Rodrigo Neri de Souza

Plataforma de Comunicação Ethernet para dispositivos embarcados em FPGAs da Xilinx

São José – SC

Agosto / 2010

Rodrigo Neri de Souza

Plataforma de Comunicação Ethernet para dispositivos embarcados em FPGAs da Xilinx

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. André Vaz da Silva Fidalgo, Dr. Eng.

Co-orientador:
Prof. Mário de Noronha Neto, Dr.

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – SC

Agosto / 2010

Monografia sob o título *"Plataforma de Comunicação Ethernet para dispositivos embarcados em FPGAs da Xilinx"*, defendida por Rodrigo Neri de Souza e aprovada em 11 de agosto de 2010, em Florianópolis, Santa Catarina, pela banca examinadora assim constituída:

Prof. Evandro Cantú, Dr. Eng.
IFSC

Prof. Gustavo Ribeiro da Costa Alves, Dr. Eng.
ISEP-IPP

Prof. Eraldo Silveira e Silva, M. Eng.
IFSC

Prefiro a rebeldia que me confia como gente,
e que jamais deixou de provar que o ser humano é maior
dos que os mecanicismos que o minimizam.
Paulo Freire

Agradecimentos

Ao professor do IFSC, Golberi de Salvador Ferreira e à Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação do IFSC, Maria Clara Kaschny Schneider, pelo empenho em tornar realidade o intercâmbio internacional de alunos do IFSC.

Ao professor do ISEP, Gustavo Ribeiro da Costa Alves, pelo apoio incondicional oferecido à mim e aos meus colegas de intercâmbio, abrindo-nos a porta do ISEP.

Ao professor do ISEP, André Vaz da Silva Fidalgo, pela oferta do projeto e pela confiança depositada em mim.

Ao professor do IFSC, Mario de Noronha Neto, por toda orientação oferecida, mesmo antes deste projeto, acreditando sempre na minha capacidade.

Ao discente IFSC, Marcelo Rodrigo Andriolli, por todo apoio técnico incondicional fornecido.

Ao professor do IFSC, Eraldo Silveira e Silva, que ao orientar o discente Marcelo Rodrigo Andriolli, acabou indiretamente me orientando.

Ao professor Jorge Pereira, Diretor do IFSC, campus São José, ao professor Volnei Velleda Rodrigues e aos funcionários do campus São José, em especial à Nina, à Meri, à Janete e à Caroline, por todo apoio em minha viagem à Portugal.

À professora, Consuelo Aparecida Sielski Santos Reitora do IFSC, pelo empenho para tornar o IFSC uma Instituição de Ensino de alta qualidade e excelência.

Ao excelentíssimo senhor Presidente da República do Brasil, Luiz Inácio “Lula”da Silva, pois nunca na história deste país foi feito tanto pela educação.

À minha nova família em Portugal, chamada de JorgeTown, a qual mostrou-me o quanto é grande este mundo, e que apesar disso, pôde estar na palma das minhas mãos, ou na casa de um português atrapalhado.

Ao meu grande amigo, “Portuga”, Hugo. Valeu Hugo!

À todos os meus amigos, pelo total apoio. Valeu Arian! Valeu Cleiber!

À minha família, meus avós, meus pais, meu irmão e à namorada dele, dentre outros que

me apoiaram completamente na empreitada da realização deste projeto. Obrigado pai e mãe, por toda educação que tenho hoje.

Em especial, quero agradecer à minha namorada por todo o apoio, paciência, generosidade, amizade, força e orientação, sem ela nada disso teria sido possível. Amot!

À Deus, que deve gostar realmente de mim.

Resumo

O projeto em questão aborda a implementação de uma plataforma de comunicação *ethernet*, para dipositivos sintetizados na lógica combinacional dos FPGAs da Xilinx. Essa plataforma fornece a estes dispositivos a capacidade de serem controlados via rede de computadores, bem como, enviar e receber dados.

A plataforma é flexível, fornecendo serviços e protocolos variados da camada TCP/IP, com intuito de oferecer várias possibilidades de comunicação via rede *ethernet*. A modulariedade da plataforma permite que vários dispositivos, sintetizados na lógica combinacional do mesmo FPGA, possam utilizá-la, a fim de atender dispositivos em conformidade com a norma IEEE1149.1. Tais dispositivos são implementados no Laboratório de Investigação em Sistemas de Teste (LABORIS), do Instituto Superior de Engenharia do Porto, onde todo o projeto foi desenvolvido.

A plataforma é baseada em um sistema embarcado, também sintetizado no FPGA, com o Petalinux como sistema operacional, e com o *hardware* desenvolvido no XPS, utilizando o microprocessador *softcore* Microblaze.

Abstract

This project deals the implementation of a ethernet communication platform for devices synthesized in a combinational logic of Xilinx FPGA. This platform provides, through computer network, a path to control these divices, as well as to send and receive data.

The platform is flexible and it provides a lot of services and protocols of TCP/IP stack, in order to provide various possibilities of ethernet communication. The platform modularity allows a series of devices synthesized in a FGPA use the platform, as devices in accordance with the standard IEEE1149.1. Such kind of devices are implemented at Laboratório de Investigação em Sistemas de Teste (LABORIS) of Instituto Superior de Engenharia do Porto, Portugal, where whole platform was developed.

The platform building is based on embedded system, synthesized in the FPGA. The embedded system hardware, developed at XPS, has a MicroBlaze softcore microprocessor, where Petalinux, the embedded operating system, runs.

Acrônimos

ARP	*Address Resolution Protocol*
CI	*Circuito Integrado*
CRAMFS	*Compressed ROM File System*
DDR	*Double-Data_Rate*
DHCP	*Dynamic Host Configuration Protocol*
DLMB	*Data-side Local Memory Bus*
FIFO	*First In First Out*
FPGA	*Field Programmable Gate Array*
FTP	*File Transfer Protocol*
GPIO	*General Purpose Input/Output*
HDL	*Hardware Description Language*
HTTP	*Hypertext Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
ILMB	*Instruction-side Local Memory Bus*
IOCTL	*Input/Output Control*
IP	*Internet Protocol*
IP *(core)*	*Intellectual Property core*
LMB	*Local Memory Bus*
LUT	*Look-Up Table*
MAC	*Media Access Control*
MMU	*Memory Management Unit*
MSB	*Most Significant Bit*
P2P	*Point-to-Point*
PHY	*Physical layer*
PLB	*Processor Local Bus*
RAM	*Random Access Memory*
ROM	*Read Only Memory*
SDK	*Software Development Kit*
SDRAM	*Synchronous Dynamic Random Access Memory*

SSL	*Secure Socket Layer*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UART	*Universal Asynchronous Receiver/Transmitter*
UDP	*User Datagram Protocol*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
XCL	*Xilinx Cache Link*
XPS	*Xilinx Platform Studio*

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Ao passo que o mundo se torna mais dinâmico e as informações são requisitadas em tempo real, as redes de computadores e a *internet* estão cada vez mais presentes, a fim de atender a demanda ao acesso rápido, em tempo real e em qualquer lugar de qualquer tipo de informação. Com isso é muito bem vindo, e muitas vezes fundamental, que um dispositivo eletrônico, que possui e/ou gera informações, possa acessar uma rede de computadores para, a partir dessa rede, ser controlado remotamente, bem como transmitir e receber dados.

Uma função para o controle remoto de dispositivos eletrônicos é a manutenção remota destes. Entende-se manutenção remota como a execução ou suporte de tarefas de manutenção em um sistema tecnológico a partir de uma localização remota, utilizando técnicas de comunicação modernas. Como exemplos temos a verificação periódica de parâmetros ou desempenho do sistema, gestão de configurações, teste e diagnóstico de problemas de *software* ou *hardware* e a atualização de componentes do sistema. Infra-estruturas de suporte ao teste em conformidade com a norma IEEE 1149.1 (JTAG) (IEEE, 2001; WIKIPEDIA, 30 jul. 2010), podem atuar como componentes de interface entre o equipamento de teste e os dispositivos a testar. Mesmo que esses tipos de infra-estruturas tenham sido originalmente desenvolvidas para suporte ao teste de interligações em placas de circuito impresso, as suas aplicações atuais apresentam normalmente diversas funcionalidades adicionais. É frequente a sua utilização para teste funcional de componentes eletrônicos, reconfiguração e ajuste de parâmetros e diversas outras funções utilizadas normalmente em tarefas de manutenção e teste. Idealmente, um sistema de manutenção remota permite a um utilizador remoto, geralmente um especialista na área de teste e manutenção, recolher informações e controlar componentes do sistema de forma a executar as tarefas pretendidas.

A manutenção remota de sistemas eletrônicos é hoje em dia uma metodologia muito utilizada em diferentes áreas. A idéia não é recente, pois já existem soluções comerciais a pelo menos 15 anos em áreas como a indústria aeroespacial por exemplo. A crescente utilização da internet e a sua integração nos processos industriais implicou no aparecimento de redes *Ethernet* (TANENBAUM, 2002; WIKIPEDIA, 25 jul. 2010) ou derivadas em diferentes áreas

do processo produtivo e em particular em componentes de automação. Existem atualmente a tecnologia, a normalização e as condições econômicas para o desenvolvimento e utilização de soluções eficientes de manutenção remota para diferentes áreas de negócio e com um segmento de mercado potencialmente grande.

Em termos práticos, a manutenção implica o teste, reconfiguração e a eventual reparação dos componentes sob manutenção. Quando estamos falando de componentes eletrônicos, é frequente que o teste seja realizável, utilizando infra-estruturas em conformidade com a norma IEEE1149.1, comumente designadas como infra-estruturas *Boundary Scan*(WIKIPEDIA, 21 may. 2010) ou JTAG.

As áreas de aplicação para monitoração e manutenção remota são extremamente abrangentes. Graças à crescente implantação de redes de computadores, é hoje tecnicamente possível estabelecer uma ligação de dados entre dois sistemas eletrônicos em praticamente qualquer ponto do globo, podendo variar a solução tecnológica específica usada (com fios, sem fios, via satélite, etc).

Pelo menos três entidades se beneficiam deste tipo de manutenção: os fabricantes dos equipamentos; os departamentos de manutenção; e o cliente final. Os benefícios econômicos podem ser consideráveis e incluem a redução de custos com deslocamentos ou transporte de materiais, a redução do tempo de instalação, reparação e possibilidade de redução dos custos globais com pessoal. Adicionalmente, são possíveis serviços mais completos, pois o tempo de reação é muito menor e pode ser automatizado. A monitoração de funcionamento e desempenho pode ser permanente, melhorando o conhecimento do sistema e permitindo a detecção precoce de problemas.

No caso específico dos controladores *Boundary Scan* com porta *Ethernet*, o objetivo depende das capacidades do sistema alvo, pois a interface com a infra-estrutura é normatizada, e as instruções e funcionalidades que podem ser utilizadas são definidas pela arquitetura do sistema sob manutenção/teste. Pode ser limitado ao teste de componentes e interligações utilizando instruções obrigatórias, ou incluindo funcionalidades avançadas como o acesso a registros internos, controle direto, leitura/escrita de memórias, depuração de microprocessadores, ou quaisquer outras funcionalidades suportadas pela infra-estrutura de teste acessível via porta JTAG.

O controlador em si poderá acrescentar algumas capacidades adicionais como o acesso simultâneo a diversos componentes, a memorização intermediária de instruções ou resultados, o ajuste de taxas de transmissão de dados, dentre outras.

Utilizando uma estrutura *Ethernet*, o componente de comunicação permite a interligação com redes convencionais e industriais. Sendo a taxa de transferência de dados, atualmente disponível, adequada à aplicações de teste convencionais.

A utilização deste conjunto de tecnologias normatizadas permite na prática o acesso remoto de forma simplificada a um controlador de teste, utilizável no teste e manutenção de sistemas com componentes em conformidade com a norma IEEE 1149.1, os quais representam uma percentagem considerável dos componentes eletrônicos disponíveis no mercado.

Existem atualmente, diversos controladores *Boundary Scan* com porta *Ethernet*, mas o recurso para componentes programáveis permitiria uma maior flexibilidade com algumas vantagens potenciais. O controlador pode ser adaptado ao problema tecnológico específico que se pretende resolver, podendo serem ajustados o *software* e *hardware* de forma a obter uma solução adequada em termos técnicos e econômicos. Simultaneamente, o equipamento de teste pode ser atualizado em conjunto com eventuais modificações do sistema testado ou modificações dos parâmetros e objetivos de teste.

Atualmente, não se encontra disponível nenhum módulo de lógica programável estável (comercial ou acadêmico), que permita o controle remoto via *Ethernet* de controladores em conformidade com a norma IEEE1149.1. Existem alguns componentes separados que poderiam ser parte de um módulo desse tipo, mas tanto quanto foi possível determinar não existe referência a uma solução que tenha sido validada num cenário operacional.

Este problema mostra uma lacuna nas pesquisas vigentes sobre manutenção remota de dispositivos eletrônicos, gerando portanto, a motivação para este trabalho.

1.1 Objetivos

Implementar uma pilha TCP/IP (TANENBAUM, 2002; WIKIPEDIA, 16 jun. 2010), de forma microprocessada, no FPGA da Spartan-3E *Starter Kit* da Xilinx, com o intuito de criar um canal de comunicação com o FPGA (MEYER-BAESE, 2007; WIKIPEDIA, 30 jul. 2010), através da porta *ethernet*. Esse canal de comunicação permitirá acessar e controlar remotamente outros dispositivos sintetizados na lógica combinacional do FPGA, bem como enviar e receber dados. É objetivo secundário, mas não obrigatório, que essa pilha TCP/IP possa ser implementada em outros modelos de FPGAs da Xilinx sem muitas modificações.

2 *Plataforma de Comunicação*

Com o intuito de tornar possível a comunicação de um dispositivo, sintetizado na lógica combinacional do FPGA, com um computador é necessário um agente que desempenhe as funções dos protocolos requeridos das camadas TCP/IP. O agente teria a função de receber dados do dispositivo de teste e enviá-los através da rede *ethernet* a um determinado computador, bem como, receber informações via *ethernet* de um computador e enviá-las ao dispositivo de teste. Para atender os requisitos desse projeto é fundamental que este agente seja, juntamente com o dispositivo de teste, sintetizado na lógica combinacional do mesmo FPGA. De agora em diante, o agente será chamado de plataforma, visto que o agente pode ser composto no FPGA de alguns dispositivos de *hardware* interligados por um barramento, bem como, possuir componentes de *software*. A figura 2.1 exemplifica superficialmente a arquitetura do modelo de comunicação proposto entre um computador e um FPGA, no exemplo a Spartan-3E *Starter Kit* que possui o FPGA da Xilinx XC3S500E Spartan-3E. Observer que *Ethernet* PHY é o circuito integrado (CI) que faz a interface com o meio analógico da *Ethernet*.

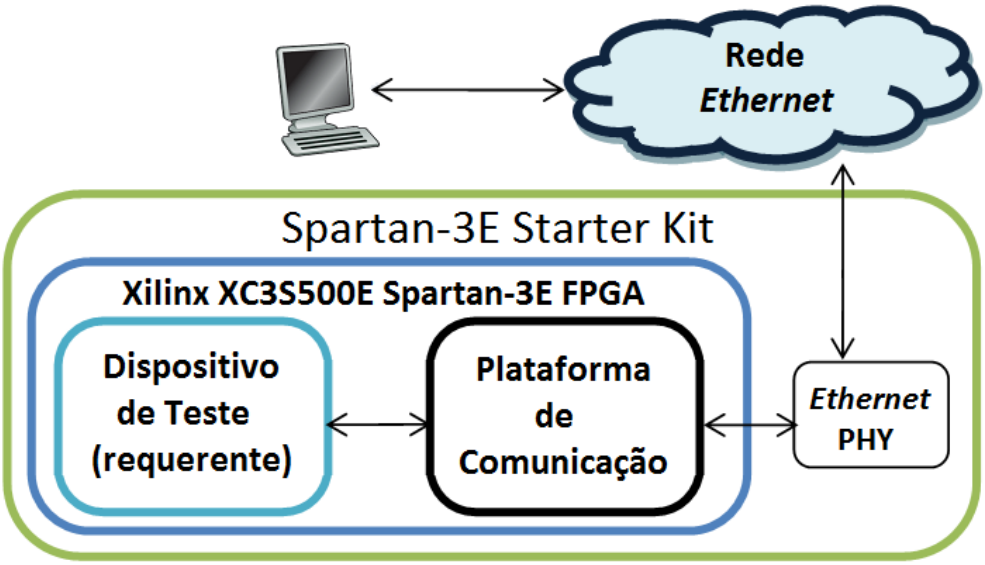

Figura 2.1: Modelo de comunicação remota entre o FPGA e o computador.

2.1 Requisitos da Plataforma de Comunicação

Partindo da função que a plataforma deve executar, foram traçadas as metas para desenvolver uma plataforma genérica que atenda, além do dispositivo de teste, outros dispositivos quaisquer, que chamaremos de dispositivo requerente. A intenção é que a plataforma possua disponível mais recursos de comunicação em rede, principalmente da camada aplicação, e não fique limitada aos requeridos pelo dispositivo de teste, que por ventura podem futuramente mudar, tentado dessa forma não limitar a capacidade de comunicação do dispositivo requerente.

A plataforma deve permitir que os dispositivos requerentes possam ser conectados a ela de uma maneira simples e eficaz, sendo necessário pouca ou nenhuma modificação no sistema de comunicação do dispositivo requerente.

Na plataforma, deve existir um componente responsável por manipular as informações recebidas, do dispositivo requerente, e formatar as informações de tal maneira que seja entendida pelo protocolo da camada aplicação, utilizado na comunicação pretendida. A partir de agora essa componente será chamada de tradutor.

A implementação desse tradutor depende do sistema de comunicação do dispositivo requerente, pois para manipular o dado recebido do dispositivo é necessário saber como esse dado é formatado. A implementação também depende do protocolo da camada aplicação, pois os dados devem estar em um formato conhecido pelo protocolo que o acessa. Assim a plataforma deve permitir que esse tradutor seja implementado de forma simples e eficaz, visto que o seu funcionamento varia de acordo com o dispositivo requerente, bem como, o tipo de comunicação pretendida.

Outro fator a ser considerado, é que a plataforma não deve ocupar demasiadamente a lógica combinacional do FPGA, pois é primordial que exista espaço considerável para sintetizar o dispositivo requerente. Além disso, é muito bem vindo que a plataforma seja passível de ser implementada nos vários modelos de FGPAs da Xilinx.

A última característica importante é que a plataforma seja estável e confiável, que funcione ininterruptamente por centenas de horas de maneira eficaz, sem erros que comprometam seu funcionamento.

2.2 Formas de implementar a Plataforma de Comunicação

Nessa seção sera mostrado algumas maneiras de implementar a plataforma de comunicação entre o dispositivo requerente e a rede de computadores.

2.2.1 HDL

Uma alternativa de implementar a plataforma de comunicação é sintetizar no FPGA as funcionalidades da plataforma, através de uma linguagem de descrição de *hardware* HDL. Contudo fazendo uma análise mais cautelosa, fica visível a grande complexidade de implementar os protocolos da pilha TCP/IP em HDL, visto que é necessário entender por completo o funcionamento de cada protocolo. Como o número de protocolos é grande, e estes são complexos, esta solução se torna inviável a nível de recursos humanos e tempo disponível.

