa ao endereço base e VALUE o valor que se deseja transmitir pelo barramento. Como o processador NIOS II possui uma arquitetura de 32 bits, o parâmetro VALUE é do tipo "unsigned int". Contudo, transações de variáveis de um byte ou de dois bytes também são permitidas.

Esta forma de acesso à periféricos é típica de dispositivos mapeados em memória, ou seja, dispositivos cujas operações de escrita e leitura realizadas pelo processador são análogas à operações de acesso à memória.

O núcleo JTAG UART possui um conjunto de registradores que controla seu funcionamento. O mapa de registradores do núcleo JTAG UART é apresentado na tabela 1.

Tabela 1. Mapa de registradores (Fonte: Manual JTAG UART core, pág. 12)

Offset	Register Name	R/W	Bit Description															
			31	...	16	15	14	...	11	10	9	8	7	...	2	1	0
0	data	RW	RAVAIL			RVALID	RESERVED						DATA					
1	control	RW	WSPACE			RESERVED				AC	WI	RI	RESERV.			WE	RE	

É importante observar que o campo DATA, que representa o topo dos buffers FIFO de entrada e de saída deste núcleo, possui apenas 8 bits. Isto significa que o núcleo JTAG UART trabalha com variáveis de um byte. Outros campos importantes são RAVAIL, que contém o número de bytes disponíveis para leitura no buffer FIFO de entrada, RVALID, que indica se o campo DATA é válido e WSPACE, que representa o espaço disponível para escrita no buffer de saída (em bytes). Os demais campos estão relacionados à configuração e execução de interrupções.

No lado do PC o acesso ao subsistema JTAG é feito através de uma API de comunicação com o servidor JTAG. A API em questão possui o nome "JTAG ATLANTIC", e pode ser encontrada na biblioteca dinâmica "jtag_atlantic.dll". As principais funções desta biblioteca são mostradas no código abaixo:

```
/* Estrutura que mantém informações sobre o estado da comunicação JTAG */
typedef struct JTAGATLANTIC JTAGATLANTIC;

/* Abre o canal de comunicação */
JTAGATLANTIC* jtagatlantic_open( const char *chain, int device_index, int
link_instance, const char *app_name );

/* Fecha o canal de comunicação */
void jtagatlantic_close( JTAGATLANTIC *link );

/* Envia dados pelo canal de comunicação aberto */
int jtagatlantic_write( JTAGATLANTIC *link, char * buf, unsigned int buflen );

/* Recebe dados do canal de comunicação aberto */
int jtagatlantic_read( JTAGATLANTIC *link, char *buf, unsigned int buflen );

/* Indica o número de bytes disponíveis para leitura */
int jtagatlantic_bytes_available( JTAGATLANTIC *link );
```

3.2.1.3 ADAPTADOR RTL

O adaptador RTL genérico é o módulo de hardware responsável pela interligação ao sistema RTOOL de um dispositivo RTL qualquer que se deseje depurar. De um lado o adaptador RTL se comunicará com o processador NIOS II através do barramento AVALON, enquanto que do outro lado o adaptador RTL se conectará diretamente às interfaces de entrada e de saída do dispositivo RTL que será depurado. O diagrama da figura 14 mostra como o adaptador RTL se encaixa no contexto do projeto RTOOL.

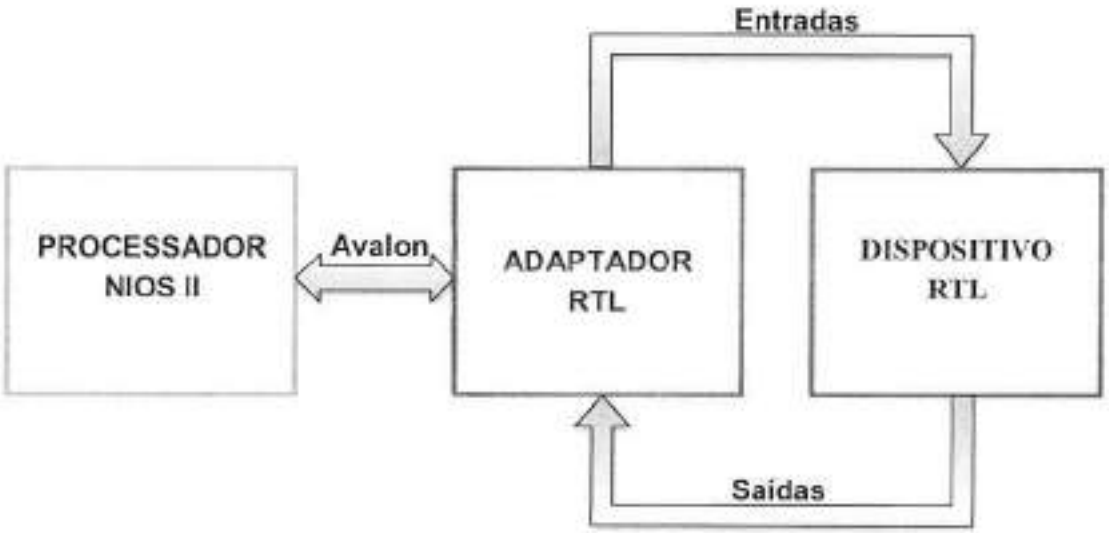

Figura 14. Diagrama do contexto no qual o adaptador RTL está inserido

O primeiro passo tomado rumo à implementação do adaptador RTL foi o estudo da especificação do barramento AVALON (disponibilizada pela ALTERA), para poder se determinar quais sinais seriam convenientes a este dispositivo de acordo com as funcionalidades que ele deve oferecer.

3.2.1.3.1 Barramento AVALON

O barramento AVALON foi desenvolvido pela Altera para se efetuar a interconexão de dispositivos em um sistema integrado construído sobre um FPGA. São suportados sete tipos diferentes de interfaces sobre o barramento AVALON, de modo a fornecer todas as funcionalidades que um barramento de dados convencional possui. As interfaces de interesse ao Adaptador RTL são descritas sucintamente na tabela 2. Na primeira coluna tem-se o nome da interface, enquanto na segunda coluna é fornecida sua descrição.

Tabela 2. Descrição das interfaces selecionadas para o Adaptador RTL (Fonte: *Avalon Interface Specification*, pág. 5)

Interface	Descrição
Avalon Memory Mapped Interface	Típica interface de leitura/escrita baseada em endereçamento.
Avalon Conduit Interface	Conjunto de sinais adicionais, não definidos na especificação do barramento Avalon, que podem ser usados para a conexão com um dispositivo externo ao SOPC.
Avalon Interrupt Interface	Interface para a implementação da funcionalidade de interrupção. Por exemplo, um periférico pode requisitar uma interrupção ao processador.
Avalon Clock Interface	Necessária à sincronização dos dispositivos conectados ao barramento Avalon. Todas as interfaces do barramento são síncronas.
Avalon Reset Interface	Permite que seja enviada uma requisição de reinicialização a um dispositivo conectado ao barramento.

Os sinais de entrada/saída especificados pelo barramento AVALON são flexíveis, ou seja, um dispositivo compatível com o barramento Avalon pode apresentar um número variável de interfaces, de acordo com suas necessidades.

A figura 15 mostra como o Adaptador RTL é interligado ao barramento AVALON, de acordo com o conjunto de interfaces selecionadas.

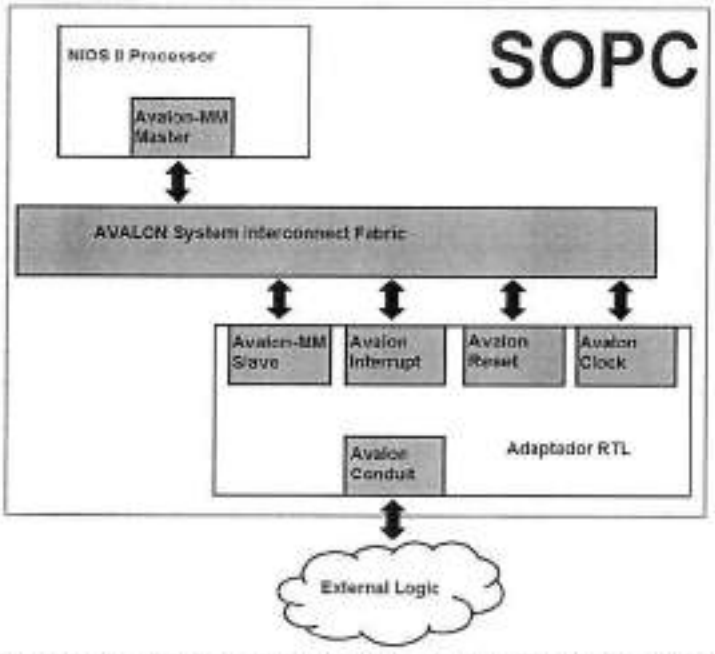

Figura 15. Diagrama de interligação entre o Adaptador RTL e o barramento AVALON

Note que a interface AVALON Conduit permite que dispositivos externos se comuniquem com o SOPC. No contexto do sistema RTOOL, o circuito lógico externo corresponde ao dispositivo RTL do usuário.

A sequência de sinalização necessária à realização de uma transferência de dados, tanto para uma operação de leitura como para uma operação de escrita, sobre o barramento Avalon é apresentada na figura 16.

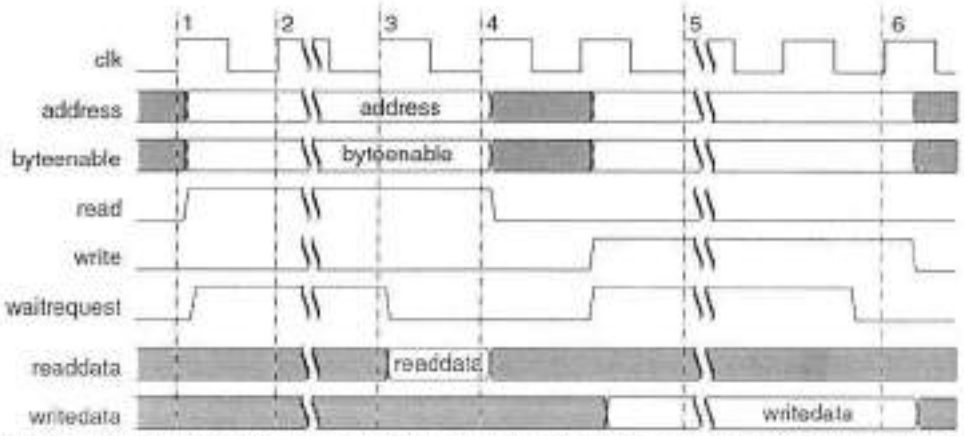

Figura 16. Carta de tempos do barramento Avalon (Fonte: *Avalon Interface Specification*, pág. 22)

Na primeira borda de subida do clock (1) tanto o sinal "write" como o sinal "read" estão em nível baixo, o que indica que o mestre do barramento não deseja realizar nenhuma operação naquele momento. Já na segunda borda de subida (2) pode-se perceber que o sinal "read" assumiu nível lógico alto, o que indica que o mestre do barramento deseja efetuar uma leitura. Contudo, devido ao fato de o escravo ter colocado o sinal "waitrequest" em nível alto, o mestre irá aguardar antes de finalizar a operação. Logo após a terceira borda de subida do clock (3) o escravo atualiza as saídas "readdata" e "waitrequest" de acordo com o valor de endereço "address" enviado pelo mestre do barramento Avalon. Consequentemente, na quarta borda de subida do clock (4) o mestre do barramento irá ler o valor requisitado e a operação de leitura estará completa.

A operação de escrita também é apresentada na figura 16, e é análoga à operação de leitura. Contudo, neste caso o mestre do barramento deve enviar o valor de "writedata" que deseja escrever no endereço "address". Novamente, a operação somente chegará ao final após o escravo do barramento colocar o sinal "waitrequest" em nível lógico baixo.

Para se enviar um comando de *reset* ao dispositivo escravo, o mestre do barramento deverá ajustar o sinal *reset* da AVALON *Reset Interface* para nível lógico alto antes da próxima borda de subida do sinal de clock, e mantê-lo estável até a subsequente borda de descida (a operação é síncrona).

Com relação às interrupções, basta que o dispositivo escravo modifique o valor do sinal "irq_n" da AVALON *Interrupt Interface* para nível lógico alto para que a requisição seja enviada ao processador NIOS II. Após a interrupção ter sido tratada, o dispositivo escravo do barramento deverá ajustar o nível lógico do sinal "irq_n" de volta para zero.

3.2.1.3.2 Funcionalidades do Adaptador RTL

A seleção das interfaces do barramento AVALON adequadas ao Adaptador RTL considerou os seguintes requisitos funcionais:

- O Adaptador RTL deve permitir que o processador NIOS II efetue operações de escrita/leitura dos sinais pertencentes à interface AVALON *Conduit Interface*, os quais serão usados para a interligação da descrição RTL do usuário ao sistema RTOOL.

- O Adaptador RTL deve permitir que o dispositivo RTL do usuário envie requisições de transferência de dados de entrada/saída ao processador NIOS II através da interface AVALON *Interrupt*.

- O clock que alimenta o dispositivo RTL do usuário deve ser configurado (frequência de operação) pelo processador NIOS II.

- O processador NIOS II, sempre que desejar, deve ser capaz de restaurar as configurações iniciais do dispositivo RTL do usuário através do envio de um comando de reset.

Levando em consideração estes requisitos funcionais, os sinais de interligação entre o Adaptador RTL e a descrição RTL do usuário, os quais pertencem à AVALON *Conduit Interface*, puderam ser definidos. O resultado é apresentado na figura 17.

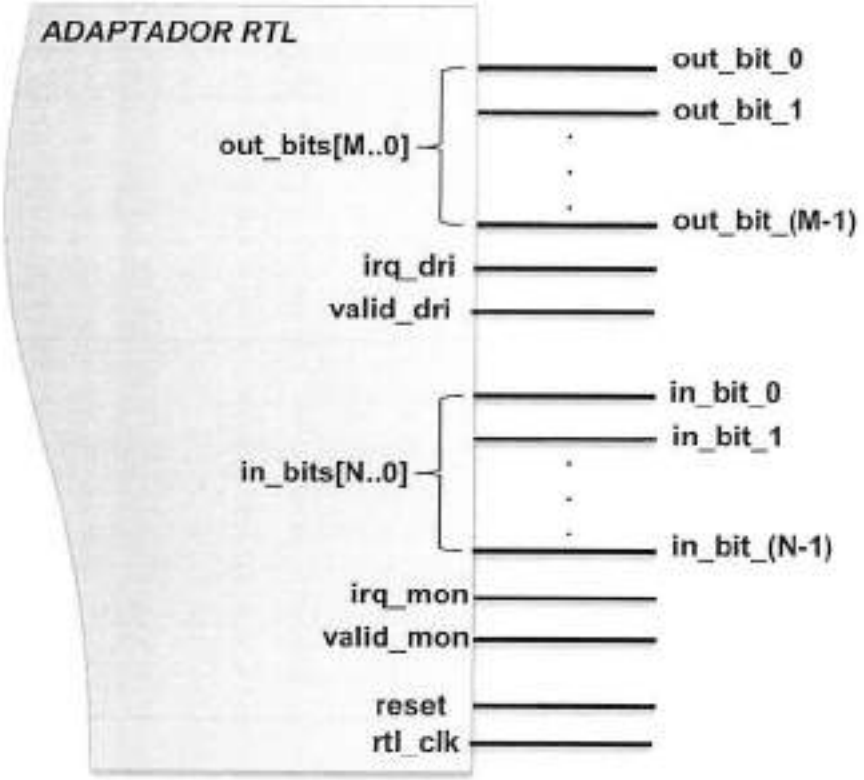

Figura 17. Implementação da AVALON Conduit Interface para o Adaptador RTL.

Há três grupos de sinais, sendo eles sinais do DRIVER, sinais do MONITOR e o sinais GLOBAIS.

No grupo de sinais do DRIVER tem-se:

- Vetor de bits "out_bits[M..0]": Bits de dados que o Adaptador RTL envia ao dispositivo RTL, cujo mapeamento lógico é definido de acordo com as interfaces de entrada presentes no DRIVER do usuário.

- Sinal de interrupção "irq_dri": Bit controlado pelo DRIVER do usuário, o qual mantém nível baixo durante operação normal e assume nível alto quando o DRIVER requisitar novos dados válidos.

- Sinal "valid_dri": Bit de indicação de dado válido no barramento "out_bits" controlado pelo Adaptador RTL, o qual mantém nível baixo durante operação normal e, após uma requisição de dados por parte do DRIVER, assume nível alto quando o vetor "out_bits" estiver pronto para ser lido.

