Pontifícia Universidade Católica do Rio Grande do Sul

Faculdade de Engenharia – Faculdade de Informática

Curso de Engenharia de Computação

DESENVOLVIMENTO DE UM WEBSERVICE PARA MONITORAMENTO REMOTO DE SISTEMAS DIGITAIS IMPLEMENTADOS EM FPGA

VOLUME FINAL DO TRABALHO DE CONCLUSÃO

Autores:
DOUGLAS MALDANER ZANCHIN
MATHEUS DOS SANTOS OLEIRO

Orientador:
Prof. Dr. Fernando Gehm Moraes

Porto Alegre, Novembro de 2011.

DOUGLAS MALDANER ZANCHIN
MATHEUS DOS SANTOS OLEIRO

DESENVOLVIMENTO DE UM WEBSERVICE PARA MONITORAMENTO REMOTO DE SISTEMAS DIGITAIS IMPLEMENTADOS EM FPGA

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

Orientador: Fernando Moraes

Porto Alegre
2011

Agradecimentos

Em primeiro lugar quero agradecer a Deus por ter me colocado no caminho certo, no momento certo, e, ainda, por ter colocado no meu caminho as pessoas certas.

Além Dele, em especial a todos da minha família que acreditaram em mim. A minha mãe que fez, e faz, de tudo para me ver feliz, independente das minhas escolhas. Ao meu irmão que por vezes foi, e é, mais do que só um irmão. A Alê que me incentiva a cada escolha que eu faço. E em especial ao meu pai. A pessoa que mais lutou para eu chegar até esse momento e, com a conclusão desse curso, realiza junto comigo esse sonho.

Dentre as pessoas certas que Deus colocou no meu caminho, quero agradecer aos três professores que, lá em 2008 para o projeto PARKS, acreditaram em mim, me mostraram realmente o que era a engenharia, e me abriram, e continuam abrindo, diversas portas: César Marcon, Fernando Moraes e Ney Calazans. Ainda gostaria de agradecer ao professor Celso Costa por ter me mostrado que o fato de dominar os Sistemas Operacionais é fundamental além de ser uma atividade profissionalizante. A professora Leticia Pöhls que me apoia e incentiva à continuar a minha carreira acadêmica.

Gostaria de agradecer ao meu companheiro nesse e em tantos outros trabalhos, Douglas Zanchin. Acredito que para superar um curso de engenharia precisamos de aliados fortes. Esse foi o meu principal aliado. Juntos e com bastante esforço superamos as dificuldades à nós propostas. Certamente em todos esses anos, para chegarmos aos nossos objetivos, passamos mais tempo juntos do que com as nossas famílias. Criamos assim um forte laço de companheirismo e verdadeira amizade. Obrigado por me ajudar a chegar até aqui e por me 'aturar' esses anos todos.

Ainda em tempo, ao nosso orientador nesse trabalho, o professor Fernando Moraes, pela compreensão nos momentos em que o trabalhou não teve a evolução que gostaríamos e pela oportunidade de desenvolver um projeto gratificante e que pode ter uma continuidade no grupo GAPH.

Por fim, gostaria de agradecer aos que acreditaram, e principalmente aos que não, em mim lá no longínquo ano de 2006, quando saí de casa para essa aventura.

Matheus Oleiro

Agradecimentos

Primeiramente agradeço a Deus por sempre ter me guiado para trilhar o melhor caminho durante esta trajetória e com seu amor Divino iluminado em cada decisão a ser tomada.

Agradeço a todos os meus familiares, pois sempre me deram força, coragem e amor quando precisei. Me incentivaram e me apoiaram em momentos de fraqueza, fazendo com que buscasse os objetivos com maior garra. Aqui cito meus pais, que me educaram e que fizeram o que podiam e também o que não podiam para me proporcionar o que não tiveram. Lembro em especial também minha noiva, sinônimo de carinho, amizade, companheirismo e felicidade.

Aos amigos que sempre estiveram junto durante este tempo de estudo, aprendizado, brincadeiras e diversão, momentos de descontração e momentos sérios. Sempre lutando juntos. Em especial lembro de meu querido colega e amigo Matheus. Tantos momentos juntos, tantas noites mal dormidas, tantas quedas, mas mais obstáculos vencidos. Tudo isso culminou com o desenvolvimento deste trabalho. Agradeço pela oportunidade de trabalhar e aprender ao seu lado.

A todos os professores que contribuíram para meu desenvolvimento pessoal e profissional. Agradeço em especial ao Prof. Dr. Fernando Moraes, nosso orientador, professor e amigo. Por acreditar que este trabalho pudesse ser realizado com sucesso e também por incentivar e auxiliar nos momentos de dificuldade e perda de rumo.

Ao grupo GAPH, pelo apoio e infra-estrutura para desenvolvimento deste trabalho. Aos membros do grupo, pela disponibilidade e auxílio em vários momentos de necessidade.

A todos aqueles que contribuíram direta ou indiretamente para que este trabalho fosse concluído com sucesso.

Douglas M. Zanchin

Índice

Índice de Figuras

Lista de Siglas

ASIC – *Application Specific Integrated Circuit*
BCTCP – *Baixo Custo Transport Control Protocol*
DDR – *Double Data Rate*
DHCP – *Dynamic Host Configuration Protocol*
DMZ – *Demilitarized Zone*
FPGA – *Field Programmable Gate Array*
FTP – *File Transfer Protocol*
HTTP – *Hypertext Transfer Protocol*
HTTPS – *HyperText Transfer Protocol Secure*
IP – *Internet Protocol*
MAC – *Media Access Control*
MII – *Media Independent Interface*
MIME – *Multipurpose Internet Mail Extensions*
OSI – *Open Systems Interconnection*
PCI – *Peripheral Component Interconnect*
PHP – *Personal Home Page*
PHY – *Physical Layer*
RFC – *Request for Comments*
TCC – *Trabalho de Conclusão de Curso*
TCP – *Transmission Control Protocol*
UDP – *User Datagram Protocol*
URL – *Uniform Resource Locator*
VHDL – *Very High Speed Integrated Circuits Hardware Description Language*
WWW – *World Wide Web*

DESENVOLVIMENTO DE UM WEBSERVICE PARA MONITORAMENTO REMOTO DE SISTEMAS DIGITAIS IMPLEMENTADOS EM FPGA

RESUMO

No presente Trabalho de Conclusão de Curso (TCC) é abordado o uso de redes Ethernet para configuração e monitoramento de dispositivos lógicos programáveis (FPGAs). Estes dispositivos estão sendo cada vez mais empregados em indústrias de sistemas eletrônicos, embarcados e de tempo real, devido à facilidade de configuração e alto nível de desempenho. Outro ponto atrativo destes dispositivos é a presença freqüente de módulos MAC Ethernet nativos, o que facilita a estruturação de interfaces Ethernet embarcadas no FPGA. Em virtude da crescente evolução da Internet e da facilidade em utilizá-la através de computadores, tablets ou celulares, o controle remoto à distância tende a ficar cada vez mais acessível. Apresenta-se no decorrer deste TCC uma solução no nível de software que realiza a comunicação de um módulo de hardware através da rede com um dispositivo hospedeiro. Este software é executado em Sistema Operacional Linux e utiliza troca de mensagens através do protocolo BCTCP [SIL11]. Para comunicação com o usuário, é apresentada uma interface web que pode ser acessada através da Internet por meio de um navegador. Com o término deste trabalho, obteve-se o desenvolvimento de uma ferramenta capaz de controlar e monitorar uma aplicação embarcada em um FPGA remotamente através da Internet com simplicidade e de forma transparente ao usuário.

1 Introdução

No atual momento, há uma grande demanda de alto desempenho e poder de processamento, por parte das indústrias de sistemas eletrônicos. Essa necessidade também é notória em sistemas embarcados e de tempo real. Para suprir estas necessidades, as empresas desenvolvedoras de sistemas embarcados estão apostando em arquiteturas de *hardware* paralelo e na integração de vários elementos de processamento [TIL11].

FPGAs são dispositivos que permitem o desenvolvimento destas arquiteturas [NOL08]. Devido a sua alta taxa de integração e baixo custo, esses dispositivos estão sendo cada vez mais utilizados em produtos de diversos segmentos. Outras características a favor dos FPGAs, é que apresentam um tempo menor de desenvolvimento de projetos e produção, se comparados a projetos ASICs [XIL11a]. Além disso, os FPGAs atuais apresentam alto nível de desempenho e longo ciclo de vida, característica antes presente só em dispositivos ASICs.

Um dispositivo FPGA é um circuito lógico programável, composto basicamente por 3 elementos (Figura 1): blocos de entrada e saída, que realizam a interface com o mundo externo; blocos lógicos configuráveis, dispostos em forma de matriz; bloco de interconexão. A funcionalidade destes blocos, bem como as interconexões desses são configurados via *software*, e podem facilmente ser reconfigurados, desempenhando assim novas atividades.

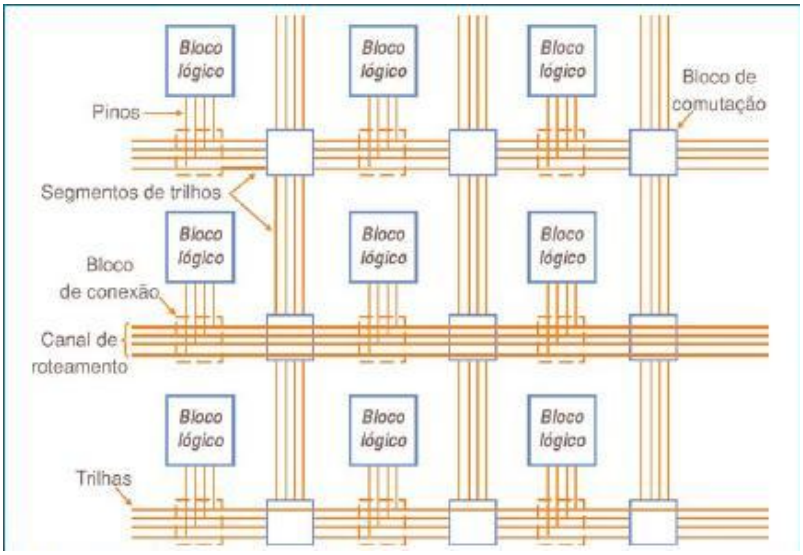

Figura 1 – Arquitetura interna de um dispositivo FPGA.

FPGAs apresentam duas características importantes: computação reconfigurável e miniaturização. O poder computacional que os FPGAs oferecem os transformam nos dispositivos mais adequados para aplicações que envolvem algoritmos onde uma rápida adaptação aos dados de entrada é requerida.

Em relação à miniaturização destes dispositivos, Xilinx e Altera - maiores fabricantes de dispositivos lógico programáveis - apresentam FPGAs com tecnologias inferiores a 65 nm. Também contam com uma grande quantidade de periféricos e componentes internos, tais como: interface para memória DDR, interface para barramento PCI, memória interna de grande capacidade, blocos DSP, além de um componente nativo: MAC Ethernet [ALT11] [XIL11b].

Este componente, MAC Ethernet, facilita o acesso do dispositivo FPGA à rede Ethernet. Ele tipicamente interage de um lado com o PHY Ethernet, que é externo ao FPGA, e de outro com a lógica desenvolvida no FPGA. Normalmente a ligação com o PHY é realizada através de uma interface MII e esta serve como interface de saída 10/100 Mbps ou, até, 1 Gbps. Já a conexão com a lógica do usuário requer o desenvolvimento de uma pilha de protocolos, a qual pode ser tanto desenvolvida em *hardware* quanto em *software*.

A presença de um MAC Ethernet nos dispositivos FPGAs simplifica o controle remoto destes. Esse controle pode propiciar atividades desde envio e recepção de tratamento das rotinas executadas pelo FPGA, até a reconfiguração do dispositivo através da rede Ethernet.

O presente trabalho tem por objetivo específico apresentar um software desenvolvido para realizar a comunicação de um FPGA através da rede Ethernet com um computador remoto, utilizando-se de troca de mensagens via protocolo UDP. Além disso, por meio de uma interface Web, possibilitar ao usuário a manipulação remota do sistema implementado no FPGA, bem como consultar resultados provenientes do mesmo.

O presente Trabalho de Conclusão de Curso (TCC) apresenta o desenvolvimento de uma ferramenta *Web* capaz de prover a comunicação entre um FPGA e um computador remoto, através da Internet. Por meio desta ferramenta, utilizando um *browser*, é possível realizar o envio de dados para o FPGA, e também receber os resultados providos do FPGA.

Apresentam-se abaixo os objetivos específicos deste TCC. Esses se relacionam diretamente com as disciplinas cursadas no curso de Engenharia de Computação, mostrando-se a estreita relação deste TCC com o mesmo.

- Domínio de uma interface de alto desempenho entre uma estação de trabalho e uma placa de prototipação (Programação de Periféricos);
- Domínio do desenvolvimento em linguagem C (Laboratório de Programação);
- Domínio de aplicações em *daemons* (Sistemas Operacionais);
- Domínio de protocolos de redes (Redes de computadores).

1.1 Motivação

Nota-se uma crescente difusão da Internet na vida quotidiana das pessoas. Essa já atende a maior parte da população, seja em seus domicílios, no local de trabalho ou estudo. Além disso, está presente em locais de comum acesso, tais como parques públicos e restaurantes.

Sua evolução é constante e busca acomodar todas as tecnologias de rede emergentes, que apresentam características e requisitos novos. Essas variam desde um simples acesso residencial até um controle industrial através da Internet. Novos modos de acesso e novas formas de serviço gerarão novas aplicações, expandindo ainda mais a Internet.

Devido à comunicação ser rápida e barata, a Internet tornou-se ubíqua. É possível encontrar diversas aplicações, que variam do popular e-mail, passam por vendas na Internet, jornalismo online, com atualizações constantes e em tempo real, chegando até modos de comunicação eficientes, tais como telefonia pela Internet e videoconferências.

Outra característica importante da Internet é a interatividade. A interatividade propicia o desenvolvimento de ferramentas para compartilhamento de arquivos, publicação de conteúdo e fotos, interação e controle de dispositivos remotamente. Um exemplo é a interação que a TV Digital com acesso a Internet proporciona ao usuário. Todas estas características deixam o meio mais atrativo e de fácil acesso e manuseio.

Com todos estes pontos a favor, o controle remoto de dispositivos a distância tende a ficar cada vez mais acessível. O controle de residências (domótica) [ISA09] e/ou o controle de dispositivos em ambientes industriais esta cada vez mais freqüente.

Em âmbito residencial, o emprego de soluções controladas remotamente, vem em busca de segurança, ou até mesmo comodidade. Para tanto, a área de domótica tem evoluído na busca por soluções que incluem, por exemplo, o monitoramento remoto de câmeras ou o controle do ar-condicionado através da Internet.

Na indústria pode-se destacar uma busca crescente pela automação. Ou seja, através de um processo automatizado, permitir que se obtenha uma melhor eficiência no processo produtivo, uma minimização de perdas e aumento da segurança. Juntamente com esta automatização, busca-se um abrangente controle sobre todo o processo produtivo, preferencialmente tudo através de um único mecanismo.

Um exemplo de automação industrial pode ser encontrada em [ALT11b]. A Figura 2**Erro! Fonte de referência não encontrada.** nos mostra a aplicação de um de um FPGA da família Cyclone, fabricado pela Altera. Essa aplicação serve como um gerenciador de uma arquitetura para controle de motores e conversores, além de apresentar uma ligação através do PHY Ethernet para a

rede interna da indústria. Trata-se de um dispositivo completo e indicado para estas aplicações, pois apresentam um processador embutido, codificadores e módulos para execução de algoritmos DSP, além de ligações para redes industriais em um único dispositivo. Também é observado em FPGAs o baixo consumo de potência, a flexibilidade para mudança dos algoritmos e suas características de segurança e redundância.

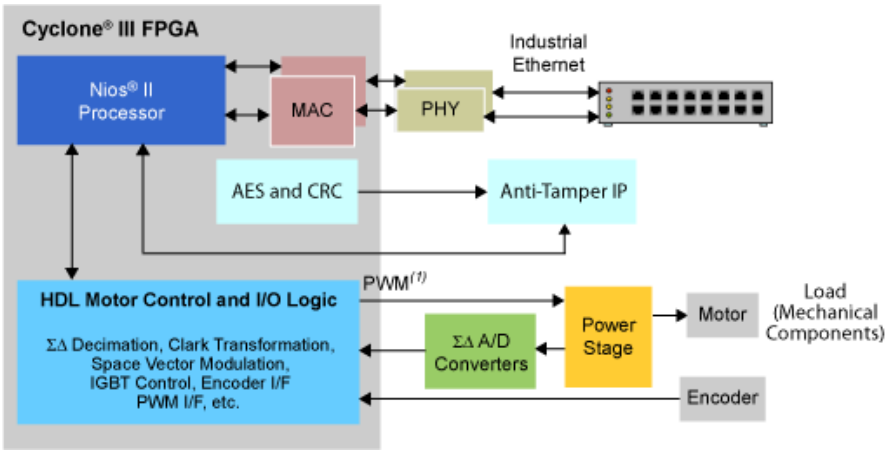

Figura 2 - Exemplo de utilização de FPGAs para controle industrial [ALT11b].

Este controle remoto de dispositivos do chão de fábrica auxilia diretamente a tomada de decisões. Espera-se ainda que essas decisões sejam tomadas o mais rapidamente possível, e dessa forma propiciar um maior nível de acerto nelas. Todo este controle pode ser feito através de redes Ethernet, de dentro da empresa ou mesmo através da Internet.

Baseados na popularização da Internet como um padrão universal de comunicação e também na grande popularização dos dispositivos móveis portadores de Internet sem fio, é possível realizar o controle de qualquer serviço disponibilizado na rede. Sendo assim, está cada vez mais fácil manter o contato com as redes de relacionamentos ou administrar uma empresa através de uma tela *touchscreen* de um celular, por exemplo.

Em diversas indústrias de componentes eletrônicos, a utilização de FPGAs em seus projetos é constante. É mais comum encontrá-los em indústrias de sistemas eletrônicos e embarcados, mas também estão substituindo e/ou trabalhando em conjunto com CLPs no chão de fábricas dos mais diversos segmentos, além de auxiliarem também na área de robótica.

Tendo em vista que estes dispositivos apresentam comumente um dispositivo PHY Ethernet, é possível utilizá-los para diversos fins através da rede. Pensando na comunicação remota com estes dispositivos, foi proposto e implementado um *software* em linguagem de programação C, no escopo deste TCC, juntamente com uma interface Web, com a finalidade de comunicação de um computador com um FPGA, independente de sua localização.