Apesar disso, existe um projeto da Universidade de Queensland, Austrália (UNIVERSITY OF QUEENSLAND, 16 jul. 2001), que poderia ser utilizado. O projeto chamado de *VHDL IP Stack*, é a implementação da pilha TCP/IP em VHDL. Entretanto, há limitações no *VHDL IP Stack* como a taxa de transmissão *ethernet* limitada a 10 Mbit/s e a ausência de vários protocolos fundamentais, TCP, HTTP, DHCP, entre outros, o que limita os recursos de comunicação em rede, visto que a implementação dos protocolos ausente não é viável. Dessa forma foi descartada qualquer tentativa de implementar os protocolos da pilha TCP/IP e, consequentemente, a Plataforma de Comunicação em HDL.

2.2.2 Sistema embarcado

Para esse projeto foi escolhido implementar a plataforma de comunicação, através de um sistema embarcado (SASS; SCHMIDT, 2010), sintetizado no FPGA, dessa maneira os protocolos da pilha TCP/IP e as outras funções necessárias seriam implementados de forma microprocessada através de *software*. A implementação de um sistema embarcado envolve componentes de *hardware* e *software*. A seguir será analisado os componentes de *hardware* e *software* de um sistema embarcado em FPGA, justificando a escolha segundo os requisitos da plataforma de comunicação.

O *hardware* do sistema embarcado

O componente de *hardware* principal de um sistema embarcado é o microprocessador. Existem disponíveis vários tipos de microprocessadores passíveis de serem implementados em FP-

GAs da Xilinx como o ZPU (OPENCORES, 2010c), MB-Lite (OPENCORES, 2010a), Wishbone High Performance Z80 (OPENCORES, 2010b), dentre outros. Contudo, um sistema embarcado não é composto somente de um microprocessador, é composto de barramento, memória RAM, memória ROM e outros dispositivos como controlador de rede MAC e UART. Integrar todos esses dispositivos dentro de um FPGA e montar o sistema embarcado é uma tarefa complexa que requer alto nível de conhecimento e tempo para ser executada, não sendo viável ao projeto em questão.

A Xilinx possui uma ferramenta de desenvolvimento de sistemas embarcados chamada de *Xilinx Platform Studio* XPS, que permite de forma interativa criar um sistema embarcado com a configuração desejada. A manipulação do XPS é constante, independente do modelo de FPGA da Xilinx, e simples ao ponto de ser possível implementar o *hardware* de um sistema embarcado somente com conhecimentos fundamentais do funcionamento de sistemas embarcados. Além disso, a ferramenta foi desenvolvida para lidar com todos os modelos de FPGAs da Xilinx que estão aptos a abrigarem um sistema embarcado, bem como, de acordo com a configuração, otimizar a ocupação de recursos e lógica combinacional do FPGA. Isso atende aos requisitos da plataforma de ocupação de lógica e abrangência dos FPGAs da Xilinx.

A ferramenta utiliza os *Intellectual Property cores* (IP *core*), dispositivos de *hardware* em HDL, da própria da Xilinx, como o microprocessador *softcore* Microblaze (MEYER-BAESE, 2007; WIKIPEDIA, 13 fev. 2010), o barramento PLB, o controlador de rede MAC *Ethernet Lite MAC* (EMACLITE), bem como, dispositivos externos conectados ao FPGA, como memória RAM e memória *Flash*, para montar todo o *hardware* do sistema embarcado.

Um IP *core* muito importante presente na ferramenta é o *General Purpose Input/Outpu* (GPIO), que possui um número configurável de portas de entrada e saída de dados, nas quais podem ser conectadas um dispositivo qualquer sintetizado no FPGA. Para isso, basta somente configurar as portas do GPIO de tal maneira que atenda as portas do dispositivo. O processo de configuração do GPIO e suas portas é simples, pois segue o mesmo conceito de configuração do XPS. O GPIO está conectado no barramento do sistema embarcado, assim o microprocessador consegue acessar as portas do GPIO e, consequentemente, o dispositivo conectado a essas portas. Isso atende, na parte de *hardware*, o requisito de simples integração entre a plataforma de comunicação e o dispositivo requerente.

Os sistemas embarcados implementados pelo XPS tendem a serem estáveis e confiáveis, considerando que, além de ser a ferramenta oficial, o XPS é comercializado a anos pela Xilinx, sendo utilizado largamente pelos seus clientes. A maturidade e consistência da ferramenta atendem o requisito de estabilidade e confiabilidade da plataforma de comunicação.

O *Software* do Sistema Embarcado

O sistema embarcado executa suas funções de acordo com o *software* processado no microprocessador. Para um sistema embarcado executar as funções da plataforma de comunicação, proposta nesse projeto, é necessário um *software* que execute as funções cabíveis da pilha TCP/IP e gerencie a comunicação com o dispositivo requerente.

A seguir será explicitado algumas formas de implementar o *software* do sistema embarcado para esse projeto.

- Pilha TCP/IP Standalone

  Considerando um sistema embarcado implementado com o XPS, o *software* pode ser criado utilizando a ferramenta da Xilinx conhecida como *Software Development Kit* SDK, que provê um ambiente de criação de *softwares*, em linguagem C/C++, para a arquitetura dos microprocessadores da Xilinx. Entretanto, programar a pilha TCP/IP é desnecessário visto que existe disponível em código fonte aberto a pilha TCP/IP *standalone* implementada em C, voltada para microprocessadores embarcados, (por exemplo, uIP TCP/IP (UIP, 2010) , lwIP (LWIP, 2010)). Contudo a pilha *standalone* necessita do respectivo *driver* do controlador de rede MAC do sistema embarcado (UIP, 6 ago. 2007).

  O *software* de controle do sistema poderia ser composto, superficialmente falando, de uma pilha TCP/IP, com respectivo *driver*, do tradutor e de um *driver* para o acesso as funcionalidades do GPIO, *driver* esse que necessitaria ser desenvolvido. Entretanto, essa composição torna mais complexo a implementação do tradutor, visto que requer conhecimentos mais aprofundados sobre a pilha TCP/IP e sistemas embarcados, bem como do *driver* do GPIO. Isso não atende, na parte de *software*, o requisito de simples integração entre a plataforma de comunicação e o dispositivo requerente. Assim, essa solução para o *software* do sistema embarcado foi descartada.

- Sistema Operacional Embarcado

  A outra solução possível é utilizar um sistema operacional que possua nativamente a pilha TCP/IP e os *drivers* necessários para controlar o sistema embarcado. Esse sistema operacional deve suportar a arquitetura do microprocessador utilizado no sistema.

  Um sistema operacional que atende a esse requisito é o Petalinux (PETALOGIX, 2010c), uma distribuição comercial de Linux mantida pela Petalogix (PETALOGIX, 2010h) para microprocessador em FPGAs da Xilinx, que suporta o microprocessador *softcore* MicroBlaze e o *hardcore* PowerPC 405 e 440 (PETALOGIX, 2010f). O Petalinux possui

drivers para vários IP *cores* da Xilinx, como *drivers* para o GPIO e o controlador de rede EMACLITE (PETALOGIX, 2010c). Além disso o Petalinux possui integração com o *Embedded Development Kit* EDK, que é a suíte de desenvolvimento de sistemas embarcados da Xilinx onde está presente o XPS (PETALOGIX, 2010c).

Na parte de rede o Petalinux é muito bem regrado, possuindo vários protocolos da pilha TCP/IP como ARP, TCP, UDP, ICMP, DHCP, e serviços como servidor HTTP, servidor *Telnet*, servidor SSL, servidor FTP, dentre outros que podem ser adicionados ou removidos de maneira simples e intuitiva através do configurador do Petalinux. Isso atende perfeitamente a disponibilidade de recursos de comunicação em rede requerida pela plataforma de comunicação.

O Petalinux possui um *script* que permite adicionar a sua lista de comando um programa desenvolvido para *user space*. Dessa forma é possível criar um programa em C para *user space* que recebe e envia dados do GPIO e adicionar esse programa ao Petalinux, utilizando o *script* referido como compilador cruzado. Isso atende o requisito de implementação simples e eficaz do tradutor da plataforma de comunicação.

Apesar do Petalinux ser uma distribuição comercial, a Petalogix, tem um parceria universitária no qual doa o Petalinux para universidades interessadas em utilizá-lo de forma acadêmica.

Com todos essas características o Petalinux foi escolhido como a solução de *software* do sistema embarcado para esse projeto.

2.3 Cenário de Implementação da Plataforma de Comunicação

A implementação da plataforma de comunicação depende do dispositivo requerente, visto que é necessário configurar as portas do GPIO de tal maneira que permita a conexão entre a plataforma e o dispositivo requerente. Dado esse fato, foi estipulado um cenário genérico em que o dispositivo requerente é composto por um dispositivo repassador, que chamaremos de *device*, e duas FIFOs uma de entrada e outra de saída de dados. O *device* recebe os dados da FIFO de entrada e, quando acionado uma chave, repassa a FIFO de saída. Esse dispositivo requerente foi implementado em VHDL e o seu código consta no Apêndice A como *requester*.

O alvo de implementação desse projeto é a Spartan-3E *Starter Kit* comercializado pela Digilent e que possui o FPGA da Xilinx XC3S500E Spartan-3E.

2.3.1 A FIFO do dispositivo requerente

A FIFO base do requerente é capaz de armazenar até 7 bytes dados, ou seja, 7 palavras de 8 bits. A Figura 2.2 mostra o diagrama de bloco da FIFO do dispositivo requerente, com as portas referentes a entrada de dados a esquerda da figura e as portas referentes saída de dados a direita. Na tabela 2.1 consta as portas da FIFO, bem como, sentido, largura e função da porta.

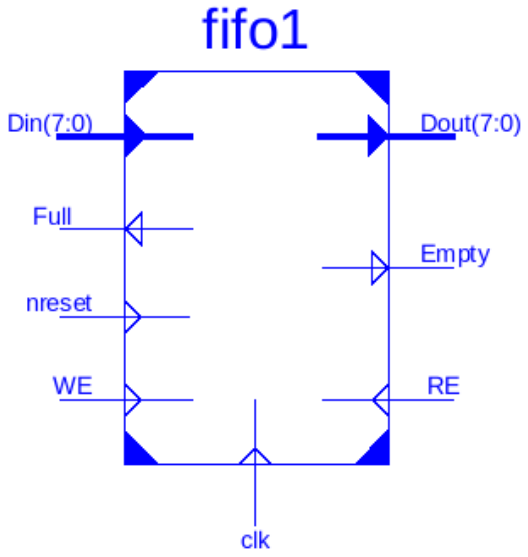

Figura 2.2: FIFO.

Porta	Sentido	Largura	Função
Din	Entrada	8 bits	Entrada de dados
Full	Saída	1 bit	Fica em 1 quando a FIFO está cheia
WE	Entrada	1 bit	Quando em 1, armazena na FIFO o valor em Din
Dout	Saída	8 bits	Saída de dados
Empty	Saída	1 bit	Fica em 1 quando a FIFO está vazia
RE	Entrada	1 bit	Quando em 1, escreve em Dout o devido dado de saída
nreset	Entrada	1 bit	Quando em 0, reinicia a FIFO, limpando a sua memória
clk	Entrada	1 bit	Clock1 para a FIFO

Tabela 2.1: Portas da FIFO.

Para escreve na FIFO, basta colocar o valor 1 na porta de entrada WE que o valor presente na porta Din será armazenado na FIFO. Antes de cada tentativa de escrita é necessário verificar, através do porta Full, se a FIFO está cheia ou não. Se estiver cheia, qualquer tentativa de escrita será em vão, pois o dado não será armazenado na FIFO.

O processo de leitura da FIFO é semelhante. Para ler da FIFO, basta colocar o valor 1 na porta de entrada RE que o devido dado de saída estará presente na porta saída Dout. Antes de cada tentativa de leitura é necessário verificar, através do porta Empty, se a FIFO está vazia ou

não. Se estiver vazia, qualquer tentativa de leitura estará equivocada, pois o dado presente na porta saída Dout será o último dado armazenado na FIFO.

A FIFO funciona na subida de *clock*, ou seja, a cada subida de *clock* são verificados os valores das portas de entrada e, de acordo com esses, são definidos os valores das portas de saída. Isso é um possível complicador, que pode invalidar o processo de escrita ou leitura, caso o dispositivo que esteja tentando executar essa ação não a faça no mesmo período de *clock* da FIFO. Por exemplo, se o valor 1 for mantido na porta de entrada WE por 3 períodos do mesmo *clock* da FIFO, o valor presente na porta Din será armazenado 3 vezes na FIFO. Assim um dispositivo que pretende escreve 1 byte na FIFO, e possui portas que demoram 7 períodos, do mesmo *clock* da FIFO, para mudar o seu valor, acabaria escrevendo 7 vezes o mesmo byte na FIFO.

Caso o mesmo dispositivo tentasse ler 1 byte da FIFO, e ela estivesse com 7 bytes armazenados, ele acabaria lendo 1 byte e perderia outros 6, visto que o tempo de mudança dos valores das portas de saída tende a ser o mesmo que o tempo de verificação do valores das portas de entrada.

Considere que e o *clock* da Spartan-3E *Starter Kit* está funcionando a 50Mhz, para assim obter maior desempenho do sistema embarcado, e a FIFO está utilizando esse mesmo *clock* como referência. Nessa situação o GPIO não conseguirá ler e escrever da FIFO corretamente, pois o tradutor, que é responsável por definir e verificar os valores das portas do GPIO, é um programa executado pelo Petalinux. Esse último é processado pelo microprocessador, que, apesar de está utilizando o mesmo *clock*, precisará de vários ciclos de *clock* para executar as funções do tradutor. Funções essas que verificam e alteram os valores das portas do GPIO.

Para sanar esse problema foi desenvolvido um sincronizador em VHDL que possui um sinal de entrada e um sinal de saída. O sinal de entrada é proveniente de uma determinada porta do GPIO, e o sinal de saída é conectado na porta de entrada WE ou RE, dependendo se o objetivo do GPIO é ler ou escrever da FIFO.

O funcionamento do sincronizador é simples, ele mantém o sinal de saída no valor 0 e ao verificar uma subida de borda de um sinal de entrada, troca o valor lógico do sinal de saída para 1, mantendo-o durante um período do mesmo *clock* da FIFO. Ao término desse período de *clock* o valor do sinal de saída retorna a 0. Esse período de *clock* inicia-se sempre na borda de decida do *clock*, para garantir que durante os sinais de subida do *clock*, onde a FIFO verifica os valores das portas de entrada WE e RE, tais valores já estejam definidos. Dessa maneira as porta do GPIO podem demorar quantos períodos de *clock* forem necessários para mudar seu estado, visto que o sincronizador fez com que as funções das portas de entrada WE e RE da FIFO, que

antes eram ativadas pelo valor de entrada naquelas portas iguais a 1, agora sejam ativadas pela borda de subida.

```
1  --Synchronizer of GPIO's WE to FIFO's WE :
2  PROCESS(WE,clk)
3     --This process creates a pulse in WE starts at a falling edge clock and finishes
4     --Notes WE is activated at rising edge clock.
5
6  variable wr : integer range 0 to 1;
7
8  BEGIN
9     if(falling_edge(clk)) then
10       if(wr = 0 and (WE  = '1'))then
11          wr := 1;
12          WE_mid <= '1'; --Set WE_mid to 1
13       elsif((WE  = '0')) then
14          wr := 0;
15          WE_mid <= '0'; --One clock after sets WE_mid to 0
16       else
17          WE_mid <= '0'; --One clock after sets WE_mid to 0
18       end if;
19    end if;
20 END PROCESS;
```

Figura 2.3: Código VHDL do sincronizador.

O sincronizador da maneira que está funcionando não limitará a taxa de bytes escritos e lidos pelo GPIO, entretanto o *clock* de operação da FIFO deve ser igual ou maior que essa taxa, que é o tempo no qual o tradutor levar para definir um valor lógico de uma porta.

O *device*, que é outra componente do dispositivo requerente, opera na mesma frequência da FIFO, então não haverá problema algum para ele ler e escrever da FIFO. Considerando isso, é necessário implementar o sincronizador somente nas portas de entrada WE, da FIFO de entrada, e RE, da FIFO de saída.

A Figura 2.3 mostra o código em VHDL que executa a função de sincronizador, considerando WE o sinal de entrada, clk o *clock* de referência e WE_mid o sinal de saída.

2.3.2 Os módulos do dispositivo requerente

Com o intuito de facilitar a conexão entre as portas dos GPIOs e as das FIFOs, foram criados dois módulos VHDL chamados de GPIO_to_FIFO e FIFO_to_GPIO, com seus diagramas de bloco mostrados nas Figuras 2.4 e 2.5. No primeiro módulo está instanciado a FIFO de entrada, do dispositivo requerente, e implementado o sincronizador, que está conectado a porta de entrada WE da FIFO. Já no módulo FIFO_to_GPIO está instanciado a FIFO de saída, do dispositivo requerente, e implementado o sincronizador, que está conectado a porta de entrada RE da FIFO. Dessa maneira para acessar as FIFOs é necessário conectar-se aos módulos. Com isso,

de agora em diante, será considerado que acessar o módulo GPIO_to_FIFO é acessar a FIFO de entrada, e acessar o módulo FIFO_to_GPIO é acessar a FIFO de saída do dispositivo requerente.

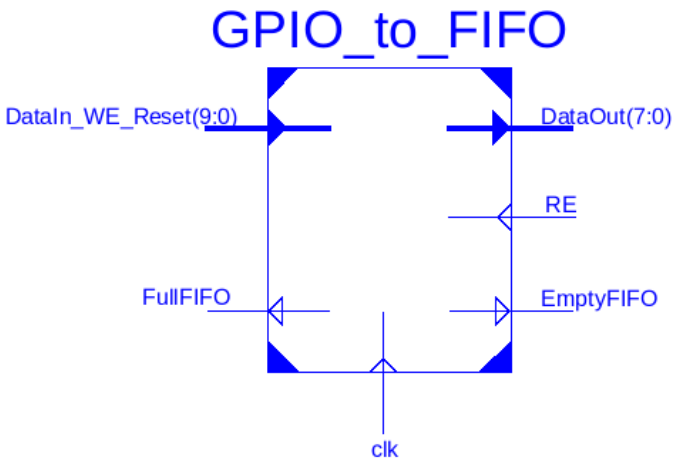

Figura 2.4: GPIO_to_ FIFO.

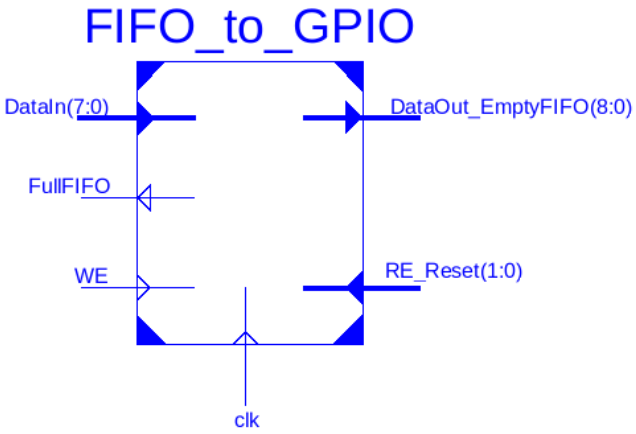

Figura 2.5: FIFO_to_GPIO.

Como as portas do GPIO são agrupadas em vetores lógicos padrão (std_logic_vector) de entrada e saída, as portas dos módulos que necessitam ser conectadas ao GPIO também foram agrupadas em vetores lógicos padrão de saída e entrada, na tentativa de facilitar ainda mais a conexão dos GPIOs aos módulos. As tabelas 2.2 e 2.3 fazem uma correspondência entre as portas da FIFO com as portas dos respectivos módulos, sendo DataIn_WE_Reset, FullFIFO, DataOut_EmptyFIFO e RE_Reset as portas dos módulos conectadas ao GPIO.