No grupo de sinais do MONITOR tem-se:

- Vetor de bits "in_bits[N..0]": Bits de dados que o dispositivo RTL envia ao Adaptador RTL, cujo mapeamento lógico é definido de acordo com as interfaces de saída presentes no MONITOR do usuário.

- Sinal de interrupção "irq_mon": Bit controlado pelo MONITOR do usuário, o qual mantém nível baixo durante operação normal e assume nível alto quando o MONITOR desejar avisar que novos dados válidos estão presentes no vetor "in_bits".

- Sinal "valid_mon": Bit de indicação de leitura do barramento "in_bits" controlado pelo Adaptador RTL, o qual mantém nível baixo durante operação normal e, após um aviso de dados prontos para leitura por parte do MONITOR, assume nível alto quando o vetor "in_bits" tiver sido lido pelo Adaptador RTL.

No grupo de sinais de GLOBAIS tem-se:

- Sinal "rtl_clk": Sinal de clock controlado pelo Adaptador RTL e que é utilizado para alimentar o dispositivo RTL do usuário, juntamente do DRIVER e do MONITOR. A frequência deste sinal de clock pode ser configurada através de uma escrita, por parte do processador NIOS, no Adaptador RTL.

- Sinal "reset": Sinal de reset controlado pelo Adaptador RTL que deve ser usado pelo RTL do usuário, juntamente do DRIVER e do MONITOR, para restaurar as condições iniciais do sistema.

A versão corrente do Adaptador RTL usa vetores de dados "in_bits" e "out_bits" de 32 bits (M=N=31) de comprimento. Isto facilitou o desenvolvimento de hardware pelo fato de que o barramento AVALON também utiliza palavras com 32 bits de comprimento. A figura 18 mostra em detalhes como o dispositivo RTL do usuário irá ser interligado ao sistema RTOOL.

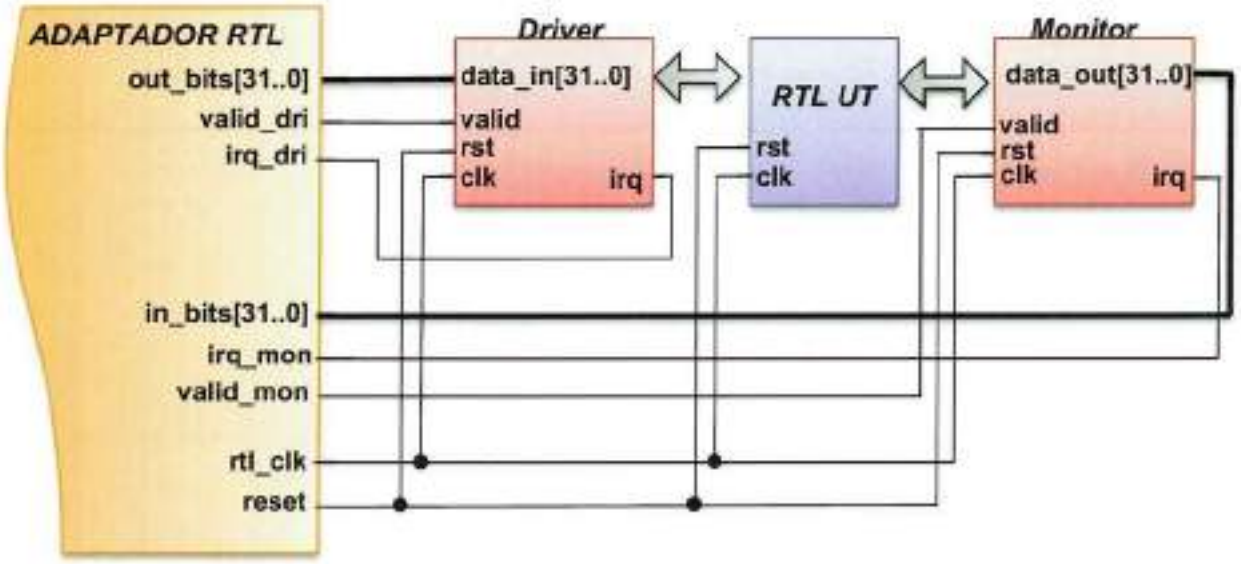

Figura 18. Detalhamento da interligação entre o Adaptador RTL e o dispositivo RTL do usuário

A seguir são apresentados exemplos de transação entre o DRIVER e o Adaptador RTL, e entre o MONITOR e o Adaptador RTL. Note que, como a comunicação entre estes dispositivos é assincrona, pois eles pertencem a dois domínios de clocks distintos, o sinal de clock é irrelevante. Contudo, o sinal de clock foi apresentado nos diagramas por conveniência.

Exemplo de requisição de dados por parte do DRIVER:

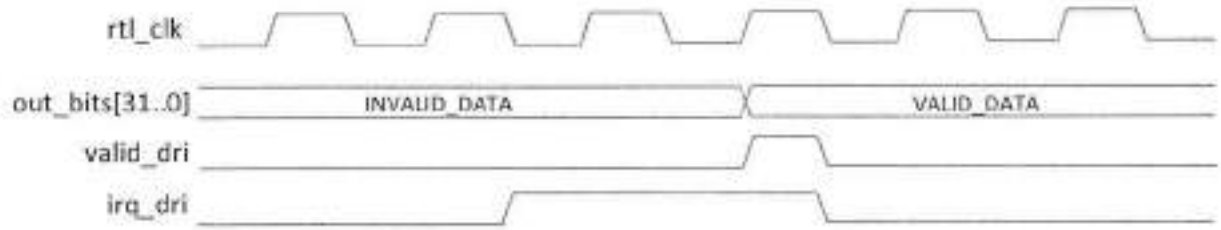

Figura 19. Carta de tempos de uma requisição do DRIVER

Inicialmente a entrada "data_in[31..0]" do driver, a qual corresponde à interface "out_bits[31..0]" do Adaptador RTL, contém dados inválidos. Quando o DRIVER do dispositivo RTL estiver pronto para receber dados de entrada ele irá gerar uma interrupção colocando o sinal "irq_dri" em nível lógico alto. O sistema RTOOL iniciará sua rotina de atualização do vetor "out_bits[31..0]", e, após terminar, colocará o sinal "valid_dri" em nível alto. Para finalizar a transação o DRIVER colocará o sinal "irq_dri" em nível lógico baixo para indicar o recebimento de dados, e o Adaptador RTL colocará o sinal "valid_dri" em nível baixo imediatamente em seguida.

Exemplo de aviso de dados válidos por parte do MONITOR:

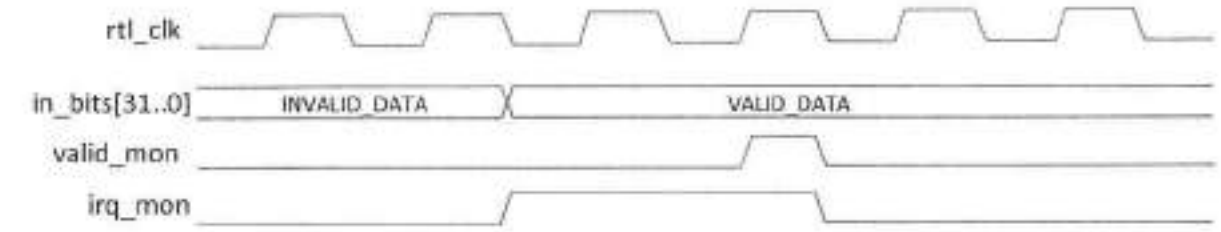

Figura 20. Carta de tempos de uma requisição do MONITOR

Quando o MONITOR atualizar sua saída ("data_out[31..0]") com novos dados válidos, ele deverá gerar uma interrupção para que o Adaptador RTL tenha conhecimento deste evento. Neste momento o Adaptador RTL irá iniciar sua rotina de leitura do vetor "in_bits[31..0]" e, após terminar, colocará o sinal "valid_mon" em nível alto. Para finalizar a transação o MONITOR colocará o sinal "irq_mon" em nível lógico baixo, e o Adaptador RTL colocará o sinal "valid_dri" em nível baixo imediatamente em seguida.

3.2.2 Software do SOPC

O software embarcado do processador NIOS II tem por objetivo gerenciar os diversos processos que ocorrem na placa. São eles: processo de leitura/escrita do PC, processo de leitura do DRIVER e processo de escrita do MONITOR.

Inicialmente, é fornecida uma descrição conceitual dos processos que foram implementados em software. Em seguida, a máquina de estados finitos desenvolvida em linguagem de programação C é apresentada e explicada.

O software desenvolvido para o processador NIOS II, na verdade, corresponde a um firmware, já que não foi usado nenhum sistema operacional.

3.2.2.1 Descrição conceitual dos processos

O diagrama da figura 21 mostra de que forma cada processo usa a memória SDRAM. O processo "Escrita/Leitura do PC" escreve e lê, enquanto o processo "Escrita do MONITOR" apenas escreve e o processo "Leitura do DRIVER" apenas lê da memória.

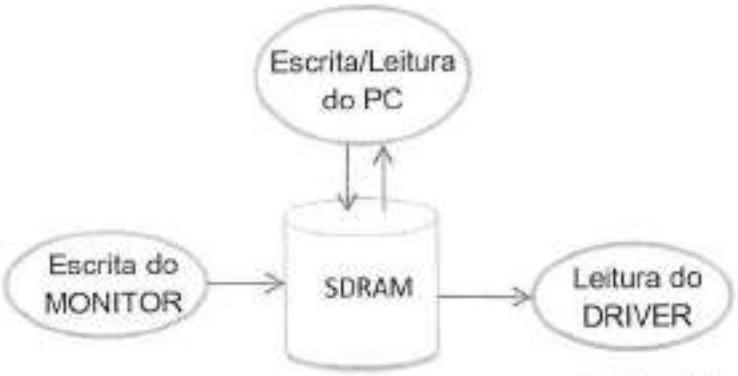

Figura 21. Interações entre os processos e a memória SDRAM

A seguir cada processo é apresentado individualmente, e suas interdependências são explicitadas.

- **Escrita/Leitura do PC:**

  Trata-se do processo em que ocorre a transferência, entre o PC e o NIOS, de vetores de entrada/saida do dispositivo RTL do usuário.

  Na operação de escrita do PC, os pacotes de dados são enviados pelo PC e recebidos pelo NIOS. Após serem recebidos pelo NIOS, os dados são armazenados na memória SDRAM, para posterior utilização pelo processo de leitura do DRIVER.

Na operação de leitura do PC, os pacotes de dados são requisitados pelo PC e, se houverem dados disponíveis na memória SDRAM, enviados pelo NIOS. A disponibilidade de dados na memória SDRAM é determinada pelo processo de escrita do MONITOR.

- **Leitura do DRIVER:**

É o processo em que os pacotes de dados enviados pelo PC, e que se encontram armazenados na memória, devem ser repassados ao DRIVER do dispositivo RTL.

A condição de início de uma transferência de dados entre o NIOS e o DRIVER é uma requisição de interrupção por parte do DRIVER. Isto fará com que pacotes de dados disponíveis na memória SDRAM sejam enviados ao DRIVER.

Cada pacote é composto por um conjunto de campos que representam os vetores de estímulo da entrada do DRIVER.

- **Escrita do MONITOR:**

Neste processo ocorre o inverso do processo de Leitura do DRIVER, ou seja, o recolhimento dos pacotes já processados pelo dispositivo RTL do usuário.

A condição de início desta transferência entre o MONITOR e o NIOS, novamente, é uma requisição de interrupção por parte do MONITOR. Esta requisição tem por objetivo informar que dados de saída estão prontos para serem coletados.

Os pacotes recolhidos são escritos na memória SDRAM para posterior transferência ao PC.

3.2.2.2 Máquina de Estados Finitos

A metodologia de desenvolvimento de software na forma de máquinas de estados finitos (MEF) é particularmente útil quando não há um sistema operacional na plataforma para a qual o software está sendo desenvolvido, pois permite atribuir um caráter multi tarefa à esta aplicação.

Em linguagem de programação C, o *framework* comumente adotado para a implementação de MEF é apresentado no código a seguir, no qual a declaração "*switch*" é adotada para se efetuar a multiplexação entre as rotinas de cada estado da máquina:

```
#define ESTADO_0 0
#define ESTADO_1 1

// variáveis de estado
```

```
unsigned int state, next_state = ESTADO_0;

while( 1 )
{
	state = next_state;  // atualiza estado

	switch( state )
	{
		case ESTADO_0:
			TarefaDoEstado0( &next_state );
		break;

		case ESTADO_1:
			TarefaDoEstado1( &next_state );
		break;

		default:
			TratamentoDeEstadoInvalido( &next_state );
		break;
	}
}
```

No *framework* fornecido, há apenas dois estados definidos, porém uma MEF com mais estados pode ser facilmente construida de forma análoga ao código em questão.

Verifique que cada estado da MEF executa sua rotina específica, a qual pode vir a alterar, ou não, a variável que determina o próximo estado, de acordo a verificação de uma condição de mudança de estado.

Referente ao software embarcado do RTOOL, o diagrama da MEF implementada é mostrado na figura 22.

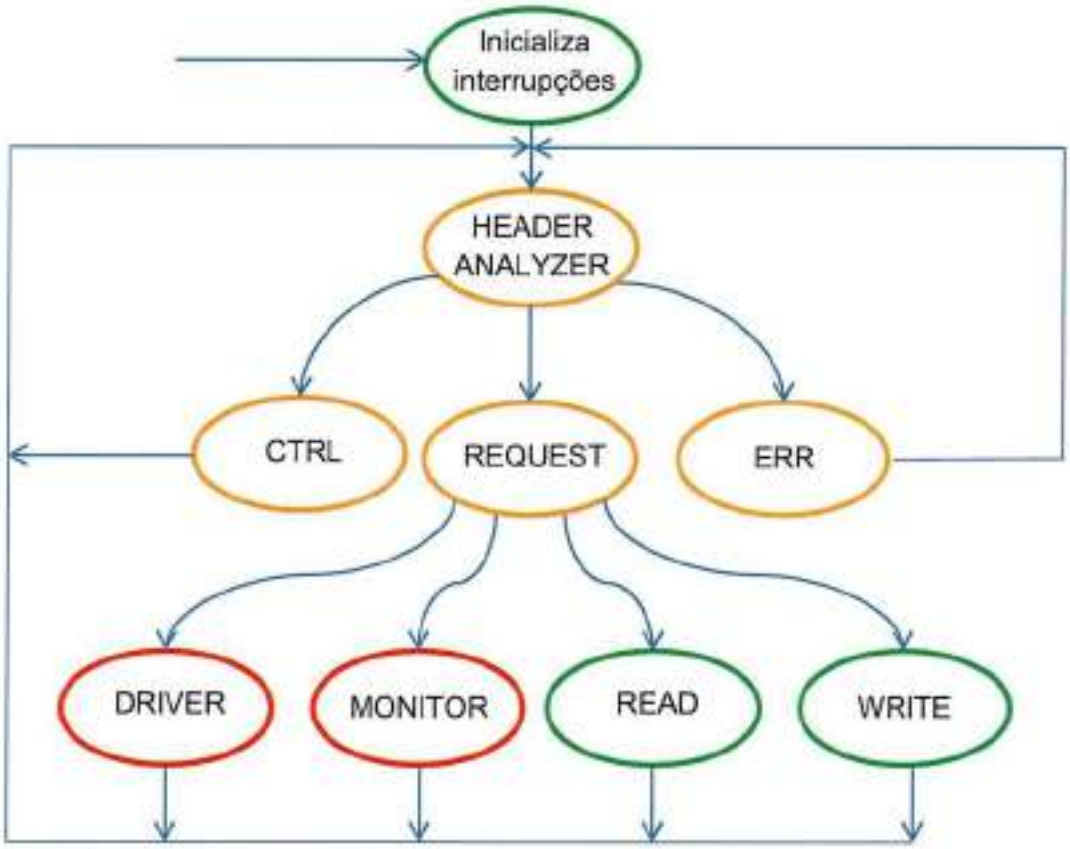

Figura 22. Diagrama da MEF do software embarcado do sistema RTOOL

As diferentes cores dos estados são explicadas a seguir:

- Em verde tem-se os estados nos quais as interrupções do processador NIOS II foram desabilitadas.
- Em laranja tem-se os estados nos quais as interrupções do processador NIOS II estão ativas.
- Em vermelho tem-se os estados que podem ser diretamente acessados (fora do percurso normal da MEF) quando uma requisição de interrupção é enviada ao processador NIOS II.