1.2 Objetivos

É esperado que com o desenvolvimento da ferramenta desenvolvida no escopo deste TCC o usuário possa controlar com facilidade, rapidez e segurança dispositivos eletrônicos - neste caso um FPGA - remotamente através de um navegador. Esse ambiente será hospedado em um servidor, podendo ser acessado através da Internet.

Busca-se também através de um único *daemon* gerenciador de todos os dispositivos, o controle com segurança destes, a fim de que um mesmo dispositivo seja acessado somente por um usuário simultaneamente.

Outro objetivo importante do presente trabalho é colocar em prática alguns dos conhecimentos adquiridos ao longo do curso, tais como redes de comunicação, arquiteturas de sistemas computacionais, linguagem de programação.

1.3 Contribuições

Este trabalho propõe o desenvolvimento de uma ferramenta capaz de propiciar o controle de dispositivos através da rede Ethernet. Para tanto foi utilizado como base uma pilha TCP/IP simplificada implementada em linguagem de descrição de *hardware* (VHDL) e o Protocolo UDP para troca de informações entre o FPGA e o *daemon.*

Como principais contribuições deste trabalho, podem-se citar a facilidade de comunicação entre um dispositivo computacional e o FPGA, o poder de transferência de dados por meio da Internet, caso o *software* tenha comunicação com a internet. Além do poder de gerenciamento de vários dispositivos simultaneamente, caso estes estejam previamente configurados.

O trabalho contribui em três frentes distintas:

- Desenvolvimento de um *hardware* simples no FPGA, o qual realiza a interface com o protocolo BCTCP, proposto por [SIL11], e realiza a comunicação do FPGA com a rede Ethernet.
- Um *daemon* é um programa que executa de forma independente, em background, ao invés de ser controlado diretamente por um usuário [TAN09] [COS10]. De modo geral, os *daemons* são inicializados durante o processo de *boot* do sistema operacional e não necessitam a intervenção do usuário. A aplicação desenvolvida procura suprir as necessidades de comunicação multidirecional, tendo em vista o gerenciamento de vários módulos de *hardware* através de um único *daemon.*
- Um *WebService* é uma aplicação que executa em um computador e tem a finalidade de possibilitar um serviço em páginas *Web*, ou seja, abertura de uma porta e o serviço de prover as

páginas quando estas forem requisitadas. As requisições para o endereço destino são feitas por meio de um navegador. Como contribuição neste tópico cita-se o desenvolvimento de páginas *web* que apresentam comunicação direta com o *daemon*, proporcionando troca de dados entre um computador e um dispositivo gerenciado por este *software*. por meio da Intranet ou também através da Internet.

1.4 Organização do Documento

O presente trabalho está organizado como segue. O Capítulo 2 apresenta uma visão geral dos itens necessários para o desenvolvimento da ferramenta, tais como os protocolos e modos de comunicação, tanto do *daemon* com o FPGA, quanto do *daemon* com as páginas PHP. É descrito também o Protocolo BCTCP, a prototipação de uma aplicação com suporte a este protocolo e aplicações *Web*. Já os Capítulos 3 e 4 apresentam a descrição do *software* desenvolvido, desde a interface com o FPGA, até a interface com o usuário (*Web*). É apresentado neste ponto, o protocolo utilizado para troca de mensagens entre os componentes da arquitetura, o mecanismo de conexão e desconexão, o envio e recebimento de dados, além do controle de erros. No Capítulo 5 são exemplificadas as arquiteturas utilizadas, os testes efetuados e os resultados obtidos através destes. Para finalizar, o Capítulo 6 apresenta a conclusão e o Capítulo 7 à bibliografia utilizada para pesquisa e desenvolvimento deste TCC.

2 Trabalhos Relacionados

A aplicação implementada no presente TCC propôs o desenvolvimento de 2 módulos principais:

(1) Um *daemon* para interagir com o hardware e com um sistema operacional;

(2) Um ambiente *Web* para realizar a comunicação entre o usuário e *daemon*.

A Figura 3 apresenta a arquitetura conceitual do sistema proposto.

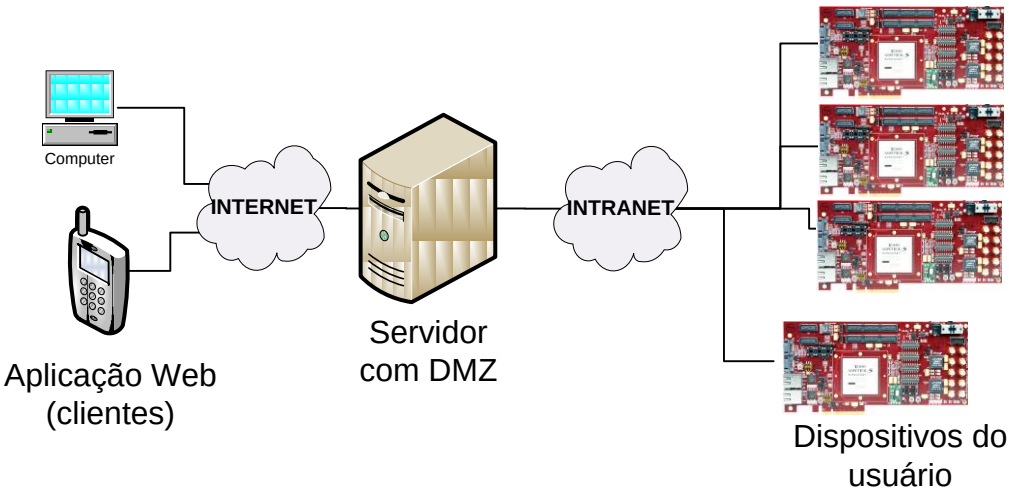

Figura 3 - Arquitetura conceitual do sistema proposto.

Nela observa-se uma arquitetura de redes de computadores. Essa arquitetura é típica quando um serviço é provido para a Internet. Os componentes dessa Figura incluem:

- *Aplicação Web*: é o aplicativo que serve como porta de entrada para a Internet. Está presente no sistema do cliente. Pode ser um *browser*, por exemplo.
- *Internet*: é a generalização de toda a rede externa usada para prover a comunicação entre clientes *web* e o servidor.
- *Servidor*: é a máquina que interconecta a Intranet, e o serviço provido pelo *daemon*, com a Internet.
- *Intranet*: é a generalização da rede interna. Não permite acessos à rede externa.
- *Dispositivos do usuário*: é a generalização de qual serviço estará sendo servido. No escopo desse projeto, é o *hardware* do usuário.

Observa-se nessa Figura a presença de um servidor. Esse servidor provê o acesso da Internet à Intranet. Para isso, o servidor deixará certos serviços e portas acessíveis por hospedeiros externos. Essa não é uma prática segura, uma vez que um cliente mal intencionado pode interferir no

funcionamento normal do sistema. Todavia, para o sistema permitir o acesso de terceiros ao servidor, é utilizado o conceito de redes DMZ (***De****M****i****litarized* ***Z****one* ou zona desmilitarizada) [TAN03]. Essas redes permitem que portas e serviços específicos da rede local sejam abertos para acesso externo. Ou seja, um servidor operando com redes DMZ permite que apenas uma parte da Intranet possa ser acessada pela Internet. Consequentemente, essa característica agrega segurança para a máquina que está habilitando o serviço para a rede.

Outra característica de segurança importante que o servidor agrega a essa rede é o isolamento dos módulos que estão sendo manipulados. Apenas o servidor tem acesso aos periféricos controlados. Assim, o servidor trata os dados recebidos e os envia para os dispositivos de *hardware*. Em virtude disso, dados incorretos não alcançam os módulos do usuário, e, portanto não provocam o funcionamento incorreto destes.

2.1 Protocolo BCTCP

Baseado em [REI09], [SIL11] desenvolveu em *hardware* um protocolo confiável que permite a comunicação de um FPGA com a Internet. Tal protocolo foi desenvolvido sobre o protocolo UDP, porém apresenta o mecanismo de controle de fluxo e erros, diferentemente do protocolo UDP tradicional. A arquitetura desenvolvida por [SIL11] a fim de implementar simplificadamente a pilha TCP/IP em hardware, denomina-se BCTCP (Baixo Consumo TCP), e tem por finalidade a comunicação de uma plataforma FPGA com a rede Ethernet.

O protocolo leva este nome devido ao baixo consumo em relação ao protocolo TCP que fora anteriormente implementado. Trata-se de uma simplificação do TCP, que executa junto ao protocolo UDP e tem como ponto positivo a verificação dos pacotes, a fim de evitar erros.

O protocolo UDP tradicional é um protocolo não orientado a conexão, e assim, pouco confiável. Para comunicação utiliza somente o endereço IP do *host* remoto e a porta destino. É tido como pouco confiável devido a não ter nenhum controle, tal como número de sequência para entrega ordenada dos datagramas ao *host* destino. Sendo assim, os dados podem chegar ao destino em ordem trocada. O único controle para verificação dos dados é o *checksum* baseado em complemento de um, de 16 bits, que também pode ser desabilitado.

Embora pouco confiável, o protocolo UDP apresenta alta eficiência, já que o pacote é composto praticamente por dados. É indicado para aplicações em tempo real, especialmente para aqueles que admitem a perda de algum conteúdo, tais como vídeo ou voz, ou até jogos.

Esta seção é dividida da seguinte forma: o item 2.1.1 apresentará a arquitetura do protocolo BCTCP, desenvolvida em VHDL, já o item 2.1.2 mostra o mecanismo de IP *Discovery*, e por fim, é

descrito o mecanismo de troca de mensagens entre um hospedeiro e um FPGA, utilizando como base o BCTCP.

2.1.1 Protocolo IP Discovery

Para que um usuário possa realizar a comunicação entre um hospedeiro e um FPGA, é necessário que o hospedeiro obtenha informações de rede do FPGA. Para isso, foi implementado uma rotina denominada IP *Discovery*. Tem como ponto positivo que a máquina hospedeira necessita somente de uma única comunicação através da rede para realização desta descoberta.

Como ponto inicial, a comunicação por parte do hospedeiro, é um envio de um pacote em *broadcast* com parâmetros que sinalizam a busca do IP de um determinado FPGA (*Project Identifier*, *FPGA Identifier* e *User Identifier*). O FPGA correspondente recebe o pacote e responde a requisição com seus parâmetros de rede (*MAC* e IP) e também valores correspondentes a aplicação e usuário, além de uma *flag* indicando se já possui alguma conexão estabelecida ou não. A Figura 4 mostra esta comunicação.

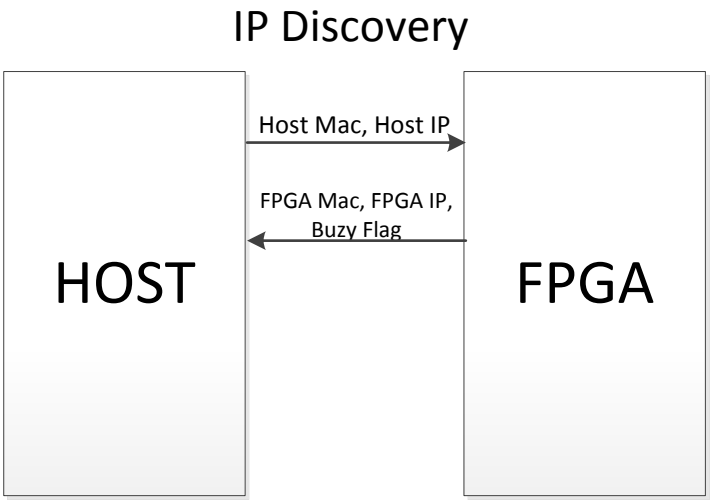

Figura 4 - Mecanismo de IP Discovery.

A comunicação entre os dois elementos se dá através da porta 9002 via protocolo UDP e nela, são trocados os seguintes dados, mantidos sempre esta ordem:

- *Source MAC Address*: 6 bytes - endereço físico (MAC) da máquina de origem. Tem a finalidade de simplificar o processo, eliminando a etapa ARP (descoberta de MAC).
- *Source IP Address*: 4 bytes - endereço lógico (IP) da máquina de origem. Tem a finalidade de eliminar o armazenamento deste dado no FPGA.
- *Project Identifier*: 4 bytes - identificador do projeto. String codificada em ASCII para

decodificação do pacote recebido.

- *FPGA Identifier*: 4 bytes - identificador de FPGA. String codificada em ASCII para decodificação do pacote recebido.
- *User Identifier*: 4 bytes - identificador de usuário. String codificada em ASCII para decodificação do pacote recebido.
- *Busy Flag*: 2 bytes - identificador de FPGA ocupado. O objetivo deste campo é informar o host que o FPGA está apto ou não a estabelecer conexão.

Através destes 24 bytes de dados, é possível garantir que somente um FPGA responda, a menos que 2 módulos possuem os mesmos campos de identificação de projeto, FPGA e usuário. No pacote de retorno, em caso da placa estar ocupada a flag *busy* conterá o valor 0xFF, caso contrário este campo conterá 0x00.

2.1.2 Troca de Mensagens utilizando o Protocolo BCTCP

Com o intuito de trabalhar com um protocolo mais robusto que o UDP, sem partir para a utilização do protocolo TCP para a verificação dos pacotes transmitidos, [SIL11] utilizou o *checksum* do pacote UDP e também o número de sequência do pacote. Em uma comunicação host ↔ FPGA, caso ocorra erro de *checksum* ou perda de pacotes, o FPGA envia uma mensagem de controle com o número de sequência esperado, e com o campo de dados nulo solicitando assim a retransmissão do último pacote.

As informações recebidas pelo FPGA são analisadas e tratadas de acordo com a operação desejada. Operações estas, definidas por *flags* de 2 bytes, mostradas na Figura 5. Além das operações permitidas, e do campo de dados, o cabeçalho BCTCP apresenta um campo de número de sequência a fim de realizar a consistência dos pacotes e solicitar retransmissão caso necessário.

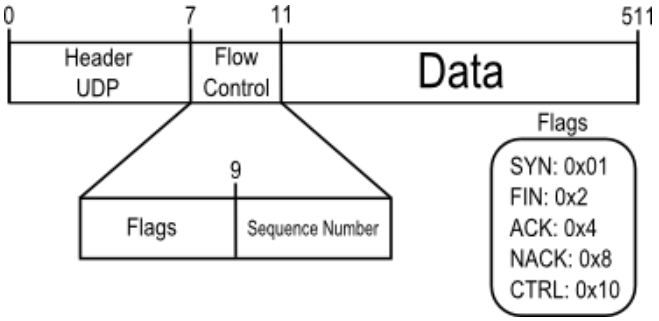

Figura 5 - Header BCTCP e *flags* de controle [SIL11].

O FPGA através do protocolo DHCP obtém automaticamente um endereço IP válido. A partir deste momento é possível estabelecer a comunicação entre um computador e o FPGA. A

primeira etapa é a etapa de sincronização (Figura 6.1) e estabelecimento de conexão. Para isso, o hospedeiro envia um pacote contendo uma requisição de conexão (SYN) e um número de sequência inicial. Caso não esteja conectado a outra máquina, o FPGA envia a resposta (SYN + ACK) e o mesmo número de sequência, sinalizando o sucesso na etapa conexão. Em caso de já estar conectado com outra aplicação o FPGA irá negar a conexão, utilizando para tanto a *flag* NACK.

Para a troca de informações entre o hospedeiro e o FPGA (Figura 6.2), a máquina envia uma sequência de pacotes de dados definidos a nível de aplicação (ACK + data), e incrementa o número de sequência a cada envio. Esta tarefa também será executada pelo FPGA controlando assim o sequenciamento de dados. Em caso de erro neste número, erro de *checksum*, ou mesmo perda de pacotes, o FPGA envia uma mensagem de controle (CTRL + NACK) com o campo de dados nulo e o número de seqüência esperado, solicitando assim o reenvio de um ou mais pacotes.

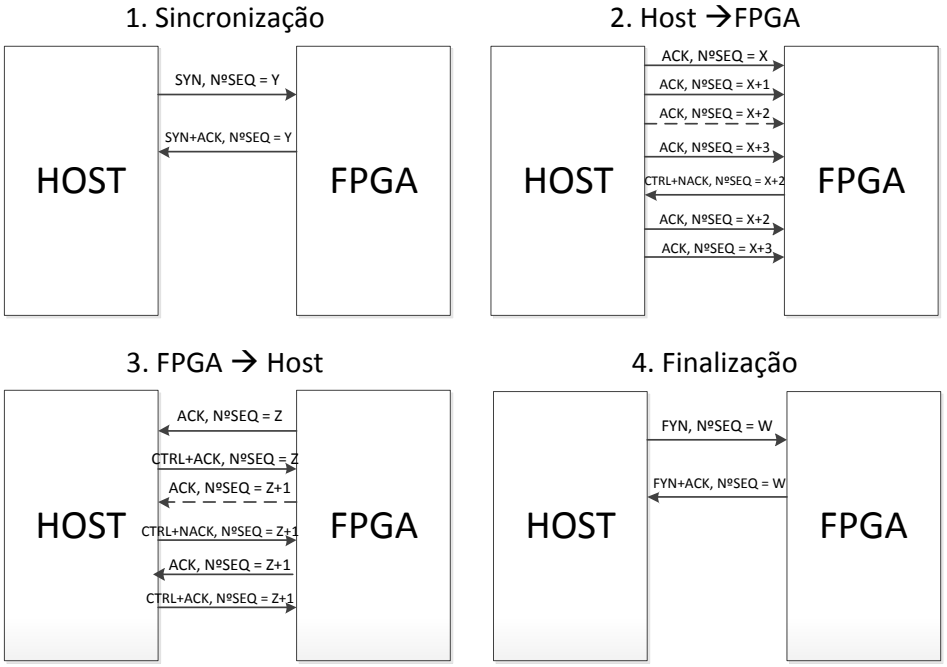

Figura 6 - Modelos de Comunicação Host ↔ Fpga.

No fluxo de controle FPGA para o computador (Figura 6.3), o processo de comunicação é diferente. A cada pacote enviado por parte do FPGA, é esperado do hospedeiro um pacote de controle (CTRL + ACK) com o mesmo número de sequência enviado, a fim de garantir que o pacote foi recebido corretamente pelo hospedeiro. Em caso de erro, o mecanismo de controle (CTRL + NACK) também funciona no sentido inverso, e os pacotes podem ser retransmitidos pelo FPGA.

Após o término da aplicação e de todas as trocas de dados e informações entre o hospedeiro e o FPGA com sucesso, é necessário a utilização de um mecanismo de finalização de conexão

(Figura 6.4), a fim de liberar o FPGA para execução de outra tarefa. Este mecanismo também é baseado em troca de mensagens, o hospedeiro envia um pacote de finalização de seção (FIN) e o FPGA responde (FIN + ACK), também acompanhados de um número de sequência, normalmente o posterior aos de dados.