É importante notar que tanto a porta nreset da FIFO do módulo GPIO_to_FIFO quanto o do módulo FIFO_to_GPIO, são portas conectadas ao GPIO, ou seja, a plataforma de comunicação que vai controlar o *Reset* dos dois módulos.

FIFO			Módulo GPIO_to_FIFO			
Porta	**Sentido**	**Largura**	**Porta**	**Sentido**	**Largura**	**Correlação com a FIFO**
Din	Entrada	8 bits	DataIn_WE_Reset	Entrada	10 bits	Os 8 primeiros bits são o Din, o 9º bit é WE e 10º bit é o nreset
WE	Entrada	1 bit				
nreset	Entrada	1 bit				
Full	Saída	1 bit	FullFIFO	Saída	1 bit	Todas as portas do GPIO são std_logic_vector, mesmo a de um pino.
Dout	Saída	8 bits	DataOut	Saída	8 bits	
Empty	Saída	1 bit	EmptyFIFO	Saída	1 bit	
RE	Entrada	1 bit	RE	Entrada	1 bit	
clk	Entrada	1 bit	clk	Entrada	1 bit	

Tabela 2.2: Correspondência das portas da FIFO e do módulo GPIO_to_FIFO.

FIFO			Módulo FIFO_to_GPIO			
Porta	**Sentido**	**Largura**	**Porta**	**Sentido**	**Largura**	**Correlação com a FIFO**
Din	Entrada	8 bits	DataIn	Entrada	8 bits	
WE	Entrada	1 bit	WE	Entrada	1 bit	
Full	Saída	1 bit	FullFIFO	Saída	1 bit	
Dout	Saída	8 bits	DataOut_EmptyFIFO	Saída	9 bits	Os 8 primeiros bits são o Dout e 9º bit é o Empty
Empty	Saída	1 bit				
RE	Entrada	1 bit	RE_Reset	Entrada	2 bits	O 1º bit é o RE o 2ª é o nreset
nreset	Entrada	1 bit				
clk	Entrada	1 bit	clk	Entrada	1 bit	

Tabela 2.3: Correspondência das portas da FIFO e do módulo FIFO_to_GPIO.

2.3.3 O *device* do dispositivo requerente

A função do *device*, Figura 2.6, é ler os dados do módulo GPIO_to_FIFO e escrever no módulo FIFO_to_GPIO, segundo o acionamento de uma chave. Toda a vez que a chave é acionada o *device*, antes de executar a função de leitura e escrita, verifica se FIFO_to_GPIO não está cheia e se GPIO_to_FIFO não está vazia, caso uma das duas condições não seja atendia o *device* não executa a sua função.

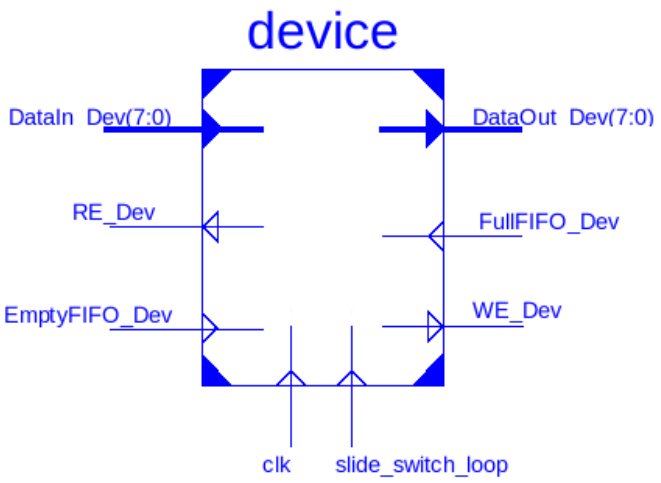

Figura 2.6: *Device*.

A chave é considerada acionada quando essa passa do estado 0 para o 1, ou seja, somente na borda de subida. Dessa maneira a chave no estado 1 não aciona *device*.

O *device* tem uma função meramente ilustrativa, para demonstrar a funcionalidade da plataforma de comunicação. É possível substituir o *device* por outro dispositivo qualquer implementado em VHDL que possua as portas como as especificadas na tabela 2.4, e consiga escrever e ler dos módulos das FIFOs no tempo apropriado. Caso não consiga atender o requisito de tempo é possível implementar o sincronizador do módulo GPIO_to_FIFO na porta RE e o sincronizador do módulo FIFO_to_GPIO na porta WE.

2.3.4 O dispositivo requerente

Para organizar de maneira mais intuitiva a hierarquia dos arquivos no ISE, foi criado um módulo VHDL chamado de *requester* no qual é instanciado e interconectado, como mostra a Figura 2.8, o módulo GPIO_to_FIFO, o módulo FIFO_to_GPIO e o *device*. Assim o módulo VHDL *requester*, mostrado na Figura 2.7, representa o dispositivo requerente.

As portas do *requester*, como mostra a tabela 2.5, são as portas do tipo *Standard Logic*

Porta	Sentido	Largura	Função
DataIn_Dev	Entrada	8 bits	Entrada de dados
RE_Dev	Saída	1 bit	Fica em 1, quando quer receber dados em DataIn_Dev
EmptyFIFO_Dev	Entrada	1 bit	Quando em 1, deixa RE_Dev em 0
DataOut_Dev	Saída	8 bits	Saída de dados
FullFIFO_Dev	Entrada	1 bit	Quando em 1, deixa WE_Dev em 0
WE_Dev	Saída	1 bit	Fica em 1, quando quer enviar os dados em DataOut_Dev
slide_switch_loop	Entrada	1 bit	Na borda de subida da chave, ative a função do device
clk	Entrada	1 bit	Clock para o Device

Tabela 2.4: Portas do Device.

Vector do módulo GPIO_to_FIFO e do FIFO_to_GPIO desenvolvidas para serem conectadas aos GPIOs, bem como o *clock* e o slide_switch_loop, que é chave de acionamento do *device*.

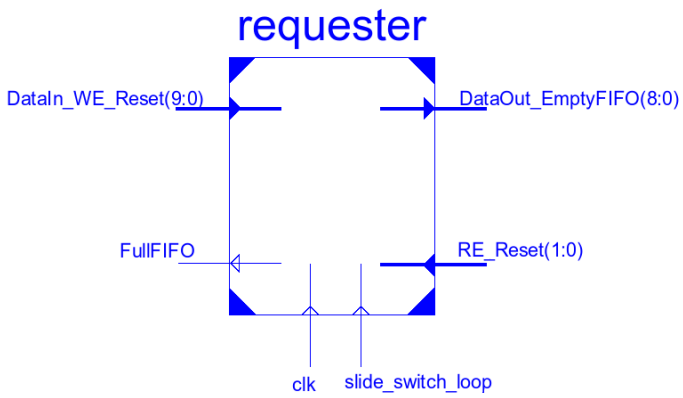

Figura 2.7: *Requester*.

Porta	Sentido	Largura	Instância correspondente
DataIn_WE_Reset	Entrada	10 bits	GPIO_to_FIFO
FullFIFO	Saída	1 bit	GPIO_to_FIFO
DataOut_EmptyFIFO	Saída	9 bit	FIFO_to_GPIO
RE_Reset	Entrada	2 bits	FIFO_to_GPIO
slide_switch_loop	Entrada	1 bit	Device
clk	Entrada	1 bit	GPIO_to_FIFO, FIFO_to_GPIO e Device

Tabela 2.5: Portas do Requester.

2.4 Implementação da Plataforma de Comunicação

A plataforma de comunicação se resume a um sistema embarcado, composto pelo Petalinux e pelo *hardware* implementado pelo XPS. Entretanto o sistema não é totalmente sintetizado dentro do FPGA, outros dispositivos da Spartan-3E *Starter Kit* conectados ao FPGA também

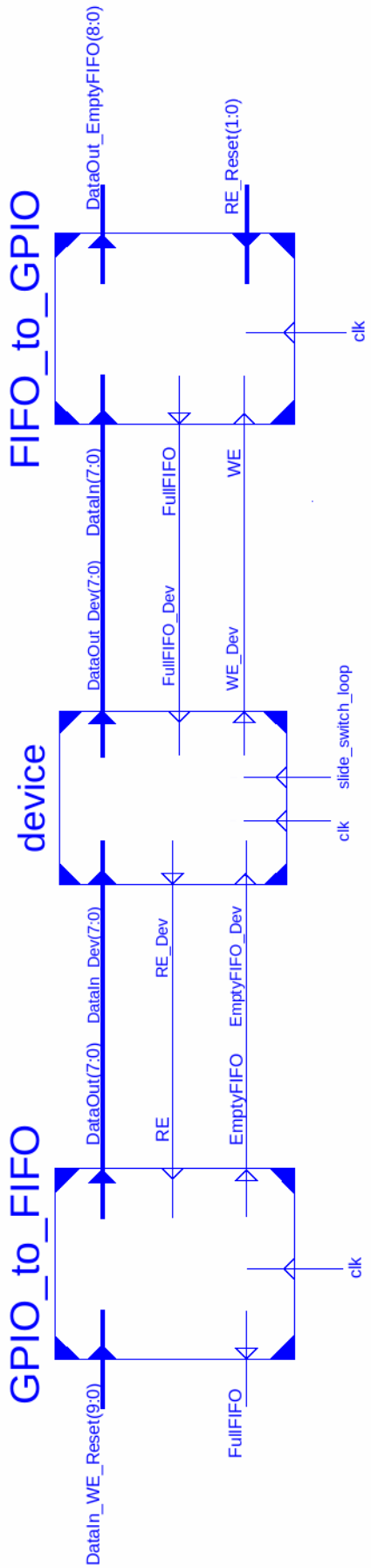

Figura 2.8: Interconexão do *Requester*.

são utilizados para compor o sistema embarcado. A tabela 2.6 mostra os CIs do *kit* que tem relação direta com o sistema embarcado.

Apesar disso a maior parte dos dispositivos do sistema embarcado está sintetizado dentro do FPGA. Muitos deles estão conectados ao barramento PLB (plb_v46 1.04.a). Todos o dispositivos conectados no PLB são escravos no barramento controlado pelo Microblaze que é o mestre. A tabela 2.7 mostra os dispositivos conectados ao PLB.

Nome do CI	**Tipo**	**Função no Sistema Embarcado**
XC3S500E	FPGA	Sintetizar grande parte do sistema embarcado
32MB Micron DDR SDRAM	Memória DDR SDRAM	Memória RAM utilizada pelo Microblaze
16MB Numonyx StrataFlash	Memória *Flash*	Memória ROM onde ficar armazenado a imagem do Petalinux e o *Bitstream*
SMSC LAN83C185 Ethernet PHY	*Ethernet* PHY	Interface com o meio analógico da *Ethernet*

Tabela 2.6: CI's do sistema embarcado.

Nome do Dispositivo	**IP**	**Função no Sistema Embarcado**
microblaze_0	microblaze 7.20.d	Microprocessador do sistema
FLASH	xps_mch_emc 3.01.a	Responsável por escrever e ler na memória *Flash*
Ethernet_MAC	xps_ethernetlite 3.01.a	Controlador de rede MAC
xps_gpio_in	xps_gpio 2.00.a	Responsável por receber dados do FIFO_to_GPIO
xps_gpio_out	xps_gpio 2.00.a.	Responsável por enviar dados ao GPIO_to_FIFO
xps_intc_0	xps_intc 2.00.a	Controlador de interrupção
xps_timer_0	xps_timer 1.01.b	Temporizador do barramento PLB
RS232_DCE	xps_uartlite 1.01.a	Terminal de usuário do sistema embarcado

Tabela 2.7: IPs do sistema embarcado conectados ao PLB.

A tabela 2.8 mostra os dispositivos, também sintetizados dentro do FPGA, que estão conectados diretamente a portas especificas do Microblaze através de um barramento próprio, ou seja, não estão conectados no PLB.

Nome	**IP**	**Barramento**	**Função no sistema embarcado**
DDR_SDRAM	mpmc 5.04.a	XCL	Controlador da memória RAM
dlmb	lmb_v10 1.00.a	DLMB	Interface de controle de dados do lmb_bram
ilmb	lmb_v10 1.00.a	ILMB	Interface de controle de instruções do lmb_bram
mdm_0	mdm 1.00.g	Xilinx P2P	Depurador, não utilizado pelo sistema embarcado

Tabela 2.8: IPs do sistema embarcado conectados ao Microblaze.

Além disso o lmb_bram (bram_block 1.00.a), que é um pequeno bloco de memória RAM sintetizado dentro no FPGA utilizado pelo Microblaze como a memória de arranque, está conectado diretamente ao DLMB e ao ILMB.

Apesar do mdm_0 ser um depurador, e não ser utilizado propriamente pelo sistema embarcado, a sua presença pode ser útil visto que ele possui varias funções, como carregar uma imagem diretamente para a memória RAM.

A Figura 2.9 mostra como está arquitetado na Spartan-3E *Starter Kit* o *hardware* do sistema embarcado, bem como a conexão com o dispositivo requerente.

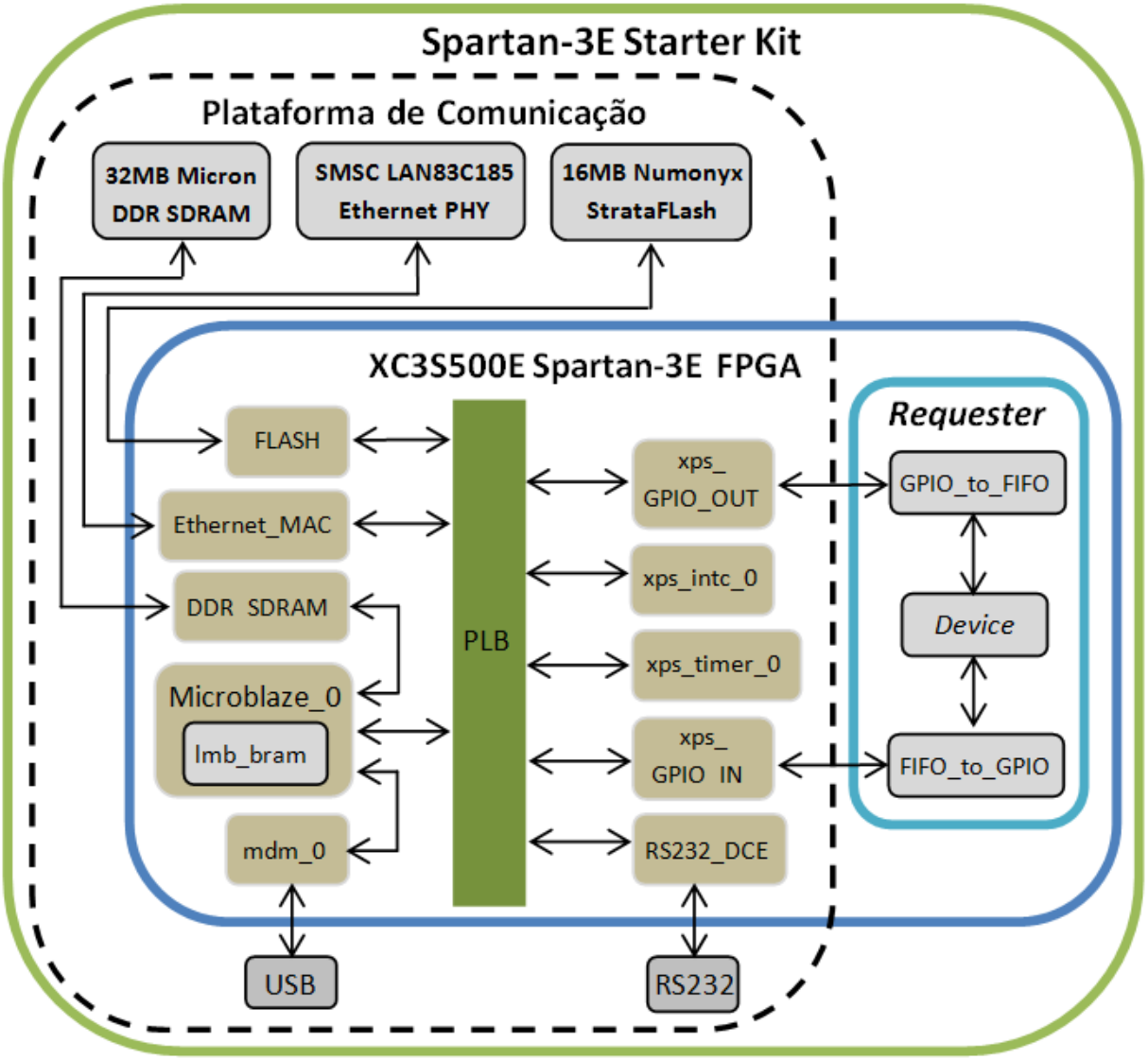

Figura 2.9: Arquitetura do Sistema Embarcado na Spartan-3E *Starter Kit*.

Para conectar o dispositivo requerente as porta do GPIO é necessário importar para o ISE o projeto do sistema embarcado criado no XPS. Assim é possível mapiar as portas do dispositivo requerente com as do GPIO. A referência (YOUTUBE, 6 feb. 2009) mostra como importar um projeto do XPS para o ISE, bem como o apêndice B mostra como implementar um projeto no XPS de tal maneira que possa ser manipulado pelo Petalinux.

2.4.1 Requisitos de *hardware* do Petalinux

O Petalinux suporta todos os dispositivos do sistema embarcado, mas alguns necessitam ser configurados de maneira especifica para o perfeito funcionamento do sistema. Utilizando como referência a documentação da Petalinux disponível em (PETALINUX, 2010) e o arquivo *"system.mhs"* contido no diretório de referências da implementação do Petalinux na Spartan3E500 RevD EDK 10.1 localizado em:

```
petalinux-v0.40-final/hardware/reference-designs/Xilinx-Spartan3E500-
RevD-lite-edk101
```

Os dispositivos microblaze_0, xps_intc_0, xps_gpio_in, xps_gpio_out e RS232_DCE são configurados como explicitado nas seções seguintes.

Configuração do Microblaze

O ponto crucial para utilizar com êxito o Petalinux é a configuração do microprocessador. O Microblaze 7.20.d possui varias configurações suportadas pelo Petalinux. Contudo somente será abordada uma configuração básica do Microblaze sem *memory management unit* (MMU), tentando dessa maneira achar um equilíbrio entre um bom desempenho e a menor ocupação dos recursos do FPGA, visto que a implementação do MMU ocupa bastante lógica combinacional.

A tabela 2.9 mostra os parâmetros de configuração das instruções do Microblaze, e seu estado de utilização, para esse projeto.

O Microblaze é configurado com uma memória *cache* de dados de 2kB e uma memória *cache* de instruções também de 2kB. Nenhuma configuração de exceção é ativada no Microblaze, bem como, não há MMU. O *processor version register* (PVR) é ativado como completo, *full*, e os outros parâmetros a cerca do PVR são definidos com o valor zero em base hexadecimal. Na parte de barramento o *processor local bus* (PLB) é ativado. Considerando que o funcionamento do Microblaze transcende o foco do projeto, não será abordado o que representa os parâmetros configurados. Recomenda-se o guia de referência do Microblaze disponível em (XILINX, 26 out. 2009) para informações mais aprofundadas.