Antes da execução do código entrar na MEF, as interrupções do processador NIOS II são habilitadas. Isto é necessário para que o DRIVER e o MONITOR do dispositivo RTL do usuário possam requisitar operações de escrita/leitura ao software embarcado.

Uma vez que as interrupções foram habilitadas, a MEF vai para seu estado inicial, o estado HEADER_ANALYZER, cuja função principal é analisar os pacotes de dados recebidos pelo processador NIOS II, os quais foram enviados pelo PC. Depois de decodificar o tipo de pacote recebido (de acordo com o protocolo definido na seção

"3.2.4.3 – Protocolo de comunicação com a placa de prototipagem") o estado HEADER_ANALYZER configura o próximo estado da MEF.

Caso o estado seguinte ao HEADER_ANALYZER seja o estado CTRL, isto significa que uma operação de controle foi requisitada. São definidas como operações de controle o comando de *reset* do sistema RTOOL, a configuração da frequência do sinal de *clock* que alimenta o hardware do usuário e a configuração do tamanho, em bits, da palavra que será escrita/lida do Adaptador RTL (32 bits). Após a realização da operação de controle, a MEF volta à seu estado inicial.

Caso o estado seguinte ao HEADER_ANALYZER seja o estado ERR, isto significa que uma mensagem de erro foi enviada pelo PC. Na versão corrente do sistema RTOOL, o único tipo de erro contemplado é o erro de "*time out*", ou seja, um erro que ocorre quando uma das partes da comunicação deixa de responder uma mensagem. O estado ERR tem o intuito de manter a sincronização entre o PC e o NIOS mesmo quando um erro de "*time out*" ocorre. Após a realização desta operação, a MEF volta à seu estado inicial.

Caso o estado seguinte ao HEADER_ANALYZER seja o estado REQUEST, isto significa que uma requisição de escrita/leitura de vetor de dados de entrada/saída do hardware do usuário foi enviada pelo PC. As requisições podem ser de dois tipos, os quais são explicados a seguir:

- A requisição WRITE é enviada pelo PC quando o mesmo deseja disponibilizar mais vetores de dados entrada ao dispositivo RTL do usuário. Como tais vetores de dados serão guardados temporariamente na memória SDRAM da placa de prototipagem (até que sejam requisitados pelo DRIVER), se faz necessária uma verificação de disponibilidade de espaço na memória na região de vetores de entrada do dispositivo RTL do usuário. Caso haja espaço disponível, uma mensagem de requisição aceita será enviada ao PC e o próximo estado da MEF será o estado WRITE. Caso não haja espaço disponível, uma mensagem de requisição negada será enviada ao PC e o próximo estado da MEF será o estado DRIVER.

- A requisição READ é enviada ao pelo PC quando o mesmo desejar ler vetores de dados de saída gerados pelo dispositivo RTL do usuário. Logo após o recebimento desta requisição, o NIOS irá verificar se existem dados suficientes para serem lidos da região de memória de vetores de saída do dispositivo RTL do usuário. Caso haja dados disponíveis, uma mensagem de requisição aceita será enviada ao PC, e o próximo estado da MEF será o estado READ. Caso não haja dados disponíveis o suficiente, uma mensagem de requisição negada será enviada ao PC e o próximo estado da MEF será o estado MONITOR.

No estado WRITE o processador NIOS II simplesmente irá receber os vetores de dados do PC e irá armazená-los na região da memória SDRAM de vetores de entrada do dispositivo RTL. O próximo estado será o estado inicial da MEF.

No estado READ o processador NIOS II simplesmente irá ler vetores de dados da memória SDRAM na região de vetores de saida do dispositivo RTL, e em seguida irá enviá-los ao PC. O próximo estado será o estado inicial da MEF.

O estado DRIVER pode ser atingido de duas formas, proveniente da execução prévia do estado REQUEST ou através de uma interrupção gerada pelo DRIVER do dispositivo RTL do usuário:

- Quando o estado DRIVER é atingido por uma interrupção, o processador NIOS II irá verificar se existem vetores de entrada disponíveis na memória SDRAM e, se houver, irá enviá-los ao DRIVER do RTL do usuário. Caso contrário, uma *flag* de sinalização de **falta de vetores de entrada** será "setada". Ao final da rotina de interrupção a MEF retorna ao estado que estava sendo executado antes da interrupção.

- Se o estado DRIVER foi atingido proveniente da execução prévia do estado REQUEST, isto significa que a região de memória de vetores de entrada está cheia. Neste caso o processador irá verificar se a *flag* de **falta de vetores de entrada** está com nível lógico alto, e, se estiver, irá transferir vetores de entrada da memória SDRAM ao DRIVER do RTL do usuário com o intuito de gerar espaço disponível na memória. Em seguida a MEF retornará à seu estado inicial.

O estado MONITOR, também, pode ser atingido de duas formas, proveniente da execução prévia do estado REQUEST ou através de uma interrupção gerada pelo MONITOR do dispositivo RTL do usuário.

- Quando o estado MONITOR é atingido por uma interrupção, o processador NIOS II irá verificar se existe espaço disponível na região da memória SDRAM de vetores de saida e, se houver, irá transferir o vetor de saída recém gerado do MONITOR à memória. Caso contrário, uma *flag* de sinalização de **vetor de saída pendente** será "setada". Ao final da rotina de interrupção a MEF retorna ao estado que estava sendo executado antes da interrupção.

- Se o estado MONITOR foi atingido proveniente da execução prévia do estado REQUEST, isto significa que a região de memória de vetores de saida está vazia. Neste caso o processador irá verificar se a *flag* de **vetor de saída**

pendente está com nível lógico alto, e, se estiver, irá transferir vetores de saída do MONITOR do RTL do usuário à região correspondente da memória SDRAM com o intuito de aumentar a disponibilidade de dados para uma futura requisição de leitura por parte do PC. Em seguida a MEF retornará à seu estado inicial.

3.2.3 Ferramenta de prototipagem rápida

Tendo por objetivo facilitar o uso do sistema RTOOL, uma ferramenta de prototipagem rápida com interface gráfica de usuário (GUI) foi desenvolvida. Este capítulo descreve os processos que tal ferramenta torna automáticos, e mostra o design da GUI criada.

3.2.3.1 Algoritmo de integração, compilação e programação

As etapas de integração do dispositivo RTL do usuário ao SOPC do RTOOL, de compilação do hardware resultante e de programação da placa de prototipagem com o hardware sintetizado e com o software embarcado são tarefas que poderiam realizadas manualmente. Todavia, dado que estas três tarefas são bastante simples e repetitivas, optou-se por implementar neste projeto um algoritmo que permite a realização destas etapas de forma automatizada.

A automatização visa reduzir a quantidade de esforço que um usuário do sistema RTOOL enfrentará ao utilizá-lo, e foi implementada através da execução direta de arquivos binários constituintes do ambiente de desenvolvimento de hardware Quartus II, da ALTERA. A figura 23 mostra uma esquematização do processo.

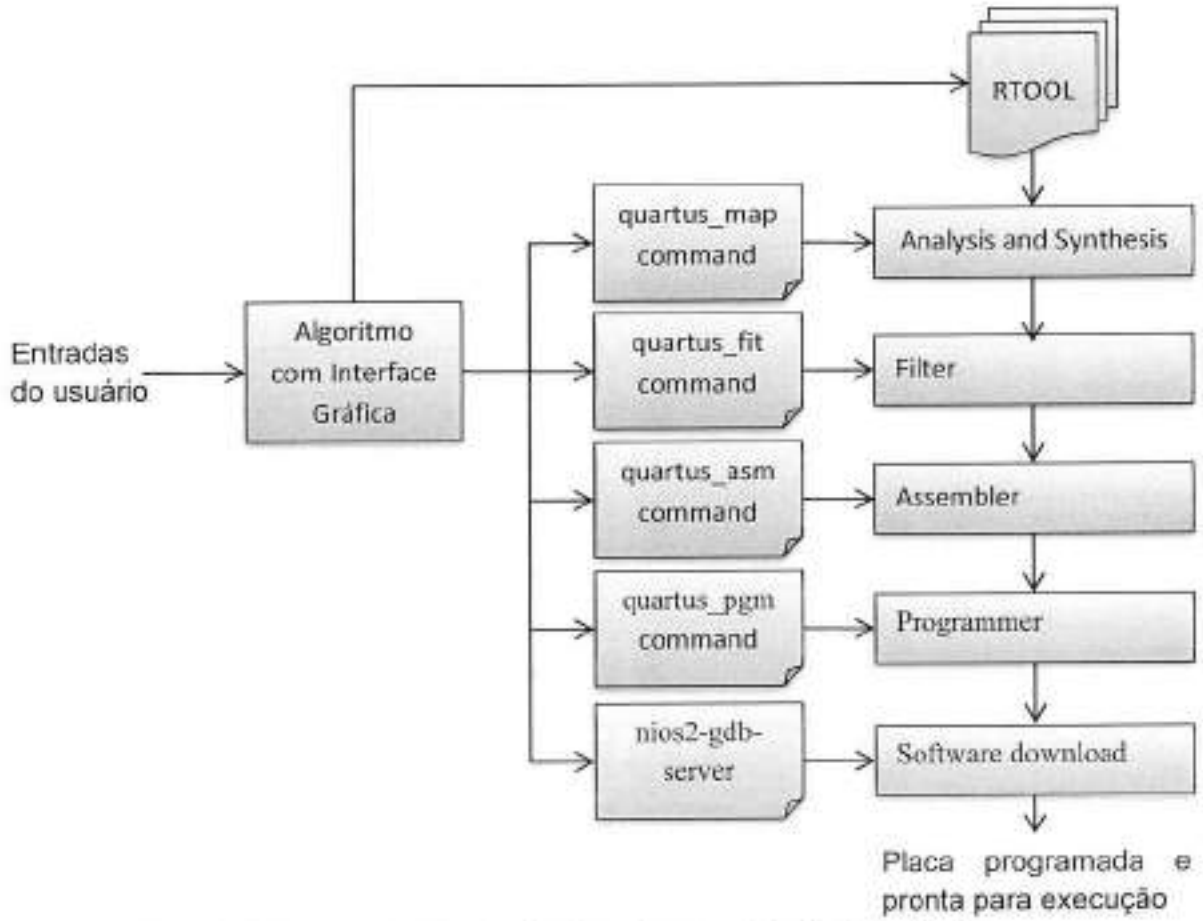

Figura 23. Esquematização do algoritmo de automatização da prototipagem

Após o usuário ter inserido as informações necessárias a respeito de seu projeto, tais como a definição de entidade "top level", os arquivos fonte de seu dispositivo RTL e a frequência de operação, o algoritmo de automatização irá modificar o código fonte (VHDL) do sistema RTOOL de modo a integrar o dispositivo RTL do usuário ao Adaptador RTL.

Uma vez que a integração do dispositivo RTL do usuário ao sistema RTOOL estiver completa, o algoritmo de automatização iniciará o processo de compilação usando os arquivos binários do software Quartus II. Primeiramente será chamado o comando de análise e síntese (quartus_map). Em seguida será chamado o comando de otimização (quartus_fit) do circuito lógico sintetizado. Depois o comando de montagem (assembler) será executado para mapear o circuito lógico ao FGPA alvo. Finalmente, o FPGA será programado com o hardware resultante usando o comando de programação (quartus_pgm) e o software embarcado será transferido ao processador NIOS II através do executável de *download* (nios2-gdb-server).

3.2.3.2 Interface gráfica de usuário da ferramenta

O design elaborado para a GUI da ferramenta de prototipagem rápida possui três abas, uma para cada etapa da utilização do sistema RTOOL.

A primeira aba, mostrada na figura 24, deve receber como entradas o diretório binário do Quartus II, o diretório binário do ambiente de desenvolvimento Eclipse para o processador NIOS II e a lista de arquivos HDL que compõe o projeto do usuário. Se algum erro ocorrer nessa etapa, uma mensagem de erro aparecerá.

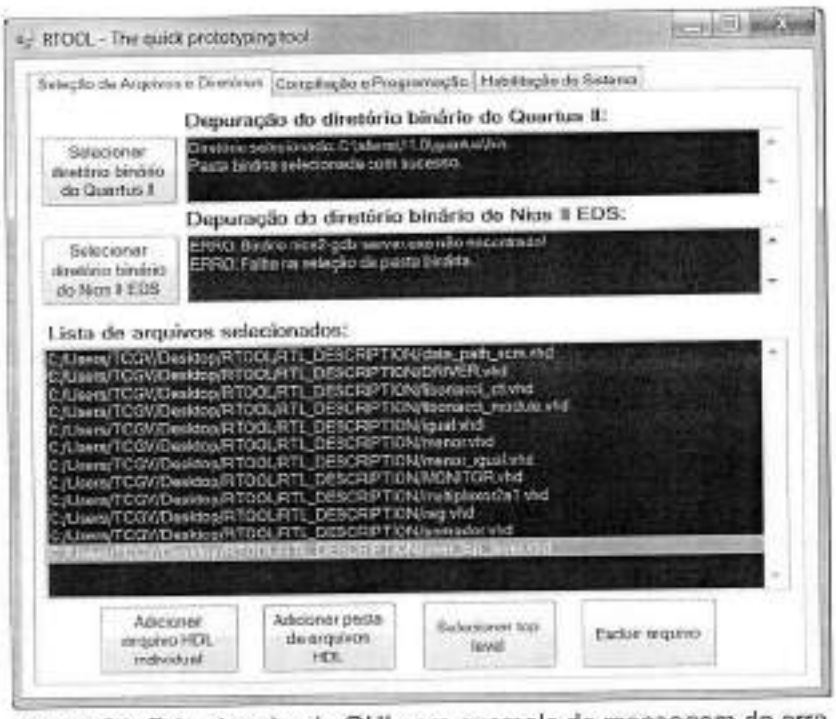

Figura 24. Primeira aba da GUI com exemplo de mensagem de erro

A segunda aba, mostrada na figura 25, permite que o usuário selecione a frequência de operação de seu hardware, compile o sistema RTOOL, programe o hardware no FPGA e em seguida faça o *download* do firmware. Novamente, mensagens de erro aparecerão se algum problema for detectado.

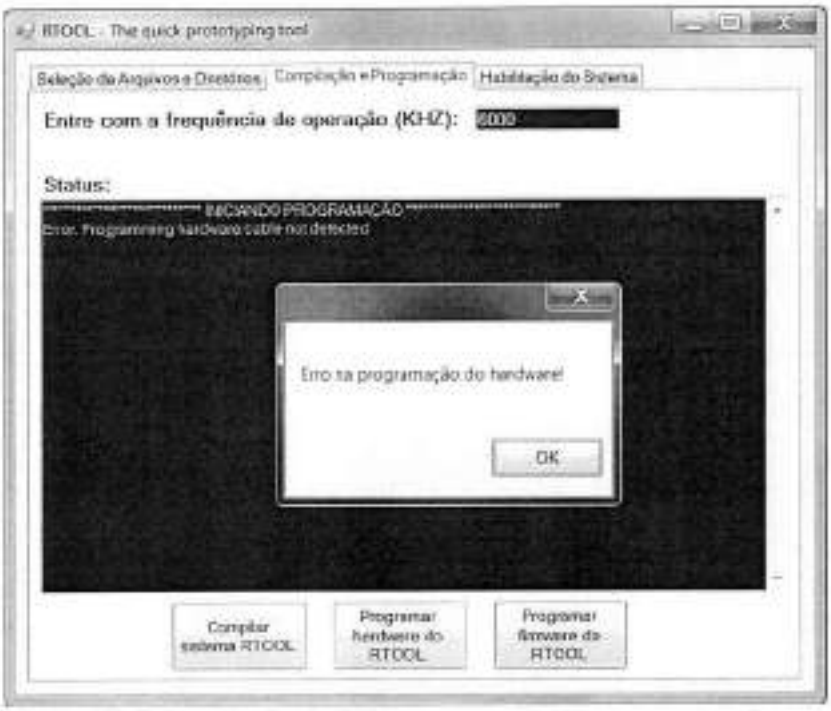

Figura 25. Segunda aba da GUI com exemplo de mensagem de erro

A última aba, mostrada na figura 26, permite que o usuário inicialize, ou interrompa, a execução do servidor da comunicação com a placa de prototipagem (detalhado no capítulo "3.2.4.2 – Driver USB com interface de rede"). É importante ressaltar que, durante a etapa de programação do hardware do RTOOL, a execução do servidor da comunicação com a placa de prototipagem deve ser interrompida, já que ambas as etapas necessitam utilizar um mesmo recurso, o canal de comunicação JTAG.