2.2 Prototipação em FPGA com suporte ao protocolo BCTCP

Como base para o desenvolvimento da aplicação no nível de *software* para comunicação remota com um dispositivo, fora utilizado o Protocolo BCTCP prototipado em uma placa fabricada pela Hitech Global e que contém um FPGA Virtex5lxt330, da fabricante Xilinx (XILINX, 2010), que conta com 51.840 *slices* e 324 blocos de memória de 36 *kbits*. A placa ainda possui dois PHY Ethernet, memória DDR2 e barramentos SATA e PCI-e.

2.2.1 Descrição da Arquitetura

O *hardware* descrito em VHDL responde conforme o protocolo descrito no item 2.1 e trata-se de uma aplicação na pilha TCP/IP. O corpo da pilha TCP/IP é apresentado na Figura 7, onde encontramos a camada de transporte TX e RX, encapsulado no protocolo UDP. Abaixo dela, a camada IP, que descreve a camada de Internet. E como base da pilha está a camada de enlace, que corresponde a também camada de enlace da pilha TCP/IP.

O MAC RX/TX é um circuito específico do FPGA (*hard-core*), que pode ser configurado através do software *CoreGen* [XIL10], contido no pacote ISE da fabricante Xilinx.

Na camada de aplicação do *hardware* são encontrados os seguintes blocos:

- *DUT* (*design under test*): implementa a função desejada pelo projetista no FPGA. Neste projeto, terá a finalidade a validação do *software* implementado para comunicação entre o dispositivo e o hospedeiro;
- *DHCP*: implementa a máquina de estados responsável pela obtenção de endereço IP válido da rede em que o FPGA está conectado;
- *IP Discovery*: tem por finalidade de prover os parâmetros de rede do FPGA ao *host* que realizará a comunicação com o dispositivo;
- *Árbitro*: localizado entre a camada de aplicação e a camada de transporte, e tem por objetivo realizar o controle de comunicação entre a aplicação e a mundo externo. As prioridades associadas ao árbitro compreendem: (1) DHCP; (2) DUT; (3) IP Discovery.

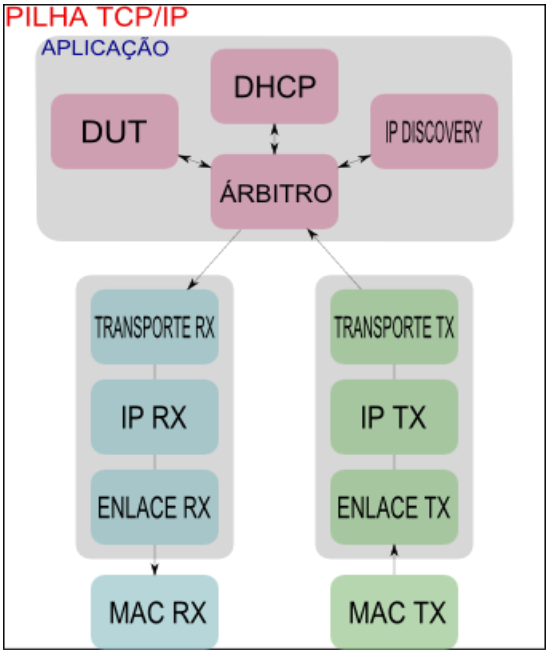

Figura 7 - Arquitetura Hardware base, prototipado em FPGA.

Para o desenvolvimento desta arquitetura [SIL11] utilizou uma arquitetura UDP já existente [REI09], onde foi incluído o mecanismo de controle de fluxo e erros na máquina de estados de recepção. A Figura 8 apresenta um diagrama mais detalhado da implementação desta arquitetura.

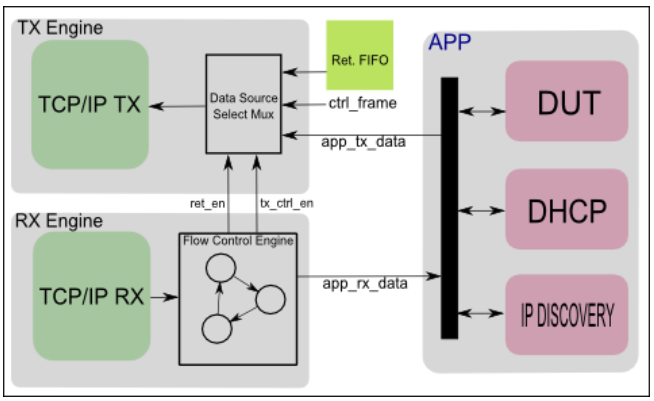

Figura 8 - Arquitetura Detalhada do *Hardware* Sintetizado

Para realizar o controle de fluxo a nível de *hardware*, [SIL11] utilizou *flags* e número de sequência de pacotes contidos no *header* do protocolo BCTCP, e dessa forma, de acordo com o valor contido nesses campos, o mecanismo aciona o bloco de transmissão, ou da mensagem de resposta ou a solicitação de retransmissão de pacote.

A análise inicial é realizada através da leitura das *flags* de controle apresentadas na Figura 5. Elas indicam a operação desejada, as quais podem ser: sincronismo, finalização de seção, envio de dados e solicitação de retransmissão. Após esta análise, passa-se para a comparação do número de sequência recebido. Este é comparado com o número de sequência esperado pelo FPGA.

Além destes controles, também é verificado o *checksum* do pacote e em caso de erro são gerados sinais de controle (*ret_en/tx_ctrl_en*) repassados a máquina *TX*. No bloco de transmissão esses sinais controlam um multiplexador, que seleciona a origem dos dados, que são provenientes das aplicações, da FIFO de retransmissão e dos pacotes de controle de seção.

A FIFO de retransmissão é carregada quando a aplicação decide enviar um pacote para o host, sendo que seus ponteiros de escrita e leitura são reinicializados quando a FIFO não está em operação, de modo à sempre haver o pacote válido para a enésima solicitação de retransmissão. Já os pacotes de controle são os pacotes de sincronismo, solicitação de retransmissão e finalização de seção, que não possuem dados de processamento válidos, mas são de fundamental importância para o correto funcionamento do protocolo BCTCP.

2.2.2 Interface com a Aplicação

O *hardware* de aplicação, neste trabalho referenciado por DUT, está localizado na camada de aplicação da pilha TCP/IP, juntamente com o mecanismo de DHCP, Protocolo IP *Discovery* e também o árbitro. Este DUT tem a função de emular o circuito do usuário e realizar a interface com o protocolo BCTCP.

Conforme pode ser observado na Figura 8, a comunicação em *hardware* entre a aplicação (DUT) e a máquina de estados TX se dá através do sinal *app_tx_data* e a máquina de estados RX utiliza para saída de dados o sinal *app_rx_data*.

2.3 Aplicação *Web*

O nome *Web* vem do termo mais amplo: *World Wide Web* (WWW) que representa um sistema de informações mundialmente distribuídas. Elas estão interligadas através de links, e é possível acessá-las através deles. Devido à grande demanda de serviços via internet, é possível encontrar desenvolvimentos de aplicações que executam alguma atividade no cliente ou no servidor, além de simplesmente fornecer a informação.

Os serviços disponibilizados na *Web* são praticamente ilimitados. Apresentam conexão com banco de dados, execução de programas em *background*, acesso remoto a estações de trabalho, além de comunicação entre os dispositivos por meio da rede. É comum também ouvirmos falar em *cloud computing*, ou seja, computação em nuvem - sistemas rodando em paralelo na Internet e sendo disponibilizados aos usuários.

Os responsáveis pela disponibilização dos serviços através da Internet denominam-se servidores. Levam este nome, pois são computadores que apresentam um arquitetura robusta, com grande capacidade de processamento e armazenamento, capazes de ficarem dias sem desligar. Existem servidores a desempenhar funções tão diversas quanto as aplicações, como permitir ligações, disponibilizar arquivos através de FTP, gerenciar o envio e recepção de correio eletrônico, etc.

Um *web server* por sua vez, trata-se de uma aplicação que executa em um computador servidor e tem a finalidade de servir páginas *web*, ou seja, executa a abertura de uma porta para prover os serviços, quando estes forem requisitadas. Normalmente a aplicação provida por estas ferramentas são as páginas. As requisições para o endereço destino é feita por meio de um navegador.

O processo se inicia com a conexão entre o computador onde está instalado o aplicativo servidor e o computador do cliente. A partir daí é processado o pedido do cliente, e conforme as restrições de segurança e a existência da informação solicitada, o servidor devolve os dados.

Uma vez que os dados estejam presentes na Internet, ou Intranet, eles não apresentam um fluxo regular de tráfego, assim, os sistemas *web* precisam lidar com entradas de dados em intervalos irregulares. Ou seja, os *websites* devem ser robustos o suficiente para atender uma demanda de entradas variáveis ao longo do tempo. Isso agrega uma dificuldade para esse tipo de *software*.

O acesso a páginas é provido por meio do protocolo HTTP. A Figura 9 mostra uma visão geral de um sistema de acesso a páginas *web*. Todas as requisições, caso o *host* destino não esteja na mesma rede do *host* origem, serão trocadas através da Internet. Caso contrário, onde a comunicação é interna, denomina-se Intranet.

Normalmente a porta configurada para a conexão no servidor remoto é a porta 80 para HTTP e 443 para HTTPS. Muitos *webservers* apresentam a possibilidade de troca dessas portas para melhor atender as necessidades dos usuários. HTTP (Protocolo de Transferência de Hipertexto) é um protocolo da camada de Aplicação do modelo OSI utilizado para transferência de dados na rede mundial de computadores. HTTPS difere somente na questão de segurança adicional, normalmente por meio de certificados digitais.

A Figura 9 apresenta também os diversos protocolos que trabalham em conjunto para o fornecimento de serviços. Para que o protocolo HTTP consiga transferir seus dados pela *Web*, é necessário que os protocolos TCP e IP tornem possível a conexão entre clientes e servidores através de *sockets* TCP/IP. Em ambos os sentidos, o conteúdo é encapsulado e transferido das camadas mais baixas até as mais altas, chegando assim até a aplicação e atendendo ao pedido e/ou executando alguma operação.

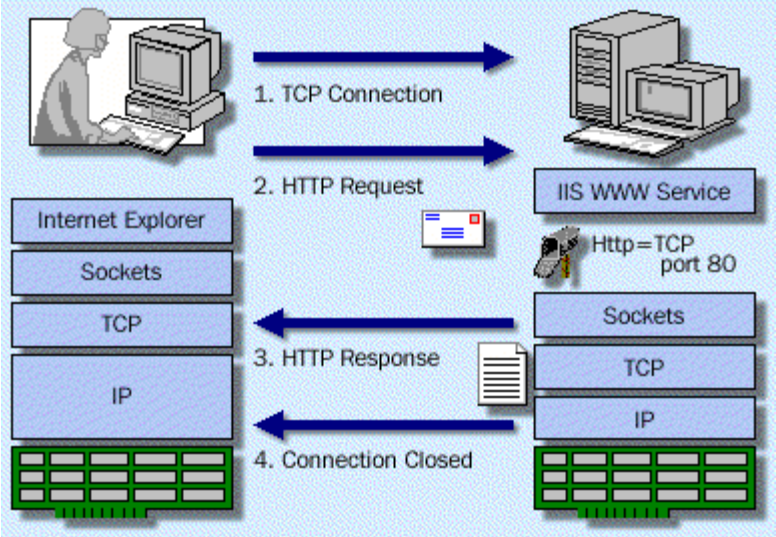

Figura 9 - Modelo de Comunicação HTTP [ICE11].

O Protocolo HTTP [FIE99] utiliza o modelo cliente-servidor como a maioria dos protocolos de rede, baseando-se no paradigma de requisição e resposta. Um programa requisitante (cliente) estabelece uma conexão com outro programa receptor (servidor) e envia-lhe uma requisição, contendo a URL solicitada, a versão do protocolo, uma mensagem MIME (padrão utilizado para codificar dados em formato de textos ASCII para serem transmitidos pela Internet) contendo os modificadores da requisição, informações sobre o cliente e, possivelmente, o conteúdo no corpo da mensagem.

Juntamente com as mensagens de resposta, o servidor encaminha uma linha de status, contendo sua versão de protocolo e um código de operação bem sucedida ou um código de erro. Ao término do envio da resposta pelo servidor encerra-se a conexão estabelecida.

Como ponto negativo, pode-se relatar que o protocolo HTTP é um protocolo *stateless*, ou seja, as conexões são encerradas ao final de cada requisição e resposta. Isso acaba acarretando maior consumo de largura de banda, CPU e memória, pois muitas vezes, são realizadas diversas requisições a fim de carregar toda a página, composta por imagens, texto, arquivos de layout. O ideal seria uma única conexão a fim de realizar o carregamento de todos os componentes da página, evitando assim um *overhead* na rede.

Para que o conteúdo seja visualizado corretamente pelo usuário, com uma boa aparência, é necessário realizar a manipulação dos dados e programação das páginas. A linguagem mais popular para o desenvolvimento de páginas para Internet é o HTML [SIL11b]. Trata-se de uma linguagem descritiva, de formatação, portanto nem sempre *browser* diferentes a interpretam da mesma

meneira. Sua descrição é baseada em *tags*, ou seja, marcas especiais que determiam o papel de cada elemento dentro do texto.

Outra linguagem de desenvolvimento *web* bastante difundida é o PHP [NIE04] [BRI11]. Essa é uma linguagem especifica para o desenvolvimento para esse tipo de sistema. Tem como principal objetivo programar sistemas simples e eficientes. Ainda, é uma linguagem estruturada e orientada a objetos e é portável, independe de plataforma.

Com a popularização da Internet, a tecnologia utilizada para desenvolvimento de páginas evoluiu gradativamente. Ao invés de conteúdos estáticos que apresentavam sempre a mesma informação, passou-se a utilizar formulário HTML, os quais são preenchidos pelo usuário e enviados novamente ao servidor. O programa remoto (*Server-Side Gateway Program*) trata as informações e retorna uma página HTML criada dinamicamente. Esta página é passada ao servidor que a entrega ao cliente. O padrão para comunicação entre o servidor *Web* e o "*Server-Side Gateway Program*" é conhecido como CGI (*Common Gateway Interface*) [RNP97].

Os programas escritos para ser executados por servidores web são também conhecidos como scripts CGI. Os scripts CGI implementam a lógica e, muitas vezes, o acesso aos dados de uma aplicação web, pois podem acessar dados armazenados no servidor ou fazer chamadas a um servidor de banco de dados local ou remoto, permitindo o acesso às informações de uma empresa por exemplo.

Estes scripts podem ser escritos em qualquer linguagem de programação, desde que o código gerado possa ser executado na maquina do servidor. Comumente são utilizados as linguagens PERL ou PHP. Estas linguagens, por terem suas rotinas executadas no servidor, são denominadas linguagens *server-side*.

Os programas que executam no *browser*, normalmente, são *scripts* (VBScript ou JavaScript) que tem a capacidade de perceber os eventos causados pelo usuário no navegador e responder de forma apropriada. Eles são embutidos no código HTML e o seu código fonte pode ser visualizado pelo usuário, pois não é um código compilado. Os scripts interagem muito com todos os elementos que formam uma pagina HTML. Por executarem no navegador do cliente, estes *scripts* são denominados *client-side*.

2.4 Daemon

Pode-se tomar como exemplo 2 aplicações: um editor de textos e um servidor *web*. O primeiro irá responder se e somente se o usuário ordenar, através de comandos via *mouse* ou teclado. Já o segundo, normalmente responde todas as requisições que chegarem até ele de forma

automática, sem a intervenção humana, sendo somente necessária a inicialização do programa para que este opere sozinho. Alguns programas são projetados de forma a utilizar informações contínuas do usuário. Ao contrário de outros que realizam suas tarefas de forma independente. Programas que transportam mensagens de correio de um local para outro, são outro exemplo dessa classe de aplicações.

Em sistemas operacionais, um *daemon*, é um programa que executa de forma independente em *background*, ao invés de ser controlado diretamente por um usuário [TAN09] [COS10]. Pode ser também descrito com um programa em execução em um computador servidor que está sempre pronto para receber solicitações de outros programas, executar determinada ação e retornar a resposta adequada.

De modo geral, os *daemons* são inicializados durante o processo de *boot* do sistema operacional e não necessitam a intervenção do usuário. Geralmente estas aplicações monitoram e interagem com requisições de rede e atividade de *hardware*, ou mesmo com outras atividades específicas. Também podem ser responsáveis pela execução de tarefas em horários pré-determinados.

Tendo em vista que os *daemons* normalmente executam funções importantes para o sistema operacional, a comunicação a estes programas deve ser feito somente por um administrador.

Nos sistemas operacionais UNIX, quando necessária a comunicação com um processo *daemon*, ela se dá através de sinais [FRE11]. Alguns deles podem ser tratados internamente pela aplicação, onde o processo toma uma iniciativa a partir do sinal recebido. Outros sinais são tratados diretamente pelo sistema operacional e obrigam o processo a mudar o fluxo de execução ou até encerrar, de acordo com o sinal. Como sinais mais conhecidos para comunicação entre processos, podem ser citados SIGKILL, que termina o processo; SIGALARM, normalmente implementa uma função interna no *daemon*; SIGSTOP pára momentaneamente um processo; SIGCONT, responsável pela continuação da execução de um processo parado. Alguns destes sinais de comunicação são executados implicitamente quando enviamos um comando *start, stop, killall* ou *restart* ao *daemon*.

3 Descrição da arquitetura desenvolvida

Foi desenvolvido um *daemon* para permitir a troca de informações entre o *WebService* e o FPGA, utilizando como base o protocolo BCTCP. Paralelamente, foi desenvolvida a página w*eb* da ferramenta. Por fim, para ser utilizado como suporte ao sistema implementado, foi desenvolvido um *software* simulador que responde a aplicação de usuário e os protocolos IP *Discovery* e BCTCP. Em relação à arquitetura conceitual, descrita na Figura 10, no escopo desse projeto a aplicação *web* pode ter acesso, além do FPGA, ao computador executando um simulador do protocolo BCTCP. A Figura 10 ilustra a arquitetura desenvolvida.

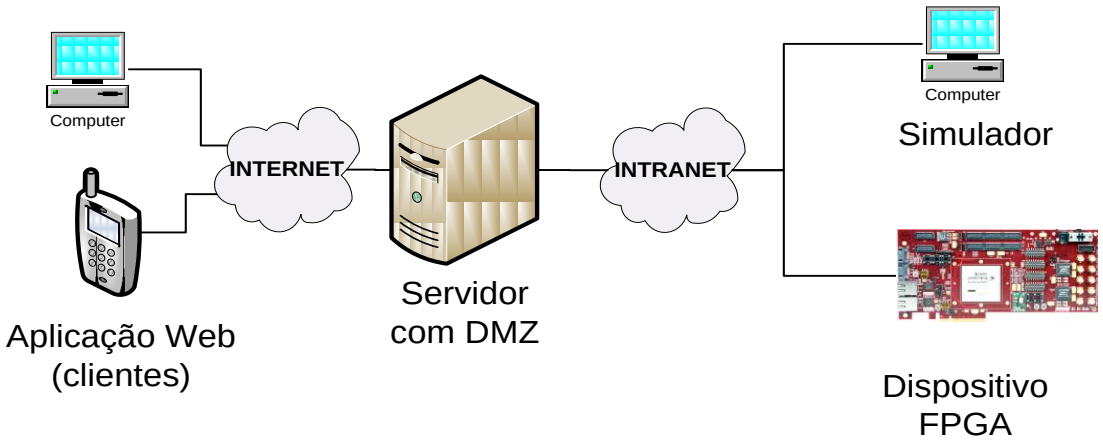

Figura 10 - Arquitetura desenvolvida.