Configuração do controlador de interrupção

O controlador de interrupção é responsável por encaminhar para o microprocessador as interrupções geradas pelos dispositivos conectados ao barramento (WIKIPEDIA, 7 jul. 2010)

Instruções/Otimização	**Estado**
Barrel Shifter	Ativado
Floating Point Unit	Não Ativado
Integer Multiplier	Ativado, Multiplicador de 32 bits
Integer Divider	Ativado
Addtional Machine Status Register Instructions	Ativado
Pattern Comparator	Ativado
Optimize area(with lower instruction throughput)	Ativado

Tabela 2.9: Parâmetros de configuração das instruções do Microblaze.

de acordo com a prioridade estabelecida. O Petalinux exige um controlador de interrupção, pois assim poderá saber quando e qual dispositivo precisa ser gerenciado. Dessa forma é fundamental que a interrupção esteja ativada nos seguintes dispositivos: xps_timer_0, Ethernet_MAC, RS232_DCE, xps_gpio_in e xps_gpio_out. Essa é a respectiva ordem decrescente de prioridade configurada no xps_intc_0.

Configuração do GPIO IN

O xps_gpio_in é o GPIO necessário para conectar o modulo FIFO_to_GPIO a plataforma de comunicação. Através das portas do GPIO o Petalinux vai poder controlar e receber informações do FIFO_to_GPIO.

Para simplificar o processo de utilização do GPIO, será utilizado os seus dois canais. O canal 1 com largura de 9 bits no sentido entrada e o canal 2 com lagura de 2 bits no sentido de saída. O XPS cria um porta do tipo *Standard Logic Vector* de 9 bits para o canal 1 e uma de 2 bits para o canal 2. Na implementação em questão o XPS nomeou essas portas de xps_gpio_in_GPIO_IO_I_pin e xps_gpio_in_GPIO2_IO_O_pin respectivamente, contudo esse nome pode ser alterado. Agora basta mapear as portas, conforme a tabela 2.10, no ISE para conectar o GPIO ao modulo FIFO_to_GPIO.

Canal	**Porta**	**Sentido**	**Largura**	**Porta do FIFO_to_GPIO**
1	xps_gpio_in_GPIO_IO_I_pin	Entrada	9 bits	DataOut_EmptyFIFO
2	xps_gpio_in_GPIO2_IO_O_pin	Saída	2 bits	RE_Reset

Tabela 2.10: Portas do GPIO IN conectadas ao FIFO_to_GPIO.

Configuração do GPIO OUT

Analogamente à subseção configuração do GPIO IN, o xps_gpio_out é o GPIO necessário para conectar a plataforma de comunicação ao modulo GPIO_to_FIFO, que será controlado pelo

o Petalinux.

Também para simplificar o processo de utilização do GPIO, será utilizado os seus dois canais. O canal 1 com largura de 10 bits no sentido de saída e o canal 2 com largura de 1 bit no sentido de entrada.

Considerando a implementação em questão o XPS criou um porta, para o canal 1, do tipo *Standard Logic Vector* de 10 bits, nomeada de xps_gpio_out_GPIO_IO_O_pin. Apesar do canal 2 ser somente um pino, o XPS também criou uma porta do tipo *Standard Logic Vector* de 1 bit, chamada de xps_gpio_out_GPIO2_IO_I_pin. Agora basta mapear as portas, conforme a tabela 2.11, no ISE para conectar o GPIO ao módulo GPIO_to_FIFO.

Canal	**Porta**	**Sentido**	**Largura**	**Porta do GPIO_to_FIFO**
1	xps_gpio_out_GPIO_IO_O_pin	Saída	10 bits	DataIn_WE_Reset
2	xps_gpio_out_GPIO2_IO_I_pin	Entrada	1 bits	FullFIFO

Tabela 2.11: Portas do GPIO OUT conectadas ao GPIO_to_FIFO.

Configuração da RS232

A RS232_DCE é o terminal de usuário do Petalinux. Todos os *softwares* e *scripts* do Petalinux que utilizam a RS232_DCE consideram que sua taxa de transmissão é de 115200 baud (PETALOGIX, 2010g). Dessa maneira a RS232_DCE deve ser sempre configurada com essa taxa.

2.4.2 Implementando o Petalinux

No panorama geral da plataforma de comunicação o Petalinux é responsável por executar parte da pilha TCP/IP e executar o tradutor, que acessa os GPIOs. Assim essas componentes devem ser configuradas e implementadas de acordo com o tipo de comunicação via rede pretendida, bem como de acordo com o dispositivo requerente de plataforma de comunicação. O Petalinux *System Development Kit* é a ferramenta necessária para configurar e compilar o Petalinux para o sistema embarcado alvo. A ferramenta possui, dentre outros recursos, menu de configuração, conhecido em inglês como *menuconfig*, *script* de automatização de rotinas envolvendo configurações e implementações no Petalinux e integração com o XPS da Xilinx.

Nos subcapítulos a seguir são tratados as configurações e implementação das duas componentes. Além disso serão explicitados outros aspectos não diretamente correlacionados com o função da plataforma de comunicação.

Configuração da pilha TCP/IP

O Petalinux possui vários protocolos e serviços da pilha TCP/IP que podem ser adicionados ou removidos através do *menuconfig*. Alguns deles como o TCP, UDP, IP, ICMP, DNS, servidor *web*, servidor para acesso *Telnet* e servidor para transferência de arquivos FTP são adicionados por padrão na compilação cruzada. Contudo, há outros protocolos que precisam ser ativados como o DHCP.

Apesar disso o Petalinux implementa somente uma parte das camadas da pilha TCP/IP. As outras partes são implementadas pelo IP *core* controlador de rede MAC (xps_ethernetlite) e pelo CI de interface com o meio analógico, *ethernet* PHY (SMSC LAN83C185 Ethernet PHY). A tabela 2.12 mostra quais camadas o Petalinux, o controlador de rede MAC e o *ethernet* PHY tem responsabilidade de implementar, bem como, alguns dos protocolos estipulado, segundo o cenário proposto de implementação da plataforma de comunicação.

<table>
<tr><th>Responsável</th><th colspan="2">Camada da Pilha TCP/IP com Subcamada OSI</th><th>Protocolo</th></tr>
<tr><td rowspan="4">Petalinux</td><td colspan="2">Aplicação</td><td>Telnet, HTTP, FTP, DHCP, DNS</td></tr>
<tr><td colspan="2">Transporte</td><td>TCP, UDP</td></tr>
<tr><td colspan="2">Internet</td><td>IP, ICMP</td></tr>
<tr><td rowspan="3">Interface de Rede (Enlace)</td><td>LLC</td><td rowspan="3">Ethernet</td></tr>
<tr><td>Controlador de Rede MAC</td><td>MAC</td></tr>
<tr><td>Ethernet PHY</td><td>Física</td></tr>
</table>

Tabela 2.12: Os dispositivos responsáveis por implemtar as camadas TCP/IP.

A ativação do serviço DHCP cliente está descrita no apêndice B, bem como a configuração de IP estático.

O servidor *web* THTTPD está ativado por padrão no Petalinux, entretanto esse servidor apresentou instabilidade de funcionamento e foi desativado. Dessa forma foi ativado, através do *menuconfig*, outro servidor *web* presente no Petalinux o Boa que funcionou de maneira estável. Para inicializar o Boa durante o arranque do sistema utilizou-se do arquivo em *shell script* de inicialização do servidor *web* THTTPD localizado em:

```
/petalinux-v0.40-final/software/petalinux-dist/romfs/etc/init.d/thttpd
```

O conteúdo do arquivo foi totalmente substituído por:

```
#!/bin/sh
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin
echo "Starting boa: "
boa &
```

Para hospedar um *web* site no Petalinux com o Boa basta colocar os arquivos html (com um arquivo *index.html* como a página principal) e os outros arquivos na pasta:

```
petalinux-v0.40-final/software/petalinux-dist/romfs/home/httpd
```

No apêndice C consta um exemplo de página *"index.html"* com um cliente *Telnet* em *Java Applet*, disponível em (JTA - TELNET/SSH FOR THE JAVA(TM) PLATFORM, 15 set. 2006), e um cliente FTP também em Java Applet, obtido em (RAD INKS (PVT), 2010). Assim fica disponível através de um navegador *web* um cliente *Telnet* e FTP necessário para comunicar e controlar o Petalinux, bem como transferir arquivos.

Para maiores informações sobre as funcionalidades do Boa consultar (BOA, 23 feb. 2005).

Para configurar o servidor DNS basta adicionar os parâmetros de configurações necessários no arquivo *"resolv.conf"* localizado em:

```
/petalinux-v0.40-final/software/petalinux-dist/romfs/etc/default
```

Contudo por padrão o Petalinux não permite a modificação dos arquivos da pasta romfs, a qual é utilizada como a fonte do sistema de arquivos. Dessa forma a alternativa é modificar o arquivo durante o arranque do Petalinux, visto que o sistema de arquivos é somente de leitura. Aproveitando-se do arquivo *thttpd*, o qual é permitido modificações, pode ser adicionado às linhas de comandos necessárias para a modificação do arquivo *"resolv.conf"*.

No caso de ser requerido um *gateway* basta adicionar a aplicação *route* no *menuconfig* e, devido ao sistema de arquivo ser somente leitura, configurar a rota durante a inicialização. Para isso basta adicionar no arquivo *thttpd* os comandos para configurar o *gateway*. Segue o exemplo do arquivo *thttpd* configurado para definir o endereço 193.136.60.25 como *gateway* padrão e os endereços 193.136.60.10 e 193.136.60.2 como servidor DNS:

```
#!/bin/sh
PATH=/bin:/sbin:/usr/bin:/usr/sbin
echo "Defining gateway: "
route add default gw 193.136.60.250
```

```
echo "Defining DNS: "
echo "nameserver 193.136.60.10" >/etc/resolv.conf
echo "nameserver 193.136.60.2"  >>/etc/resolv.conf
echo "Starting boa: "
boa &
```

Há outros serviços como SMTP, Openssl e Asterisk PBX que podem ser adicionados ao Petalinux através do *menuconfig*, entretanto para o cenário proposto de implementação da plataforma de comunicação, os serviços *Telnet*, HTTP e FTP, configurados como servidores, são suficientes.

Implementação do Tradutor

Como explicitado no capítulo 2.1, o tradutor é responsável por receber e enviar dados do dispositivo requerente e formatar esses dados de maneira que seja entendida pelo serviço da camada aplicação utilizado na comunicação. Assim as duas principais funções do tradutor é, através dos GPIOs, enviar e receber dados do dispositivo requerente e formatar os dados para o serviço da camada aplicação.

A fim de facilitar a função de formatar os dados para o serviço da camada aplicação, o tradutor foi desenvolvido, em linguagem de programação C, como um *software* em *user space* e adicionado aos comandos do Petalinux utilizando o *script petalinux-new-app*(PETALOGIX, 2010e)-(PETALOGIX, 2010a), que faz parte do Petalinux *System Development Kit*. Assim os serviços da camada aplicação podem acessar o tradutor facilmente, visto que para enviar ou receber dados basta executar a linha de comando do Petalinux referente ao tradutor, passando os argumentos necessários e tratando os retornos do comando. Arquitetado dessa maneira, o tradutor pode por exemplo ser acessado através de comandos via *Telnet*. No apêndice B é mostrado como utilizar o *script petalinux-new-app*.

Considerando o cenário proposto de implementação da plataforma de comunicação, foram desenvolvidos 6 programas tradutores, cujos nomes representam a linha de comando necessária para executá-los no Petalinux. A tabela 2.13 mostra os programas segundo seus nomes, funções, argumentos a serem passados e os seus retornos.

O tradutor comunica-se com o dispositivo requerente através dos GPIOs. Para acessar o GPIO o tradutor utiliza comandos IOCTL da biblioteca xgpio_ioctl.h, que é a biblioteca em C no linux para o GPIO da Xilinx. Essa biblioteca em C provê uma estrutura, em inglês *struct*, chamada de xgpio_ioctl_data, para representar nos programas tradutores as portas do GPIO.

Nome	Função	Argumento	Retorno
writereset	Reiniciar e limpar memória do GPIO_to_FIFO	Nenhum	Verificação do exito do comando
writefifo	Escreve um dado em hexadecimal na GPIO_to_FIFO	Dado em hexadecimal	Verificação do exito do comando
writefifofile	Escreve dados em hexadecimais contido em um arquivo no GPIO_to_FIFO	Nome do arquivo com os valores hexadecimais.	Se GPIO_to_FIFO estiver cheia, pergunta se deve continuar a tentar escrever
readreset	Reiniciar e limpar memória do FIFO_to_GPIO	Nenhum	Verificação do exito do comando
readfifo	Lê um dado hexadecimal da FIFO_to_GPIO	Dado em hexadecimal	O dado hexadecimal lido
readfifofile	Armazena em um arquivo os dados hexadecimais lidos do FIFO_to_GPIO	Nome do arquivo para armazenar os dados	Se FIFO_to_GPIO estiver vazia, pergunta se deve continuar a tentar ler

Tabela 2.13: Programas tradutores.

Assim, para definir e obter os valores das portas dos GPIOs basta aplicar o comando IOCTL na estrutura referente a porta do GPIO. Nos 6 programas tradutores foram definidas as mesmas estruturas para representar as portas dos GPIOs. A tabela 2.14 associa as estruturas em *softwares* com as portas em *hardware* dos GPIOs.

Software	***Hardware***				
Nome da estrutura	**GPIO**	**Canal**	**Porta do GPIO**	**Sentido**	**Largura**
gpio_in_data_empty	GPIO IN	1	xps_gpio_in_GPIO_IO_I_pin	Entrada	9 bits
gpio_out_RE_Reset	GPIO IN	2	xps_gpio_in_GPIO2_IO_O_pin	Saída	2 bits
gpio_out_data_WE_Reset	GPIO OUT	1	xps_gpio_out_GPIO_IO_O_pin	Saída	10 bits
gpio_in_fullFIFO	GPIO OUT	2	xps_gpio_out_GPIO2_IO_I_pin	Entrada	1 bits

Tabela 2.14: Estruturas em *software* associadas as portas dos GPIOs.

O GPIO IN e o GPIO OUT estão conectados, respectivamente no FIFO_to_GPIO e no GPIO_to_FIFO. Entretanto, estes últimos somente tem a função de sincronizar a FIFO estanciada dentro deles, assim o sua lógica de funcionamento é a mesma da FIFO. Com isso a lógica de operação dos 6 programas é baseada no paradigma de funcionamento da FIFO, descrito no capítulo 2.3.1, bem como na função do programa em si. Os códigos comentados dos 6 programas estão no apêndice D.

Em relação ao desempenho, a velocidade de funcionamento do GPIO dependerá do tempo que o Microblaze leva para processar as instruções IOCTL do programa tradutor. Este tempo não tende a ser constatante, uma vez que o microprocessador pode ter que processar várias tarefas em um sistema operacional, tarefas estas que podem demorar uma quantidade ciclos de

máquina variados.

Outro fator variável no processamento é a ocupação da memória *cache*. Isso implica que a velocidade de leitura e escrita de dados no GPIO não será constante. Considerando que a leitura dos dados do FIFO_to_GPIO é o ponto crítico que pode colocar em risco a capacidade da plataforma de comunicação atender o dispositivo requerente, o programa *readfifofile* foi desenvolvido para ler dados de maneira mais rápida. Para verificar a velocidade de leitura foi criado um cenário de teste no qual o sinal RE presente na porta xps_gpio_in_GPIO2_IO_O_pin do GPIO IN é analisado com um osciloscópio, enquanto o comando *readfifofile* é executado.

A Figura 2.10 mostra que o RE é na maior parte ativado e desativado em períodos em torno de 300μs, contudo houve um intervalo no qual o período para ativação e desativação do RE foi em torno de 500μs. Assim, é possível afirmar somente que o programa *readfifofile* pode chegar a ler dados do FIFO_to_GPIO a uma velocidade de 3,3khz, mas isso não significa que os dados sempre serão lidos a essa frequência.

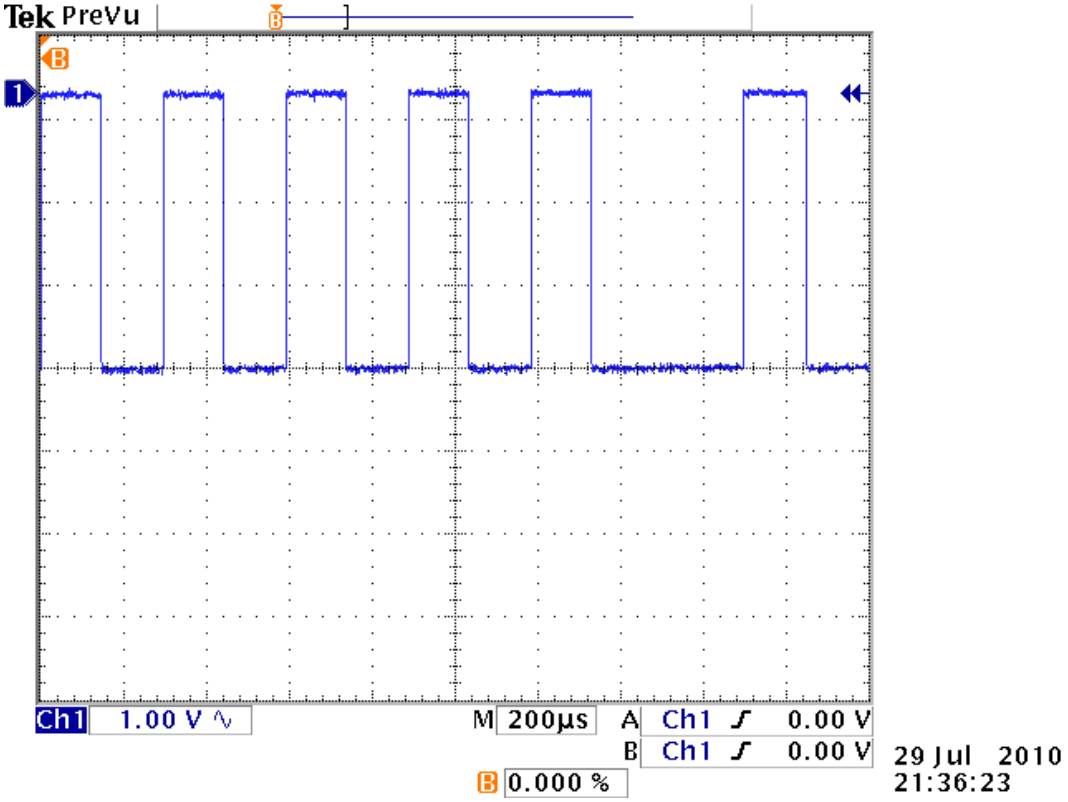

Figura 2.10: Análise temporal do sinal RE enviado ao FIFO_to_GPIO.

Bootloader

Para inicializar o Petalinux é necessário um *Bootloader* para copiar o Petalinux armazenado na memória *Flash* para a memória SDRAM, de onde ele será executado pelo Microblaze.

A Petalogix disponibiliza uma solução de arranque em dois estágios para Petalinux (PETALOGIX, 2010d).