Figura 26. Terceira aba da GUI com exemplo de mensagem de erro

3.2.4 Interface de Programação de Aplicativo

Uma visão em camadas da API do sistema RTOOL é apresentada na figura 27. Em verde, tem-se a aplicação do usuário, em laranja os módulos de software da API do sistema RTOOL e em azul o servidor JTAG fornecido pela ALTERA.

Figura 27. Camadas da API do sistema RTOOL

A aplicação do usuário terá acesso direto à API, a qual é fornecida na forma de uma biblioteca em linguagem C, contendo funções de escrita e leitura de vetores de dados de entrada e saída do dispositivo RTL do usuário.

A biblioteca em C não contém a implementação das rotinas de comunicação com a placa de prototipagem. Em vez disso, as funções da biblioteca C enviam requisições de transações para o Driver USB através de comunicação interprocesso. Isto permitiu reduzir o tamanho da compilação da biblioteca C, que por sua vez visa diminuir o tamanho da compilação do aplicativo do usuário.

O gerenciamento da comunicação com a placa de prototipagem foi designado ao Driver USB, cuja implementação se deu como um processo independente da aplicação do usuário. Consequentemente, é preciso fornecer uma forma de comunicação interprocessos entre a aplicação do usuário e o driver USB. A forma escolhida foi a utilização de uma interface de rede (biblioteca Socket), a qual permite a simplificação da comunicação entre diferentes processos dentro de um sistema operacional.

O protocolo de comunicação entre o PC e a placa de prototipagem, usado pelo Driver USB, foi elaborado com o intuito de garantir a confiabilidade de transações com grande volume de dados.

3.2.4.1 Biblioteca em linguagem C

O usuário da ferramenta RTOOL, ao desenvolver a aplicação de verificação funcional de seu projeto RTL, irá utilizar as funções da biblioteca C para "resetar" a placa de prototipagem, configurar a frequência do sinal de clock que alimenta seu dispositivo RTL, enviar vetores de dados de entrada ao DRIVER e ler os vetores de dados de saída do MONITOR.

A seguir é apresentada a interface desta biblioteca (arquivo **"rtool_int.h"**):

```
#ifndef _RTOOL_INT_H_
#define _RTOOL_INT_H_

// Código de erro que pode ser devolvido pelas funções
// "RtoolWrite" e "RtoolRead" caso a requisição da operação
// correspondente tenha sido negada
#define REQUEST_DENIED     -4

// A função "RtoolStart" tenta se conectar ao Driver USB que
// gerencia a comunicação com a placa de prototipagem. Se a
// conexão for efetuada com sucesso, o valor zero será retornado,
// caso contrário um valor negativo de retorno indicará um erro.
int RtoolStart();

// A função "RtoolReset" envia um comando de Reset à placa de
// prototipagem. Ao receber o comando a placa deverá limpar sua
// memória SDRAM externa, reinicializar o dispositivo RTL do
// usuário e reinicializar seu software.
// Se a operação for realizada com sucesso, o valor zero será
// retornado, caso contrário um valor negativo de retorno
// indicará um erro.
int RtoolReset();

// A função "RtoolSync" configura o período do sinal de clock
// que alimenta o dispositivo RTL do usuário. A fórmula que
// relaciona o período real e o parâmetro "clk_per" é:
//           T = (1+clk_per)/(50 MHZ)
// Se a operação for realizada com sucesso, o valor zero será
// retornado, caso contrário um valor negativo de retorno
// indicará um erro.
int RtoolSync( unsigned int clk_per );

// A função "RtoolSizeIn" configura o comprimento, em bits,
// da interface de dados do DRIVER do usuário. Na versão
// corrente do sistema RTOOL o valor do parâmetro "size"
// deve ser, OBRIGATÓRIAMENTE, igual a 32.
// Se a operação for realizada com sucesso, o valor zero será
// retornado, caso contrário um valor negativo de retorno
// indicará um erro.
int RtoolSizeIn( unsigned int size );

// A função "RtoolSizeOut" configura o comprimento, em bits,
```

```
// da interface de dados do MONITOR do usuário. Na versão
// corrente do sistema RTOOL o valor do parâmetro "size"
// deve ser, OBRIGATÓRIAMENTE, igual a 32.
// Se a operação for realizada com sucesso, o valor zero será
// retornado, caso contrário um valor negativo de retorno
// indicará um erro.
int RtoolSizeOut( unsigned int size );

// A função "RtoolWrite" envia uma requisição de escrita de
// dados à placa de prototipagem. O tamanho da requisição, em
// bytes, é determinado pelo parâmetro "len".
// Caso a requisição seja aceita pela placa, um vetor de dados
// com a posição inicial de memória apontada pelo parâmetro
// "buf", e com tamanho "len", será transmitido à placa.
// Se a operação for realizada com sucesso, o valor zero será
// retornado, caso contrário um valor negativo de retorno
// indicará um erro (o erro REQUEST_DENIED não é fatal).
int RtoolWrite( unsigned int *buf, int len );

// A função "RtoolRead" envia uma requisição de leitura de
// dados à placa de prototipagem. O tamanho da requisição, em
// bytes, é determinado pelo parâmetro "len".
// Caso a requisição seja aceita pela placa, os dados recebidos
// pelo PC serão escritos no vetor cuja posição inicial de
// memória é determinada pelo parâmetro "buf".
// Se a operação for realizada com sucesso, o valor zero será
// retornado, caso contrário um valor negativo de retorno
// indicará um erro (o erro REQUEST_DENIED não é fatal).
int RtoolRead( unsigned int *buf, int len );

// Uma chamada à função "RtoolClose" faz com que a conexão
// préviaamente estabelecida com o Driver USB seja fechada.
// Se a operação for realizada com sucesso, o valor zero será
// retornado, caso contrário um valor negativo de retorno
// indicará um erro.
int RtoolClose();

#endif
```

Para aumentar o desempenho de sua aplicação de verificação funcional, recomenda-se que o usuário efetue chamadas às funções "RtoolWrite()" e "RtoolRead()" para transações de tamanho mínimo igual a 1 KB. Caso contrário o desempenho do sistema será reduzido devido a uma maior carga de operações de controle no canal de comunicação.

O diagrama da figura 28 mostra um fluxo típico de utilização da API do sistema RTOOL por uma aplicação do usuário:

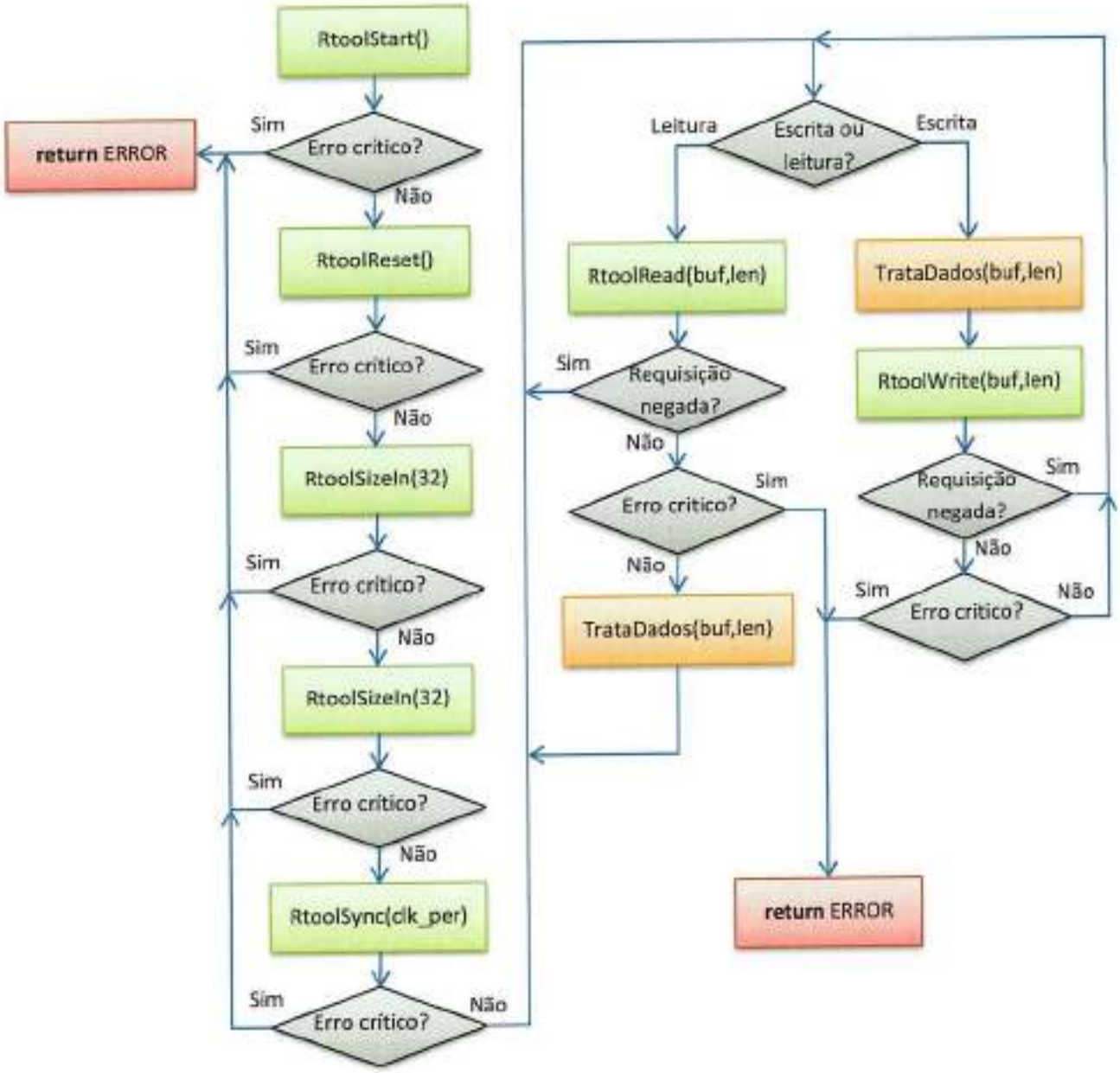

Figura 28. Fluxo de utilização da API do sistema RTOOL

Na figura 28 foram destacadas em verde as chamadas à API do RTOOL, em vermelho os estados de finalização da execução do aplicativo devido à ocorrência de erro e em laranja rotinas de implementação do usuário. Note que no diagrama a função "RtoolClose()" não foi chamada, porém sempre que o usuário desejar interromper sua aplicação ele poderá chamar tal função para que a biblioteca C se desconecte do Driver USB.

3.2.4.2 Driver USB com interface de rede

Apesar de ser descrito como um Driver USB, este módulo de software da API do sistema RTOOL na verdade não implementa um driver USB em modo *kernel* no sistema operacional Windows. O que é feito é o acesso às funcionalidades do servidor JTAG fornecido pela ALTERA, o qual, de fato, implementa um driver USB em modo de usuário e utiliza um driver em modo *kernel* nativo do sistema operacional. Logo, o nome "Driver USB" foi escolhido por conveniência, apenas, para indicar aspectos de baixo nível da API do RTOOL.

O Driver USB é responsável pelo gerenciamento da comunicação entre a aplicação do usuário e a placa de prototipagem, ou seja, ele atua como um intermediário nas transações de dados e operações de controle.

A comunicação com a aplicação do usuário é feita na forma de uma interface de rede, mais especificamente usando-se o protocolo TCP/IP. Já a comunicação com a placa de prototipagem é realizada através de um protocolo (especificado na seção "3.2.4.3 Protocolo de comunicação da placa de prototipagem") sobre o canal de comunicação fornecido pelo servidor JTAG. A figura 29 mostra um diagrama de blocos do Driver USB e suas interações com o servidor JTAG e a aplicação do usuário.

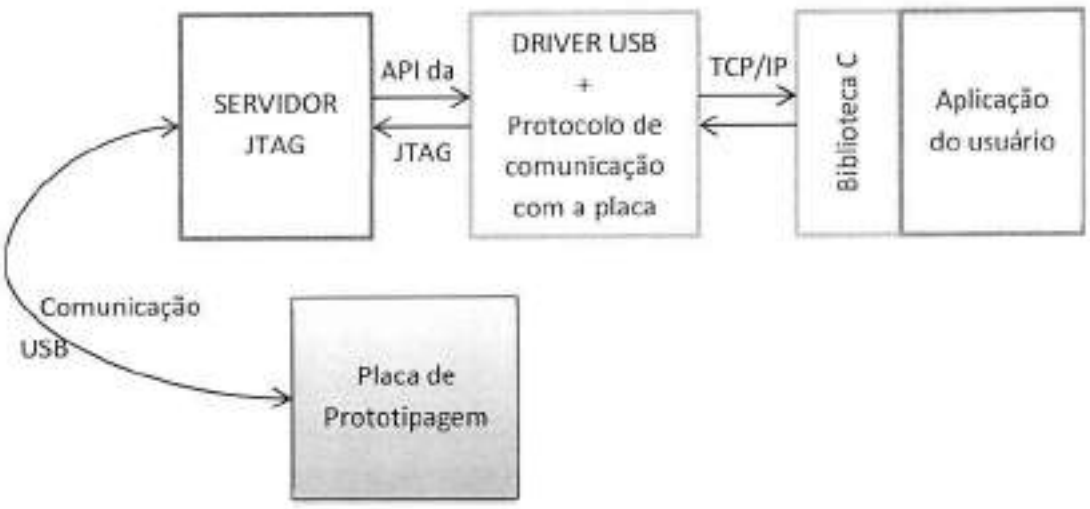

Figura 29. Diagrama de blocos do Driver USB e suas interações

Perceba que a biblioteca C torna-se parte integrante da aplicação do usuário, enquanto o Driver USB é um processo independente, e sua implementação contempla o protocolo de comunicação com a placa de prototipagem, o qual é utilizado sobre o canal de transmissão de dados fornecido pelo servidor JTAG da ALTERA.

3.2.4.3 Protocolo de comunicação da placa de prototipagem

Sobre o canal de comunicação USB entre a placa de prototipagem e o computador foi adotado o protocolo definido pelos autores e que é detalhado neste capítulo. Este protocolo tem por objetivo o estabelecimento de um mecanismo de transmissão de dados estruturado que seja confiável. A estrutura proposta para o pacote de comunicação é mostrada na figura 30.

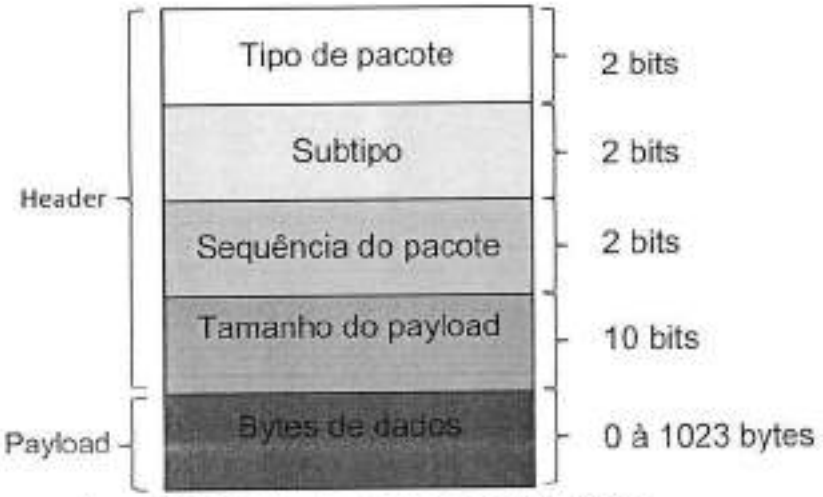

Figura 30. Estrutura do pacote de comunicação

O campo "Tipo de pacote" identifica o pacote para que o receptor do mesmo possa tratá-lo de forma apropriada. A tabela 3 apresenta tipos definidos pelo protocolo, e suas respectivas representações binárias:

Tabela 3. Tipos de pacotes definidos pelo protocolo

Valor	Tipo de pacote
00	Dados (DATA)
01	Controle (CTRL)
10	Requisição (REQ)
11	Error Status (ERR)

Os tipos de pacotes apresentados acima são usados em conjunto para se efetuar a transmissão de dados entre o computador (PC) e a placa de emulação que contém o processador NIOS II.

Cada tipo de pacote poderá apresentar diferentes subtipos, os quais serão explicados mais adiante.

O campo "Sequência de pacote" contém um número sequêncial de controle que é incrementado a cada novo pacote transmitido para permitir que a perda de pacotes individuais seja detectada. Dado que este campo possui apenas dois bits de tamanho, poderá haver apenas 4 valores distintos de sequência, e ao se incrementar o valor máximo ('11') o valor inicial zero ('00') será novamente atingido.