Na arquitetura, ilustrada na Figura 10, o servidor permite a conexão com um ou mais clientes. Contudo, caso haja a conexão de mais de um cliente, o servidor gerencia qual cliente tem acesso ao dispositivo.

Uma vez que o cliente obteve conexão com o FPGA, ele está apto a enviar dados ao mesmo. Os dados são enviados no formato de um arquivo de texto. Esse arquivo é carregado pelo cliente no servidor através da interface *web* do *WebService*. O tamanho do arquivo, o número de dados por linha, e a semântica do conteúdo enviado dependem do *hardware* de aplicação em execução no dispositivo FPGA.

Após o cliente carregar o arquivo no servidor, o *daemon* envia os dados contidos no arquivo para o FPGA. Ele lê cada linha deste arquivo e a envia para o FPGA. O dispositivo, através do *hardware* de aplicação, processa os dados. Após o processamento destes dados, o FPGA retorna,

caso exista, a saída para o *daemon*. O *daemon* armazena essa saída em um arquivo texto no servidor local.

Após esse processamento, a saída resultante da manipulação do FPGA estará disponível no servidor. Deste modo, o cliente pode obter a saída através do *download* de um arquivo de através da página acessada.

O restante deste Capítulo está organizado como segue. A Seção 3.1 descreve o *hardware* de aplicação que desenvolvido. A Seção 3.2 apresenta o *daemon* implementado. Por fim, a Seção 3.3 descreve o *Web Service*.

3.1 Descrição do Hardware Sintetizado no Dispositivo FPGA

Com o objetivo de validar as trocas de mensagens entre o *daemon* e o FPGA, utilizando o protocolo BCTCP, [SIL11] desenvolveu-se um *hardware* de aplicação. Esse tem o comportamento de uma aplicação sobre o FPGA. Até o presente momento, a prototipação somente foi realizada na placa Virtex5lxt330, da fabricante XILINX, Figura 11.

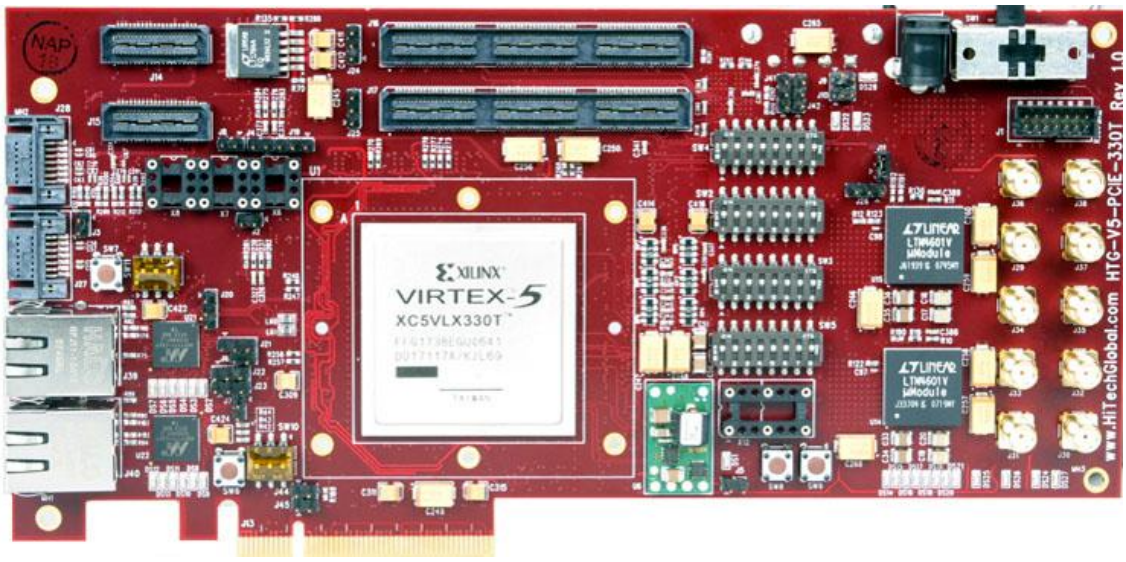

Figura 11 - Virtex5lxt330.

Cabe ainda salientar que o *hardware* de aplicação desenvolvido é simples e não tem um tratamento específico para os pacotes UDP recebidos. Esta aplicação recebe 3 pacotes provenientes da camada de transporte. A cada pacote recebido, é realizada a soma de todos os dados contidos em cada pacote. Após recebê-los, e efetuar o cálculo, o *hardware* de aplicação envia para a camada de transporte um pacote contendo a soma de todos os dados dos pacotes recebidos.

Esse *hardware* de aplicação (Figura 12) é composto por três máquinas de estados. Ele tem comunicação direta com o *hardware* do Módulo Ethernet. As máquinas de estados têm as suas

transições baseadas nos sinais emitidos pelo módulo. Uma máquina é utilizada para receber os pacotes do módulo, nesse caso a máquina de recebimento. Outra tem o objetivo de realizar o cálculo dos dados, é a máquina de cálculo. A terceira é a máquina responsável pelo envio dos dados calculados para o módulo, nesse caso, a máquina de envio.

Quando um dado, no Módulo Ethernet, está pronto para ser processado pelo *hardware* de aplicação, o módulo sinaliza o *hardware* através do sinal **en_in**. Ao receber esse sinal, a máquina de recebimento do *hardware* de aplicação lê o tamanho do dos dados a serem recebidos, através do sinal **frame_size_in**. Após descobrir o tamanho dos dados a serem recebidos, a máquina realiza a leitura dos dados e os armazena na FIFO. No momento que os dados estiverem prontos na FIFO, o sinal **process_execute** recebe o valor '1'.

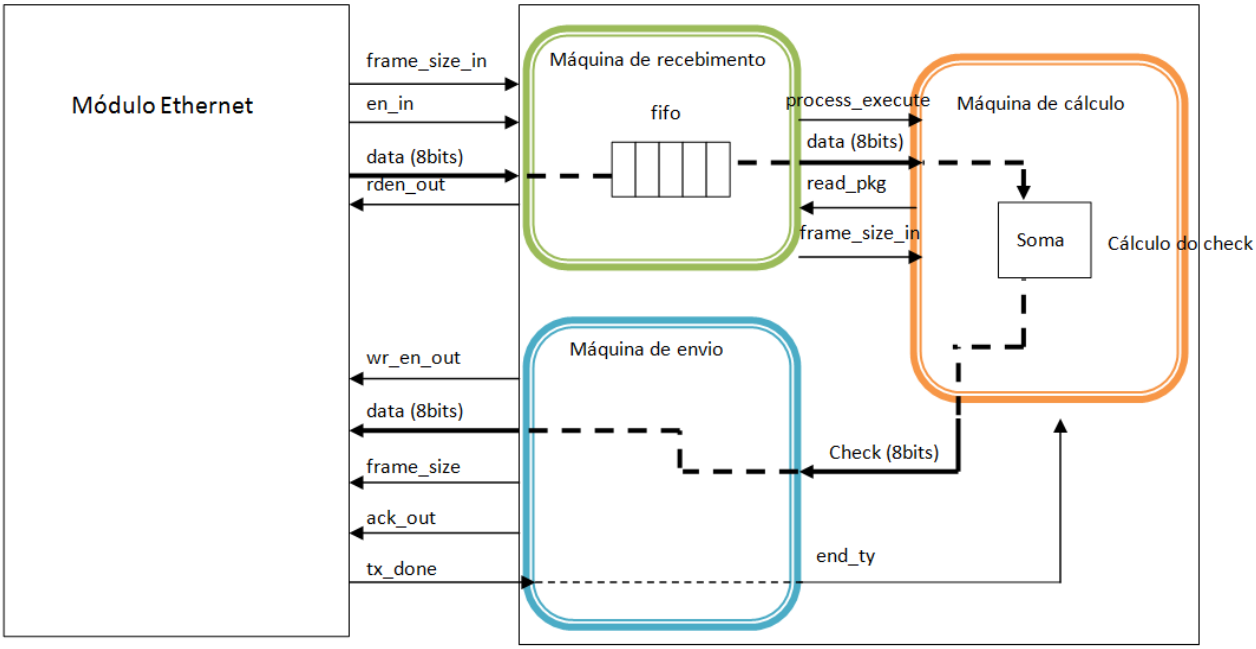

Figura 12 – Hardware de Aplicação.

A máquina de cálculo aguarda até que o sinal **process_execute** esteja com valor '1'. Após, a máquina de cálculo bloqueia a FIFO, através do sinal **read_pkg**, para escrita e começa a retirar os dados da fila. Os dados que são retirados da fila são somados, byte a byte, e serão armazenados na variável **Check**. Após todos os dados serem somados, o sinal **send_frm** receberá o valor '1' e a FIFO é liberada.

A máquina de envio fica em espera enquanto o sinal **send_frm** não for '1'. Após o inicio da máquina, os dados presentes na variável **Check** são enviados para o Módulo Ethernet através do canal **data**. O tamanho dos dados é informado ao módulo pelo sinal **frame_size**. O sinal **ack_out** e

o sinal **tx_done** servem para controle de recebimentos de dados e envio de dados entre o módulo e o *hardware* de aplicação. O sinal **wr _en_out** serve para sinalizar o módulo que existe um dado pronto para ser lido.

O módulo Ethernet apresentado na Figura 12 é a abstração do protocolo BCTCP que [SIL11] desenvolveu. Este se comunica conforme as requisições providas da aplicação e de estímulos externos, ou seja, pacotes destinados a este FPGA, e interpretados e respondidos através do BCTCP.

3.2 Simulador do *Hardware*

O simulador foi desenvolvido com o objetivo de permitir que o sistema fosse executado, em tempo de projeto, sem a necessidade de um FPGA. No caso, ele é um *software* de apoio para o desenvolvimento *stand-alone*. Contudo, hoje o simulador também está integrado à arquitetura do projeto e está servindo como mais um dispositivo integrante do sistema.

Neste simulador, os protocolo BCTCP e *IP Discovery* foram implementados na sua totalidade. Ele ainda simula o mesmo *hardware* de aplicação descrito na seção anterior. Sendo assim, comporta-se de forma semelhante ao hardware prototipado em FPGA. Esse *software* executa em sistemas operacionais Linux.

3.3 Daemon Desenvolvido

Com o objetivo de prover a conexão entra o FPGA e o *Web Service*, foi desenvolvido um *daemon*, em linguagem C. Essa aplicação é executada em sistema operacional Linux.

Para trocar mensagens com o FPGA, o *daemon* utiliza mensagens do protocolo de comunicação UDP. Para isso, envia pacotes por *socket* UDP codificado com o protocolo BCTCP ou com o protocolo IP Discovery. Este *socket* opera nas portas 9001 e 9002. A porta 9002 é utilizada somente para troca de pacotes do protocolo *IP Discovery*. A porta 9001 é utilizada pelo protocolo BCTCP, para a troca de dados.

Para as trocas de mensagem entre o *daemon* e o *Web Service*, os dados são passados através de *sockets* entre processos locais, do tipo AF_UNIX. Os dois sistemas são executados em paralelo, cada um esperando os dados providos pelo outro. Eles têm por objetivo manipular os dados entre o FGPA e o usuário.

O *daemon* controla todos os módulos de *hardware* que estão executando o protocolo BCTCP na Intranet. Para saber quais os dispositivos podem ser controlados, ele utiliza o

mecanismo de descoberta de IP do protocolo *IP Discovery*. Além disso, o *software* permite a conexão simultânea de vários clientes do *WEB Server*. Contudo, um cliente só poderá enviar dados para um dispositivo que não está em uso.

3.4 Serviço WEB Desenvolvido

O *Web Service* tem por objetivo realizar a comunicação entre o usuário e o FPGA, sendo transparente a idéia do *daemon* que estará realizando o tratamento dos pacotes trocados entre eles. Os serviços providos para esta comunicação são realizados por um Web *Server*, neste projeto o Apache, que irá servir as páginas, além das próprias páginas que realizam o envio das mensagens para o *daemon.*

Como idéia inicial, partiu-se de uma arquitetura 1↔1, ou seja, um FPGA controlado por somente um cliente, sem mecanismo de conexão. Porém foi verificado que qualquer outro usuário que obtivesse endereço onde está executando o servidor Web poderia ter acesso ao mesmo FPGA. Para solucionar este problema, foi adotado um mecanismo de conexão com um determinado FPGA e também um mecanismo de controle de sessão de usuário, a fim de identificar cada acesso que está chegando ao servidor.

Com as soluções escolhidas, foi possível manter um controle maior sobre os acessos às páginas e conseqüentemente aos FPGAs, além de propiciar o direcionamento das respostas providas pelo dispositivo somente para um único usuário, ou seja, para o usuário que está utilizando o FPGA. Dessa forma também, é possível prover a comunicação M↔N, com vários FPGAs sendo controladas por diferentes usuários.

Conforme já descrito, as páginas escritas em PHP têm a vantagem de serem *server-side*, ou seja, executam as instruções no servidor e exibem ao usuário somente a saída através de *tags*, como se fosse uma página HTML simples. Portanto, isto favorece a comunicação no protocolo *Web-Daemon* através de *socket* local. Sendo assim PHP foi a principal linguagem escolhida para a criação das páginas. Também foi utilizada a linguagem HTML e scripts descritos em *JavaScript* e Flash.

Dentre os campos apresentados nas páginas, pode-se destacar, além do mecanismo de conexão e desconexão, um campo para envio de um arquivo de dados destinado ao FPGA e também uma tela de visualização dos resultados obtidos deste dispositivo, sendo também possível realizar o *download* destes resultados.

4 Detalhamento da implementação

No presente capítulo será apresentado o detalhamento da implementação deste TCC. Serão descritas neste capítulo, todo o processo para a implementação das funcionalidades desenvolvidas, Serão abordados o desenvolvimento de um protocolo para comunicação entre o *Web Service* e o *daemon*, o desenvolvimento do *daemon*, o desenvolvimento das páginas *web* e também a implementação de um simulador para efetivação de testes.

No item 4.1 será apresentado o protocolo *Web-Daemon*, contendo a descrição dos modos de comunicação entre as páginsa e o *daemon*. No item que segue, será exposta a implementação do *software daemon*, realizando a descrição das rotinas e dos mecanismos utilizados para a comunicação deste com o *Web Service* e também com o módulo de *hardware*. O item 4.3 apresenta a descrição do simulador, onde será descrito sua funcionalidade e também os modos utilização deste software. Para finalizar, será exposto o desenvolvimento das páginas *web*, descrevendo suas telas, rotinas e formas de comunicação com o *daemon*.

4.1 Protocolo *Web-Daemon*

Com o objetivo de estabelecer um padrão na comunicação entre o *Web Service* e o *daemon*, foi desenvolvido um protocolo para as trocas de mensagens. Esse protocolo baseia-se no processo de identificação de dispositivos FPGAs (módulos de *hardware*) definido no protocolo BCTCP, através de palavras de 4 bytes cada uma, como segue:

- FPGA_ID: identificador do dispositivo;
- PROJECT_ID: identificador do projeto sintetizado no dispositivo;
- USER_ID: identificador de usuário.

Todas as comunicações realizadas entre o *daemon* e as páginas *web* são realizadas através de *sockets* entre processos. Estas comunicações iniciam-se quando um usuário abre a página do *Web Service*. Nesse ponto, o processo solicita para o *daemon* o status de todos os módulos de hardware. Para isso, é enviado para o *daemon* pelo *Web Service*, um pacote contendo a *string* “NEW_USER”. Ao receber esse pacote, o *daemon* deve responder com um pacote contendo as informações de todos os módulos de hardware que por ele são controladas. Esse pacote é composto por:

<nº_de_módulos_controlados>;<FPGA_ID>;<PROJECT_ID>;<USER_ID>;<STATUS>;<...>

O primeiro campo do pacote é o número de módulos que estão sendo controladas pelo *daemon*. Após, estão presentes os dados de identificação de cada FPGA, como mostrado

anteriormente. O campo "STATUS" informa se o módulo está em uso ou não. Caso exista mais de um módulo controlado pelo *daemon*, as demais informações mantêm o mesmo padrão de transmissão no pacote.

Quando um usuário seleciona um dispositivo e inicia uma conexão com esse, o *Web Service* envia para o *daemon* uma mensagem informando essa tentativa de conexão. Para isso, é enviado um pacote com a *string* "CONNECT". Ainda, nesse mesmo pacote, após a *string* inicial, é enviado o *session_id* da página associada ao cliente e, também, o FPGA_ID requisitado pelo cliente. Caso o *daemon* consiga estabelecer a conexão com o FPGA, ele envia a *string* "CLIENT_CREATED" é retornada para o *WEB Service*. Se o *software* não conseguir alocar o dispositivo, é enviada a mensagem "CANT_CONNECT".

Após o estabelecimento da conexão, é iniciada a etapa de envio de dados para o FPGA. Nesse ponto, o usuário, através do *Web Service*, realiza o *upload* do arquivo de dados no sistema. Após, a página comunica ao *daemon* que o arquivo a ser enviado para o módulo de hardware já foi carregado. Para isso, a página envia para o *software* a mensagem "FILE_OK".

Após realizar o upload do arquivo e sinalizar o *daemon* com a mensagem de "FILE_OK", a página começa a questionar periodicamente o daemon sobre o status do processamento. Isso ocorre com o objetivo de saber se o processo já foi finalizado. Para esta solicitação, é utilizada a mensagem "END_PROCESS". Em caso de término, a resposta enviada pelo daemon é "YES". Caso contrário, a resposta enviada é "NO". Essas mensagens estão sintetizadas e ilustradas na Tabela 1.

O processamento do FPGA será realizado a partir dos dados contidos em um arquivo. Esse arquivo é referenciado como arquivo de dados. Esse precisa ser coerente com a aplicação prototipada no dispositivo, ou seja, a taxa de envio e recebimento de dados além do tamanho dos mesmos.