O Primeiro arranque é realizado pelo *FS-Boot*, um pequeno *Bootloader*, que pode ser armazenado em uma BRAM de 8 Kbytes e tem como intuito principal inicializar o *Bootloader* principal. O *U-Boot*, *Bootloader* principal, é responsável por inicializar o Petalinux. Ele possui várias funcionalidades que permitem, por exemplo: manipular uma memória *Flash*, copiando e apagando dados dessa memória; fazer downloads de arquivos via protocolo TFTP; cria um sistema de arquivo NFS, dentre outros recursos (PETALOGIX, 2010b). Contudo, ele é demasiadamente grande quando comparado com o *FS-Boot*, tornando proibitiva a sua utilização em uma BRAM. Visto que o Microblaze inicializa pela BRAM (PETALOGIX, 2010d), justifica-se utilizar um sistema de arranque em dois estágios, para assim obter as funcionalidades de um *Bootloader* robusto e ocupar pouca BRAM. Entretanto, o *FS-Boot* apresentou uma característica inconveniente, somente inicializa se houver uma conexão ativa no terminal de usuário, ou seja, uma conexão com *FS-Boot* através da RS232.

Tentando superar esse inconveniente, utilizou-se nesse projeto o *Bootloader* criado pelo XPS, que não apresentou essa característica, inicializando normalmente, mesmo sem conexão ativa no terminal de usuário, durante o reset da Spartan-3E *Starter Kit*. Entretanto, constatou-se que no caso do *kit* ficar desligado por um período de tempo, o *Bootloader* não inicializa sem uma conexão ativa no terminal de usuário. Não foi possível determinar, em tempo hábil, o porquê desta situação. No apêndice B, está descrito como criar o *Bootloader* no XPS, bem como utilizar o *U-Boot* para guardar a imagem do Petalinux na memória *Flash*.

3 *Resultados Obtidos*

O principal resultado obtido neste projeto foi a plataforma de comunicação, que concede conectividade via rede de computadores aos dispositivos sintetizados dentro do mesmo FPGA. A forma como a plataforma é implementada, descrita no apêndice B, utiliza as ferramentas da Xilinx, tornando-a modular em relação ao modelo do FPGA. A utilização do Petalinux como o sistema operacional embarcado traz uma grande flexibilidade a nível de manipulação da plataforma, bem como dos serviços de comunicação em redes.

No cenário proposto de comunicação, o desempenho da plataforma foi eficiente, mostrando sua grande utilidade. A Figura 3.1 mostra um *website* com um cliente Telnet e um cliente FTP, ambos em Java Applet, conectados à plataforma de comunicação sintetizada em uma Spartan-3E *Starter Kit*. Este *website* está hospedado na plataforma de comunicação e o *kit* está conectado na rede do Instituto Superior de Engenharia do Porto em Portugal, estando a plataforma configurada com um IP público.

Na Figura 3.1 também é possível verificar a utilização, via Telnet, dos comandos tradutores. Assim, foi possível enviar e receber dados via rede de computadores, de um dispositivo sintetizado em um FPGA.

A nível de *hardware*, a principal contribuição é o desenvolvimento de uma plataforma de comunicação, baseada em um sistema embarcado implementado com o XPS. O sistema é baseado no microprocessador *softcore* Microblaze, configurando sem MMU, tentando assim diminuir a ocupação lógica do FPGA. Na Spartan-3E *Starter Kit* a plataforma de comunicação está ocupando 4237 *Slices*, 4238 *Slice Flip Flops* e 5345 LUTs do FPGA XC3S500E Spartan-3E. Além disso, é possível adicionar, no sistema embarcado, GPIOs para conectar outros dispositivos à plataforma de comunicação.

Uma contribuição em *hardware* também muito importante é o sincronizado VHDL, que na prática transforma uma porta ativada pelo nível em uma porta ativada pela borda. Isto auxilia muito na comunicação e na interligação das portas de dois dispositivos com frequências de operações distintas.

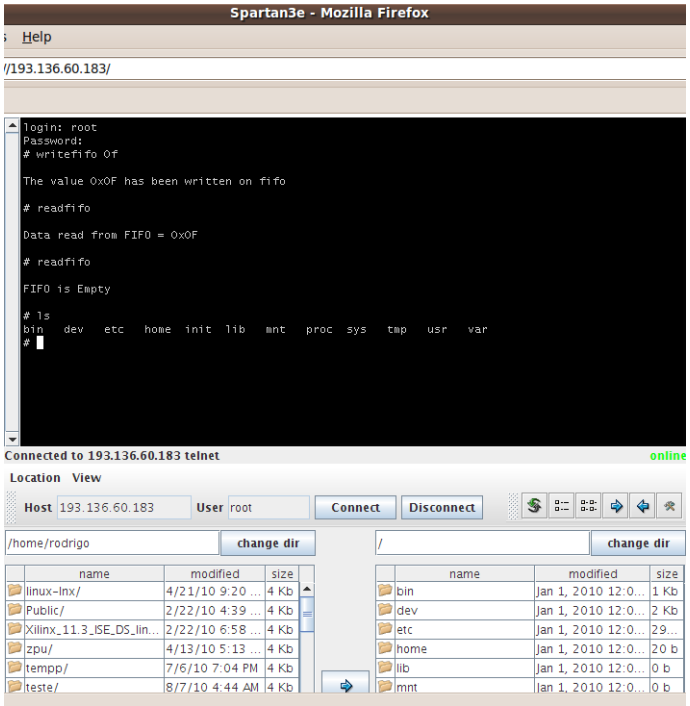

Figura 3.1: Acesso via navegador a plataforma de comunicação.

Ainda em *hardware*, foi implementado um dispositivo em VHDL que, quando acionado por uma chave, lê os dados de uma FIFO e escreve em outra, a fim de testar o funcionamento da plataforma. O dispositivo consta no apêndice A como *Device*.

Na parte de *software*, a contribuição principal é a implementação funcional do Petalinux no sistema embarcado criado pelo XPS, com todos os *Drivers* necessários a este projeto funcionando. Este sistema operacional robusto e confiável traz vários serviços de rede de computadores, bem como flexibilidade na criação de aplicações.

Outra contribuição importante em *software* são os 6 programas tradutores, com funções distintas, implementados em C, que acessam e controlam as portas do GPIO. Estes programas, contidos no apêndice D, desenvolvidos para o *user space*, são invocados como um comando do Petalinux. Desta forma, eles podem ser acessados por serviços da camada aplicação. Na prática isto significa que eles podem ser invocados via rede de computadores.

4 Conclusões

A execução do cenário de implementação da plataforma de comunicação mostrou que a implementação de uma pilha TCP/IP, de forma microprocessada, na Spartan-3E *Starter Kit* da Xilinx, com o objetivo de criar um canal de comunicação com o FPGA através da porta *Ethernet*, foi alcançado. A plataforma de comunicação implementada neste projeto atende de maneira eficaz os dispositivos, sintetizados no FPGAs da Xilinx, que necessitam de comunicação via rede de computadores.

Levando em consideração a robustez e flexibilidade da plataforma, fica claro que ela atende abrangentemente uma gama de dispositivos, tais como vários controladores em conformidade com a norma IEEE1149.1, sintetizados na lógica combinacional do FPGA, uma contribuição importante e de vanguarda nas pesquisas vigentes sobre manutenção remota de dispositivos eletrônicos. Isto é uma demonstração da potencialidade da plataforma.

A manipulação fácil e consistente do XPS torna, praticamente transparente, muitos dos conceitos da implementação do *hardware* de um sistema embarcado, inclusive o modelo de FPGA. Combinando isto à flexibilidade e solidez do Petalinux, cria-se um ambiente simples, porém rico em opções e funcionalidades de implementação de sistemas embarcados. Basear a plataforma de comunicação nestas duas ferramentas, é, de certa forma, transferir um pouco da transparência e da simplicidade de implementação para a plataforma de comunicação.

Quanto a um dispositivo utilizar a plataforma para comunicar-se com uma rede de computadores, existem 3 fatores diretamente relacionados à comunicação em si, a configuração do(s) serviço(s) da pilha TCP/IP, a implementação do tradutor e a configuração do GPIO. Os outros fatores restantes da implementação da plataforma, estão relacionados ao funcionamento da mesma, e não ao funcionamento da plataforma. Assim, ao mesmo tempo que os ambientes de desenvolvimento, das ferramentas que implementam a plataforma, tornam-se mais simples e transparente, a implementação da plataforma torna-se também mais simples e transparente para o usuário.

Seguindo este sentido, resta então simplificar ainda mais a configuração/implementação

dos 3 fatores diretamente relacionados à comunicação entre o dispositivo requerente e as redes de computadores, como por exemplo: automatizar a implementação do tradutor, automatizar o processo de criação da plataforma no XPS e automatizar o processo de compilação do Petalinux, dentre outros.

Contudo, a frequência de operação da GPIO, controlada pelo Petalinux, é um fator problemático para a plataforma quando o dispositivo requerente necessita de comunicação via rede com altas taxas de transferência de dados. Desta forma, com a finalidade de melhorar o desempenho da plataforma, é altamente recomendado pesquisar uma solução, como por exemplo conectar o dispositivo requerente diretamente ao barramento do sistema embarcado.

Apesar disso, a plataforma de comunicação se mostrou muito eficaz e versátil à nível de aplicações e serviços de redes de computadores, sendo extremamente útil para fornecer aos dispositivos sintetizados no FPGA acesso à redes de computadores, no qual poderão trocar dados e serem controlados remotamente.

APÊNDICE A – Códigos VHDL

Código A.1: *Requester*

```
1  ----------------------------------------------------------------------------
2  --      Requester VHDL
3  --
4  --      Copyright (C) 2010 Rodrigo Neri de Souza
5  --
6  --      This program is free software: you can redistribute it and/or modify
7  --      it under the terms of the GNU General Public License as published by
8  --      the Free Software Foundation, either version 3 of the License, or
9  --      any later version.
10 --
11 --      This program is distributed in the hope that it will be useful,
12 --      but WITHOUT ANY WARRANTY; without even the implied warranty of
13 --      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 --      GNU General Public License for more details.
15 --
16 --      You should have received a copy of the GNU General Public License
17 --      along with this program.  If not, see <http://www.gnu.org/licenses/>.
18 --
19 ----------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 entity requester is
26     Port (      clk : IN std_logic;
27                 slide_switch_loop : in  STD_LOGIC;      -- Enable loop
                       device
28                 --Gpio write:
29 DataIn_WE_Reset : in  STD_LOGIC_VECTOR (9 downto 0); -- Reset is active-low
30                 FullFIFO : out  std_logic_vector(0 downto 0);
31                 --Gpio read:
```

```
32                  DataOut_EmptyFIFO: out  STD_LOGIC_VECTOR (8 downto 0);
33                  RE_Reset : in  STD_LOGIC_VECTOR(1 downto 0)
34          );
35 end requester;
36
37 architecture Behavioral of requester is
38
39 --Signals to connect GPIO_to_FIFO componet to device componet:
40 signal DataOut_to_DataIn_Dev : std_logic_vector(7 downto 0);
41 signal RE_Dev_to_RE : std_logic;
42 signal EmptyFIFO_to_EmptyFIFO_Dev : std_logic;
43
44 --Signals to connect FIFO_to_GPIO componet to device componet:
45 signal DataOut_Dev_to_DataIn : std_logic_vector(7 downto 0);
46 signal WE_Dev_to_WE : std_logic;
47 signal FullFIFO_to_FullFIFO_Dev : std_logic;
48
49         COMPONENT GPIO_to_FIFO
50         PORT(
51                 clk : IN std_logic;
52                 DataIn_WE_Reset : IN std_logic_vector(9 downto 0);
53                 RE : IN std_logic;
54                 FullFIFO : OUT std_logic_vector(0 to 0);
55                 DataOut : OUT std_logic_vector(7 downto 0);
56                 EmptyFIFO : OUT std_logic
57                 );
58         END COMPONENT;
59
60         COMPONENT device
61         PORT(
62                 slide_switch_loop : IN std_logic;
63                 clk : IN std_logic;
64                 DataIn_Dev : IN std_logic_vector(7 downto 0);
65                 EmptyFIFO_Dev : IN std_logic;
66                 FullFIFO_Dev : IN std_logic;
67                 RE_Dev : OUT std_logic;
68                 DataOut_Dev : OUT std_logic_vector(7 downto 0);
69                 WE_Dev : OUT std_logic
70                 );
71         END COMPONENT;
72
73         COMPONENT FIFO_to_GPIO
74         PORT(
```

```
75                  clk : IN std_logic;
76                  DataIn : IN std_logic_vector(7 downto 0);
77                  WE : IN std_logic;
78                  RE_Reset : IN std_logic_vector(1 downto 0);
79                  FullFIFO : OUT std_logic;
80                  DataOut_EmptyFIFO : OUT std_logic_vector(8 downto 0)
81                  );
82          END COMPONENT;
83
84  begin
85
86          Inst_GPIO_to_FIFO: GPIO_to_FIFO PORT MAP(
87                  clk => clk,
88                  DataIn_WE_Reset => DataIn_WE_Reset,
89                  FullFIFO => FullFIFO,
90                  DataOut => DataOut_to_DataIn_Dev,
91                  RE => RE_Dev_to_RE,
92                  EmptyFIFO => EmptyFIFO_to_EmptyFIFO_Dev
93          );
94
95          Inst_device: device PORT MAP(
96                  slide_switch_loop => slide_switch_loop,
97                  clk => clk,
98                  DataIn_Dev => DataOut_to_DataIn_Dev,
99                  RE_Dev => RE_Dev_to_RE,
100                 EmptyFIFO_Dev => EmptyFIFO_to_EmptyFIFO_Dev,
101                 DataOut_Dev => DataOut_Dev_to_DataIn,
102                 WE_Dev => WE_Dev_to_WE,
103                 FullFIFO_Dev => FullFIFO_to_FullFIFO_Dev
104         );
105
106         Inst_FIFO_to_GPIO: FIFO_to_GPIO PORT MAP(
107                 clk => clk,
108                 DataIn => DataOut_Dev_to_DataIn,
109                 WE => WE_Dev_to_WE,
110                 FullFIFO => FullFIFO_to_FullFIFO_Dev,
111                 DataOut_EmptyFIFO => DataOut_EmptyFIFO,
112                 RE_Reset => RE_Reset
113         );
114
115 end Behavioral;
```

Código A.2: GPIO_to_FIFO

```
1  ------------------------------------------------------------------------------
2  --      GPIO_to_FIFO VHDL
3  --
4  --      Copyright (C) 2010 Rodrigo Neri de Souza
5  --
6  --      This program is free software: you can redistribute it and/or modify
7  --      it under the terms of the GNU General Public License as published by
8  --      the Free Software Foundation, either version 3 of the License, or
9  --      any later version.
10 --
11 --      This program is distributed in the hope that it will be useful,
12 --      but WITHOUT ANY WARRANTY; without even the implied warranty of
13 --      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 --      GNU General Public License for more details.
15 --
16 --      You should have received a copy of the GNU General Public License
17 --      along with this program.  If not, see <http://www.gnu.org/licenses/>.
18 --
19 ------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 entity GPIO_to_FIFO is
26     Port (        clk : IN std_logic;
27                   --Gpio write:
28 DataIn_WE_Reset : in  STD_LOGIC_VECTOR (9 downto 0); -- Reset is active-low
29                   FullFIFO : out  std_logic_vector(0 downto 0);
30                   --FIFO out:
31                   DataOut : OUT std_logic_vector(7 downto 0);
32                   RE : IN std_logic;
33                   EmptyFIFO : OUT std_logic
34                                 );
35 end GPIO_to_FIFO;
36
37 architecture Behavioral of GPIO_to_FIFO is
38
39 COMPONENT fifo1
40         PORT(
41                   Din : IN std_logic_vector(7 downto 0);
42                   clk : IN std_logic;
```

```
43                  nreset : IN std_logic;
44                  RE : IN std_logic;
45                  WE : IN std_logic;
46                  Dout : OUT std_logic_vector(7 downto 0);
47                  Full : OUT std_logic;
48                  Empty : OUT std_logic
49                  );
50          END COMPONENT;
51
52 --Gpio Write to FIFO IN signals:
53 signal DataIn_mid                : std_logic_vector(7 downto 0);
54 signal WE                : std_logic;
55 signal WE_mid    : std_logic;
56 signal Reset     : std_logic;
57 signal FIFO_mid: std_logic;
58 signal DataOut_mid : std_logic_vector(7 downto 0);
59 begin
60
61 DataOut <= DataOut_mid;
62 --Gpio data out to FIFO data in
63 DataIn_mid <= DataIn_WE_Reset(7 downto 0);
64 --Gpio WE to Synchronizer process
65 WE <= DataIn_WE_Reset(8);
66 --Gpio Reset to FIFO Reset
67 Reset <= DataIn_WE_Reset(9);
68 --Convert FIFO_mid std_logic signal to FullFIFO std_logic_vector port
69 FullFIFO(0) <= FIFO_mid;
70
71 --Synchronizer of GPIO's WE to FIFO's WE :
72 PROCESS(WE, clk)
73          --This process creates a pulse in WE starts at a falling edge clock
               and
74          --finishes in the next falling edge, that active WE.
75          --Notes WE is activated at rising edge clock.
76
77 variable wr : integer range 0 to 1;
78
79 BEGIN
80          if(falling_edge(clk)) then
81                  if(wr = 0 and (WE = '1'))then
82                          wr := 1;
83                          WE_mid <= '1';  --Set WE_mid to 1
84                  elsif((WE = '0')) then
```

```
85                          wr := 0;
86                          WE_mid <= '0';  --One clock after sets WE_mid to 0
87                  else
88                          WE_mid <= '0';  --One clock after sets WE_mid to 0
89                  end if;
90          end if;
91  END PROCESS;
92
93  Inst_fifo1: fifo1 PORT MAP(
94                  clk => clk,
95
96                  Din => DataIn_mid,
97                  nreset => Reset,
98                  WE => WE_mid,
99                  Full => FIFO_mid,
100
101                 Dout => DataOut_mid,
102                 RE => RE,
103                 Empty => EmptyFIFO
104         );
105 end Behavioral;
```

Código A.3: FIFO_to_GPIO

```
1   ----------------------------------------------------------------------
2   --      FIFO_to_GPIO VHDL
3   --
4   --      Copyright (C) 2010 Rodrigo Neri de Souza
5   --
6   --      This program is free software: you can redistribute it and/or modify
7   --      it under the terms of the GNU General Public License as published by
8   --      the Free Software Foundation, either version 3 of the License, or
9   --      any later version.
10  --
11  --      This program is distributed in the hope that it will be useful,
12  --      but WITHOUT ANY WARRANTY; without even the implied warranty of
13  --      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14  --      GNU General Public License for more details.
15  --
16  --      You should have received a copy of the GNU General Public License
17  --      along with this program.  If not, see <http://www.gnu.org/licenses/>.
18  --
19  ----------------------------------------------------------------------
20  library IEEE;
```

```
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 entity FIFO_to_GPIO is
26     Port (      clk : IN std_logic;
27                         --FIFO in:
28                                 DataIn : IN std_logic_vector(7 downto 0);
29                                 WE : IN std_logic;
30                                 FullFIFO : OUT std_logic;
31                         --Gpio read:
32                                 DataOut_EmptyFIFO: out STD_LOGIC_VECTOR (8
                                       downto 0);
33                                 RE_Reset                    : in
                                      STD_LOGIC_VECTOR(1 downto 0)
34                                 );
35 end FIFO_to_GPIO;
36
37 architecture Behavioral of FIFO_to_GPIO is
38
39 COMPONENT fifo2
40         PORT(
41                 Din : IN std_logic_vector(7 downto 0);
42                 clk : IN std_logic;
43                 nreset : IN std_logic;  --Reset is active-low.
44                 RE : IN std_logic;
45                 WE : IN std_logic;
46                 Dout : OUT std_logic_vector(7 downto 0);
47                 Full : OUT std_logic;
48                 Empty : OUT std_logic
49                 );
50         END COMPONENT;
51
52 --Gpio Read from FIFO OUT signals:
53 signal DataOut_mid                 : std_logic_vector(7 downto 0);
54 signal RE_mid                              : std_logic;
55 signal EmptyFIFO_mid               : std_logic;
56 signal RE                                          : std_logic;
57 signal Reset                               : std_logic;
58
59 begin
60
61 RE <= RE_Reset(0);
```

```
62 Reset <= RE_Reset(1);
63
64 --Gpio data in from FIFO data out
65 DataOut_EmptyFIFO(8) <= EmptyFIFO_mid;
66 DataOut_EmptyFIFO(7 downto 0) <= DataOut_mid;
67
68 --Synchronizer of GPIO's RE to FIFO's RE:
69 PROCESS(RE, clk)
70 --This process creates a pulse in RE starts at a rising edge clock and
71 --finishes in the next rising edge, that active RE.
72 --Notes RE activated at falling edge clock.
73
74 variable rd : integer range 0 to 1;
75
76 BEGIN
77
78         if(falling_edge(clk)) then
79                 if(rd = 0 and RE = '1')then
80                         rd := 1;
81                         RE_mid <= '1';  --Set RE_mid to 1
82                 elsif(RE = '0') then
83                         rd := 0;
84                         RE_mid <= '0';  --One clock after sets RE_mid to 0
85                 else
86                         RE_mid <= '0';  --One clock after sets RE_mid to 0
87                 end if;
88         end if;
89
90 END PROCESS;
91
92 Inst_fifo2: fifo2 PORT MAP(
93                 clk => clk,
94
95                 Din => DataIn,
96                 nreset => Reset,
97                 WE => WE,
98                 Full => FullFIFO,
99
100                Dout => DataOut_mid,
101                RE => RE_mid,
102                Empty => EmptyFIFO_mid
103        );
104 end Behavioral;
```