Como o número de bytes no "*payload*" do pacote é variável, o campo "Tamanho do payload" indicará quantos bytes de dados estarão sendo enviados em um determinado pacote de comunicação. Este campo possui 10 bits de comprimento, podendo assumir valores inteiros entre 0 e 1023.

Será adotada a seguinte notação para representar os pacotes: [TIPO; SUBTIPO; SEQUÊNCIA; TAMANHO DO PAYLOAD; PAYLOAD]. A seguir são apresentados os contextos nos quais cada tipo de pacote esta inserido.

3.2.4.3.1 Transmissão de dados

A transmissão de dados de leitura ou escrita se inicia com o envio de uma requisição ("Tipo de pacote" = "10"), por parte do PC, ao NIOS II. Os subtipos permitidos para este tipo de pacote estão definidos na tabela 4.

Tabela 4. Subtipos permitidos para o pacote REQ

Valor	Subtipo do pacote REQ
00	Dados Leitura (RD)
01	Dados Escrita (WR)
10	Requisição Aceita (ACK)
11	Requisição negada (NACK)

A requisição enviada pelo PC deve conter, além do subtipo do pacote, o número de bytes que o computador deseja transferir. O número de bytes da transferência de dados estará contido no *payload* do pacote, que conterá um número inteiro de 3 bytes. Logo, um pacote de requisição apresentará dois formatos possíveis:

Requisição de leitura por parte do PC	[REQ; RD; SEQ; 3; RD_SIZE]
Requisição de escrita por parte do PC	[REQ; WR; SEQ; 3; WR_SIZE]

Sendo SEQ um número de sequência adequado e RD_SIZE e WR_SIZE os números de bytes que se deseja transferir para leitura e escrita, respectivamente. Consequentemente, as respostas possíveis para estes tipos de requisição serão:

Requisição aceita pelo NIOS	[REQ; ACK; SEQ; 3; SIZE]
Requisição rejeitada pelo NIOS	[REQ; NACK; SEQ; 3; SIZE]

Sendo SEQ novamente um número de sequência adequado e SIZE o número de bytes que se deseja transferir (para leitura ou escrita). O número de bytes de uma

transferência não é limitado pelo protocolo, o que significa que transferências maiores do que 1023 bytes deverão ser fragmentadas em diversos pacotes.

Após a requisição ter sido aceita pelo NIOS, o prosseguimento da transmissão de dados se dará através do envio de pacotes do tipo DATA, ou seja, pacotes cujo campo "Tipo de pacote" estarão preenchido com o valor binário "00".

O tipo DATA permite dois subtipos de pacotes, de leitura ou de escrita, como apresentado na tabela 5.

Tabela 5. Subtipos permitidos para o pacote DATA

Valor	Subtipo do pacote DATA
00	Dados leitura (RD)
01	Dados escrita (WR)
10	Reservado
11	Reservado

No caso em que uma requisição de leitura de dados foi enviada pelo PC ao NIOS, o prosseguimento da transmissão de dados se dará com o NIOS enviando um fluxo de pacotes do tipo DATA e subtipo RD. Após o número de bytes previamente requisitado pelo PC ter sido entregado pelo NIOS, o PC enviará um pacote do tipo ERR ("Tipo de pacote" = "11") e subtipo SUCCESS para para finalizar a operação, o qual será respondido pelo NIOS com um pacote idêntico.

A figura 31 mostra a sequência de transmissão de pacotes que deve ocorrer para uma operação de leitura de dados bem sucedida.

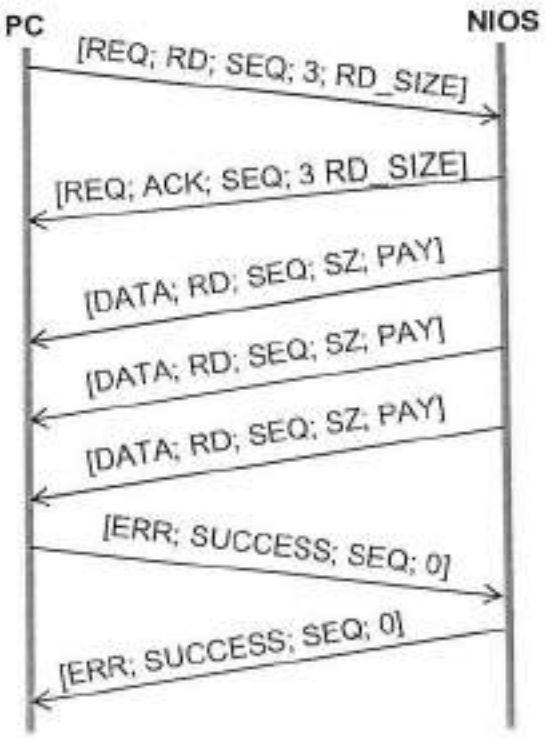

Figura 31. Sequência de pacotes para a operação de leitura

No segundo caso, em que uma requisição de escrita de dados foi enviada pelo PC ao NIOS, o prosseguimento da transmissão de dados se dará com o PC enviando um fluxo de pacotes do tipo DATA e subtipo WR. Após o número de bytes previamente requisitado pelo PC ter sido entregado ao NIOS, o NIOS enviará um pacote do tipo ERR ("Tipo de pacote" = "11") e subtipo SUCCESS para finalizar a operação, o qual será respondido pelo PC com um pacote idêntico.

A figura 32 mostra a sequência de transmissão de pacotes que deve ocorrer para uma operação de escrita de dados bem sucedida.

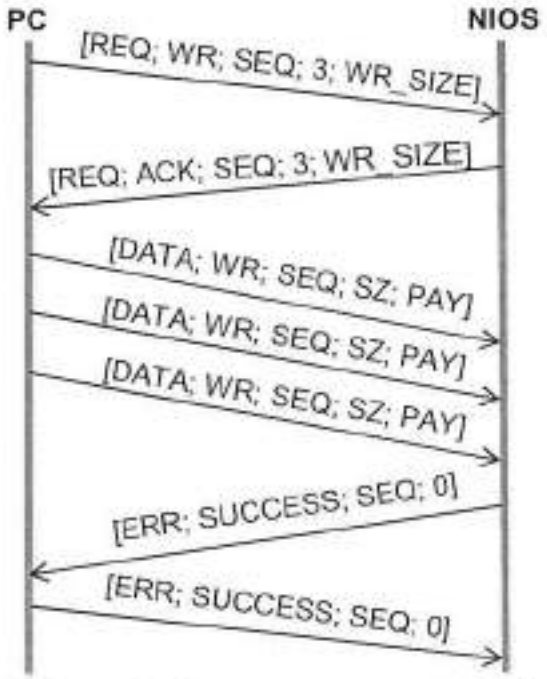

Figura 32. Sequência de pacotes para a operação de escrita

No caso em que o PC enviar uma requisição de leitura ou escrita ao NIOS e receber um pacote rejeitando a requisição, o mesmo deverá finalizar a operação e tentar realizá-la novamente mais tarde.

3.2.4.3.2 Operações de controle

Uma operação de controle é iniciada com o envio de um pacote de controle por parte do PC. O pacote do tipo CTRL conterá no campo "Tipo de pacote" o valor binário "01", e poderá apresentar os subtipos da tabela 6.

Tabela 6. Subtipos permitidos para o pacote CTRL

Valor	Subtipo do pacote CTRL
00	Reset na placa (RST)
01	Sincronização (SYNC)
10	Tamanho de Entradas (SZ_IN)
11	Tamanho de Saidas (SZ_OUT)

O subtipo RST (valor binário "00") é um pedido de reset que o PC envia para que o NIOS reinicie a execução de seu software. Isto faz com que todos os valores armazenados em memória externa sejam apagados e também reinicia o adaptador RTL. Na figura 33 é apresentada a sequência de transmissão de pacotes que deve ocorrer para uma operação de reset bem sucedida.

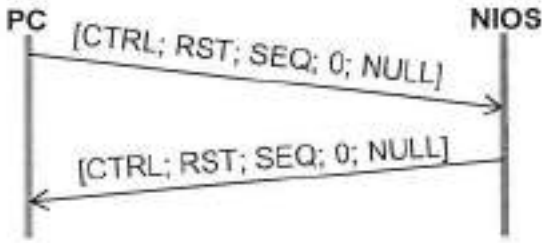

Figura 33. Sequência de pacotes para a operação de reset

Se a operação RST for bem sucedida, o NIOS irá enviar um eco da mensagem ao PC para, imediatamente em seguida, iniciar o processo de reset.

O pacote do subtipo SYNC (valor binário "01") é enviado pelo PC para efetuar a configuração da frequência do sinal de *clock* que alimenta o hardware do usuário. Um pacote de resposta por parte do NIOS idêntico ao pacote enviado pelo PC indicará o sucesso da operação. Na figura 34 é apresentada a sequência de transmissão de pacotes esperada.

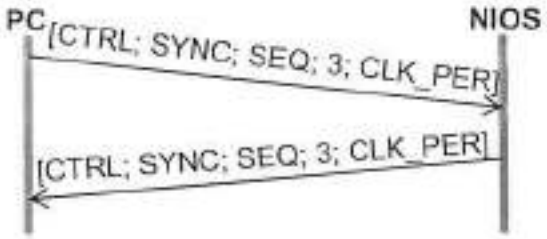

Figura 34. Sequência de pacotes para a operação de sincronização

O campo CLK_PER é um valor inteiro sem sinal de 3 bytes, e representa o número de períodos adicionais que devem ser somados ao *clock* base para se gerar o *clock* do hardware do usuário ("rtl_clk"). Por exemplo, um valor de CLK_PER igual a zero indica que o sinal "rtl_clk" terá a mesma frequência do *clock* base (50 MHZ), enquanto um valor de CLK_PER igual a um significa que o período do sinal "rtl_clk" será igual ao período base adicionado de um período, ou seja, a frequência do sinal "rtl_clk" será de 25 MHZ.

Os subtipos SZ_IN e SZ_OUT (valores binários "10" e "11", respectivamente), são utilizados pelo PC para se configurar os tamanhos, em bytes, dos blocos de dados que serão lidos e transmitidos às interfaces de leitura e escrita do Adaptador RTL. É imprescindível que esses valores sejam configurados logo ao início da execução do software embarcado e que eles sejam compatíveis com os tamanhos físicos das interfaces de saída e de entrada do adaptador RTL.

A figura 35 mostra a sequência de transmissão de pacotes que deve ocorrer para uma operação de configuração SZ_IN bem sucedida:

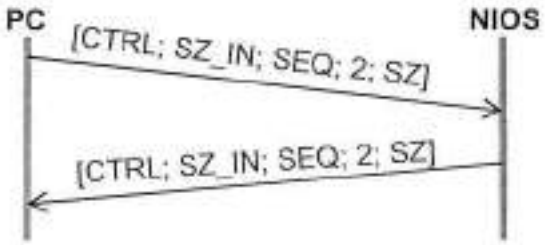

Figura 35. Sequência de pacotes para a configuração do tamanho da interface de entrada

Sendo, como convencionado, SEQ um número de sequência adequado e SZ um payload de 2 bytes contendo o tamanho do bloco de leitura que se deseja utilizar. Uma configuração bem sucedida receberá um pacote de eco por parte do NIOS.

A figura 36 mostra a sequência de transmissão de pacotes que deve ocorrer para uma operação de configuração SZ_OUT bem sucedida:

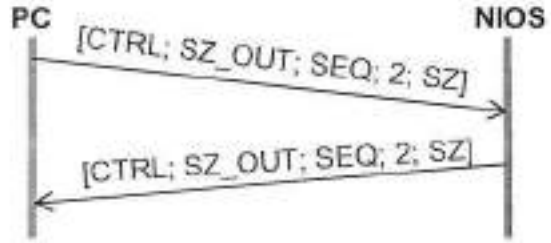

Figura 36. Sequência de pacotes para a configuração do tamanho da interface de saída

Sendo, como convencionado, SEQ um número de sequência adequado e SZ um payload de 2 bytes contendo o tamanho do bloco de escrita que se deseja utilizar. Uma configuração bem sucedida receberá um pacote de eco por parte do NIOS.

3.2.4.3.3 Detecção e tratamento de erro

O canal de comunicação USB escolhido pelo grupo para ser utilizado na transmissão de dados do sistema RTOOL é extremamente confiável, e por isso não será implementado nenhum mecanismo de verificação de consistência de dados, já que isto acarretaria num incremento de complexidade do protocolo de comunicação e numa diminuição do desempenho do software embarcado.

Além disso, pacotes de dados com inconsistência não chegam a atingir a camada de aplicação, pois são descartados em níveis mais baixos do sistema de comunicação, tornando desnecessária a implementação da verificação de consistência de dados neste protocolo. Contudo, isto levanta a possibilidade de perda integral de pacotes, a qual deve ser percebida pelos membros do canal de comunicação para se evitar o mal funcionamento do sistema RTOOL.

Tendo isto em consideração, este protocolo define o tipo de pacote de erro, o qual pode ser utilizado na sinalização de ocorrências eventuais de perda integral de pacotes de dados. A tabela 7 apresenta os subtipos do pacote "Error Status".

Tabela 7. Subtipos permitidos para o pacote ERR

Valor	Subtipo Error Status
00	Sem Erro (SUCCESS)
01	Falta pacote (MISS)
10	Indefinido (UNDEF)
11	Reservado

Optou-se pela implementação de um mecanismo simples de prevenção de erro devido à perda de pacotes na versão inicial do projeto RTOOL, de modo que se uma perda de pacote for detectada através de um salto inesperado entre dois valores consecutivos do campo "Sequência de pacote", então toda a operação em execução será considerada inválida. Porém, caso esse mecanismo simples venha a reduzir consideravelmente a eficiência do canal de comunicação, então uma versão mais elaborada será proposta.

4. RESULTADOS

Testes iniciais foram realizados com o propósito de verificar a confiabilidade do sistema e o ganho de tempo que a emulação, utilizando a ferramenta RTOOL, fornece em relação à simulação convencional através de softwares.

4.1 Dispositivo RTL usado nos testes

Para a emulação utilizou-se um módulo RTL de cálculo do número Fibonacci, sem erros de projeto, que implementa em hardware, rigorosamente, o seguinte algoritmo:

```
int fibonacci( int x )
{
      int i, n=1, n_ant=0, aux;
      if( x==0 )
            return 0;
      if( x==1 )
            return 1;
      for( i=1; i<x; i++ )
      {
            aux = n;
            n = n+n_ant;
            n_ant = aux;
      }
      return n;
}
```

Apesar de sua compilação resultar em um circuito digital relativamente pequeno (443 elementos lógicos), o código em linguagem VHDL desenvolvido para o módulo Fibonacci é razoavelmente extenso (mais de 800 linhas), e não foi apresentado em sua totalidade neste documento por conveniência.

O motivo pelo qual um algoritmo em linguagem C com 15 linhas de código se transformou em uma descrição RTL, em linguagem VHDL, com mais de 800 linhas de código está no fato de que cada operação matemática e cada variável declarada requer um sub-módulo em hardware para ser implementada. Por exemplo, a operação de soma requer que a seguinte entidade seja adicionada ao projeto:

```
--Descrição do circuito feito por:
--MSc. Mario Raffo : mraffo@lme.usp.br
--Membro do GSEIS
--Doutorando em Engenharia Elétrica
--Universidade de São Paulo

library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY somador IS
   GENERIC( NUMBITS   : NATURAL := 32 );
   PORT(    SIGNAL x  : IN STD_LOGIC_VECTOR(NUMBITS-1 DOWNTO 0);
            SIGNAL y  : IN STD_LOGIC_VECTOR(NUMBITS-1 DOWNTO 0);
            SIGNAL XY : OUT STD_LOGIC_VECTOR(NUMBITS-1 DOWNTO 0));
END somador;

architecture Behavioral of somador is

    SIGNAL xy_temp : STD_LOGIC_VECTOR(NUMBITS DOWNTO 0);

begin

    xy_temp <= ('0'&x) + ('0'&y);
    xy <= xy_temp(NUMBITS-1 DOWNTO 0);

end Behavioral;
```

A operação de soma, executada pelo somador em hardware fornecido, é usada em diversas etapas da rotina de cálculo do número de Fibonacci. Consequentemente, em hardware, foi necessário adicionar um conjunto de multiplexadores à entrada do somador para que os operandos adequados fossem selecionados. O código abaixo mostra a descrição VHDL do multiplexador usado.

```
--Descrição do circuito feito por:
--MSc. Mario Raffo : mraffo@lme.usp.br
--Membro do GSEIS
--Doutorando em Engenharia Elétrica
--Universidade de São Paulo

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY multiplexor2a1 IS
    GENERIC( NUMBITS    : NATURAL := 32);
    PORT(    SIGNAL a   : IN STD_LOGIC_VECTOR(NUMBITS-1 DOWNTO 0);
             SIGNAL b   : IN STD_LOGIC_VECTOR(NUMBITS-1 DOWNTO 0);
             SIGNAL sel : IN STD_LOGIC;
             SIGNAL f   : OUT STD_LOGIC_VECTOR(NUMBITS-1 DOWNTO 0));
END multiplexor2a1;

ARCHITECTURE mux2a1 OF multiplexor2a1 IS

BEGIN

    WITH sel SELECT
        f <= a WHEN '0',
        b WHEN OTHERS;

END mux2a1;
```

Logo, um diagrama do circuito digital equivalente à interligação de multiplexadores à entrada do somador seria tal qual apresentado na figura 37.