Para isso, no inicio do arquivo de dados, é definida a quantidade de pacotes que o FPGA manipula antes de enviar uma resposta. Nesse trabalho, essa taxa é chamada de *burst*. Além desse parâmetro, no arquivo está presente ainda o tamanho dos dados a serem enviados, chamado aqui de *length*. O arquivo de dados a serem processados pelo dispositivo tem o formato apresentado na Figura 13.

Nesse arquivo, o primeiro número que aparece, no caso o '3', é o *burst* dessa comunicação. O segundo item, o '32', é o *length*. O restante do arquivo contém os dados a serem processados e enviados para o dispositivo FPGA.

Tabela 1 - Mensagens do protocolo Web-Daemon

Conteúdo da Mensagem	Origem	Destino	Finalidade	Quando é utilizada
NEW_USER	*WEB Service*	*Daemon*	Solicitar ao *Daemon* o status de todas as páginas	Quando uma página é aberta
<nro>;<F_ID>;<P_ID>;<U_ID>;<S> Onde: Nro: número módulos de hardware conectadas ao *daemon* F_ID: FPGA_ID P_ID: PROJECT_ID U_ID: USER_ID S: STATUS	*Daemon*	*WEB Service*	Informar para o *Web Service* o status de todos os módulos de hardware	Em resposta a uma mensagem de NEW_USER
CONNECT <session_id> <FPGA_ID>	*WEB Service*	*Daemon*	Solicitar ao *Daemon* que inicie a comunicação do cliente identificado por <session_id> com o FPGA identificado por <FPGA_ID>	Quando um novo cliente no *WEB Service* seleciona um módulo de hardware para iniciar o processamento
CLIENT_CREATED	*Daemon*	*WEB Service*	Informar ao WEB Service que o *Daemon* criou o novo cliente	Em resposta a uma solicitação de CONNECT quando ocorre sucesso
CANT_CONNECT	*Daemon*	*WEB Service*	Informar ao WEB Service que o *Daemon* não criou o novo cliente	Em resposta a uma solicitação de CONNECT quando não ocorre sucesso
END_PROCESS	*WEB Service*	*Daemon*	Solicitar ao *Daemon* se um processamento finalizou	É enviada periodicamente
YES	*Daemon*	*WEB Service*	Resposta do *Daemon* a quando o processamento foi finalizado	Enviada em resposta ao questionamento de END_PROCESS
NO	*Daemon*	*WEB Service*	Resposta do *Daemon* a quando o processamento ainda não foi finalizado	Enviada em resposta ao questionamento de END_PROCESS
FILE_OK	*WEB Service*	*Daemon*	Informa ao *Daemon* que o arquivo de dados está pronto para ser enviado para a placa	Quando é desejado enviar dados para o módulo de hardware
EXIT <session_id> <FPGA_ID>	*WEB Service*	*Daemon*	Informa ao *Daemon* que a comunicação do cliente identificado por <session_id> que estava associado com o FPGA identificado por <FPGA_ID>, seja fechado	Quando cliente solicita desconexão ou fecha a página, sem finalizar o processo

```
3 32
00000000000000000000000000000010
00000000000000000000000000000020
00000000000000000000000000000030
00000000000000000000000000000040
00000000000000000000000000000050
00000000000000000000000000000060
00000000000000000000000000000070
00000000000000000000000000000080
00000000000000000000000000000090
00000000000000000000000000000100
00000000000000000000000000000110
00000000000000000000000000000120
```

Figura 13 - Exemplo de arquivo de dados.

O resultado do processamento é enviado do dispositivo FPGA para o *daemon*. Esse *software* armazena o conteúdo do campo *data* de cada pacote UDP recebido em um arquivo de saída. Após o processamento completo, esse arquivo é disponibilizado para *download*. A seguir, é exibido na Figura 14, contendo um exemplo de arquivo de saída. Cada linha desse arquivo corresponde ao processamento de 3 linhas do arquivo de dados, representado na Figura 13. Então, a primeira linha do arquivo de saída corresponde as 3 primeiras linhas do arquivo de dados. Esse procedimento se repete até o final do arquivo de dados, conforme descrito no *hardware* de aplicação, 3.1.

```
60000000000000000000000000000000
15000000000000000000000000000000
24000000000000000000000000000000
33000000000000000000000000000000
```

Figura 14 - Exemplo do arquivo de saída.

Caso seja realizado o *upload* de um arquivo que contenha o campo de *length* superior a 1500 bytes, o *daemon* precisará realizar a fragmentação dos dados para envio ao FPGA. Esta fragmentação se deve ao fato de que o pacote UDP deve ser limitado em 1500 bytes, devido a uma limitação da rede Ethernet. Sendo assim, é necessário que juntamente com a segunda parte dos dados seja enviado o *header* do protocolo BCTCP.

4.2 Daemon de Comunicação

Como visto anteriormente, *daemons* são softwares que operam independente da ação do usuário. Para isso, neste trabalho o *daemon* implementado recebe como estímulo pacotes de sistemas externos, no caso o *Web Service* e o FPGA. Além disso, como configuração inicial, o *daemon* realiza a leitura de um arquivo de configuração.

O arquivo de configuração inicial do *daemon* contém os dados respectivos de todos os FPGAs que o sistema pode controlar. Esses dados seguem o padrão de identificação de FPGAs do protocolo BCTCP, que são: o identificador do FPGA, Figura 15, destaque 1, o identificador do projeto Figura 15, destaque 2 e o identificador do usuário, Figura 15, destaque 3. Assim, para adicionar um novo módulo de *hardware* para o sistema controlar, deve-se acrescentar os dados dessa placa em uma nova linha no arquivo de configuração do *daemon*, e reiniciá-lo para que possa atualizar seu conteúdo. Para realizar esta etapa, é preciso garantir que nenhum usuário esteja acessando nenhum dispositivo por meio do *daemon*.

O *daemon* provê a comunicação do *Web Service* com o FPGA. Esse *software* recebe pacotes de requisições da interface *Web*, de acordo com o protocolo descrito acima. A seguir, trata as

mensagens recebidas. Essas mensagens contêm o identificador do cliente que está utilizando o *Web Service* e com qual FPGA o cliente deseja realizar a comunicação.

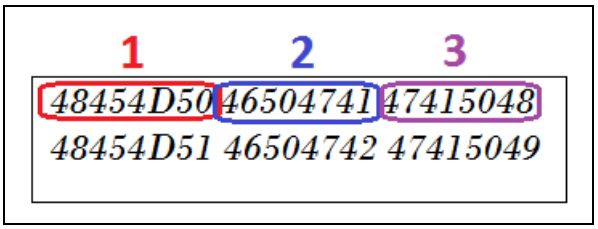

Figura 15 - Exemplo de arquivo de configuração.

O *daemon* desenvolvido utiliza uma arquitetura baseada em processos paralelos. Para isso, foi utilizado o mecanismo de software *multi-thread*. Foram implementadas *threads* para a comunicação da página inicial do *Web Service* com o *daemon*, para a manipulação de conexão e desconexão de usuários junto ao *daemon*, além de *threads* para troca de mensagens entre o *daemon* e o FPGA.

Esse sistema ainda mantém em duas estruturas globais todos os atributos inerentes a comunicação. Uma dessas estruturas é alocada na forma de uma lista que mantém informações sobre todos os módulos controlados pelo *daemon*. As informações mantidas são as presentes na Figura 16, a *struct list_of_fpga*. Essa lista é preenchida com os parâmetros recebidos a cada descoberta de um novo módulo de *hardware*. A outra estrutura também é alocada na forma de uma lista. Ela mantém uma relação dos clientes conectados ao *daemon* e que estão utilizando um módulo. Essa estrutura é preenchida a cada nova conexão de clientes no *daemon*.

```
struct list_of_fpga {
        struct bctcp_data fpga;
        struct sockaddr_in board_address;
        int end_process;
        int status;
        int user_session_id[64];
        char file_output[32];
        char file_input[32];
        u_int16_t seq_number_htof;
        u_int16_t seq_number_ftoh;
};
```

Figura 16 - Pseudo-código *struct list_of_fpga*.

A estrutura que gerencia as informações sobre os FPGAs, Figura 16, mantém para cada dispositivo a flag ‘end_process’ para sinalizar a conclusão do processamento, o status de cada

placa, o identificador do usuário que está manipulando o módulo de hardware, na variável user_session_id, o nome dos arquivos de entrada e saída, file_input e file_output respectivamente, além dos números de seqüência do protocolo.

```
struct list_of_clients {
        pthread_t th_in;
        pthread_t th_out;
        char session_id[64];
        u_int fpga_id;
};
```

Figura 17 - Pseudo-código da *struct list_of_clients*.

Na primeira execução do *daemon* inicia-se o processo de descoberta de *IPs* de todos os FPGAs a serem controlados. Para realizar esse processo, é chamada a rotina *discovery_all_boards*, ilustrada na Figura 18. Essa rotina lê uma linha por vez do arquivo de configuração. Posteriormente, a cada linha lida, é realizado o processamento das linhas lidas do arquivo, com o objetivo de obterem-se os parâmetros de cada FPGA a ser buscada na rede. Logo a seguir, é realizado um teste para saber se o *daemon* ainda pode controlar mais um módulo. Caso possa, é chamada a rotina *bctcp_ip_discovery* e para essa rotina é passado o índice disponível da lista global dos módulos controlados. Após a descoberta de IP, é setada uma *flag* para o módulo em questão, na lista de módulos, para sinalizar que esse dispositivo está liberado para o uso por um cliente. No final da execução da rotina *discovery_all_boards* a estrutura que mantém as informações de todos módulos deve estar preenchida.

```
1: discovery_all_boards() {
2:      struct list_of_fpga placa;
3:      abre_arquivo_de_configuração;
4:      enquanto (não_fim_do_arquivo) {
5:              linha = ler_linha_do_arquivo;
6:              dados = parser(linha);
7:              placa = bctcp_ip_discovery(dados);
8:              placa.status = PLACA_LIVRE;
9:              placas_ativas++;
10:     }
11:     fclose (F);
12:     return retval;
13: }
```

Figura 18 - Pseudo-código *discovery_all_boards*.

O funcionamento da rotina *bctcp_ip_discovery*, Figura 19, é baseado no protocolo *IP Discovery*. Para isso, primeiramente a rotina abre um *socket* UDP com destino *broadcast* e endereçado para a porta definida como a porta de serviço do *IP Discovery*. O campo *data* desse pacote é preenchido com os parâmetros que identificam o FPGA no qual é desejado descobrir o IP. A seguir, é enviado esse pacote para a rede. Após o envio, a rotina *bctcp_ip_discovery* aguarda até 5 segundos por uma resposta do FPGA. Caso o dispositivo não responda, considera-se que esse está fora de operação. Se a resposta recebida conter a *flag* de BUSY e ativa (0xFF), é considerado que o FPGA já está sendo utilizado por outro processo e esse módulo fica fora de operação. Quando a resposta apresentar a *flag* BUSY desativada (0x00) significa que o módulo está disponível e que agora o *daemon* passará a ter o controle sobre ele. Os dados presentes no campo data do pacote UDP recebido completam os parâmetros ainda pendentes na lista de módulos para aquele FPGA. Após realizar esse tratamento, é enviado um pacote UDP endereçado diretamente para o IP do dispositivo com as *flags* de SYN e de ACK ativas. Isso realiza a sincronização com o referido dispositivo. Além disso, o envio do pacote com essas *flags* caracteriza que o *daemon* recebeu corretamente o pacote.

```
1: bctcp_ip_discovery(dados_do_fpga) {
2:              struct list_of_fpga placa;
3:              sock = socket(end_destino_broadcast, porta_ip_discovery);
4:              campo_data = dados_do_fpga;
5:              enviar(sock, campo_data);
6:              enquanto (não_passou_5s) {
7:                      espera (receber(sock, placa));
8:              }
9:              se não_respondeu {
10:                     placa.status = FORA_DE_OPERAÇÃO;
11:                     retorna placa;
12:             }
13:             se (placa.status == BUSY) {
14:                     retorna placa;
15:             }
16:             se (placa.flag == ACK) {
17:                     sock_placa = socket(placa.endereço, porta_bctcp);
18:                     enviar(sock_placa, SYN+ACK);
19:                     retornar placa;
20:             }
21: }
```

Figura 19 - Pseudo-código *bctcp_ip_discovery*

Após o preenchimento da lista de módulos, é iniciada a *thread connection_manager*. Essa *thread* provê a comunicação da página inicial com o *daemon*, informando o *status* de todos os módulos manipulados pelo *software*. Para isso, é estabelecida uma comunicação via *socket* do tipo AF_UNIX, ou *socket* entre processos. Esse canal de comunicação foi chamado de *socket_master*.

Além disso, a característica principal dessa rotina é criar novos clientes no *daemon*. Após informar para a página quais os módulos disponíveis, o cliente pode selecionar um módulo para iniciar uma comunicação. Quando o cliente seleciona um módulo, o *Web Service* envia uma mensagem de "NEW_CLIENT", conforme o protocolo *Web-Daemon* descreve.

A thread *connection_manager* é o processo responsável por receber essa mensagem e iniciar o processo de novo cliente. Esse processo é realizado pela rotina *create_client*. Essa rotina, primeiramente analisa se ainda é possível incluir um novo cliente na lista de clientes. Caso seja, são criadas duas novas *threads*. Uma delas, a *webservice_input*, é utilizada para tratar as mensagens recebidas da página pelo cliente e ainda inicia o processo de envio de dados para processamento no módulo. A outra, a *webservice_output*, é utilizada para enviar as respostas do processamento dos dados pelo FPGA para a página. Além disso, a *connection_manager* ainda é responsável por tratar a desconexão de um cliente do sistema. A Figura 20 aborda o fluxo de funcionamento dessa *thread*.

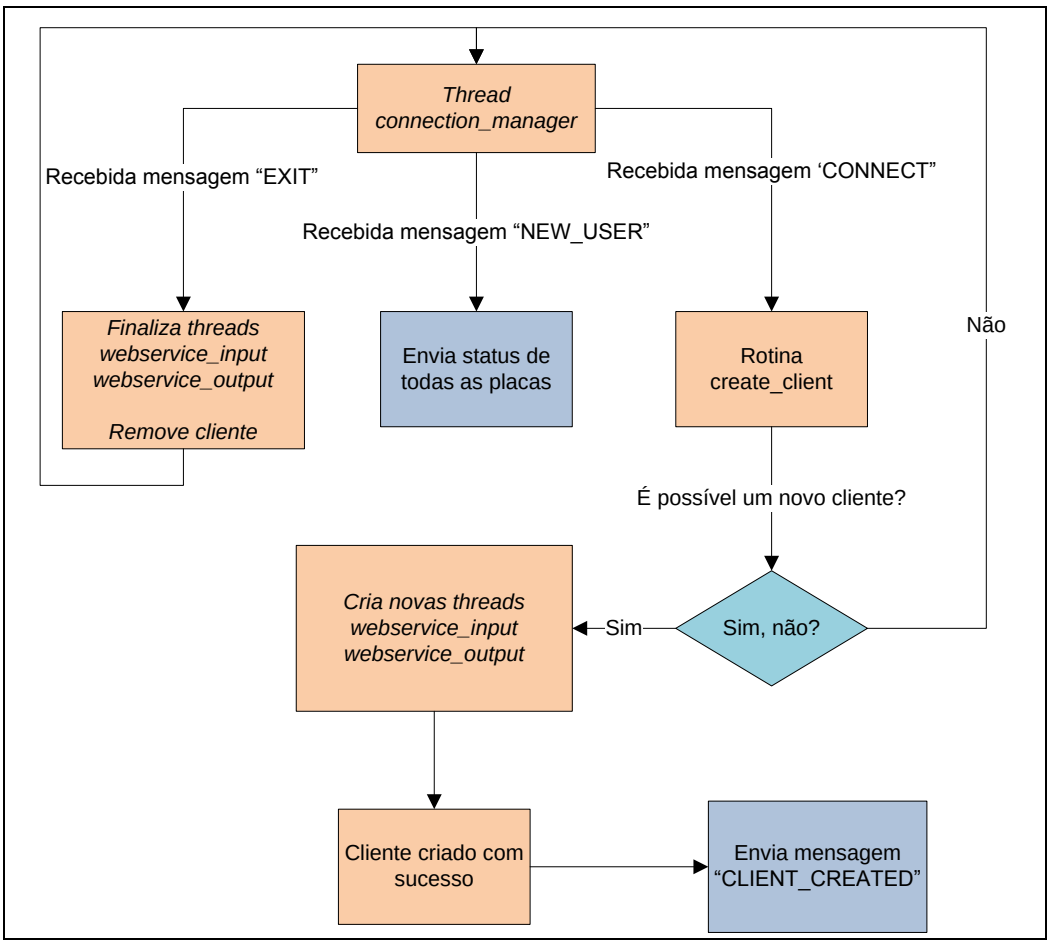

Figura 20 - Fluxograma da *thread connection_manager*.

A *thread webservice_input* é iniciada quando um novo cliente conecta-se ao *daemon*. Essa rotina é responsável por iniciar o processo de comunicação entre o *daemon* e o FPGA. Primeiramente, essa *thread* inicia um novo *socket* de comunicação do *software* com o *Web Service*. Esse *socket* será o canal de comunicação por onde irá trafegar a informação de que o arquivo de dados a ser processado pelo FPGA está pronto. Para isso, a *thread* aguarda uma mensagem do *Web Service* com o conteúdo "FILE_OK". Quando o *daemon* recebe essa mensagem, significa que o arquivo com os dados de entrada está pronto. Esse é identificado com o nome do FPGA que o usuário escolheu. Nesse ponto, é chamada a rotina *send_data_to_board*.

A rotina *send_data_to_board*, Figura 21, primeiramente abre um *socket* de comunicação do tipo UDP entre o *daemon* e o FPGA. Após esse processo, a rotina realiza a leitura da primeira linha do arquivo de dados. Essa linha contém a informação de qual será o *burst* de pacotes enviados e o tamanho de cada pacote. As linhas posteriores desse arquivo são os dados destinados ao processamento. Cada nova linha do arquivo é lida e encapsulada dentro do campo *data* de um pacote no formato BCTCP. Esse pacote contém ainda o *header* do protocolo BCTCP, ou seja, o campo ACK e o número de sequência da comunicação. Após a montagem do pacote, é realizado o envio desse para o FPGA. São enviados, seqüencialmente, a quantidade de pacotes definidos no *burst* e em seguida é aguardada uma resposta do FPGA. Quando a resposta do dispositivo é recebida, primeiramente são verificadas as *flags* SYN e ACK do BCTCP. Caso essas *flags* estejam presentes no pacote, significa que o pacote oriundo do FPGA é válido e está correto. Sendo assim, o conteúdo recebido no campo *data* do pacote BCTCP é armazenado no arquivo de saída. Após o envio de todas as linhas do arquivo de dados para o módulo e do recebimento da resposta do FPGA, a *flag* de controle do *daemon*, *end_process*, é sinalizada. Essa *flag* indica que o processamento para aquele dispositivo finalizou.