Código A.4: *Device*

```
1  ------------------------------------------------------------------------
2  --     Device VHDL
3  --
4  --     Copyright (C) 2010 Rodrigo Neri de Souza
5  --
6  --     This program is free software: you can redistribute it and/or modify
7  --     it under the terms of the GNU General Public License as published by
8  --     the Free Software Foundation, either version 3 of the License, or
9  --     any later version.
10 --
11 --     This program is distributed in the hope that it will be useful,
12 --     but WITHOUT ANY WARRANTY; without even the implied warranty of
13 --     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 --     GNU General Public License for more details.
15 --
16 --     You should have received a copy of the GNU General Public License
17 --     along with this program.  If not, see <http://www.gnu.org/licenses/>.
18 --
19 ------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24 use IEEE.NUMERIC_STD.ALL;
25
26 entity device is
27     Port ( slide_switch_loop    : in  STD_LOGIC;                    -- Enable
           loop
28             clk : in  STD_LOGIC;                                     --
               Process' clock
29             --Ports to connect at FIFO out
30             DataIn_Dev        : in  STD_LOGIC_VECTOR (7 downto 0); --
               Data in
31             RE_Dev   : out  STD_LOGIC;                               --
               Enable to Read
32             EmptyFIFO_Dev : in  STD_LOGIC;                      -- FIFO
               empty flag
33             --Ports to connect at FIFO in
34             DataOut_Dev : out  STD_LOGIC_VECTOR (7 downto 0);-- Data
               out
```

```
35                  WE_Dev : out  STD_LOGIC;                                    --
                        Enable to Write
36                  FullFIFO_Dev              : in  STD_LOGIC          -- FIFO
                        full flag
37                          );
38 end device;
39
40 architecture Behavioral of device is
41
42 begin
43
44 DataOut_Dev<= DataIn_Dev;
45
46 PROCESS(slide_switch_loop, clk, EmptyFIFO_Dev, FullFIFO_Dev)
47
48 variable rd_count : integer range 0 to 1;
49 variable wr_count : integer range 0 to 1;
50 variable trigger : integer range 0 to 1;
51 variable d_switch_loop : integer range 0 to 1;
52 variable cntsl : integer range 0 to 250000;
53
54 BEGIN
55
56 --Turn slide_switch_loop in a digital switch stable
57
58         if(slide_switch_loop = '1') then
59                 if(clk'event and clk='0') then
60                         if(cntsl=250000)then
61                                 cntsl := 0;
62                                 d_switch_loop := 1;
63                         else
64                                 cntsl := cntsl+1;
65                         end if;
66                 end if;
67         elsif(slide_switch_loop = '0') then
68                 if(clk'event and clk='0') then
69                         if(cntsl=250000)then
70                                 cntsl := 0;
71                                 d_switch_loop := 0;
72                         else
73                                 cntsl := cntsl+1;
74                         end if;
75                 end if;
```

```
76          end if;
77
78
79  --Read FIFO_1 and Write FIFO_2
80
81          if(falling_edge(clk)) then
82                  if(rd_count=0 and FullFIFO_Dev='0' and EmptyFIFO_Dev='0'
                       and d_switch_loop=1)then
83                          rd_count := 1;
84                          RE_Dev <= '1';  --Set RE_Dev to 1
85                          trigger := 1;
86                  elsif(d_switch_loop = 0) then
87                          rd_count := 0;
88                          RE_Dev <= '0';  --One clock after sets RE_Dev to 0
89                  else
90                          RE_Dev <= '0';  --One clock after sets RE_Dev to 0
91
92                          if(FullFIFO_Dev = '0' and trigger = 1)then
93                          --trigger does write process starts after read
                               process
94                                  WE_Dev <= '1';  --Set WE_Dev to 1
95                                  trigger := 0;
96                          else
97                                  WE_Dev <= '0';  --One clock after sets
                                       WE_Dev to 0
98                          end if;
99
100                 end if;
101         end if;
102
103 END PROCESS;
104
105 end Behavioral;
```

APÊNDICE B – Roteiro para embarcar o PetaLinux

Descrição: Esse roteiro foi desenvolvido em um computador com o sistema operacional Ubuntu 9.10 64 bits para configurar e embarcar o PetaLinux v0.40 em um sistema embarcado, criado pelo XPS 11.5, contido na Spartan-3E *Starter Kit*. A informações aqui contidas estão baseadas na documentação oficial do PetaLinux [1].

B.1 Instalando o PetaLinux

1. Para descompactar o PetaLinux execute o seguinte comando no terminal de linha de comando do Ubuntu [2]:

   ```
   [user@host home]$ tar xzf petalinux-v0.40-final.tar.gz
   ```

2. Definindo no Ubuntu as variáveis do PetaLinux. Execute os seguintes comandos [2]:

   ```
   $ cd PetaLinux
   $ source ./settings.sh
   PetaLinux environment set to '/home/user/PetaLinux'
   ```

   Lembre-se que é necessário executar o passo B.1.2 todas as vezes que for manipular o PetaLinux.

3. Como usuário root crie a pasta /tftpboot onde serão armazenadas as imagens do PetaLinux compilado. Execute os seguintes comandos [2]:

   ```
   $ su
   Password: ********
   # mkdir /tftpboot
   # chmod -R 777 /tftpboot
   # exit
   $ sudo chown -R nobody /tftpboot
   ```

4. Instalando um servidor TFTP no Ubuntu para transferir as imagens do PetaLinux para o *Bootloader*. Execute o comando [3]:

```
$ sudo apt-get install tftp
```

5. Modifique (crie se não existir) o arquivo /etc/xinetd.d/tftp com o seguinte conteúdo:

```
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
user            = nobody
server          = /usr/sbin/in.tftpd
server_args     = /tftpboot
disable         = no
}
```

6. Inicie o servidor TFTP. Execute o comando:

```
$ sudo /etc/init.d/xinetd start
```

B.2 Configurado o sistema embarcado do XPS

1. Copie o projeto do sistema embarcado do XPS para $PetaLinux/hardware/user-platforms. Execute o seguinte comando [4]:

```
$ cp -rf my_hardware_project $PETALINUX/hardware/user-platforms
```

2. Adicione o PetaLinux BSB ao projeto do XPS. Crie um link simbólico no projeto do XPS para o PetaLinux BSB [4]:

```
$ cd $PETALINUX/hardware/user-platforms/my_hardware_project
$ ln -s ../../edk_user_repository edk_user_repository
```

3. Inclua, como mostrado abaixo, a variável *ModuleSearchPath* no arquivo system.xmp, para o XPS saber onde estão localizados os arquivos do PetaLinux BSP [4]:

```
XmpVersion: 9.1.01
IntStyle: default
ModuleSearchPath: edk_user_repository/
MHS File: system.mhs
MSS File: system.mss
```

4. Abra com o XPS o projeto do sistema embarcado e modifique os parâmetros OS_NAME e OS_VER do arquivo system.mss como se mostra [4]:

```
BEGIN OS
PARAMETER OS_NAME = petalinux
PARAMETER OS_VER = 1.00.b
```

5. Configurar os parâmetros do PetaLinux no XPS. No XPS vá em *Software* > *Software Platform Settings* e configure em *OS and Lib Configuration* os seguintes parâmetros com os seguintes argumentos [5]:

```
main_memory:  DDR_SDRAM
flash_memory:  FLASH
lmb_memory:  dlmb
```

6. Gere o *Bitstream*, o *Libraries and BSPs* e saia do XPS [6].

B.3 Configurando o PetaLinux

1. Construindo o PetaLinux. Execute os seguintes comandos [7]:

```
$ cd $PETALINUX/software/petalinux-dist
$ make menuconfig
```

2. Definindo o FPGA no qual será embarcado o PetaLinux [7]. No *menuconfig* vá em *Vendor/Product Selection*, e defina o *Vendor* como Xilinx e o *Product* como Spartan3E500-RevD-lite-edk101.

3. Configurando os parâmetros de usuário do Petalinux [8]. Retorne ao menu principal do *menuconfig* e vá em *Kernel/Library/Defaults Selection*, marque a opção *Customize Vendor/User Settings*, defina o *Kernel Version* como linux-2.6.x e o *Libc Version* como *None*. Salve a configurações do *Kernel* e saia do *menuconfig*.

4. Configurando as definições do sistema [8]. Ao sair do *menuconfig* anterior, outro *menuconfig* aparecerá, nesse vá em *System Settings*.

5. Configurando os parâmetros do endereço de rede [8] [9]. Vá em *Network Addresses* e marque *Obtain IP address automatically* se o serviço cliente DHCP for requerido. No caso de endereço IP fixo, defina *Static IP address* com o IP desejado.

6. Definindo o nome do sistema embarcado e a senha de root [9]. Volte ao *System Settings* e defina, se preferir, um nome e uma senha diferente da padrão em *Default host name* e *Default root password* respectivamente.

7. Definindo o sistema de arquivo [9]. Em *Root filesystem type* defina o tipo de sistema de arquivo como CRAMFS.

8. Armazenando a imagem do PetaLinux na pasta /tftpboot [9]. Marque a opção *Copy final image to tftpboot* e defina *tftpboot directory* com o endereço /tftpboot.

9. Construindo o *Bootloader U-boot* [9]. Marque a opção *Build u-boot* para compilar e criar uma imagem do *U-boot* para o Petalinux.

10. Adicionando o servidor *web* BOA. No *menuconfig* retorne ao menu principal e vá em *Network Applications* e marque na lista a aplicação chamada boa.

11. Adicionando a aplicação *route* para permitir a configuração do *gateway* padrão. Também em *Network Applications* selecione na lista a aplicação *route*. Volte ao menu principal e vá em *BusyBox* e marque na lista novamente a aplicação *route*. Saia do *menuconfig* e salve as configurações.

12. Utilize a ferramenta *petalinux-copy-autoconfig* para copiar os parâmetros de *hardware* do sistema embarcado necessários para configurar o *Kernel* e o *Bootloader*. Entre no diretório do projeto e execute o seguinte comando [5]:

```
$ cd $PETALINUX/hardware/user-platforms/my_hardware_project
$ petalinux-copy-autoconfig
```

13. Compilado o PetaLinux, execute os passos B.3.1, B.3.2, B.3.3 e saia do segundo *menuconfig*. Execute os seguintes comandos [10]:

```
$ cd $PETALINUX/software/petalinux-dist
$ yes "" | make oldconfig dep all
```

14. Em caso de erro referente ao *U-Boot*, compile o *U-Boot* separadamente [11]. Execute os comandos abaixo 2 vezes e repita o passo B.3.13. Repita essa seqüência quantas vezes forem necessárias até o erro referente o *U-Boot* desaparecer.

```
$ make u-boot
$ make image
```

15. Criando as aplicações em C ao PetaLinux. Para adicionar os programas *writereset*, *writefifo*, *writefifofile*, *readreset*, *readfifo* e *readfifofile* execute os seguintes comandos [12]:

```
$ cd $PETALINUX/software/user-apps
$ petalinux-new-app writereset
$ petalinux-new-app writefifo
$ petalinux-new-app writefifofile
$ petalinux-new-app readreset
$ petalinux-new-app readfifo
$ petalinux-new-app readfifofile
```

16. Adicionando os códigos em C as respectivas aplicações. Em cada uma das pastas criadas pelo comando *petalinux-new-app* existe um arquivo em C com o mesmo nome da pasta. Nestes arquivos deverá estar escrito a aplicação em C. Portanto, escreva em cada um desses arquivos o respectivo código constante no apendice D.

17. Compilando as aplicações e adicionando-as ao sistema de arquivo. Execute os comandos a seguir [12]:

```
$ cd $PETALINUX/software/user-apps/writereset
$ make
$ make romfs
$ cd $PETALINUX/software/user-apps/writefifo
$ make
$ make romfs
$ cd $PETALINUX/software/user-apps/writefifofile
$ make
$ make romfs
$ cd $PETALINUX/software/user-apps/readreset
$ make
$ make romfs
```

```
$ cd $PETALINUX/software/user-apps/readfifo
$ make
$ make romfs
$ cd $PETALINUX/software/user-apps/readfifofile
$ make
$ make romfs
```

18. Adicionando os arquivos do *website*. Adicione no diretório $PETALINUX/software/petalinux-dist/romfs/home/httpd os arquivos do *website*, como por exemplo a página *index.html* do apêndice C e os arquivos em formato já requeridos pelo *index*.

19. Configurações durante o arranque do PetaLinux. Apague o conteúdo do arquivo *thttpd* localizado em $PETALINUX/software/petalinux-dist/romfs/etc/init.d e insira os comandos que necessitam estar no arranque. Segue abaixo o conteúdo de um arquivo *thttpd* exemplo, no qual os comandos estão: definindo um *gateway* padrão com endereço 193.136.60.250; definindo o endereço 193.136.60.10 e 193.136.60.2 como servidor DNS; ativando o servidor *web* BOA e executando o comando *writereset* e *readreset*.

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin
echo "Defining gateway: "
route add default gw 193.136.60.250
echo "Defining DNS: "
echo "nameserver 193.136.60.10" >/etc/resolv.conf
echo "nameserver 193.136.60.2"  >>/etc/resolv.conf
echo "Starting boa: "
boa &
echo "Resetting writefifo: "
writereset
echo "Resetting readfifo: "
readreset
```

20. Acessando as definições do sistema. Execute os passos B.3.1, B.3.2 e B.3.3.

21. Removendo o servidor *web* THTTPD. Agora, no outro *menuconfig*, vá em *Network Applications* e desmarque a aplicação *thttpd*. Saia do *menuconfig* e salve as configurações.

22. Compilando a imagem final do PetaLinux. Execute o passo B.3.13.

B.4 Criando o *Bootloader*

1. Criando o *Bootloader* para o *U-Boot* e armazenando o *U-Boot* na memória *Flash* da Spartan-3E *Starter Kit*. Abra com o XPS o projeto do sistema embarcado. Na barra de menu vá em *Device Configuration* e selecione *Program Flash Memory*. Na janela que aparecer defina *File to Program* como /tftpboot/u-boot.sre, defina *Instance Name* do *Flash Memory Properties* como memória *Flash* e defina *Instance Name* do *Scratch Memory Properties* como memória DDR SDRAM. Marque também a opção *Create Flash Bootloader Application* para que o XPS crie um *Bootloader* para o *U-Boot*. Marque o *Bootloader* criado para inicializar nas BRAMs e saia do XPS.

B.5 Importando o projeto do sistema embarcado para o ISE.

1. Não é do escopo deste roteiro explicar como exportar projetos do XPS para o ISE. O roteiro acerca desta integração é encontrado na refêrencia [13]. Contudo, isto é fundamental para o funcionamento da plataforma de comunicação. Uma vez realizado isto, gere o *Bitstream* com o *Bootloader* carregado na BRAM.

B.6 Armazenando o *hardware* do sistema embarcado na Spartan-3E *Starter Kit*

1. Criando, apartir do *Bitstream*, um arquivo MCS para armazenar o *hardware* do sistema embarcado na Spartan-3E *Starter Kit*. Pelo terminal de comandos do Ubuntu entre na pasta onde encontra-se o *Bitstream* do projeto e execute o seguinte comando:

   ```
   $ promgen -w -p mcs -u 0 bitstream_do_projeto.bit
   ```

2. Armazenando o *hardware* no *kit*. Utilize o programa *impact* da Xilinx para com o arquivo MCS gerado anteriormente guardar o *hardware* no *kit*. Após isso, toda vez que o *kit* for ligado o *hardware* armazenado no *kit* será sintetizado no FPGA automaticamente.

B.7 Armazenando o Petalinux na memória *Flash*

1. Utilizado *U-Boot* para guardar o Petalinux na *Flash*. Para isto é necessário conectar-se a RS232 do *kit*, que é o terminal de usuário do sistema embarcado. Assim é possível enviar, através de um programa terminal de comunicação serial, comandos ao *U-Boot*.

2. Configurando o *U-Boot* para funcionamento via rede de computadores. É necessário o endereço IP do computador onde estão salvas as imagens do Petalinux geradas nos passos anteriores, bem como um IP válido para o *kit*. Execute os seguintes comandos no terminal de comunicação serial:

   ```
   set serverip "IP do computador com as imagens"
   set ipaddr "IP disponível para o kit"
   ```

   Os dois IP's acima devem estar na mesma rede.

3. Apagar na memória *Flash* a parte que será utilizada para armazenar a imagem no formato *ub*. Deve ser apagada uma quantidade de setores da *Flash* que compreenda o tamanho da imagem. O setor inicial onde será armazenada a imagem é o setor onde consta o endereço no qual o *U-Boot* tentou carregar a *image.ub*. Para saber o setor final, some o tamanho da imagem ao endereço do setor inicial. Para verificar os setores da *Flash*, bem como o seus endereços, execute o seguinte comando[15]:

   ```
   flinfo
   ```

   Na implementação em questão, o setor 4 foi o inicial e o 32, mais do que necessário, foi o final. Assim o comando para apagar os setores da *Flash* ficou assim:

   ```
   erase 1:4-32
   ```

4. Compiando a imagem do Petelinux via TFTP. Para isso é necessário o endereço base da memória SDRAM, que pode ser obtido no XPS. No projeto em questão o endereço da memória é 0x44000000, assim o comando ficou como mostrado abaixo [14]:

   ```
   tftp 0x44000000 image.ub
   ```

5. Copiar imagem formato *ub* para a memória *Flash*. No projeto em questão o endereço da *Flash* é 0x82080000 e o tamanho da imagem é de 0x32b0d0. Para descobrir o tamanho da imagem basta verificar os *Bytes* transferidos no retorno do comando do passo B.7.4. Assim execute o comando a seguir [14]:

   ```
   cp.b 0x44000000 0x82080000 0x32b0d0
   ```

6. Pronto. Agora a imagem do Petalinux está armazenada na memória *Flash* sendo inicializada pelo *U-Boot*, que por sua vez é inicializado pelo *Bootloader* do XPS, contido na BRAM, sempre que o *kit* é ligado.

Referências do Apêndice

1. <http://www.petalogix.com/univ/documentation>
2. <http://www.petalogix.com/univ/documentation/petalinux-installation-guide/petalinux-installation-procedure>
3. <http://www.petalogix.com/univ/documentation/petalinux-installation-guide/networking-configuration-procedures>
4. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Customising/NewHardware\#AddingNewHardwareProject>
5. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Basics/AutoConfig>
6. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Basics/BuildingHardware>
7. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Basics/SelectPlatform>
8. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Basics/PetaLinuxConfiguration>
9. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Customising/SystemConfigMenu>
10. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Basics/BuildingPetaLinux>
11. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Bootloaders/UBoot/BuildUBoot>
12. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Customising/UserApps>
13. <http://www.youtube.com/watch?v=R5wZ89BcBPo>
14. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Bootloaders/UBoot/UBDownload>
15. <http://www.petalogix.com/resources/documentation/petalinux/userguide/Bootloaders/UBoot/UBProgFlash>

APÊNDICE C – Página HTML (index.html)

Descrição: Página HTML com um cliente *Telnet* e um cliente FTP, ambos em *Java Applet*. Os arquivos JAR estão disponíveis respectivamente em http://javassh.org/space/Installing+the+Applet e http://radinks.net/ftp/applet/demo.php .

Código C.1: Index.htm

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4  <title>Spartan3e</title>
5  </head>
6  <body>
7  <table align = "center">
8  <tr>
9  <td align = "center">
10 <applet CODEBASE="."
11     ARCHIVE="jta26.jar"
12     CODE="de.mud.jta.Applet"
13     WIDTH=710 HEIGHT=360>
14 <param name="config" value="applet.conf">
15 </applet>
16 </td>
17 </tr>
18 <tr>
19 <td align ="center">
20
21 <script language="JavaScript">
22 <!--
23 var _info = navigator.userAgent;
24 var ie = (_info.indexOf("MSIE") > 0);
25 var win = (_info.indexOf("Win") > 0);
26
27 if(win)
28 {
```

```
29
30 if(ie)
31 {
32 document.writeln('<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805
      F499D93"');
33 document.writeln('width= "710" height= "420"');
34 document.writeln('codebase="http://java.sun.com/update/1.5.0/
35 jinstall-1_5_0-windows-i586.cab#version=1,5">');
36 }
37 else
38 {
39 document.writeln('<object type="application/x-java-applet;version=1.5"');
40 document.writeln('width= "710" height= "420"  >');
41 }
42 document.writeln('<param name="archive" value="ftpicons.zip,RadFTPDemo.jar"
      >');
                                        document.writeln('<param name="code"
      value="com.radinks.ftp.FTPAppletDemo">');
43 document.writeln('<param name="name" value="Rad FTP">');
44 }
45 else
46 {
47 /* mac and linux */
48 document.writeln('<applet ');
49 document.writeln'                        archive  = "ftpicons.zip,RadFTPDemo
      .jar"');
50 document.writeln'                        code     = "com.radinks.ftp.
      FTPAppletDemo"');
51 document.writeln'                        name     = "Rad FTP"');
52 document.writeln'                        hspace   = "0"');
53 document.writeln'                        vspace   = "0"');
54 document.writeln'                        width = "710"');
55 document.writeln'                        height = "420"');
56 document.writeln'                                align    = "middle">');
57 }
58
59 if(win)
60 {
61 document.writeln('</object>');
62 }
63 else
64 {
65 document.writeln('</applet>');
```

```
66 }
67 //-->
68 </script>
69 </td>
70 </tr>
71 </table>
72 </body>
73 </html>
```