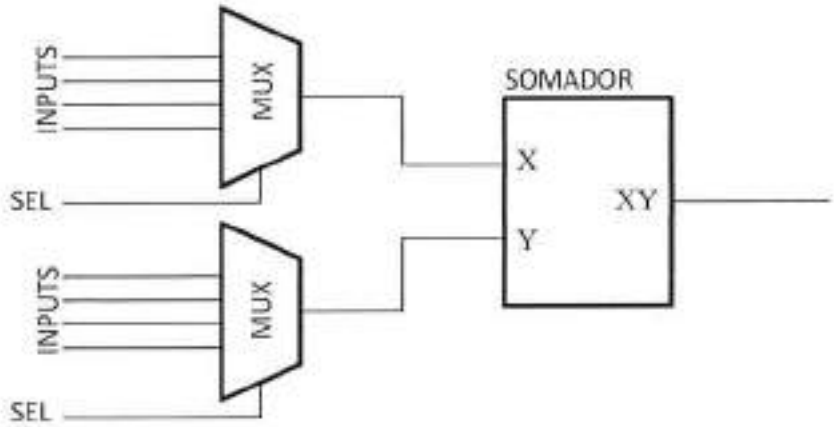

Figura 37. Diagrama do circuito digital

Se para cada operação de soma fosse instanciado um somador, não haveria a necessidade da utilização de multiplexadores de entrada, porque cada conjunto de operandos estaria interligado diretamente às entradas de um somador. Contudo, isto acarretaria no aumento do tamanho do hardware sintetizado pelo compilador VHDL.

Com o intuito de se compatibilizar as interfaces do RTL que calcula o número de Fibonacci com o sistema RTOOL, foram acrescidos a ele módulos DRIVER e MONITOR em conformidade com a interface de comunicação assíncrona do Adaptador RTL. A figura 38 mostra um esquema desta modificação do hardware.

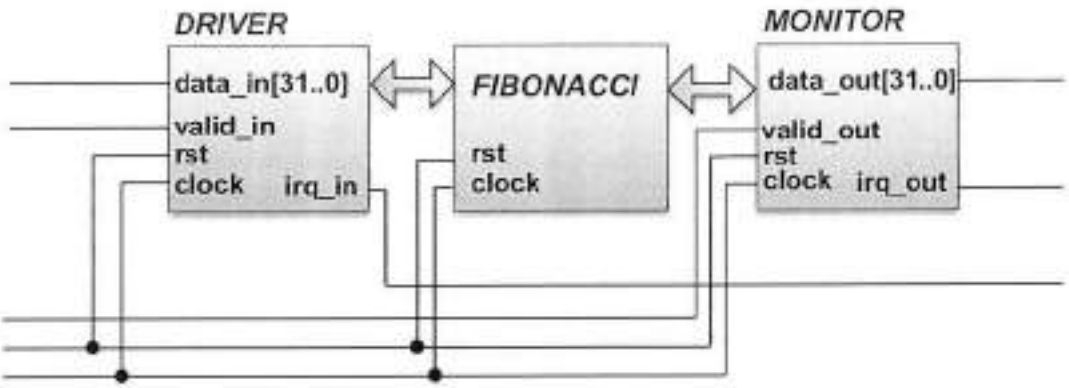

Figura 38. Módulo Fibonacci acrescido de DRIVER e MONITOR. As interfaces específicas do módulo Fibonacci foram omitidas por não serem relevantes nesta figura

Os códigos do DRIVER e MONITOR desenvolvidos para este teste encontram-se abaixo, e servem como exemplos de implementação da comunicação assíncrona com o Adaptador RTL.

Código do DRIVER para o módulo FIBONACCI:

```
ENTITY fibonacci_driver IS
      PORT
      (
            clock    :  IN  STD_LOGIC;
            status   :  IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
            data_in  :  IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
            valid_in :  IN  STD_LOGIC;
            rst      :  IN  STD_LOGIC;
            irq_in   :  OUT  STD_LOGIC := '0';
            go_i     :  OUT  STD_LOGIC := '0' ;
            data_out :  OUT  STD_LOGIC_VECTOR(31 DOWNTO 0)
      );
END fibonacci_driver;

ARCHITECTURE TCGV OF fibonacci_driver IS

      signal state : std_logic_vector( 2 downto 0 ) := "000";

BEGIN

      process (clock)
      begin
            if( clock'event and clock='0' ) then
                  -- reseta o DRIVER de forma sincrona
                  if( rst='1' ) then
                        state <= "000";
                        irq_in <= '0';
                        go_i <= '0';
                  -- gera interrupção para pedir novo dado de entrada
                  elsif( status="01" and state="000" ) then
                        irq_in <= '1';
                        go_i <= '0';
                        state <= "001";
                  -- aguarda indicação de dado válido
                  elsif( valid_in='1' and state="001" ) then
                        data_out <= data_in;
                        irq_in <= '0';
                        go_i <= '0';
                        state <= "010";
                  -- finaliza transação com o Adaptador RTL
                  elsif( valid_in='0' and state="010" ) then
                        state <= "011";
                  -- Envia dado ao RTL UT
                  elsif( state="011" ) then
                        go_i <= '1';
                        state <= "100";
                  -- Aguarda finalização do processamento RTL
                  elsif( status/="01" and state="100" ) then
                        go_i <= '0';
                        state <= "000";
                  end if;
            end if;
      end process;


END TCGV;
```

Código do MONITOR para o módulo FIBONACCI:

```
ENTITY fibonacci_monitor IS
      PORT
      (
            clock          :  IN  STD_LOGIC;
            recv           :  IN  STD_LOGIC;
            data_in        :  IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
            valid_out      :  IN  STD_LOGIC;
            rst            :  IN  STD_LOGIC;
            irq_out        :  OUT  STD_LOGIC := '0';
            data_out       :  OUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
            data_accepted  :  OUT  STD_LOGIC := '0'
      );
END fibonacci_monitor;

ARCHITECTURE TCGV OF fibonacci_monitor IS

      signal state : std_logic_vector( 1 downto 0 ) := "00";

BEGIN

      process (clock)
      begin
            if( clock'event and clock='0' ) then
                  -- reseta o MONITOR de forma sincrona
                  if( rst='1' ) then
                        state <= "00";
                        irq_out <= '0';
                        data_accepted <= '0';
                  -- Aguarda RTL gerar uma saida
                  elsif( recv='1' and state="00" ) then
                        data_accepted <= '1';
                        data_out <= data_in;
                        state <= "01";
                  -- Gera uma interrupção para enviar dados ao Adaptador
                  elsif( recv='0' and state="01" ) then
                        data_accepted <= '0';
                        irq_out <= '1';
                        state <= "10";
                  -- Aguarda confirmação de recebimento
                  elsif( valid_out='1' and state="10" ) then
                        irq_out <= '0';
                        state <= "11";
                  -- Finaliza transação com o Adaptador RTL
                  elsif( valid_out='0' and state="11" ) then
                        state <= "00";
                  end if;
            end if;
      end process;

END TCGV;
```

O mapeamento lógico entre os sinais do DRIVER o do MONITOR com os sinais do Adaptador RTL são apresentados na tabela 8.

Tabela 8. Mapeamento com a interface do Adaptador RTL.

Adaptador RTL	DRIVER	MONITOR
rtl_clk	clock	clock
rst	rst	rst
out_bits	data_in	-
irq_dri	irq_in	-
valid_dri	valid_in	-
in_bits	-	data_out
irq_mon	-	irq_out
valid_mon	-	valid_out

Os demais portos do DRIVER e do MONITOR são referentes às interfaces do módulo de cálculo do número de Fibonacci.

4.2 Teste de confiabilidade

Primeiramente foi realizado um teste para se averiguar a confiabilidade do sistema RTOOL, ou seja, um teste abrangente para validar a consistência do sistema como um todo.

O teste consistiu no envio de 128 milhões de números inteiros aleatórios com 32 bits de comprimento (512 MB de dados) à placa de prototipagem para posterior verificação do número de Fibonacci calculado. O tamanho das transações também foi definido aleatoriamente para que pacotes de diversos tamanhos fossem utilizados durante o teste. O Diagrama da figura 39 esquematiza o processo de geração de vetores de entrada e saída.

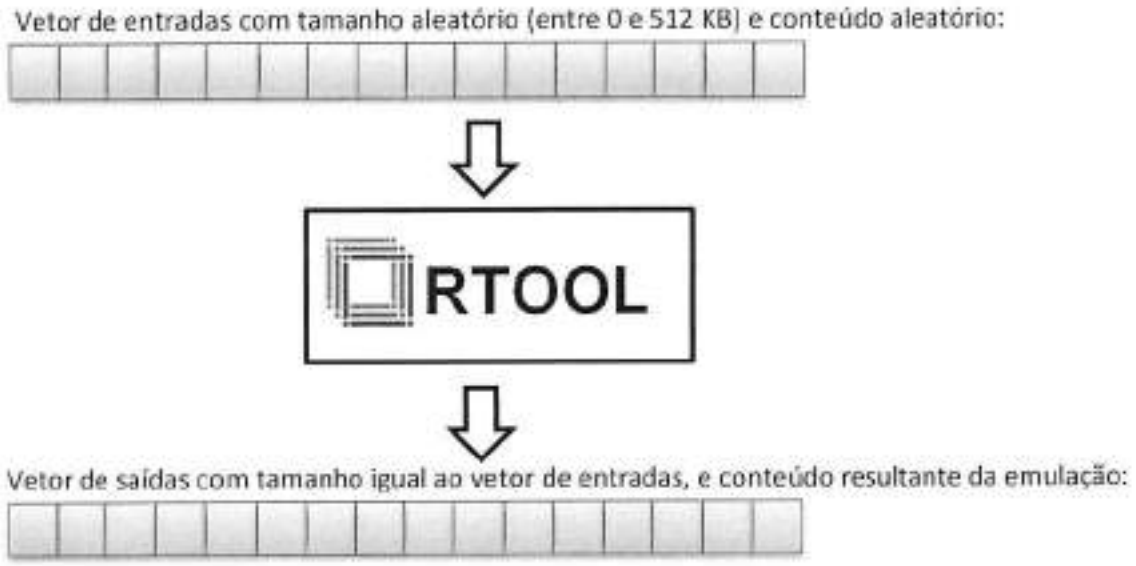

Figura 39. Processo de geração de vetores de entrada e saída

Em cada iteração do laço principal da aplicação de teste de confiabilidade (ANEXO A), um vetor de entradas é gerado, com as características apresentadas na figura 39, e enviado ao RTOOL. Em seguida uma requisição de leitura é feita para que a aplicação possa receber o resultado do cálculo executado pelo hardware do módulo Fibonacci. O resultado então será conferido para saber se há erros, e em seguida uma nova iteração será executada. A condição de parada do laço é atingida quando o número total de palavras enviadas (transações acumuladas) exceder 128 milhões.

Os resultados do teste de confiabilidade são apresentados na tabela 9.

Tabela 9. Teste de confiabilidade

Quantidade de palavras enviadas	128000000
Quantidade de erros no cálculo	0
Frequência do *clock* do RTL Fibonacci	5 MHZ
Tempo de processamento	3 horas e 12 minutos

Este primeiro teste foi importante para certificar que as rotinas de gerenciamento de memória estão funcionando corretamente, uma vez que o fluxo total de dados (enviado mais recebido), por ter excedido o montante de um gigabyte, exercitou exaustivamente os *buffers* circulares implementados sobre a memória SDRAM.

4.3 Teste de desempenho

O segundo teste teve o intuito de se avaliar o desempenho da ferramenta RTOOL em comparação à simulação com softwares convencionais. No caso da simulação, usou-se um computador com as características de hardware e software apresentadas na tabela 10.

Tabela 10. Características do PC usado na simulação

Processador	Intel i7-2630QM @ 2 GHZ
Sistema Operacional	Windows 7 Home Edition
Versão de Software de simulação	ModelSim 6.6d Starter Edition

Para o teste de desempenho foi utilizado um vetor de entrada de 63400 posições de 32 bits. O conteúdo do vetor foi uma sequência linear crescente. Este mesmo vetor foi utilizado para excitar tanto a entrada do RTL emulado quanto do simulado, como mostra a figura 40.

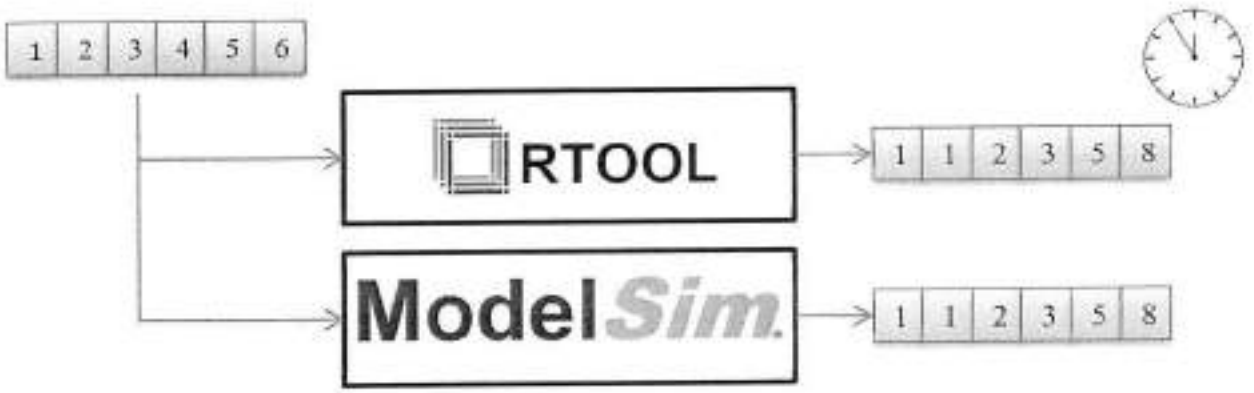

Figura 40. Comparação de desempenho

Pelo fato do módulo Fibonacci, especificamente, gerar uma palavra (inteiro de 32 bits) de resultado para cada palavra de entrada, um vetor de entrada com 63400 inteiros (190,2 KB) produziu um vetor de saída com 63400 inteiros.

O resultado obtido na comparação de desempenho foi acima do esperado. Previa-se inicialmente que a emulação fosse gerar, em média, resultados dez vezes mais rapidamente do que a simulação (Serrestou, Beroulle, Robach, 2007), no entanto o desempenho da ferramenta foi substancialmente maior, tal qual apresentado na tabela 11.

Tabela 11. Comparação de desempenho entre PC e sistema RTOOL

Método	Simulação no software ModelSim	Sistema RTOOL
Erros	0	0
Tempo de processamento	307,90 segundos	11,42 segundos
***Throughput* (Resultados/s)**	823,64 B/s	22,2 KB/s

Enquanto o simulador Modelsim precisou de 307,90 segundos para processar os dados, a emulação via RTOOL necessitou de apenas 11,42 segundos, ou seja, um ganho de tempo de aproximadamente 27 vezes.

O código em linguagem C da aplicação desenvolvida para este teste de desempenho é fornecido no ANEXO B, e o *testbench* usado no software ModelSim é fornecido no anexo C.

5. DISCUSSÃO

Os resultados dos testes iniciais mostraram um desempenho bastante superior da emulação sobre a simulação. O tempo gasto pela emulação utilizando o RTOOL foi 27 vezes menor que o gasto na simulação pelo MODELSIM. Espera-se que para descrições RTLs de maior complexidade esta diferença seja mais acentuada. Há, entretanto, alguns fatores limitantes para o desempenho do RTOOL em módulos maiores de RTL.

A taxa de transferência da interface JTAG-UART é limitada a 1.2 Mbit/s. Em projetos mais complexos em que os vetores de entrada e saída são maiores, isso pode se tornar um problema, pois embora a emulação seja bastante rápida, a velocidade do tráfego de dados fica limitada à taxa máxima de transferência do canal.