Assim como a thread *webservice_input*, a *thread webservice_output* também é iniciada quando um novo cliente conecta-se ao *daemon*. Essa última monitora a finalização do processamento realizado pelo FPGA, através da *flag end_process*. Além disso, ela também é responsável por responder as requisições periódicas, *polling*, realizadas pelo *Web Service*. Para isso, um novo *socket* entre a página e o *daemon* é iniciado. Por esse *socket*, o *Web Service* envia mensagens de "END_PROCESS" para o *daemon* com o objetivo de saber o status do processamento. O *daemon* por sua vez verifica a *flag end_process*. Enquanto esta *flag* não é sinalizada pela *thread webservice_input*, o *daemon* responde para o *Web Service* que a solicitação está em processamento. Após a sinalização da *flag*, na próxima solicitação da página, a *webservice_output* responde que o processamento finalizou e o que o *download* do arquivo de saída pode ser realizado.

```
1: send_data_to_board(board) {
2:              sock = socket(board, PORTA_BCTCP);
3:              abre_arquivo_de_dados;
4:              linha = le_linha();
5:              burst = parser(linha[0]);
6:              length = parser(linha[1]);
7:              enquanto (não_fim_do_arquivo) {
8:              linha = le_linha();
9:              enviar(sock, linha, length);
10:             num_envios++;
11:             if (num_envios > burst) {
12:                             receber(sock, retorno);
13:                             escrever_arquivo(arquivo_saida, retorno);
14:                             num_envios = 0;
15:                     }
16:             }
17:             end_process = 1;
19: }
```

Figura 21 - Pseudo-Código *send_data_to_board.*

Ao término deste processo, é possível fechar o *socket* de comunicação entre a página de requisição de status e o *daemon*, aberto no processo *webservice_output*. Porém o *socket* que espera o recebimento de sinalização de arquivo destinado ao FPGA não pode ser finalizado, pois é possível que o usuário realize uma nova submissão de dados para processamento no dispositivo.

Todos os *sockets* abertos serão fechados somente quando o *daemon* receber a *string* "EXIT", sinalizando a desconexão do usuário.

4.3 Simulador

Com o objetivo de validar o protocolo BCTCP, bem como o funcionamento do *daemon* e do *Web Service*, foi desenvolvido um simulador que implementa o protocolo. Esse simulador é um *software*, escrito em linguagem C, executado em um computador com sistema operacional Linux. Ele apresenta o comportamento da aplicação sintetizada no FPGA, onde estão presentes o protocolo IP *Dicovery* e o protocolo BCTCP, bem como a aplicação do usuário. Além disso, o simulador permite que possam ser controlados dispositivos distintos de FPGAs pelo *daemon*. Esse *software* serve ainda como um apoio na etapa de projeto.

Primeiramente, o simulador é identificado como se fosse um FPGA. Para isso, de acordo com o protocolo BCTCP, ele recebe um FPGA_ID, um PROJECT_ID e um USER_ID. Essas são as referências do simulador na rede. A seguir, é aberto um *socket* UDP entre o simulador e a rede.

Por esse canal são trafegados os pacotes tanto para o protocolo IP *Discovery*, quanto os pacotes do protocolo BCTCP.

O funcionamento desse simulador é o mais simples possível. Ele é seqüencial e apenas responde a pacotes que são a ele enviados.

Em primeiro lugar no simulador, o *socket* aberto é endereçado para a porta do protocolo IP *Discovery*. Isso ocorre porque esse *software* precisa responder para a rede os seus endereços de IP e de MAC, quando o *daemon* executar o processo de descoberta de módulos. Nesse ponto, o simulador é reconhecido pelo *daemon* como um dispositivo ativo e operando sob o protocolo BCTCP. Além disso, o *daemon* é o responsável por enviar e receber pacotes do simulador.

Após responder esse pacote de descoberta, o *socket* do simulador passa a ser endereçado para a porta do protocolo BCTCP. A partir desse ponto, o simulador recebe do *daemon* o número de pacotes tal como está definido no *burst* do arquivo de dados. A seguir, realiza uma soma *byte* a *byte* de todos os dados do pacote e devolve para o *daemon*. Essa característica visa ser a mesma realizada no hardware do usuário sintetizado nos FPGAs.

Além disso, o controle do tráfego de dados é realizado segundo o protocolo BCTCP. Para isso, é utilizada a lógica dos números de sequência previstas no protocolo, bem como os pacotes com as *flags* de CTRL, ACK, NACK, SYN e FIN.

O simulador para ser executado precisa receber como parâmetros:

<mac_simulador> <ip_simulador> <PROJECT_ID> <FPGA_ID> <USER_ID> <ip_daemon>

Os dois primeiros parâmetros são os identificadores da interface de rede do computador onde é executado o simulador. Os três próximos campos indicam os identificadores utilizados pelo simulador e o último campo trata-se do endereço IP do servidor.

4.4 Web Service

Conforme descrito no capítulo anterior, o *Web Service* desenvolvido neste trabalho é composto por um servidor *web* e também pelas páginas que implementam as rotinas de comunicação e execução das instruções a elas repassadas.

O servidor *web* utilizado durante o desenvolvimento deste trabalho foi o *software* XAMPP [APA11]. Trata-se de um *software* livre, que implementa um servidor *web* com muitas funcionalidades: base de dados MySQL, o servidor *web* Apache e os interpretadores para linguagens de script: PHP e PERL.

Este programa está liberado sob a licença GNU e atua como um servidor *web* livre, de fácil utilização e capaz de interpretar páginas dinâmicas. Atualmente XAMPP está disponível para Microsoft Windows, GNU/Linux, Solaris, e MacOS X, mas foi testado somente em ambiente Linux, no mesmo computador onde o *daemon* estava em execução.

A maior parte da programação das páginas *web* realizou-se através da linguagem PHP, porém também foi utilizado HTML e scripts em *JavaScript* e *JQuery* (uma extensão de *JavaScript*), além de animações em Flash. Ao todo, para uma melhor organização e uma comunicação que provê a comunicação M↔N, foram desenvolvidas 7 páginas: *index.php*, *envio.php*, *send.php*, *retorno.php*, *termino.php*, *download.php*, *exit.php*.

O acesso a estas páginas acontece por um navegador padrão, por meio de qualquer sistema operacional, como por exemplo, o *Internet Explorer* ou o *Mozilla Firefox*. Além disso, o acesso ocorre por meio do endereço do servidor de onde está sendo executado o *daemon* de controle.

Esse endereço tanto pode ser um endereço da rede local quanto pode ser uma URL para acesso da rede externa. Caso seja um endereço local, o acesso se dará ao digitar o IP da máquina servidora no navegador. Quando o acesso for realizado através de uma URL, deve ser digitado o endereço do site onde está hospedado o servidor.

Ao iniciar o processo, o usuário será direcionado para a página *index.php*. Esta página apresentará uma lista com todos os dispositivos que estiverem na rede e que sejam controlados pelo *daemon.* Além disso, ela apresenta um botão para selecionar o FPGA que o usuário queira utilizar. Outro botão presente é o botão "CONECTAR". Esse realiza o envio destas informações ao *daemon*, com o objetivo de efetivar esta conexão e posterior comunicação.

Antes de a página inicial ser exibida ao usuário, ela envia uma requisição ao *daemon* solicitando por meio de um pacote contendo a string "NEW_USER" a lista de FPGAs ativos (linha 6 da Figura 22). A comunicação entre os dois processos ocorre através de um socket denominado *socket_master*. A resposta a este pedido, é recebido, na variável "$out" (linha 7 da Figura 22), ou seja, o número de dispositivos disponíveis, e seus respectivos identificadores de Projeto, FPGA e Usuário, além de seu estado (livre ou ocupado). Todos estes dados se apresentam separados por um caractere ";". O estado dos FPGAs é um dado do tipo inteiro, representado pelo número 1 quando um módulo está em uso e o número 0 indica que esse está livre e pode receber conexões.

De posse destes dados, é montada uma tabela com todas estas informações e direcionadas ao usuário através da página *web*, por meio do protocolo HTTP. O campo para seleção do FPGA só é habilitado caso ele esteja livre, ou seja, o campo indicando o estado contém o número 0. Caso todos os FPGAs estejam sendo utilizados, o botão "CONECTAR" também não estará disponível e o usuário deverá tentar novamente o acesso, em outra oportunidade.

```
1:    session_start();
2:    $_SESSION["id_session"] = session_id();
3:    ...
4:    $sock = socket_create(AF_UNIX, SOCK_STREAM, 0);
5:    socket_connect($sock, LOCAL_MASTER);
6:    socket_write($sock, NEW_USER, strlen(NEW_USER))
7:    $out = socket_read($sock, 1024);
8:    socket_close($sock);
9:    $new = strchr($out, ';');
10:   $num_fpgas = $out-$new;
11:   $new = substr($new, 1);
12:   ...
13:   <td><input type="radio" name="fpga" value="',$i,'"></td>
14:   <td>$FPGA_ID[$i]'</td>
15:   <td >$PROJECT_ID[$i])</td>
16:   <td>$USER_ID[$i]</td>
17:   <td >  if ($STATUS_ID[$i]) echo 'OCUPADO';
18:              else echo'LIVRE';
19:   </td>
```

Figura 22 - Pseudo Código do envio de "NEW_USER" e recebimento dos dados dos dispositivos.

Além de todas estas comunicações e trocas de mensagens, a página inicial realiza a identificação de usuário através de um número de sessão, diferente para cada computador. Esta identificação ocorre por meio de uma rotina denominada *session_start( )*. Nessa função são inicializadas as variáveis globais e definidas constantes que podem ser transferidas para outras sessões, ou seja, para outras páginas. O identificador da sessão pode ser acessado pela chamada *session_id( )*.

As sessões são métodos de preservar determinados dados ativos enquanto o navegador do cliente estiver aberto, ou enquanto a sessão não expirar (por inatividade, ou por meio da rotina *session_write_close( )*. Uma vez iniciada a sessão, e definidas variáveis dentro dela, é possível acessar estas variáveis em outras páginas, desde que a sessão seja a mesma. Como exemplo prático de sessão, pode-se citar um serviço de e-mail. O usuário efetua *login* em um servidor e este cria uma sessão momentânea, até que o usuário efetue *logoff*.

As variáveis declaradas dentro de uma sessão são conhecidas como variáveis de sessão. Estas variáveis serão específicas do usuário de modo que podem coexistir diversas variáveis de sessão do mesmo tipo com diferentes valores para cada uma das sessões. Estas sessões têm o seu próprio identificador de sessão que será único e específico.

Assim, este identificador de sessão será transferido para as demais páginas através da própria sessão, pois ela só será finalizada quando o usuário solicitar. Esta identificação é repassada

com a finalidade de fazer com que todas as posteriores comunicações entre o usuário e o ocorram através de um *socket* referenciado pelo identificador da sessão. Ao clicar no botão “CONECTAR”, é aberta uma nova página, a *envio.php*. No endereço URL dessa página, estão presentes o *session_id* iniciado na página anterior e o FPGA_ID do módulo que o usuário deseja iniciar a comunicação.

Primeiramente a nova página aberta, ainda através do *socket_master*, informará ao *daemon* que necessita iniciar a comunicação com o FPGA referenciado pelo ID recebido como parâmetro pela URL. Para informar o *daemon* dessa solicitação, é enviada uma mensagem de requisição com a *string* que apresenta o seguinte formato: "*connect* <*fpga_id*> <*session_id*>". Esses parâmetros são providos pela página anterior. Caso a requisição ocorra com sucesso, um novo campo é aberto para o envio do arquivo de dados. Em contrapartida, caso a solicitação não ocorra com sucesso, será exibida uma mensagem de erro, informando ao usuário falha na conexão e este deverá retornar para a página anterior e refazer sua escolha. A Figura 23 traz um pseudocódigo da montagem do pacote para envio contendo o "*connect*", além de apresentar também o formulário para realizar o *upload* do arquivo de dados.

```
1:    $msg = sprintf ("connect %s %s", $fpga, $id_session);
2:    socket_write($sock, $msg, strlen($msg));
3:    ...
4:    <form enctype="multipart/form-data" action="send.php?fpga=$fpga" method="post">
5:          Arquivo a ser enviado:<br>
6:          <input id="file" name="file" type="file" />
7:          <input type="submit" value="Enviar" />
8:    </form>
```

Figura 23 - Pseudocódigo da mensagem de connect e campo para envio de dados

No campo de submissão de arquivos, o usuário deve selecionar o arquivo de dados para processamento no FPGA. Após a seleção do arquivo, deve ser utilizado o botão “Enviar” para realizar o envio propriamente dito. Ao clicar neste botão, por meio da página *send.php*, o arquivo será enviado e o *daemon* sinalizado que o arquivo está pronto para ser consumido, por meio da mensagem contendo a *string* "FILE_OK". Esta comunicação é realizada por meio de um novo *socket* referenciado pelo identificador de usuário. Todo este processo está ilustrado na Figura 24.

Após o envio de dados com sucesso, o usuário será direcionado para uma nova página, a *retorno.php*, que realiza requisições a cada 5 segundos, solicitando o *status* do processo solicitado ao FPGA. Caso ocorra erro na transmissão do arquivo, será apresentada uma tela exibindo que o processo não foi finalizado corretamente. Esta página irá receber, também via URL, a identificação do módulo em questão. Isso servirá para que seja identificado em qual FPGA ocorreu o problema.

```
1:      if(!$_FILES)
2:              echo 'Nenhum arquivo enviado!';
3:      else {
4:              move_uploaded_file ($destino.$arq_nome)
5:              socket_write($sock, "FILE_OK")
6:      }
```

Figura 24 - Pseudocódigo do envio de arquivo e a comunicação do *web* com o *daemon*.

Assim como a página *send.php*, a página *retorno.php* também se comunica com o *daemon* através de um *socket* referenciado pelo identificador de usuário e envia um pacote para o *software* contendo a string "END_PROCESS", com o objetivo de verificar se o processo atingiu o final da execução. As possíveis respostas enviadas pelo *daemon* são as *strings*: "YES", quando a execução foi finalizada, e "NO", caso o processo ainda esteja em execução.

De acordo com a resposta recebida é tomada uma decisão. Caso o processo não tenha terminado, a saída temporária do processamento é exibida durante o tempo em que o processo estiver em execução. Essa saída é obtida através da leitura do arquivo de saída que está sendo construído. Ou seja, é exibido o conteúdo do arquivo de saída enquanto o *daemon* paralelamente atualiza esse arquivo. Ainda, as requisições serão atualizadas a cada 5 segundos, através do mecanismo de, conforme mostra a linha 2 da Figura 25. Com a finalização do processo, será exibida uma página final, que leva o nome de *termino.php*.

A página *termino.php*, leva esse nome pois trata-se da última página a ser exibida durante o processo de envio e recebimento de dados. Ela conta com um botão "ENVIAR ARQUIVO", para realizar a transmissão de novos arquivos de dados ao FPGA. O uso desse botão irá direcionar o usuário para a página *envio.php*. Além desse,um botão chamado “DOWNLOAD” é exibido. Ao clicar nesse botão, o usuário efetua o *download* do arquivo de saída do processamento do FPGA. Ou seja, um arquivo que contém todas as informações provenientes do dispositivo que estão sendo exibidas na tela.

Este mecanismo de *download* é realizado pela página *download.php*. Essa recebe, por parâmetro, o nome do arquivo de dados, disponibilizado para *download* ao usuário. Esse arquivo tem o nome padrão de *output_file.txt*, conforme mostra a Figura 26, que contém o código da descrição deste mecanismo.

Além dos 2 botões já descritos, esta página, apresenta o botão "DESCONECTAR". Esse botão serve para que o usuário libere a utilização de um módulo que estava para ele reservado. O mecanismo de desconexão é feito pela página *exit.php*. Essa se comunica diretamente com o

daemon através socket_master e o informa qual cliente está encerrando a conexão. Essa comunicação ocorre por meio do envio de um pacote contendo a seguinte string: "exit <fpga_id> <session_id>". Na tela, será exibida uma mensagem informando que a conexão foi terminada.

```
1:    $url = retorno.php;
2:    $timeout=5;
3:    header('Refresh: ' , $timeout , $url);
4:    ...
5:    socket_write($sock, "END_PROCESS");
6:    $resposta = socket_read($sock, 10);
7:    se ($resposta == "no") {
8:          exibe( );
9:    }
10:   senao se ($resposta == "yes") {
11:         $url=retorno_term.php;
12:         $timeout=0;
13:         header('Refresh: ' , $timeout , $url);
14:   }
```

Figura 25 – Pseudocódigo contendo o *polling* e a solicitação de estado do processamento

```php
<?php

// nome do arquivo recebido via URL
$fullPath = $_GET['download_file'];

if (($fd = fopen ($fullPath, "r")) {
    $fsize = filesize($fullPath);

    $novoNome = 'output_file.txt';

    // Configuramos os headers que serão enviados para o browser
    header('Cache-control: private');
    header('Content-Description: File Transfer');
    header('Content-Disposition: attachment; filename="'.$novoNome.'"');
    header('Content-Type: application/octet-stream');
    header('Content-Transfer-Encoding: binary');
    header("Content-length: $fsize");
    header('Cache-Control: must-revalidate, post-check=0, pre-check=0');
    header('Pragma: public');
    header('Expires: 0');

    // Envia o arquivo para o cliente
    readfile($fullPath);
}
fclose ($fd);
exit;
?>
```

Figura 26 - Código da página que disponibiliza o arquivo para *download*.

Através de um mecanismo em JavaScript, é possível determinar os eventos promovidos pelo usuário nas páginas *web*, tais como clique em um botão, a atualização ou o fechamento de um página. Com o auxilio deste mecanismo, caso o usuário feche alguma das páginas em algum momento, sem realizar a desconexão através do botão próprio para tal, também é chamada a página *exit.php* para realizar a desconexão do usuário e liberar o FPGA de maneira correta.

5 Validação da Arquitetura

Neste capítulo é reportada a validação da ferramenta descrita nos Capítulos anteriores. Para tal, são validados separadamente o *Web Service*, *Daemon* e Simulador apresentados neste trabalho. A primeira parte desta validação tem por objetivo verificar a correta funcionalidade do *Web Service* através da simulação da conexão de um cliente e o envio de dados para um FPGA. A segunda parte compreende esta mesma simulação, entretanto em um ponto de vista dos pacotes recebidos e enviados pelo *daemon*. Ainda utilizando essa simulação, é feita a análise do funcionamento do simulador. Por fim, nesse mesmo cenário, é reportado o teste utilizando o FPGA.