APÊNDICE D – Códigos em C dos tradutores

Código D.1: Writefifofile

```
/*********************************************************************
**      Writefifofile
**
**      Copyright (C) 2010 Rodrigo Neri de Souza
**
**      This program is free software: you can redistribute it and/or modify
**      it under the terms of the GNU General Public License as published by
**      the Free Software Foundation, either version 3 of the License, or
**      any later version.
**
**      This program is distributed in the hope that it will be useful,
**      but WITHOUT ANY WARRANTY; without even the implied warranty of
**      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
**      GNU General Public License for more details.
**
**      You should have received a copy of the GNU General Public License
**      along with this program.  If not, see <http://www.gnu.org/licenses/>.
**
*********************************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <ctype.h>              /* For isxdigit function */
#include <linux/xgpio_ioctl.h>
/* Linux Xilinx GPIO library for ioctl functions */
#define GPIO_DEV "/dev/gpio1"   /* GPIO's name in /dev */
#define size 200                /* Limit of letter read */


int main(int argc, char *argv[])
{
        FILE *pFile;                    /* Pointer to File */
```

```
33          char char_of_letter_hex[1];      /* Char to store the letters read
                from file  */
34          char path[50];                   /* Char to store the File name */
35          char q[1];                       /* Char to store the answer of
                question */
36          char data_WE_Reset_C[4];         /* Char stores send data */
37          int int_letter_hex[size];        /* Int array to store the hex read
                from file  */
38          int int_ascii_of_letter;
39          /* Int to store the ASCII value of letter read from file  */
40          int n = 0;                       /* The number of letter read */
41          int c = 0;                       /* Counter to run int_letter_hex */
42
43
44          /* Check if user entered file name */
45          if(argc < 2)
46          {
47                  printf("\nInvalid file name. Please enter a valid file name
                        . Example: file.txt\n\n");
48                  return 0;
49          }
50
51          sprintf(path, "/var/tmp/%s", argv[1]);
52          pFile = fopen (path,"rb");       /* Open the file */
53
54
55          if (pFile!=NULL)
56          {
57                  while((!feof(pFile))&&(n < size))
58                  /* Read file until end of file or limit of letter read  */
59                  {
60                          int_ascii_of_letter = fgetc (pFile);
61                          /* Get the ASCII value of a letter */
62
63                          sprintf(char_of_letter_hex, "%c",
                                int_ascii_of_letter);
64                          /* Convert int_ascii_of_letter in a letter and
                                store in a char */
65
66                          if (isxdigit(char_of_letter_hex[0]))
67                          /* Check if the letter is a hex number */
68                          {
```

```
69                                  int_letter_hex[n] = strtoul (
                                        char_of_letter_hex ,NULL,16);
70                                  /* Convert char_of_letter_hex in a integer
                                        hex number and store in a array */
71                                  n++;
72                          }
73                  }
74
75                  if(n == 0)      /* Check if there is ant hexadecimal number
                        in the file */
76                  {
77                          printf("\nThere is not any hexadecimal number in
                                the file\n\n");
78                  }
79                  else if(n == size)      /* Check if there are more numbers
                        than the limit*/
80                  {
81                          printf("\nThe maximum hexadecimal number in the
                                file is %d.\n\n", size);
82                  }
83                  else if(n % 2 != 0)     /* Check if sequence numbers is
                        even */
84                  {
85                          printf("\nThere is a missing hexadecimal number in
                                the file.
86                          Sequence numbers must be even.\n\n");
87                          close(pFile);
88                          return errno;
89                  }
90                  else
91                  {
92                          printf("\nThe hexadecimal numbers of the file are:\
                                n");
93                          /* Print the hexadecimal numbers in the file */
94                          while(c < n)
95                          {
96                                  printf("%1x%1x\n",int_letter_hex[c],
                                        int_letter_hex[c+1]);
97                                  c++;
98                                  c++;
99                          }
100                         printf("\nThe file is Ok!\n\n");
101                 }
```

```
102
103                 /* Open the device */
104                 int fd = open(GPIO_DEV, O_RDWR);
105                 /* The file descriptor of GPIO_DEV */
106
107                 /* Check if fd was opened successfully */
108                 if(fd==-1)
109                 {
110                         printf("\n\nUnable to open %s\nContinuing anyway
                            ...\n\n", GPIO_DEV);
111                 }
112
113                 /* xgpio_ioctl_data struct */
114                 struct xgpio_ioctl_data gpio_in_fullFIFO;
115                 /* Struct for gpio_in_fullFIFO */
116                 memset(&gpio_in_fullFIFO, 0, sizeof(gpio_in_fullFIFO));
117                 /* Clean gpio_in_fullFIFO */
118                 gpio_in_fullFIFO.chan = 2;                              /*
                    The gpio channel */
119                 gpio_in_fullFIFO.mask = 0x00000001;
120                 /* Gpio mask, that sets channel data pins. In this case,
                    there
121                 is 1 data pins in channel, that is connected FIFO Full Flag
                    */
122
123                 /* xgpio_ioctl_data struct */
124                 struct xgpio_ioctl_data gpio_out_data_WE_Reset;
125                 /* Struct for gpio_out_data_WE_Reset */
126                 memset(&gpio_out_data_WE_Reset, 0, sizeof(
                    gpio_out_data_WE_Reset));
127                 /* Clean gpio_out_data_WE_Reset */
128                 gpio_out_data_WE_Reset.chan = 1;
                                        /* The gpio channel */
129                 gpio_out_data_WE_Reset.mask = 0x000003FF;
130                 /* Gpio mask, that sets channel data pins. In this case,
                    there are 10
131                 data pins in channel. The first 8 are connected to FIFO
                    Data Out, the
132                 nineth is connected to FIFO Write Enable Flag and the last
                    one (MSB)
133                 is connected FIFO Reset. FIFO Resett is ACTIVE-LOW */
134                 gpio_out_data_WE_Reset.data = 0x000003FF;
135                 /* Int variable stores data of channel */
```

```
136
137                 /* Important: before we write we must set the proper data
                        direction.
138                 The data direction is 0 on boot (all write), but may have
                        changed for
139                 whatever reason.
140                 Doing a write will set it only AFTER the operation, so it's
                         no good.
141                 The correct way to set the direction is using TRISTATE
                        which sets the
142                 data direction without reading. */
143
144                 /* Set gpio_out_data_WE_Reset as TRISTATE */
145                 if(0 > (ioctl(fd, XGPIO_TRISTATE, &gpio_out_data_WE_Reset))
                        )    /
146                 * Check if Ioctl command that sets GPIO as tristate was
                        done sucessfully */
147                 {
148                         printf("\nioctl TRISTATE failed\n");
149                         close(pFile);
150                         close(fd);
151                         return errno;
152                 }
153
154                 c = 0;
155                 while(c < n)
156                 {
157                         /* Set gpio_in_fullFIFO as input */
158                         ioctl(fd, XGPIO_IN, &gpio_in_fullFIFO);
159                         /* Ioctl command, that sets gpio_in_fullFIFO as
                                input.
160                         It updates gpio_in_data_empty.data with data input
                                value  */
161
162
163                         /* Check if FIFO is Full */
164                         if(gpio_in_fullFIFO.data == 0x00000001)
165                         {
166                                 printf("\nFIFO is Full.");
167                                 printf("\nPress enter to still write, type
                                        E to exit.\n");
168                                 fgets(q,2,stdin);
169                                 if((!strcmp(q,"e"))||(!strcmp(q,"E")))
```

```
170                                     {
171                                             close(fd);
172                                             close(pFile);
173                                             return 0;
174                                     }
175                             }
176                             else /* If FIFO is not Full, it can be write */
177                             {
178                                     sprintf(data_WE_Reset_C, "3%x%x",
                                            int_letter_hex[c],int_letter_hex[c+1]);
179                                             /* Build a string with this format:
                                                    "3(letters read from file)".
180                                             It means in binary "11(letters read
                                                    from file)", so this value
181                                             actives FIFO Write Enable */
182
183                                     gpio_out_data_WE_Reset.data = strtoul (
                                            data_WE_Reset_C,NULL,16);
184                                     /* Convert data_WE_Reset_C in a integer and
                                            put it in
185                                     gpio_out_data_WE_Reset.data */
186
187                                     /* Active FIFO Write Enable and write Data
                                            in FIFO data in */
188                                     ioctl(fd, XGPIO_OUT, &
                                            gpio_out_data_WE_Reset);
189                                     /* Ioctl command, that sets
                                            gpio_out_data_WE_Reset as output.
190                                     It puts the gpio_out_RE_Reset.data at
                                            output, */
191
192
193                                     sprintf(data_WE_Reset_C, "2%x%x",
                                            int_letter_hex[c],int_letter_hex[c+1]);
194                                     /* Build a string with this format: "2(
                                            letters read from file)".
195                                     It means in binary "10(letters read from
                                            file)", so this value
196                                     disable FIFO Write Enable */
197                                     gpio_out_data_WE_Reset.data = strtoul (
                                            data_WE_Reset_C,NULL,16);
198                                     /* Convert data_WE_Reset_C in a integer and
                                            put it in
```

```
                                gpio_out_data_WE_Reset.data */

                                /* Disable FIFO Write Enable */
                                ioctl(fd, XGPIO_OUT, &
                                    gpio_out_data_WE_Reset);
                                /* Ioctl command, that sets
                                    gpio_out_data_WE_Reset as output.
                                It puts the gpio_out_RE_Reset.data at
                                    output, */

                                c=c+2;
                        }
                }

                close(fd);      /* Close the file descriptor of GPIO_DEV
                    */
        }
        else
        {
                printf("\nIt is not possible to open %s\n\n", path);
                close(pFile);
                return errno;
        }

        close(pFile);   /* Close the pointer to File */
        return 0;


}
```

Código D.2: Writefifo

```
/*****************************************************************
**      Writefifo
**
**      Copyright (C) 2010 Rodrigo Neri de Souza
**
**      This program is free software: you can redistribute it and/or modify
**      it under the terms of the GNU General Public License as published by
**      the Free Software Foundation, either version 3 of the License, or
**      any later version.
**
**      This program is distributed in the hope that it will be useful,
**      but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
13 **     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 **     GNU General Public License for more details.
15 **
16 **     You should have received a copy of the GNU General Public License
17 **     along with this program.  If not, see <http://www.gnu.org/licenses/>.
18 **
19 ***********************************************************************/
20 #include <stdio.h>
21 #include <fcntl.h>
22 #include <errno.h>
23 #include <ctype.h>              /* For isxdigit function */
24 #include <linux/xgpio_ioctl.h>
25 /* Linux Xilinx GPIO library for ioctl functions */
26 #define GPIO_DEV "/dev/gpio1"   /* GPIO's name in /dev */
27
28
29 int main(int argc, char *argv[])
30 {
31         char data_WE_Reset_C[4];        /* Char to store send data */
32
33         /* Check if argv is two digits number */
34         if(2 != strlen(argv[1]))
35         {
36                 printf ("\nYou have to enter a hexadecimal number with two
                      digits. Example: 0F\n\n");
37                 return 0;
38         }
39         /* Check if argv is hexadecimal number */
40         if (!(isxdigit(argv[1][0]) && isxdigit(argv[1][1])))
41         {
42                 printf ("\nYou have to enter a hexadecimal number. Example:
                       0F\n\n");
43                 return 0;
44         }
45
46         /* Open the device */
47         int fd = open(GPIO_DEV, O_RDWR);        /* The file descriptor of
              GPIO_DEV */
48
49         /* Check if fd was opened successfully */
50         if(fd==-1)
51         {
```

```
52                  printf("\n\nUnable to open %s\nContinuing anyway...\n\n",
                        GPIO_DEV);
53          }
54
55          /* xgpio_ioctl_data struct */
56          struct xgpio_ioctl_data gpio_in_fullFIFO;
57          /* Struct for gpio_in_fullFIFO */
58          memset(&gpio_in_fullFIFO, 0, sizeof(gpio_in_fullFIFO));
59          /* Clean gpio_in_fullFIFO */
60          gpio_in_fullFIFO.chan = 2;                              /* The gpio
                 channel */
61          gpio_in_fullFIFO.mask = 0x00000001;
62          /* Gpio mask, that sets channel data pins. In this case, there is 1
63          data pins in channel, that is connected FIFO Full Flag*/
64
65          /* xgpio_ioctl_data struct */
66          struct xgpio_ioctl_data gpio_out_data_WE_Reset;
67          /* Struct for gpio_out_data_WE_Reset */
68          memset(&gpio_out_data_WE_Reset, 0, sizeof(gpio_out_data_WE_Reset));
69          /* Clean gpio_out_data_WE_Reset */
70          gpio_out_data_WE_Reset.chan = 1;
                    /* The gpio channel */
71          gpio_out_data_WE_Reset.mask = 0x000003FF;
72          /* Gpio mask, that sets channel data pins. In this case, there are
                10
73          data pins in channel. The first 8 are connected to FIFO Data Out,
                the
74          nineth is connected to FIFO Write Enable Flag and the last one (MSB
                )
75          is connected FIFO Reset. FIFO Reset is ACTIVE-LOW */
76          gpio_out_data_WE_Reset.data = 0x000003FF;
77          /* Int variable stores data of channel */
78
79          /* Important: before we write we must set the proper data direction
                .
80          The data direction is 0 on boot (all write), but may have changed
                for
81          whatever reason.
82          Doing a write will set it only AFTER the operation, so it's no good
                .
83          The correct way to set the direction is using TRISTATE which sets
                the
84          data direction without reading. */
```

```
85
86          /* Set gpio_out_data_WE_Reset as TRISTATE */
87          if(0 > (ioctl(fd, XGPIO_TRISTATE, &gpio_out_data_WE_Reset)))
88          /* Check if Ioctl command that sets GPIO as tristate was done
               sucessfully */
89          {
90                  printf("\nioctl TRISTATE failed");
91                  close(fd);
92                  return errno;
93          }
94
95          /* Set gpio_in_fullFIFO as input */
96          if(0 > (ioctl(fd, XGPIO_IN, &gpio_in_fullFIFO)))
97          /* Check if Ioctl command, that sets gpio_in_fullFIFO as input,
98          was done sucessfully. It updates gpio_in_data_empty.data with data
               input value */
99          {
100                 printf("\nioctl failed\n");
101                 close(fd);
102                 return errno;
103         }
104         /* Check if FIFO is Full */
105         if(gpio_in_fullFIFO.data == 0x00000001)
106         {
107                 printf("\nFIFO is Full\n");
108                 close(fd);
109                 return 0;
110         }
111         else /* If FIFO is not Full, it can be write */
112         {
113                 sprintf(data_WE_Reset_C, "3%s", argv[1]);
114                 /* Build a string with this format: "3(hex data typed by
                       user)".
115                 It means in binary "11(user data in binary)", so this value
                       actives
116                 FIFO Write Enable */
117
118                 gpio_out_data_WE_Reset.data =  strtoul (data_WE_Reset_C,
                       NULL,16);
119                 /* Convert data_WE_Reset_C in a integer and put it in
120                 gpio_out_data_WE_Reset.data */
121
```

```
122                 /* Active FIFO Write Enable and write Data in FIFO data in
                        */
123                 if (0 > (ioctl(fd, XGPIO_OUT, &gpio_out_data_WE_Reset)))
124                 /* Check if Ioctl command, that sets gpio_out_data_WE_Reset
                        as output,
125                  was done sucessfully. It puts the gpio_out_RE_Reset.data
                        at output, */
126                 {
127                         printf("\nioctl failed\n");
128                         close(fd);
129                         return errno;
130                 }
131
132
133                 sprintf(data_WE_Reset_C, "2%s", argv[1]);
134                 /* Build a string with this format: "2(hex data typed by
                        user)".
135                 It means in binary "10(user data in binary)", so this value
136                 disable FIFO Write Enable */
137                 gpio_out_data_WE_Reset.data =  strtoul (data_WE_Reset_C,
                        NULL,16);
138                 /* Convert data_WE_Reset_C in a integer and put it in
139                 gpio_out_data_WE_Reset.data */
140
141                 /* Disable FIFO Write Enable */
142                 if (0 > (ioctl(fd, XGPIO_OUT, &gpio_out_data_WE_Reset)))
143                 /* Check if Ioctl command, that sets gpio_out_data_WE_Reset
144                 as output, was done sucessfully. It puts the
                        gpio_out_RE_Reset.
145                 data at output, */
146                 {
147                         printf("\nioctl failed\n");
148                         close(fd);
149                         return errno;
150                 }
151                 printf("\nThe value 0x%02X has been written on fifo\n\n",
152                 (gpio_out_data_WE_Reset.data - 0x00000200));
153                 /* Take FIFO write enable  out of data print */
154         }
155         close(fd);      /* Close the file descriptor of GPIO_DEV  */
156
157         return 0;
158 }
```