Aliado a isso, o fato da memória externa SDRAM possuir apenas 8 Mbytes não só limita o tamanho dos pacotes de transação, como também diminui o desempenho do sistema, visto que na falta de memória disponível para leitura ou escrita do PC, este fica ocioso.

Além da questão do desempenho, outra característica que deve ser observada é com relação à implementação do DRIVER e MONITOR.

Devido à simplicidade do módulo testado, tanto o DRIVER quanto o MONITOR foram desenvolvidos de modo bastante simples. Nota-se que ambos possuem na descrição funcional apenas um único processo síncrono, que é responsável tanto pela captura de dados a partir do adaptador RTL, como pelo estímulo do RTL. Todavia, projetos mais complexos executam simultaneamente vários processos síncronos e assíncronos. Desta forma, estes componentes, da maneira como estão implementados, não são factíveis para uso em projetos reais.

Atualmente o método de teste de projetos de hardware mais difundido no mercado tem sido o da verificação funcional (Romero, 2005) que examina a corretude do módulo sob teste a partir da comparação entre o modelo de referência, escrito em linguagem de alto nível (C, C++, Java ou Matlab) e um modelo RTL, escrito em linguagens HDL (VHDL, VERILOG ou SYSTEM C). Esta comparação é feita por meio da simulação (Bergeron, 2003).

O desenvolvimento de um DRIVER e um MONITOR adequado, portanto, foi feito observando este contexto.

De um modo geral, os usuários adeptos a esta forma de depuração possuem os módulos de DRIVER e MONITOR desenvolvidos para simulação em softwares.

Desta maneira, o processo de captura de vetores de estimulo é feito acessando arquivos que os contém.

Para utilizar a ferramenta RTOOL, no entanto, é preciso alterar este processo. Para os projetistas que desenvolveram estes módulos em VHDL ou VERILOG, linguagens sintetizáveis em FPGA, essa tarefa é trivial, pois basta realizar as modificações seguindo o protocolo de comunicação com o adaptador RTL, conforme ilustrado na carta de tempos nas figuras 19 e 20.

No entanto, os usuários que desenvolveram o DRIVER e MONITOR em SystemC, acessam os vetores de entrada a partir de estruturas de alto nível similares aos *structures* da linguagem C que agrupam diversos tipos de dados distintos. Para que estes usuários consigam fazer uso da ferramenta RTOOL, existem no mercado softwares capazes de converter os seus módulos implementados em SystemC, não sintetizável em FPGA, em linguagens sintetizáveis, VHDL ou VERILOG. Porém, a falta de experiência destes usuários em manipular linguagens HDL sintetizáveis, para criar um processo de captura de dados equivalente àquele escrito em SystemC, torna-se um obstáculo para o uso da ferramenta.

Com o intuito de viabilizar o RTOOL para uma maior gama de usuários, os autores decidiram disponibilizar modelos de código de registro de vetores de entrada e envio de vetores de resposta facilmente adaptável de acordo com as necessidades de cada projeto. Os códigos encontram-se disponíveis no apêndice D.

A figura 41, ilustra como os vetores de entrada organizados em forma de estruturas, no caso do SystemC, podem ser acessados a nível de registradores.

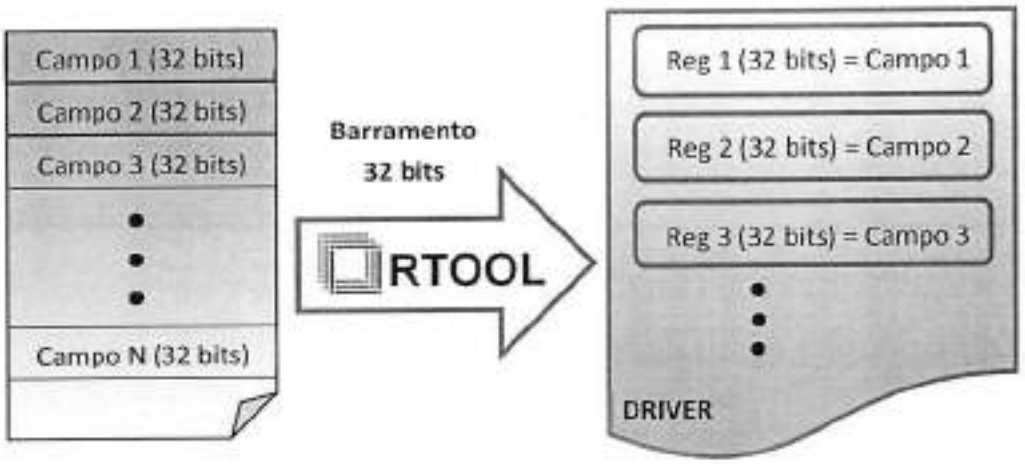

Figura 41. Esquema que mostra a equivalência entre os campos que compõem as estruturas de alto nível e os registradores presentes no DRIVER

Por fim, além dos limitantes de desempenho e de implementação, há também a limitação física no que diz respeito ao tamanho do FPGA, que comporta 35 mil

elementos lógicos. Estimando que o RTOOL vá consumir cerca de 10% deste total com a sua arquitetura composta pelo processador, memória SDRAM e adaptador RTL, sobram ainda 31,5 mil elementos lógicos, o que pode ser insuficiente para alguns projetos maiores.

6. CONCLUSÃO

O sistema RTOOL, uma ferramenta de prototipagem rápida de descrições RTL para fins de depuração, apresentou desempenho acima do esperado para o circuito de teste utilizado. Observou-se um ganho de desempenho de 27 vezes com relação à execução de simulação com o software ModelSim, da Menthor Graphics, sobre um computador com especificações atuais.

Os autores percebem que cada metodologia de depuração de descrições RTL é pertinente dentro de uma determinada etapa do desenvolvimento de um projeto de hardware digital. Enquanto a simulação convencional apresenta, em média, menor desempenho, ela se mostra como uma solução muito mais versátil do ponto de vista de mapeamento de erros (figura 42) de projeto, porque permite que todos os sinais e processos internos do hardware em depuração sejam analisados. Todavia, essa maior versatilidade vem com um preço, a redução de desempenho na simulação de circuitos de grande porte.

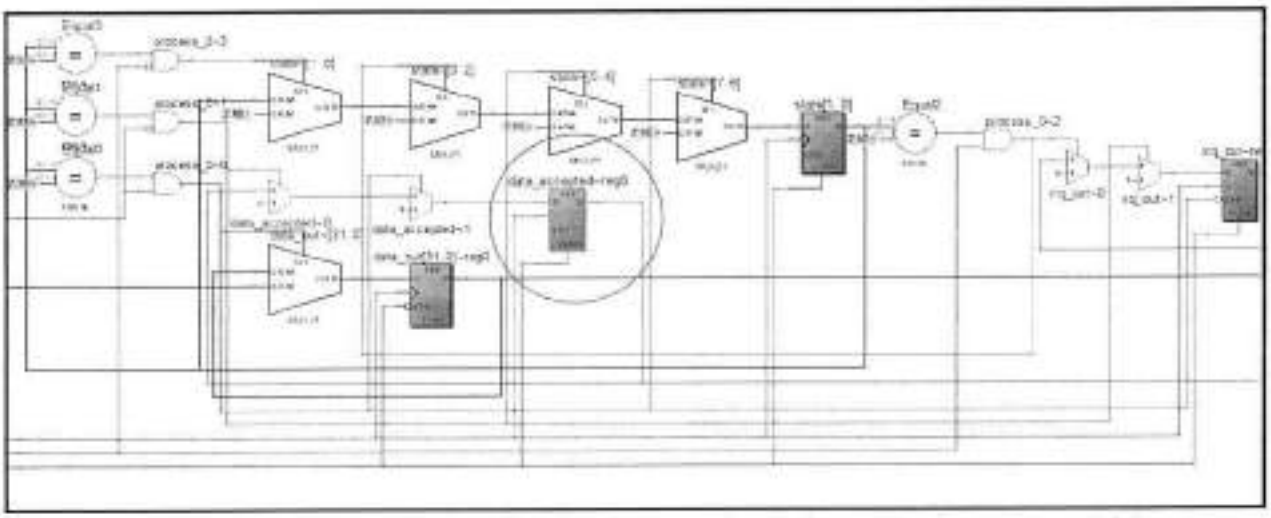

Figura 42. Através da simulação convencional pode-se analisar todos os sinais e processos internos do hardware, com o intuito de se mapear precisamente a fonte de erro

Logo, a simulação convencional adequa-se mais às etapas de desenvolvimento dos sub-módulos de uma descrição RTL, em que se deseja validar os componentes de menor porte constituintes de um circuito digital mais complexo.

Uma ferramenta de prototipagem, por sua vez, é mais adequada à etapa de depuração "*top-level*", ou seja, na depuração de um projeto em sua etapa final de desenvolvimento, na qual todos os módulos de menor porte constituintes da descrição RTL já passaram por um estágio de depuração individual e foram interligados entre si para a construção da entidade *top-level*. O objetivo desta etapa

é a execução de uma depuração exaustiva do hardware para se analisar se ele funciona corretamente para um conjunto abrangente de sinais de entrada.

O uso de emulação na verificação funcional projetos de hardware vem crescendo nos últimos anos devido ao aumento da complexidade destes projetos (Wilson Reaserch Group, 2010), abrindo espaço para ferramentas que visem automatizar este processo.

As características fundamentais que devem ser apresentadas por uma ferramenta de auxílio à emulação de hardware são: confiabilidade, desempenho e usabilidade.

Se a ferramenta não garantir resultados confiáveis, ou seja, caso erros internos da própria ferramenta venham a interferir nos resultados da emulação do projeto do usuário, não haverá demanda para tal ferramenta. O sistema RTOOL, em uma análise inicial, forneceu resultados que mostram que sua arquitetura de hardware e seu conjunto de softwares de comunicação de dados são robustos e confiáveis.

Uma ferramenta, mesmo que comprovadamente confiável, não terá mercado se seu desempenho for pobre. Os testes efetuados com o sistema RTOOL mostraram que ele pode fornecer ganhos expressivos de desempenho para circuitos digitais de grande porte.

Por último, porém não menos importante, está a usabilidade do sistema. É de senso comum que projetistas possuem certa inércia para com alterações em suas metodologias de trabalho, ou seja, possuem resistência à adoção de novas ferramentas em detrimento das ferramentas às quais estão acostumados. Portanto, é importante que uma nova ferramenta seja de fácil utilização, caso contrário sua adoção na indústria será reduzida. O sistema RTOOL buscou especificar interfaces simples para a integração de uma descrição RTL que se deseje depurar, tendo em vista reduzir o esforço que um novo usuário enfrentará ao decidir utilizá-lo. Também optou-se pelo desenvolvimento de uma interface gráfica simples e intuitiva para guiar o usuário nas diferentes etapas do uso do RTOOL.

O caráter inovador da solução consiste no fato de ela ser geral e, portanto, facilmente adaptável aos diferentes projetos de teste. Essa generalidade deriva do modelo de testbench geral de Bergeron (Bergeron, 2003) para o qual a ferramenta foi desenvolvida. Desta forma as especificidades de cada projeto são contempladas pelo módulo do DRIVER e MONITOR, cujo desenvolvimento fica a cargo do usuário, enquanto o restante da arquitetura mantém-se fixa.

Como visto na discussão dos resultados, um dos fatores que podem vir a limitar o desempenho do sistema RTOOL é sua taxa de transmissão de dados limitada à cerca de 1,2 Mbit/s. Logo, o aprimoramento futuro desta ferramenta deve buscar

alternativas de comunicação de dados com maior taxa efetiva, como, por exemplo, a interface ethernet. Aliado à isso, deverá ser feita uma otimização do software embarcado do sistema RTOOL e um aumento da frequência de operação do processador embarcado NIOS II para que o sistema suporte o incremento na taxa de transmissão de dados.

O projeto RTOOL foi bastante completo na área de sistemas eletrônicos, envolvendo o estudo de protocolos de comunicação de baixo nível (ex: Barramento AVALON), arquiteturas de hardware (ex: Processador NIOS II), protocolos de comunicação de alto nível (ex: USB, TCP/IP), metodologias de desenvolvimento de software embarcado (ex: Máquina de estados finitos), elaboração de interfaces gráficas (através de ferramentas profissionais como o Microsoft Visual Studio), etc. Logo, os autores estão satisfeitos com os resultados obtidos tendo em vista a abrangência de disciplinas pertinentes ao projeto.

Ambos os objetivos traçados no começo do projeto puderam ser atingidos, os quais são novamente apresentados abaixo por conveniência:

- A meta deste projeto, estimada a partir de resultados observados em Serrestou, Beroulle, Robach, 2007, será uma melhora de desempenho no tempo de verificação da descrição RTL de dez vezes.

- O sistema RTOOL deve ser de fácil implantação e de fácil utilização, ou seja, o esforço do usuário nas tarefas de integração da sua descrição RTL ao hardware do sistema RTOOL, de compilação do hardware integrado, de programação do FPGA núcleo do sistema e de utilização da API do sistema RTOOL deve ser mínimo.

O primeiro objetivo foi atingido pelo desempenho observado na etapa de testes, enquanto o segundo foi contemplado pelo desenvolvimento de uma GUI amigável para o sistema RTOOL.

Os autores acreditam que este trabalho será uma contribuição útil ao acervo do departamento de Engenharia de Sistemas Eletrônicos da Escola Politécnica da USP, fornecendo conhecimentos teóricos pertinentes ao processo de depuração de hardware através da adoção da metodologia de emulação.

7. REFERÊNCIAS

- J. Bergeron, "Writing Testbenches: Functional Verification of HDL Model", 2. ed. Kluwer Academic Publishers, Boston, 2003.
- Rogers, Everett M. "Diffusion of Innovations", 5. ed. New York: Free Press, 2003. p. 273.
- Serrestou, Youssef; Beroulle, Vincent; Robach, Chantal. "Impact of Hardware Emulation on the Verification Quality Improvement". Valência, Espanha, 2007.
- Squillero, Giovanni; Reorda, Matteo Sonza; Corno, Fulvio. "RT-level ITC'99 Benchmarks and First ATPG Results". IEEE Design & Test of Computers. Setembro, 2000.
- Tocci, R J.; Widmer, N S. "Sistemas Digitais – Princípios e Aplicações" 10. ed. Pearson, São Paulo , 2007. Cap. 3, p. 82.
- Tsai, Bihru. "Accelerating VHDL Model Execution". Fremont, California.
- Wilson research group. "The 2010 Wilson research group and Mentor Graphics functional verification study". 2010

8. BIBLIOGRAFIA

- Avalon Interface Specifications. Version MNL-AVABUSREF-2.0. Altera Corporation, 2011.
- DE2 Development and Education Board User Manual. Version 1.4. Altera Corporation, 2006.
- Embedded Peripherals IP. Version UG-01085-11.0. Altera Corporation, 2011.
- Hennessy, John L., and David A. Patterson. Computer Architecture: A Quantitative Approach. 4th ed. Morgan Kaufmann, 2006.
- JTAG UART Core. Version NII51009-7.1.0. Altera Corporation, 2007.
- Nios II Hardware Development Tutorial. Version TU-N2HWDV-4.0. Altera Corporation, 2011.
- Vahid, Frank, and Tony Givargis. Embedded System Design: A Unified Hardware/Software Introduction. John Wiley & Sons, 2002.
- Vahid, Frank, Jie Gong, Sanjiv Narayan, and Daniel D. Gajski. Specification and Design of Embedded Systems. 1st ed. Prentice Hall, 1994.
- VHDL Reference Manual. Synario Design Automation, 1997