O arquivo de configuração utilizado no *daemon* para a etapa de descoberta de módulos é o mesmo apresentado na Figura 15. Ainda, embora no arquivo estejam configurados dois módulos, nesse cenário somente foi habilitado um módulo de *hardware*. Isso permite validar o mecanismo de *timeout* da etapa de descoberta de módulos. O arquivo de dados a ser enviado é o ilustrado na Figura 13.

5.1 Validação do WEB Service

Para efetuar a validação do *Web Service*, utilizou-se um navegador comum. Para isso, foi aberto na ferramenta o endereço do servidor onde está hospedado o site. A tela inicial apresentada na Figura 27. Observa-se que o cliente tem a visualização de um conjunto de módulos e os seus respectivos *status*. Para que essa informação fosse exibida pela página, é necessário que a página envie a mensagem "NEW_USER" para o *daemon*. Esse por sua vez, responde com os parâmetros de todos os módulos configurados, conforme a descrição do protocolo *WEB-Daemon* no 4.1.

O cliente pode selecionar um destes módulos e iniciar uma conexão. Pressupõe-se nesse ponto que o cliente tem conhecimento sobre *hardware* de aplicação que está sintetizado em cada módulo de *hardware* e que essa informação é levada em consideração no momento de realizar a escolha e, posteriormente, o *upload* do arquivo de dados a ser processado.

Após selecionar um módulo e clicar no botão "Conectar", é aberta a tela de envio de dados, apresentada na Figura 28. Quando o cliente clica no botão "Conectar", o *Web Service* envia uma mensagem contendo a *string* "CONECTAR" para o *daemon*. Após esse processo, o cliente deve realizar o *upload* do arquivo de dados. Depois de carregar o arquivo, o cliente deve clicar no botão "ENVIAR", Figura 28. Nesse ponto, a mensagem com a *string* "FILE_OK" é enviada para o *daemon*.

Figura 27 - Página Inicial do Web Service apresentando o único FPGA na rede.

Figura 28 - Página de envio de Dados.

Após o envio dos dados, a tela da Figura 29 é exibida. Essa tela exibe a saída temporária do processamento. Enquanto a execução não é finalizada, as mensagens resultantes do processamento, enviadas pelo FPGA, são exibidas nesta tela.

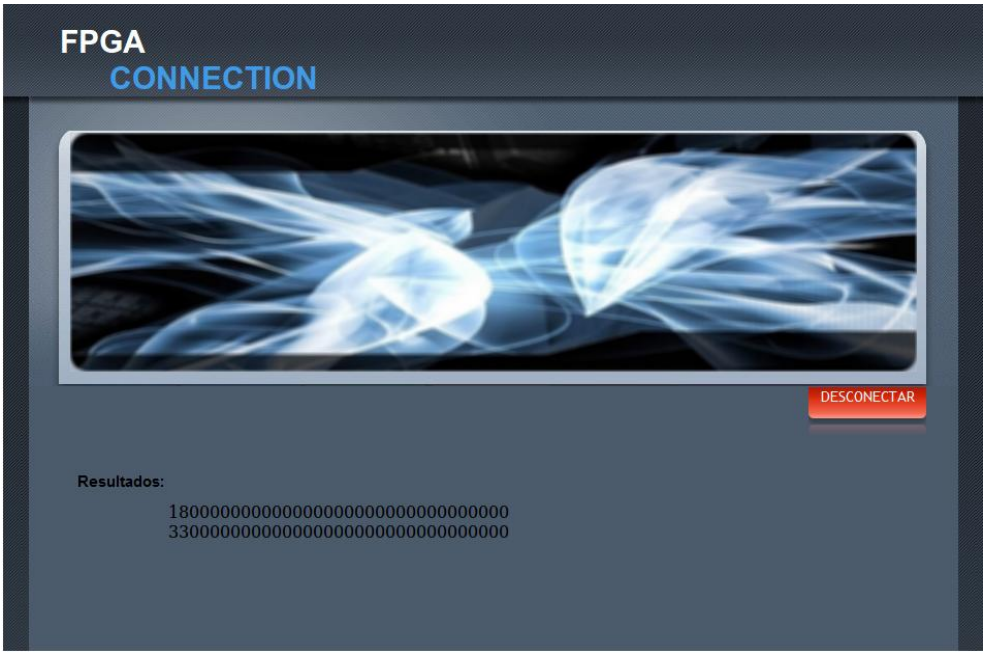

Figura 29 - Resultados parciais exibidos durante o processamento.

Após o término da execução, é exibida a página da Figura 30. Nesse ponto o cliente pode realizar 3 operações: (*i*) realizar o download do arquivo de saída, utilizando o botão “DOWNLOAD”; (*ii*) retornar a página de envio de dados, clicando em “ENVIAR ARQUIVO”; (*iii*) solicitar a desconexão do FPGA, através do botão "DESCONECTAR". Se esta for a opção escolhida, é enviada uma mensagem contendo a *string* "EXIT" ao *daemon*.

Figura 30 - Término do Processamento.

5.2 Validação do *Daemon*

Para realizar a validação do *daemon*, utilizou-se o mesmo teste realizado na validação do *WEB Service*. O *software* foi executado no sistema operacional Linux Ubuntu. Para efeito de ilustração, ele foi executado com o modo de *debug* ativo. Esse modo permite que todas as mensagens e pacotes recebidos pelo *daemon* sejam exibidos na tela. Para iniciar o daemon, foi executado o seguinte comando:

./daemon -ip <ip_da_maquina_local> -mac <end_mac_da_interface> -debug_mode

Ao executar esse comando, o *daemon* realiza o seu *start-up*. Inicialmente, ocorre a leitura do arquivo de configuração. Logo em seguida ocorre a descoberta de todos os dispositivos previamente inseridos no arquivo e que estão presentes na rede. Para isso, é enviado o pacote de descoberta de IP para cada dispositivo. A Figura 31, destaque 1, ilustra a inicialização. Ainda, foi realizado o teste para descoberta de um segundo módulo que não estava presente na rede. A rotina tenta descobrir o endereço IP dessa placa. Como a resposta não é recebida antes do tempo de espera por uma resposta expirar, Figura 31, destaque 2, a placa não é adicionada na lista.

Após iniciar a descoberta de módulos, o *daemon* inicia a thread *connection_manager*. Agora, o *software* estará apto para receber as requisições do *Web Service*. Nesse ponto é realizada a abertura da página principal do serviço *Web*. Isso desencadeia o envio da mensagem "NEW_USER" transmitido pela página para o *daemon*. O *software* ao receber essa mensagem responde com todos os dispositivos que por ele são controlados, conforme a Figura 31, destaque 3.

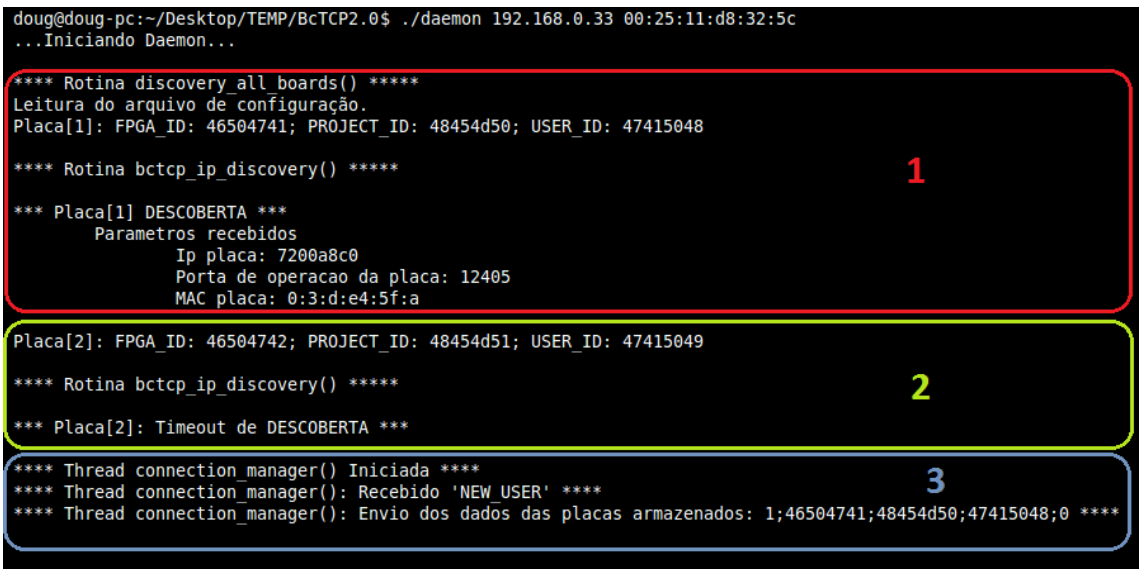

Figura 31 - Recebimento do "NEW_USER" proveniente do Web Service

Após informar o *status* de todos os módulos por ele controlado, o *daemon* aguarda a tentativa de conexão de um usuário. Essa tentativa ocorre quando o *software* recebe do *socket* a mensagem “CONNECT”. Primeiramente o *daemon* procura uma posição vaga na lista de clientes. Caso encontre, são iniciadas as *threads* correspondentes àquele cliente, os *sockets* para comunicação são abertos e a mensagem de “CLIENT_CREATED” é enviada para o *Web Service*, Figura 32. Caso não exista mais uma posição vaga na lista de clientes, é retornada a mensagem de “CANT_CONNECT”.

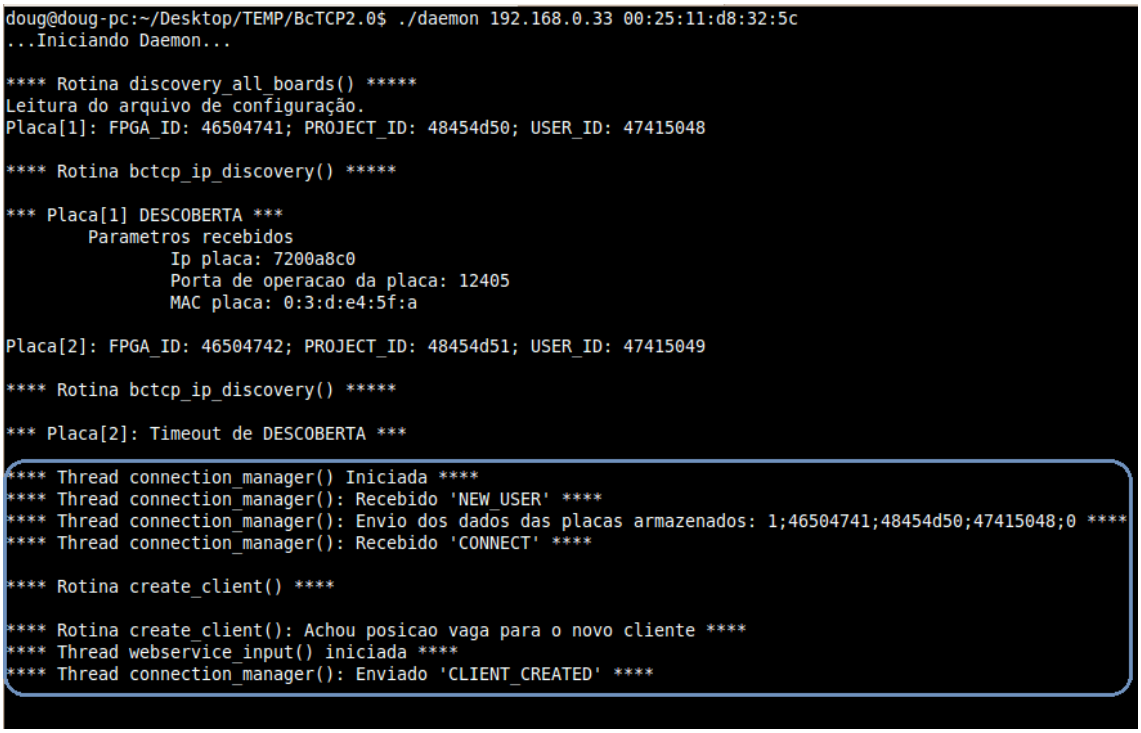

Figura 32 - Resposta do *daemon* informando que realizou a conexão com sucesso.

Depois da criação das *threads* e da abertura dos canais de comunicação, o *daemon* aguarda a sinalização de que o *upload* do arquivo foi realizado com sucesso. Essa sinalização ocorre quando o *Web Service* envia a mensagem “FILE_OK”, Figura 33, destaque 1. Nesse ponto o *software* começa a ler as linhas do arquivo a ser enviado para o dispositivo. Em um primeiro momento, o *daemon* realiza a leitura do *burst* e do tamanho dos dados, Figura 33, destaque 4. A seguir, ele realiza a leitura de uma linha por vez do arquivo. O conteúdo dessa linha é encapsulado em um pacote BCTCP e é enviado para o dispositivo. Essa seqüência repete-se até que todas as linhas do arquivo sejam lidas na sua totalidade.

Observa-se ainda que é realizado o controle do fluxo de dados. Isso ocorre através do número de seqüência dos pacotes. Além disso, também é realizada a verificação das *flags* enviadas pelo módulo de *hardware*, Figura 33, destaque 3.

Paralelamente, enquanto o módulo de *hardware* processa os dados recebidos e, eventualmente, todas as linhas do arquivo ainda não foram lidas e enviadas para o FPGA, o *Web Service* pode enviar requisições para saber informações sobre o status do processamento. Essas requisições são recebidas pela *thread webservice_output*. Essa *thread* verifica se o processamento foi finalizado, através da *flag* "END_PROCESS". Caso esse ainda não tenha sido finalizado, a *thread* envia a mensagem de "NO" para a página, Figura 33, destaque 5. Quando a requisição for realizada e o processo finalizado, o retorno para o *WEB Service* será "YES", Figura 33, destaque 2.

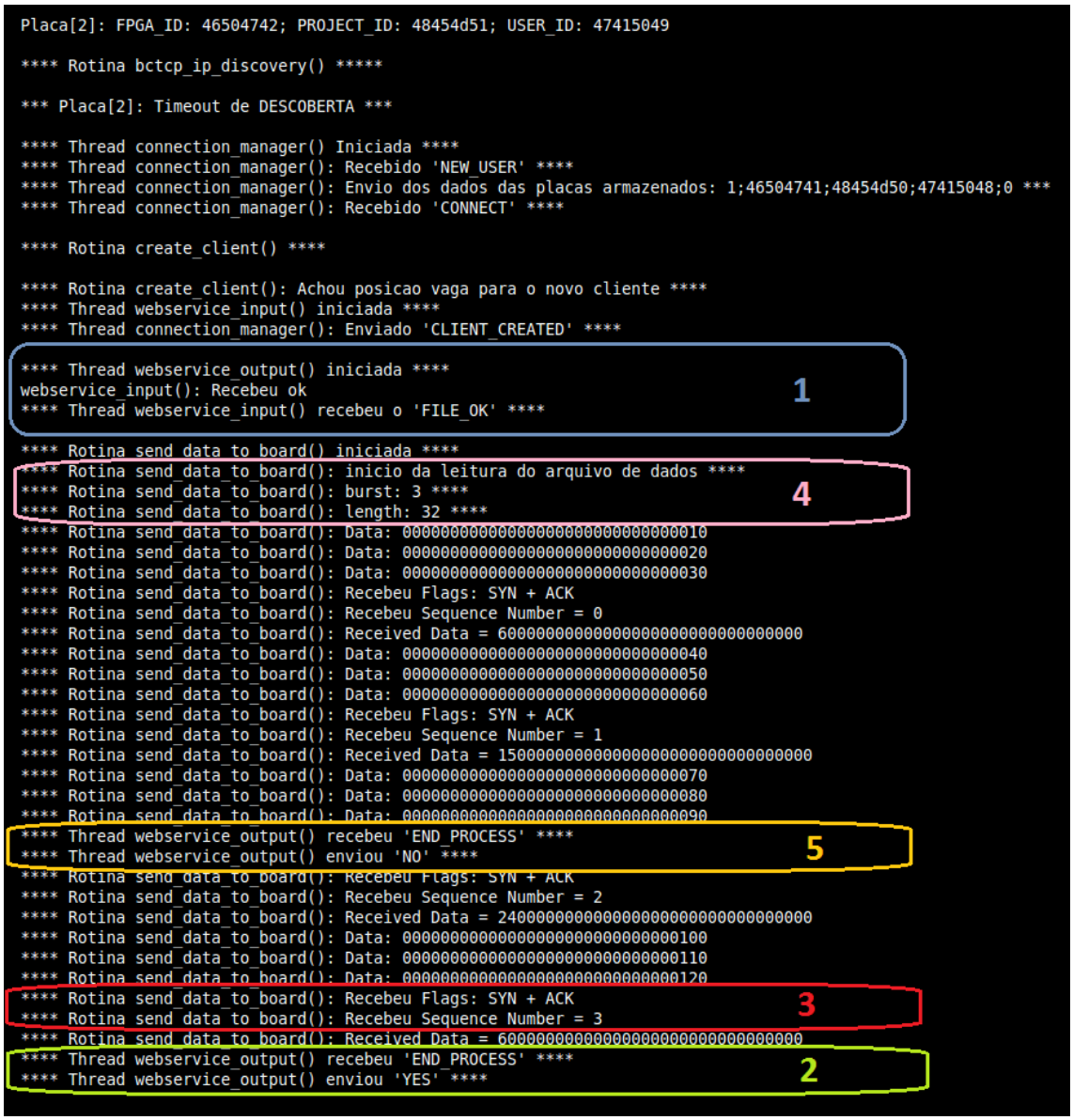

Figura 33 - Processamento dos pacotes.

5.3 Validação Funcional do Simulador

A validação do simulador ocorreu com o mesmo teste realizado nas duas validações anteriores. O *software* do simulador foi executado no sistema operacional Linux Ubuntu. Para executar o simulador, foi utilizado o seguinte comando:

./simulador <mac_sim> <ip_sim> <PROJECT_ID> <FPGA_ID> <USER_ID> <ip_daemon>

Assim, o simulador foi configurado para ser o módulo de *hardware* identificado pela primeira linha do arquivo ilustrado na Figura 15. O FPGA_ID é o "*48454d50*", o PROJECT_ID é o "*47415048*" e o USER_ID é o "*47415048*".

Em primeiro lugar, o simulador fica aguardando uma requisição de conexão. Para isso, é aberto um *socket* UDP para monitorar a porta do protocolo *IP Discovery*. Após esse canal de comunicação ser aberto, é aguardado um pacote do protocolo de descoberta de IP. Quando esse é recebido, o simulador responde com os seus parâmetros de rede. Nesse caso, o endereço IP e o endereço MAC do hospedeiro de onde está sendo executado o simulador, Figura 34, destaque 1.