APÊNDICE A – CÓDIGO DO TESTE DE CONFIABILIDADE

```
#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "rtool_int.h"

// Alocação estática de buffers de entrada/saída
unsigned int buf_out[262144], buf_in[262144];

// Modelo de referência para o cálculo do número
// de Fibonacci
int fibonacci( int x )
{
	int i, n=1, n_ant=0, aux;
	if( x==0 )
		return 0;
	if( x==1 )
		return 1;
	for( i=1; i<x; i++ )
	{
		aux = n;
		n = n+n_ant;
		n_ant = aux;
	}
	return n;
}

// Efetua procedimento de inicialização da ferramenta,
// e se um erro ocorrer imprime uma mensagem e finaliza
// o programa.
void Initialize()
{
    int err;

    // Executa função de conexão com o Driver USB
    if( (err=RtoolStart()) != 0 )
    {
        printf( "ERRO: Inicializacao RTOOL %d!\n", err );
        exit(-1);
    }

    // Envia um comando de Reset à placa de prototipagem
    if( (err=RtoolReset()) != 0 )
    {
        printf( "ERRO: Reset de firmware do RTOOL %d!\n", err );
        exit(-1);
    }

    // Configura tamanho da interface do DRIVER (fixo em 32 bits!)
    if( (err=RtoolSizeIn(32)) != 0 )
    {
        printf( "ERRO: Configuracao SZ_IN do RTOOL %d!\n", err );
```

```
        exit(-1);
    }

    // Configura tamanho da interface do MONITOR (fixo em 32 bits!)
    if( (err=RtoolSizeOut(32)) != 0 )
    {
        printf( "ERRO: Configuracao SZ_OUT do RTOOL %d!\n", err );
        exit(-1);
    }

    // Configura clock de 5 MHZ (T=(1+clk_per)/50000000)
    if( (err=RtoolSync(4)) != 0 )
    {
        printf( "ERRO: Configuracao do clock do RTOOL %d!\n", err );
        exit(-1);
    }

    // semente para geração de números aleatórios
    srand( time(NULL) );
}

// Gera um vetor de entradas de forma aleatória
void GenerateInputs( unsigned int *buf, int len )
{
    int i;
    for( i=0; i<len; i++ )
        buf[i] = rand()%256;
}

// Limpa conteúdo do vetor
void ClearBuffer( unsigned int *buf, int len )
{
    int i;
    for( i=0; i<len; i++ )
        buf[i] = 0;
}

// A função Checker() usa o modelo de referência para
// verificar se o resultado calculado em hardware é
// compatível com o resultado calculado em software.
int Checker( unsigned int *buf_num, unsigned int *buf_fibo, int len )
{
    int i, err=0;
    for( i=0; i<len; i++ )
    {
        if( fibonacci( buf_num[i] ) != buf_fibo[i] )
            err++;
    }
    return err;
}

// Rotina principal
int main()
{
    int i, err, err_total, transf, transf_total, t;

    // Inicialização do sistema
    Initialize();
```

```
    // Cada iteração do loop efetua uma transação com
    // tamanho diferente, de modo a verificar o comportamento
    // do sistema RTOOL em diversos casos
    err_total = 0;
    for( transf_total=0; transf_total<128000000; transf_total+=transf )
    {
        // Configura tamanho randômico da transferência
        // (entre 0 e 512 KB) e atualiza vetores
        transf = rand()%(512*1024);
        GenerateInputs( buf_out, transf );
        ClearBuffer( buf_in, transf );

        // Envia vetor de dados à placa de prototipagem através da
        // função RtoolWrite(). Se um erro ocorrer, o usuário será
        // notificado.
        if( (err=RtoolWrite( buf_out, transf*sizeof(unsigned int) )) != 0 )
        {
            printf( "ERRO: WRITE %d!\n", err );
            exit(-1);
        }

        // Requisita a leitura de dados e verifica o tipo de erro.
        // Se o erro for REQUEST_DENIED isto significa que a placa
        // de prototipagem ainda não terminou os cálculos.
        do
        {
            err = RtoolRead( buf_in, transf*sizeof(unsigned int) );
            if( err<0 && err!=REQUEST_DENIED )
            {
                printf( "ERRO: READ %d!\n", err );
                exit(-1);
            }
            printf( "A placa ainda nao terminou o calculo...\n" );
            Sleep( 500 );
        } while( err==REQUEST_DENIED );

        // Efetua a verificação do resultado da emulação e incrementa
        // variável do valor total de erros.
        err = Checker( buf_out, buf_in, transf );
        printf( "PARCIAL = %d erros em %d palavras\n", err, transf );
        err_total += err;
    }

    // Finaliza conexão com o Driver USB, termina programa e
    // apresenta resultados
    printf( "TOTAL = %d erros em %d palavras\n", err_total, transf_total );
    RtoolClose();
    return 0;
}
```

APÊNDICE B – CÓDIGO DO TESTE DE DESEMPENHO

```
#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "rtool_int.h"

// Alocação estática de buffers de entrada/saida
unsigned int buf_out[262144], buf_in[262144];

// Modelo de referência para o cálculo do número
// de Fibonacci
int fibonacci( int x )
{
	int i, n=1, n_ant=0, aux;
	if( x==0 )
		return 0;
	if( x==1 )
		return 1;
	for( i=1; i<x; i++ )
	{
		aux = n;
		n = n+n_ant;
		n_ant = aux;
	}
	return n;
}

// Efetua procedimento de inicialização da ferramenta,
// e se um erro ocorrer imprime uma mensagem e finaliza
// o programa.
void Initialize()
{
    int err;

    // Executa função de conexão com o Driver USB
    if( (err=RtoolStart()) != 0 )
    {
        printf( "ERRO: Inicializacao RTOOL %d!\n", err );
        exit(-1);
    }

    // Envia um comando de Reset à placa de prototipagem
    if( (err=RtoolReset()) != 0 )
    {
        printf( "ERRO: Reset de firmware do RTOOL %d!\n", err );
        exit(-1);
    }

    // Configura tamanho da interface do DRIVER (fixo em 32 bits!)
    if( (err=RtoolSizeIn(32)) != 0 )
    {
        printf( "ERRO: Configuracao SZ_IN do RTOOL %d!\n", err );
```

```
        exit(-1);
    }

    // Configura tamanho da interface do MONITOR (fixo em 32 bits!)
    if( (err=RtoolSizeOut(32)) != 0 )
    {
        printf( "ERRO: Configuracao SZ_OUT do RTOOL %d!\n", err );
        exit(-1);
    }

    // Configura clock de 5 MHZ (T=(1+clk_per)/50000000)
    if( (err=RtoolSync(4)) != 0 )
    {
        printf( "ERRO: Configuracao do clock do RTOOL %d!\n", err );
        exit(-1);
    }

    // semente para geração de números aleatórios
    srand( time(NULL) );
}

// Gera um vetor de entradas de forma aleatória
void GenerateInputs( unsigned int *buf, int len )
{
    int i;
    for( i=0; i<len; i++ )
        buf[i] = i%256;
}

// Limpa conteúdo do vetor
void ClearBuffer( unsigned int *buf, int len )
{
    int i;
    for( i=0; i<len; i++ )
        buf[i] = 0;
}

// A função Checker() usa o modelo de referência para
// verificar se o resultado calculado em hardware é
// compatível com o resultado calculado em software.
int Checker( unsigned int *buf_num, unsigned int *buf_fibo, int len )
{
    int i, err=0;
    for( i=0; i<len; i++ )
    {
        if( fibonacci( buf_num[i] ) != buf_fibo[i] )
            err++;
    }
    return err;
}

// Rotina principal
int main()
{
    int i, err, transf, t;

    // Inicialização do sistema
    Initialize();
```

```
    // Configura tamanho da transferência e
    // inicializa vetores
    transf = 63420;
    GenerateInputs( buf_out, transf );
    ClearBuffer( buf_in, transf );

    // Guarda tempo inicial para a análise
    // de desempenho
    t = clock();

    // Envia vetor de dados à placa de prototipagem através da
    // função RtoolWrite(). Se um erro ocorrer, o usuário será
    // notificado.
    if( (err=RtoolWrite( buf_out, transf*sizeof(unsigned int) )) != 0 )
    {
        printf( "ERRO: WRITE %d!\n", err );
        exit(-1);
    }

    // Requisita a leitura de dados e verifica o tipo de erro.
    // Se o erro for REQUEST_DENIED isto significa que a placa
    // de prototipagem ainda não terminou os cálculos.
    do
    {
        err = RtoolRead( buf_in, transf*sizeof(unsigned int) );
        if( err<0 && err!=REQUEST_DENIED )
        {
            printf( "ERRO: READ %d!\n", err );
            exit(-1);
        }
        printf( "A placa ainda nao terminou o calculo...\n" );
        Sleep( 500 );
    } while( err==REQUEST_DENIED );

    // Verifica o intervalo total decorrido entre a o envio
    // do buffer de entradas do RTL e o recebimento do buffer
    // de saidas calculadas
    t = clock()-t;
    printf( "Tempo de calculo em hardware = %d (ms)\n", t );

    // Efetua a verificação do resultado da emulação e imprime
    // uma mensagem com os resultados.
    err = Checker( buf_out, buf_in, transf );
    printf( "%d erros em %d palavras\n", err, transf );

    // Finaliza conexão com o Driver USB e termina programa
    RtoolClose();
    return 0;
}
```

APÊNDICE C – CÓDIGO DO TESTBENCH DO MODELSIM

```
LIBRARY ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

LIBRARY work;

ENTITY testbench IS
END testbench;

ARCHITECTURE bdf_type OF testbench IS

  COMPONENT rtl_block
    PORT( rst       :  IN  STD_LOGIC;
          rtl_clk   :  IN  STD_LOGIC;
          valid_dri :  IN  STD_LOGIC;
          valid_mon :  IN  STD_LOGIC;
          out_bits  :  IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
          irq_mon   :  OUT  STD_LOGIC;
          irq_dri   :  OUT  STD_LOGIC;
          in_bits   :  OUT  STD_LOGIC_VECTOR(31 DOWNTO 0) );
  END COMPONENT;

  signal rst       :  STD_LOGIC := '1';
  signal rtl_clk   :  STD_LOGIC := '0';
  signal valid_dri :  STD_LOGIC := '0';
  signal valid_mon :  STD_LOGIC := '0';
  signal out_bits  :  STD_LOGIC_VECTOR(31 DOWNTO 0);
  signal irq_mon   :  STD_LOGIC := '0';
  signal irq_dri   :  STD_LOGIC := '0';
  signal in_bits   :  STD_LOGIC_VECTOR(31 DOWNTO 0);
  signal cout      :  integer range 0 to 2147483647 := 0;

BEGIN

  rtl_block_inst : rtl_block
  PORT MAP( rst => rst,
            rtl_clk => rtl_clk,
            valid_dri => valid_dri,
            valid_mon => valid_mon,
            out_bits => out_bits,
            irq_mon => irq_mon,
            irq_dri => irq_dri,
            in_bits => in_bits );

  -- processo da interrupção IRQ_DRI
  process (irq_dri)
    variable aux : integer range 0 to 255 := 0;
  begin
    if( irq_dri'event and irq_dri='1' ) then
        out_bits <= std_logic_vector(to_unsigned(aux,32));
        cout <= cout+1;
        valid_dri <= '1';
        if( aux=255 ) then
            aux := 0;
        else
            aux := aux+1;
```

```
        end if;
    elsif( irq_dri'event and irq_dri='0' ) then
        valid_dri <= '0';
    end if;
  end process;

  -- processo da interrupção IRQ_MON
  process (irq_mon)
    variable aux : STD_LOGIC_VECTOR(31 DOWNTO 0);
  begin
    if( irq_mon'event and irq_mon='1' ) then
        aux := in_bits;
        valid_mon <= '1';
    elsif( irq_dri'event and irq_dri='0' ) then
        valid_mon <= '0';
    end if;
  end process;

END bdf_type;
```

APÊNDICE D – MODELO DE DRIVER E MONITOR

A estratégia adotada no caso do DRIVER foi de criar um processo síncrono para o registro dos vetores de entrada, separando-o do processo de estímulo do RTL. Deste modo o projetista acessa os registradores que contém estruturas de transação de modo análogo ao que era feito no acesso às estruturas em linguagens de alto nível. No MONITOR, estratégia semelhante foi adotada, ou seja, foi criado um processo responsável pela comunicação com o RTL e outro pelo envio de vetores de saída ao adaptador RTL. Acredita-se que implementando dessa forma, usuários da ferramenta possam reaproveitar trechos do código VHDL do DRIVER e MONITOR, em especial, aqueles referentes à comunicação com o adaptador RTL que não sofrem grandes variações de projeto para projeto.

Código do DRIVER do módulo CRC (trechos):

```
entity driver is
    Port (  -- Interface com o Adaptador RTL
            clock     : in  STD_LOGIC;
            data_in   : in  STD_LOGIC_VECTOR (31 downto 0);
            valid_dri: in  STD_LOGIC;
            reset     : in  STD_LOGIC;
            irq_dri   : out STD_LOGIC;
                        .
                        .
                        .
              -- Interface com o RTL
                        .
                        .
                        .
            );
architecture behavioral of driver is
-- Registradores que armazenam a estrutura de transação
signal reg1: STD_LOGIC_VECTOR (31 downto 0);
signal reg2: STD_LOGIC_VECTOR (31 downto 0);
signal reg3: STD_LOGIC_VECTOR (31 downto 0);
                    .
                    .
                    .
signal regN: STD_LOGIC_VECTOR (31 downto 0);
-- Estados da Máquina de Estados do processo de registro da estrutura de
transação
type r_state is (r_request,r_register,r_wait);
signal reg_state: r_state := r_request;

-- Sinais internos do módulo DRIVER
signal reg_counter: integer range 0 to 127 := 0;
signal reg_enable: STD_LOGIC := '0';

begin
      REG_PROC: process (reset, clock)
      begin
            if reset = '1' then
                  reg_counter <= 0;
```

```
                irq_dri <= '0';
                reg_state <= r_request;
                reg1 <= (others=> '0');
                reg2 <= (others=> '0');
                reg3 <= (others=> '0');
                              .
                              .
                              .
                regN <= (others=> '0');
          else
                if reg_enable = '1' then
                      case reg_state is
                            when r_request =>
                                  irq_dri <= '1';
                                  reg_state <= r_register;
                            when r_register =>
                                  if valid_dri = '1' then
                                        if falling_edge(clock) then
                                              if reg_counter = 0 then
                                                    reg1 <= data_in;
                                              elsif reg_counter = 1 then
                                                    reg2 <= data_in;
                                              elsif reg_counter = 2 then
                                                    reg3 <= data_in;
                                                          .
                                                          .
                                                          .
                                              elsif reg_counter = N then
                                                    regN <= data_in;

                                              irq_dri <= '0';
                                              reg_state <= r_wait;
                                        end if;
                                  end if;
                            when r_wait =>
                                  if valid_dri = '0' then
                                        reg_state <= r_request;
                                  end if;
                            when others =>
                                  NULL;
                      end case;
                else
                      irq_dri <= '0';
                end if;
          end if;
    end process;

    -- Inicia aqui o(s) processo(s) do DRIVER responsáveis pelo estimulo
    do RTL, ou seja, geração de sinais de interface do RTL. O contador de
    registro, ou seja, o reg_counter deve ser somado e zerado em momentos
    apropriados que variam de acordo com a implementação do DRIVER. Da
    mesma forma, o sinal de enable para registro e enable para o RTL deve
    ser apropriadamente setado.
end behavioral;
```

Código do MONITOR do módulo CRC (trechos):

```
entity monitor is
      Port (  -- Interface com o Adaptador RTL
            clock     : in  STD_LOGIC;
            valid_mon: in STD_LOGIC;
            reset     : in STD_LOGIC;
            data_out : out STD_LOGIC_VECTOR (31 downto 0);
            irq_mon   : out STD_LOGIC;
            -- Interface com o CRC

                        .
                        .
                        .

      );
end monitor;

Architecture behavioral of monitor is
-- Estados da Máquina de Estados do processo de transmissão da estrutura de
transação
type t_state is (t_transmit, t_wait);
signal ttx_state: t_state := t_transmit;
-- Registrador que armazena CRC serializado vindo do RTL
signal mon_buffer: STD_LOGIC_VECTOR (31 downto 0);
signal ttx_enable: STD_LOGIC;
signal ttx_done: STD_LOGIC;

begin
      TTX_PROC: process(reset, clock, ttx_enable)
      begin
            if reset = '1' then
                  data_out <= (others => '0');
                  irq_mon <= '0';
                  ttx_done <= '0';
            elsif ttx_enable = '1' then
            -- Transmissao para o adaptador RTL
                  if falling_edge(clock) then
                        case ttx_state is
                              when t_transmit =>
                                    data_out <= mon_buffer;
                                    irq_mon <= '1';
                                    if valid_mon = '1' then
                                          irq_mon <= '0';
                                          ttx_state <= t_wait;
                                          ttx_done <= '0';
                                    end if;
                              when t_wait =>
                                    if valid_mon = '0' then
                                          ttx_state <= t_transmit;
                                          ttx_done <= '1';
                                    end if;
                              when others =>
                                    NULL;
                        end case;
                  end if;
            end if;
      end process;
      -- Inicia aqui o(s) processo(s) do MONITOR responsável pelo
      armazenamento da(s) resposta(s) produzida pelo RTL, ou seja,
```

```
        atualização correta do buffer de saida mon_buffer, que varia de
        acordo com o projeto. Da mesma forma, o sinal de enable para envio da
        resposta e enable para o RTL deve ser apropriadamente setado.

end behavioral;
```