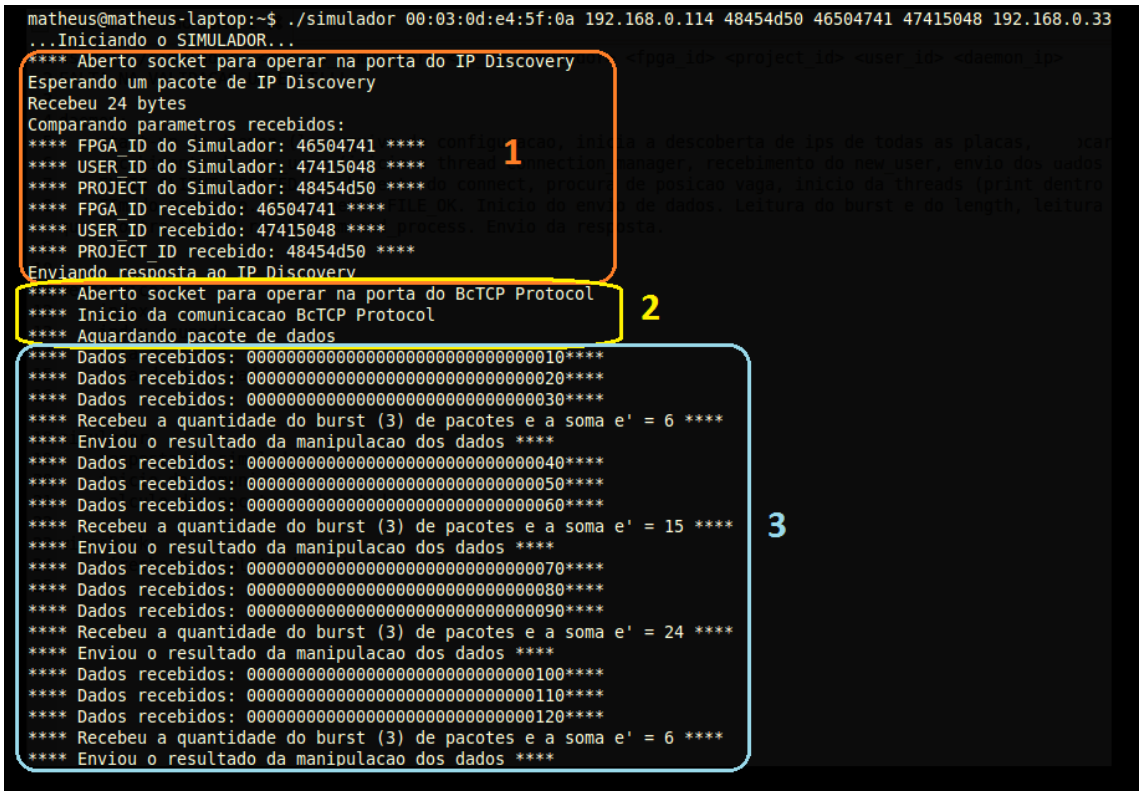

Figura 34 - Processo de IP *Discovery* e processamento de pacotes recebidos.

Após o processo de descoberta, o simulador passa a ser controlado pelo *daemon* e a operar com o protocolo BCTCP. Para isso, o *socket* aberto para atuar na porta do *IP Discovery* é fechado e um novo *socket* é criado, agora monitorando os pacotes da porta do BCTCP, Figura 34, destaque 2.

Após essa etapa, o simulador, aguarda pacotes do *daemon* para iniciar o processamento. Quando esses pacotes são recebidos, o simulador realiza a soma de todos os bytes recebidos e armazena em uma variável. Isso se repete até que o valor do *burst* seja atingido. Ao alcançar esse valor, o simulador envia o conteúdo armazenado na variável onde foi somado o conteúdo de todos os pacotes recebidos. O cálculo e o envio desse pacote são exemplificados na Figura 34, destaque 3.

5.4 Validação Funcional do *Hardware* de Aplicação no Ambiente de Simulação

Para a validação do *hardware* de aplicação, foi utilizado o mesmo cenário da validação anterior. Com o objetivo de ilustrar a comunicação do FPGA com o *daemon*, foi utilizado o *software Wireshark*. Esse aplicativo permite a captura de pacotes da rede.

O *Wireshark* foi executado na máquina onde o *daemon* foi iniciado. Isso foi realizado com o objetivo de capturar os pacotes trocados entre o módulo de *hardware* e o *daemon*. Para capturar esses pacotes foi aplicado um filtro no aplicativo, Figura 35, destaque 6. Esse filtro corresponde às portas dos protocolos *IP Discovery* e BCTCP.

Primeiramente é iniciado o processo de descoberta de módulos. Para isso, o *daemon* envia um pacote *broadcast* para rede, Figura 35, destaque 1. Esse pacote contém o ID do módulo a ser descoberto, Figura 35 - Troca de pacotes *daemon* ↔ Módulo de *Hardware*. Quando o módulo recebe esse pacote, ele verifica que o ID do pacote corresponde ao seu identificador. O módulo envia então um pacote diretamente para o *daemon* contendo os parâmetros de rede, no caso endereço IP e endereço MAC, Figura 35, destaque 2. Esse pacote ainda contém a *flag* de SYN ativa. Quando o *daemon* recebe esse pacote, ele responde com um pacote com as *flags* de SYN e de ACK ativas. Nesse ponto, é estabelecida a conexão, Figura 35, destaque 3.

Após o estabelecimento da conexão, é iniciado o envio dos pacotes de dados do *daemon* para o módulo. Esse recebe os pacotes e realiza o processamento, de acordo com o *hardware* do usuário descrito no 3.1. A cada 3 pacotes recebidos pelo módulo, um de resposta é enviado para o *daemon*, Figura 35, destaque 4.

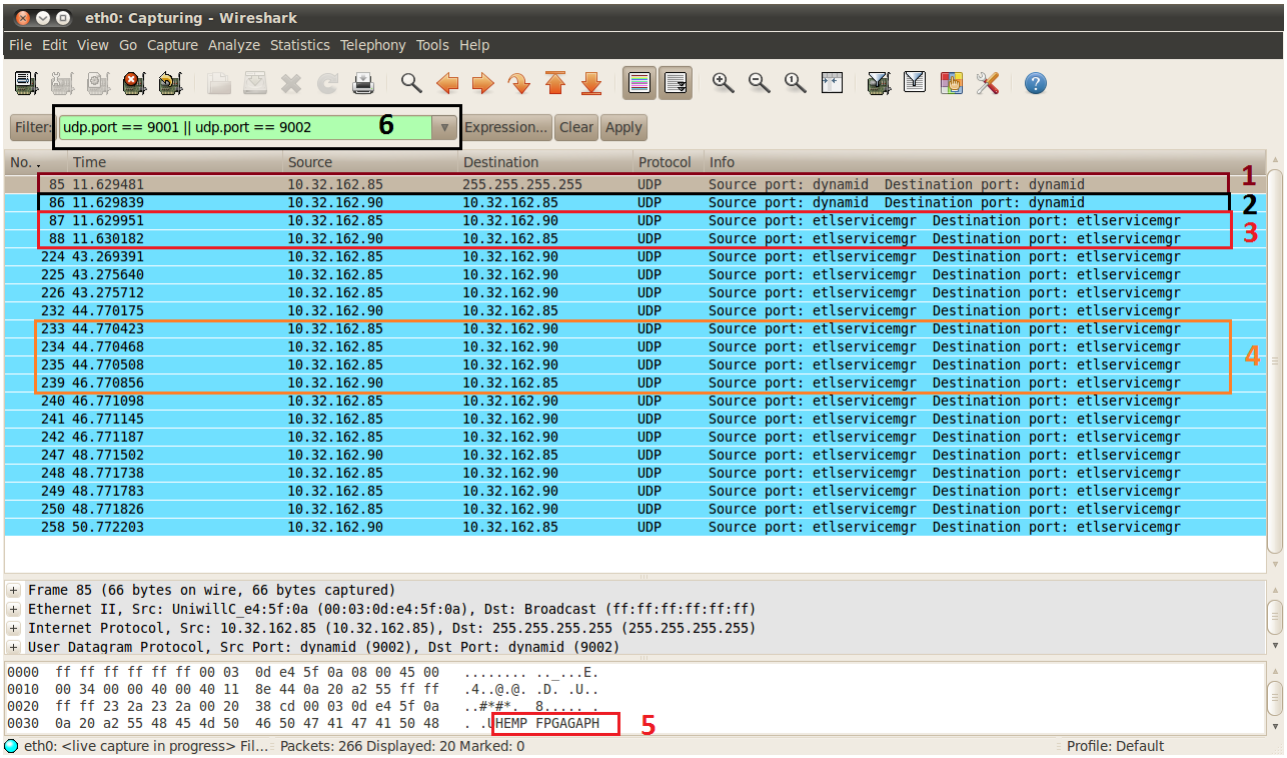

Figura 35 - Troca de pacotes *daemon* ↔ Módulo de *Hardware*.

6 Conclusão e Trabalhos Futuros

Este trabalho teve início com o estudo do protocolo BCTCP. Este protocolo é responsável pela troca de informações entre um módulo de hardware e um computador hospedeiro. O intuito deste estudo foi permitir a implementação de uma ferramenta que, baseada neste protocolo, possibilite realizar a comunicação de módulos de hardware através da rede Ethernet. Estudou-se também a possibilidade de comunicação de páginas web e um daemon executando em um servidor.

Com o estudo realizado, apresenta-se um mecanismo no nível de software, capaz de realizar o acesso e controle de projetos em FPGA utilizando para tal o protocolo BCTCP. Este mecanismo é composto por um software daemon e por páginas web, que se comunicam entre si por meio de sockets AF_UNIX.

Com o término deste trabalho, é perceptível que os objetivos iniciais foram quase que em sua totalidade atendidos. Como objetivo principal, esperava-se que o software pudesse executar em um sistema operacional Linux, operando juntamente com um servidor web, a fim de realizar o controle de vários FPGAs simultaneamente, através da Intranet ou Internet. Este objetivo foi alcançado. Como objetivo secundário, buscava-se a prototipação em hardware do protocolo BCTCP, juntamente com uma aplicação. O intuito desse novo módulo respondendo o protocolo era ampliar os casos de testes. Este objetivo foi atendido parcialmente, através de um simulador que responde como se fosse um FPGA na rede. Além disso, o conjunto de aplicações (daemon e páginas web) apresenta como ponto positivo a facilidade de configuração e facilidade no envio/recebimento de dados.

A maior dificuldade encontrada durante a realização deste trabalho foi a busca por um modo eficiente de realizar a comunicação entre o daemon implementado e as páginas web que são visualizadas pelo usuário. Com um maior estudo da linguagem PHP, pode-se concluir que essa atenderia a demanda e poderia contribuir para que a comunicação fosse realizada toda em nível de servidor, por meio de sockets, entregando ao usuário somente os resultados das requisições.

Acredita-se que este trabalho contribuiu para a formação dos autores, pois através dele foi possível consolidar vários conceitos a respeito de desenvolvimento de software, bem como aprofundar conhecimentos sobre redes de computadores e seus protocolos. Também se podem colocar em prática as habilidades adquiridas durante o curso de Engenharia de Computação.

Como trabalhos futuros, é possível listar melhorias e testes em cinco frentes: melhorias no daemon, testes mais exaustivos, prototipação do protocolo em mais FPGAs, integração com

diferentes plataformas e também execução do software em um servidor para acesso através da Internet.

Com relação ao daemon, é necessária a criação de um mecanismo onde o daemon possa realizar nova leitura do arquivo de configuração que contém os FPGAs disponíveis na rede, caso seja modificado algum parâmetro, sem que o serviço seja reiniciado e/ou que pare de realizar a comunicação. Como solução, pode ser criada uma rotina contemplando esta nova descoberta de FPGAs. Esta rotina deve ser chamada sempre que o daemon receber uma interrupção por meio de sinais, utilizados para comunicação com aplicações. Outra atividade para melhoramento deste daemon, trata-se de direcionar todos os logs para um arquivo. Dessa forma é possível manter um histórico de suas atividades.

Referente à execução de aplicações e monitoramento de resultados por meio web, se faz necessário mais testes. Estes testes incluiriam desde o upload de um arquivo contendo maior quantidade de dados, até o controle de diversos dispositivos ao mesmo tempo por parte do daemon.

Para maiores testes e validações, é indicado a prototipação do protocolo BCTCP em outros FPGAs. A realização desta etapa consiste na geração do CoreGen para a interface MII específica e a adaptação de clock e sinais internos para serem compatíveis com o FPGA escolhido. Também é necessário que a aplicação de usuário responda o protocolo.

Com a finalidade de obter uma maior flexibilidade para os arquivos de dados a serem destinados ao FPGA, é possível a integração com diferentes plataformas, tais como a Hermes e a Atlas. Isso irá possibilitar que após a execução e obtenção de arquivos providos destas aplicações, estes sejam direcionados ao FPGA por meio da aplicação web desenvolvida.

Após estas etapas, é de grande valia a integração do software daemon ao servidor que se encontra no GAPH/PUCRS. Com esta integração é possível o acesso a qualquer dispositivo controlado pelo daemon através da Internet, pois este servidor já apresenta DMZ, isolando assim a Intranet da Internet.

7 Referências Bibliográficas

[APA11] Apache Friends - Xampp. Disponível em **http://www.apachefriends.org/pt_br/xampp.html**. Acesso em Novembro 2011.

[ALB01] Albuquerque, F. "TCP/IP - Internet, Protocolos & Tecnologias", Axcel Books: Rio de Janeiro, 2001, Terceira Edição, 362p.

[ALT11] Altera Corporation, "10/100/1000 Mbps Ethernet MAC". Disponível em **http://www.altera.com/products/ip/iup/ethernet/m-mtip-1000mbps_fulldup_ethermac.html**. Acesso em Agosto de 2011.

[ALT11b] Altera Corporation "Cyclone III FPGA: Industrial Applications", Disponível em **http://www.altera.com/products/devices/cyclone3/mkts-apps/cy3-mkts-ind.html**. Acesso em Agosto 2011.

[BRI11] Brito, D. "Criação de Sites na Era da Web 2.0", Brasport, 2011, Primeira Edição, 222p.

[COS10] Costa, C. M. "Sistemas operacionais: Programação Concorrente com Pthreads", Edipucrs: Porto Alegre, 2010, 211p.

[DRO97] Doms, R. "Dynamic Host Configuration Protocol. RFC 2131". Bucknell University, Lewisburg, USA, 1997. Disponível em **http://www.ietf.org/rfc/rfc2131.txt**. Acesso em Setembro 2011.

[FIE99] Fielding, R. " Hypertext Transfer Protocol -- HTTP/1.1". June, 1999. Disponível em **http://www.ietf.org/rfc/rfc2616.txt**. Acesso em Novembro 2011.

[FRE11] FreeBSD Handbook "Daemons, sinais e controle de processos". Capítulo 3. UNIX Básico. Disponível em **http://doc.fug.com.br/handbook/basics-daemons.html**. Acesso em Novembro 2011.

[ICE11] ICEB "Implicações do Uso dos Protocolos HTTP e TCP no Desenvolvimento de Apliacações para Internet". Disponível em **http://www.iceb.ufop.br/decom/prof/tiago/disciplinas/2006/sistDist/trabalhos/httpXtcp.htm**. Acesso em Novembro 2011.

[ISA09] Isabel, S.M.; Alberto, C.D. "Internet Configurable Distributed Domotic Network", IEEE Latin-American Conference, LATINCOM '09, 2009, 4p.

[NIE04] Niederauer, J. "Desenvolvendo Websites com PHP", Novatec Editora: São Paulo, 2004, 272p.

[NOL08] Nollet, V.; Avasare P.; Eeckhaut H.; Verkest D.; Corporaal H. "Run-Time Management of a MPSoC Containing FPGA Fabric Tiles", IEEE Transactions On Very Large Scale Integration Vlsi Systems (2008), vol: 16-1, pp: 24-33.

[POS80] Postel, J. B. "User Datagram Protocol. RFC 768". Information Sciences Institute, Marina del Rey, California, USA, 1980. Disponível em **http://tools.ietf.org/pdf/rfc768.pdf**. Acesso em Setembro 2011.

[POS81] Postel, J. B. "Transmission Control Protocol. RFC 793". University of Southern California, Information Sciences Institute, Marina del Rey, California, USA, 1981. Disponível em: **http://tools.ietf.org/pdf/rfc793.pdf**. Acesso em Setembro 2011.

[REI09] Reinbrecht, C. R. W., Scartezzini, G.; Rosa T. R. "Desenvolvimento de um ambiente de execução de aplicações embarcadas para a plataforma multiprocessada HeMPs",

Trabalho de Conclusão de Curso, Engenharia de Computação, PUCRS, 2009. Disponível em **http://revistaseletronicas.pucrs.br/ojs/index.php/graduacao/article/viewFile/6046/4353.** Acesso em Agosto 2011.

[RNP97] Araujo, J. G. R. "O Desenvolvimento de Aplicações WEB", Boletim bimestral sobre tecnologia de redes produzido e publicado pela RNP – Rede Nacional de Ensino e Pesquisa, Vol. 1, Num. 5. 03 de Outubro de 1997. Disponível em **http://www.rnp.br/newsgen/9710/n5-3.html**. Acesso em Novembro 2011.

[SIL11] Silva, J.S. "Infraestrutura para controle de projetos em FPGAs através do Protocolo Ethernet", Trabalho de Conclusão de Curso, Engenharia Elétrica, PUCRS, 2011, 10p.

[SIL11b] Silva, M.S. "HTML5 - A Linguagem de marcação que revolucionou a web", Novatec Editora: São Paulo, 2011, 320p.

[TAN03] Tanenbaum, A. S. "Redes de Computadores - Tradução da 4ª Edição Americana", Campus Elsevier, 2003, 968p.

[TAN09] Tanenbaum, A. S.; Woodhull A.S. "Sistemas Operacionais Modernos - Projeto e Implementação", Bookman: Porto Alegre, 1999, Segunda Edição, 344p.

[TIL11] Tilera Corporation, "Technology". Disponível em **http://www.tilera.com/technology**. Acesso em Setembro 2011.

[XIL10] Xilinx Inc "Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v4.8", UG240 April 19, 2010. Disponível em **http://www.xilinx.com/support/documentation/ip_documentation/v4_emac_gsg240.pdf**. Acesso em Setembro 2011.

[XIL11a] Xilinx Inc. "Getting Started". Disponível em **http://www.xilinx.com/company/gettingstarted/index.htm**. Acesso em Setembro 2011.

[XIL11b] Xilinx Inc, "Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide UG194 (v1.10). Fevereiro, 2011. Disponível em **http://www.xilinx.com/support/documentation/user_guides/ug194.pdf**. Acesso em Setembro 2011.

[XIL11c] Xilinx Inc. "ML401/ML402/ML403 Evaluation Platform - User Guide". UG080 (v2.5) May 24, 2006. Disponível em **http://www.xilinx.com/support/documentation/boards_and_kits/ug080.pdf**. Acesso em Setembro 2011.