Código D.3: Writereset

```
/*********************************************************************
**      Writereset
**
**      Copyright (C) 2010 Rodrigo Neri de Souza
**
**      This program is free software: you can redistribute it and/or modify
**      it under the terms of the GNU General Public License as published by
**      the Free Software Foundation, either version 3 of the License, or
**      any later version.
**
**      This program is distributed in the hope that it will be useful,
**      but WITHOUT ANY WARRANTY; without even the implied warranty of
**      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
**      GNU General Public License for more details.
**
**      You should have received a copy of the GNU General Public License
**      along with this program.  If not, see <http://www.gnu.org/licenses/>.
**
*********************************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <linux/xgpio_ioctl.h>  /* Linux Xilinx GPIO library for ioctl functions */
#define GPIO_DEV "/dev/gpio1"   /* GPIO's name in /dev */


int main(void)
{
        /* Open the device */
        int fd = open(GPIO_DEV, O_RDWR);        /* The file descriptor of GPIO_DEV */

        /* Check if fd was opened successfully */
        if(fd==-1)
        {
                printf("\n\nUnable to open %s\nContinuing anyway...\n\n", GPIO_DEV);
        }

```

```
38          /* xgpio_ioctl_data struct */
39          struct xgpio_ioctl_data gpio_out_data_WE_Reset;
40          /* Struct for gpio_out_data_WE_Reset */
41          memset(&gpio_out_data_WE_Reset, 0, sizeof(gpio_out_data_WE_Reset));
42          /* Clean gpio_out_data_WE_Reset */
43          gpio_out_data_WE_Reset.chan = 1;
                    /* The gpio channel */
44          gpio_out_data_WE_Reset.mask = 0x000003FF;
45          /* Gpio mask, that sets channel data pins. In this case, there are
                10
46          data pins in channel. The first 8 are connected to FIFO Data Out,
                the
47          nineth is connected to FIFO Write Enable Flag and the last one (MSB
                )
48          is connected FIFO Reset. FIFO Resett is ACTIVE-LOW */
49          gpio_out_data_WE_Reset.data = 0x000003FF;
50          /* Int variable stores data of channel */
51
52          /* Important: before we write we must set the proper data direction
                .
53          The data direction is 0 on boot (all write), but may have changed
                for
54          whatever reason.
55          Doing a write will set it only AFTER the operation, so it's no good
                .
56          The correct way to set the direction is using TRISTATE which sets
                the
57          data direction without reading. */
58
59          /* Set gpio_out_data_WE_Reset as TRISTATE */
60          if(0 > (ioctl(fd, XGPIO_TRISTATE, &gpio_out_data_WE_Reset)))
61          /* Check if Ioctl command that sets GPIO as tristate was done
                sucessfully */
62          {
63                  printf("\nioctl TRISTATE failed");
64                  close(fd);
65                  return errno;
66          }
67
68          gpio_out_data_WE_Reset.data = 0x00000000;
69          /* Set gpio_out_data_WE_Reset as 0b0000000000 */
70
71          /* Active FIFO Reset */
```

```
72          if (0 > (ioctl(fd, XGPIO_OUT, &gpio_out_data_WE_Reset)))
73          /* Check if Ioctl command, that sets gpio_out_data_WE_Reset
74          as output, was done sucessfully. It puts the gpio_out_RE_Reset.data
                at output, */
75          {
76                  printf("\nioctl failed\n");
77                  close(fd);
78                  return errno;
79          }
80
81          gpio_out_data_WE_Reset.data = 0x00000200;
82          /* Set gpio_out_data_WE_Reset as 0b1000000000 */
83
84          /* Disable FIFO Reset */
85          if (0 > (ioctl(fd, XGPIO_OUT, &gpio_out_data_WE_Reset)))
86          /* Check if Ioctl command, that sets gpio_out_data_WE_Reset as
               output,
87          was done sucessfully. It puts the gpio_out_RE_Reset.data at output,
                */
88          {
89                  printf("\nioctl failed\n");
90                  close(fd);
91                  return errno;
92          }
93
94          close(fd);      /* Close the file descriptor of GPIO_DEV  */
95
96          printf("\nFIFO Write has been reset\n\n");
97
98          return 0;
99  }
```

Código D.4: Readreset

```
1   /*********************************************************************
2   **      Readreset
3   **
4   **      Copyright (C) 2010 Rodrigo Neri de Souza
5   **
6   **      This program is free software: you can redistribute it and/or modify
7   **      it under the terms of the GNU General Public License as published by
8   **      the Free Software Foundation, either version 3 of the License, or
9   **      any later version.
10  **
```

```
11 **     This program is distributed in the hope that it will be useful,
12 **     but WITHOUT ANY WARRANTY; without even the implied warranty of
13 **     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 **     GNU General Public License for more details.
15 **
16 **     You should have received a copy of the GNU General Public License
17 **     along with this program.  If not, see <http://www.gnu.org/licenses/>.
18 **
19 ***********************************************************************/
20 #include <stdio.h>
21 #include <fcntl.h>
22 #include <errno.h>
23 #include <linux/xgpio_ioctl.h>  /* Linux Xilinx GPIO library for ioctl
       functions */
24 #define GPIO_DEV "/dev/gpio0"   /* GPIO's name in /dev */
25
26
27 int main(void)
28 {
29         /* Open the device */
30         int fd = open(GPIO_DEV, O_RDWR);         /* The file descriptor of
               GPIO_DEV */
31
32         /* Check if fd was opened successfully */
33         if(fd==-1)
34         {
35                 printf("\n\nUnable to open %s\nTrying reset anyway...\n",
                       GPIO_DEV);
36         }
37
38         /* xgpio_ioctl_data struct */
39         struct xgpio_ioctl_data gpio_out_RE_Reset;
40         /* Struct for gpio_out_RE_Reset */
41         memset(&gpio_out_RE_Reset, 0, sizeof(gpio_out_RE_Reset));
42         /* Clean gpio_out_RE_Reset struct */
43         gpio_out_RE_Reset.chan = 2;                                    /*
               The gpio channel */
44         gpio_out_RE_Reset.mask = 0x00000003;
45         /* Gpio mask, that sets channel data pins. In this case,
46         there are 2 data pins in channel, the LSB, that is connected
47         to FIFO Read Enable flag, and the second one (MSB), that is
48         connected to FIFO Reset. FIFO Reset is ACTIVE-LOW */
49         gpio_out_RE_Reset.data = 0x00000003;
```

```
50          /* Int variable stores data of channel */
51
52          /* Important: before we write we must set the proper data direction
                .
53          The data direction is 0 on boot (all write), but may have changed
                for
54          whatever reason.
55          Doing a write will set it only AFTER the operation, so it's no good
                .
56          The correct way to set the direction is using TRISTATE which sets
                the
57          data direction without reading. */
58
59          /* Set gpio_out_RE_Reset as TRISTATE */
60          if(0 > (ioctl(fd, XGPIO_TRISTATE, &gpio_out_RE_Reset)))
61          /* Check if Ioctl command, that sets GPIO as tristate was,
62          done sucessfully */
63          {
64                  printf("\nioctl TRISTATE failed\n");
65                  close(fd);
66                  return errno;
67          }
68
69          gpio_out_RE_Reset.data = 0x00000000;
70          /* Set gpio_out_RE_Reset.data as 0b00 */
71
72          /* Active FIFO Reset */
73          if (0 > (ioctl(fd, XGPIO_OUT, &gpio_out_RE_Reset)))
74          /* Check if Ioctl command, that sets gpio_out_RE_Reset as output,
75          was done sucessfully. It puts the gpio_out_RE_Reset.data at output,
                */
76          {
77                  printf("\nioctl failed\n");
78                  close(fd);
79                  return errno;
80          }
81
82          gpio_out_RE_Reset.data = 0x00000002;
83          /* Set gpio_out_RE_Reset.data as 0b10 */
84
85          /* Disable FIFO Reset */
86          if (0 > (ioctl(fd, XGPIO_OUT, &gpio_out_RE_Reset)))
87          /* Check if Ioctl command, that sets gpio_out_RE_Reset as output,
```

```
88          was done sucessfully. It puts the gpio_out_RE_Reset.data at output,
                */
89          {
90                  printf("\nioctl failed\n");
91                  close(fd);
92                  return errno;
93          }
94
95          close(fd);      /* Close the file descriptor of GPIO_DEV  */
96
97          printf("\nFIFO Read has been reset\n\n");
98
99          return 0;
100 }
```

Código D.5: Readfifo

```
1   /*********************************************************************
2   **      Readfifo
3   **
4   **      Copyright (C) 2010 Rodrigo Neri de Souza
5   **
6   **      This program is free software: you can redistribute it and/or modify
7   **      it under the terms of the GNU General Public License as published by
8   **      the Free Software Foundation, either version 3 of the License, or
9   **      any later version.
10  **
11  **      This program is distributed in the hope that it will be useful,
12  **      but WITHOUT ANY WARRANTY; without even the implied warranty of
13  **      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14  **      GNU General Public License for more details.
15  **
16  **      You should have received a copy of the GNU General Public License
17  **      along with this program.  If not, see <http://www.gnu.org/licenses/>.
18  **
19  *********************************************************************/
20  #include <stdio.h>
21  #include <fcntl.h>
22  #include <errno.h>
23  #include <linux/xgpio_ioctl.h>
24  /* Linux Xilinx GPIO library for ioctl functions */
25  #define GPIO_DEV "/dev/gpio0"   /* GPIO's name in /dev */
26
27
```

```
28 int main(void)
29 {
30         /* Open the device */
31         int fd = open(GPIO_DEV, O_RDWR);          /* The file descriptor of
               GPIO_DEV */
32         char data[3];                             /* Char to store the
               received data */
33
34         /* Check if fd was opened successfully */
35         if(fd==-1)
36         {
37                 printf("\n\nUnable to open %s\nContinuing anyway...\n\n",
                       GPIO_DEV);
38         }
39
40         /* xgpio_ioctl_data struct */
41         struct xgpio_ioctl_data gpio_out_RE_Reset;
42         /* Struct for gpio_out_RE_Reset */
43         memset(&gpio_out_RE_Reset, 0, sizeof(gpio_out_RE_Reset));
44         /* Clean gpio_out_RE_Reset struct */
45         gpio_out_RE_Reset.chan = 2;                                    /*
               The gpio channel */
46         gpio_out_RE_Reset.mask = 0x00000003;
47         /* Gpio mask, that sets channel data pins. In this case,
48         there are 2 data pins in channel, the LSB, that is connected to
49         FIFO Read Enable flag, and the second one (MSB), that is connected
50         to FIFO Reset. FIFO Resett is ACTIVE-LOW */
51         gpio_out_RE_Reset.data = 0x00000003;
52         /* Int variable stores data of channel */
53
54         /* xgpio_ioctl_data struct */
55         struct xgpio_ioctl_data gpio_in_data_empty;
56         /* Struct for gpio_in_data_empty */
57         memset(&gpio_in_data_empty, 0, sizeof(gpio_in_data_empty));
58         /* Clean gpio_in_data_empty struct */
59         gpio_in_data_empty.chan = 1;                                   /*
               The gpio channel */
60         gpio_in_data_empty.mask = 0x000001FF;
61         /* Gpio mask, that sets channel data pins. In this case, there are
               9
62         data pins in channel, the first 8, that are connected to FIFO Data
               Out,
63         and the last one (MSB), that is connected to FIFO Empty Flag */
```

```
64
65
66          /* Important: before we write we must set the proper data direction
            .
67          The data direction is 0 on boot (all write), but may have changed
            for
68          whatever reason.
69          Doing a write will set it only AFTER the operation, so it's no good
            .
70          The correct way to set the direction is using TRISTATE which sets
            the
71          data direction without reading. */
72
73          /* Set gpio_out_RE_Reset as TRISTATE */
74          if(0 > (ioctl(fd, XGPIO_TRISTATE, &gpio_out_RE_Reset)))
75          /* Check if Ioctl command, that sets gpio_out_RE_Reset as tristate,
76          was done sucessfully */
77          {
78                  printf("\nioctl TRISTATE failed\n");
79                  close(fd);
80                  return errno;
81          }
82
83          /* Set gpio_in_data_empty as input */
84          if(0 > (ioctl(fd, XGPIO_IN, &gpio_in_data_empty)))
85          /* Check if Ioctl command, that sets gpio_in_data_empty as input,
86          was done sucessfully.  It updates gpio_in_data_empty.data with
87          data input value  */
88          {
89                  printf("\nioctl failed\n");
90                  close(fd);
91                  return errno;
92          }
93
94          /* Check if FIFO is empty. The FIFO Empty Flag is the MSB of
95          gpio_in_data_empty.data */
96          if(gpio_in_data_empty.data > 0xFF)
97          /* gpio_in_data_empty.data greater than 0XFF means FIFO Empty Flag
98          is 1, so gpio_in_data_empty.data is 0x1FF */
99          {
100                 printf("\nFIFO is Empty\n\n");
101                 close(fd);
102                 return 0;
```

```
103         }
104         else    /* If FIFO is not empty, it can be read */
105         {
106                 gpio_out_RE_Reset.data = 0x00000003;
107                 /* Set gpio_out_RE_Reset.data as 0b11 */
108
109                 /* Active FIFO Read Enable */
110                 if (0 > (ioctl(fd, XGPIO_OUT, &gpio_out_RE_Reset)))
111                 /* Check if Ioctl command, that sets gpio_out_RE_Reset as
                        output,
112                  was done sucessfully. It puts the gpio_out_RE_Reset.data
                         at output, */
113                 {
114                         printf("\nioctl failed\n");
115                         close(fd);
116                         return errno;
117                 }
118
119                 gpio_out_RE_Reset.data = 0x00000002;
120                 /* Set gpio_out_RE_Reset.data as 0b10 */
121
122                 /* Disable FIFO Read Enable */
123                 if (0 > (ioctl(fd, XGPIO_OUT, &gpio_out_RE_Reset)))
124                 /* Check if Ioctl command, that sets gpio_out_RE_Reset as
                        output,
125                 was done sucessfully. It puts the gpio_out_RE_Reset.data at
                         output, */
126                 {
127                         printf("\nioctl failed\n");
128                         close(fd);
129                         return errno;
130                 }
131                 /* Read Data from FIFO data out */
132                 if(0 > (ioctl(fd, XGPIO_IN, &gpio_in_data_empty)))
133                 /* Check if Ioctl command, that sets gpio_in_data_empty as
                        input,
134                 was done sucessfully.It updates gpio_in_data_empty.data
                        with
135                 data input value  */
136                 {
137                         printf("\nioctl failed\n");
138                         close(fd);
139                         return errno;
```

```
140                 }
141                 /* Print the data read from FIFO */
142                 if(gpio_in_data_empty.data > 0xff)
143                 /* gpio_in_data_empty.data greater than 0XFF means
144                 FIFO Empty Flag is 1 */
145                 {
146                         sprintf(data, "%02X", (gpio_in_data_empty.data - 0
                            x00000100));
147                         /* Take FIFO Empty Flag out of data print */
148                         printf("\nData read from FIFO = 0x%s\n\n",data);
149                 }
150                 else
151                 /* gpio_in_data_empty.data is not greater than 0XFF
152                  means FIFO Empty Flag is 0 */
153                 {
154                         sprintf(data, "%02X",  gpio_in_data_empty.data);
155                         /* It is not necessary take FIFO Empty Flag out of
                            data print,
156                         because FIFO Empty Flag is 0 */
157                         printf("\nData read from FIFO = 0x%s\n\n",data);
158                 }
159         }
160         close(fd);      /* Close the file descriptor of GPIO_DEV  */
161
162         return 0;
163 }
```

Código D.6: Readfifofile

```
1  /*********************************************************************
2  **     Readfifofile
3  **
4  **     Copyright (C) 2010 Rodrigo Neri de Souza
5  **
6  **     This program is free software: you can redistribute it and/or modify
7  **     it under the terms of the GNU General Public License as published by
8  **     the Free Software Foundation, either version 3 of the License, or
9  **     any later version.
10 **
11 **     This program is distributed in the hope that it will be useful,
12 **     but WITHOUT ANY WARRANTY; without even the implied warranty of
13 **     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 **     GNU General Public License for more details.
15 **
```

```
16 **     You should have received a copy of the GNU General Public License
17 **     along with this program.  If not, see <http://www.gnu.org/licenses/>.
18 **
19 ***********************************************************************/
20 #include <stdio.h>
21 #include <fcntl.h>
22 #include <errno.h>
23 #include <ctype.h>
24 #include <stdbool.h>
25 #include <linux/xgpio_ioctl.h>
26 /* Linux Xilinx GPIO library for ioctl functions */
27 #define GPIO_DEV "/dev/gpio0"   /* GPIO's name in /dev */
28 #define size 200                /* Limit of letter written */
29
30 int main(int argc, char *argv[])
31 {
32
33         FILE *pFile;            /* Pointer to File */
34         char data[3];           /* Char to store the received data */
35         int data_int[size];     /* Int array to store all the received data
               */
36         char q[1];              /* Char to store the answer of question */
37         char path[50];          /* Char to store the File name */
38         int n = 0;              /* Int to count the amount of hex number
               from FIFO */
39         bool flag = false;
40         /* Bool to check if FIFO Empty Flag must be taken out */
41         int i;                  /* Int to run data_int */
42
43         /* Check if user entered file name */
44         if(argc < 2)
45         {
46                 printf("\nInvalid file name. Please enter a valid file name
                       . Example: file.txt\n\n");
47                 return 0;
48         }
49         /* Create the file */
50         sprintf(path, "/var/tmp/%s", argv[1]);
51         pFile = fopen (path,"wb");
52
53         /* Open the device */
54         int fd = open(GPIO_DEV, O_RDWR);        /* The file descriptor of
               GPIO_DEV */
```

```
55
56          /* Check if fd was opened successfully */
57          if(fd==-1)
58          {
59                  printf("\n\nUnable to open %s\nContinuing anyway...\n\n",
                        GPIO_DEV);
60          }
61
62
63          /* xgpio_ioctl_data struct */
64          struct xgpio_ioctl_data gpio_out_RE_Reset;
65          /* Struct for gpio_out_RE_Reset */
66          memset(&gpio_out_RE_Reset, 0, sizeof(gpio_out_RE_Reset));
67          /* Clean gpio_out_RE_Reset struct */
68          gpio_out_RE_Reset.chan = 2;                                    /*
                The gpio channel */
69          gpio_out_RE_Reset.mask = 0x00000003;
70          /* Gpio mask, that sets channel data pins. In this case,
71          there are 2 data pins in channel, the LSB, that is connected
72          to FIFO Read Enable flag, and the second one (MSB), that is
73          connected to FIFO Reset. FIFO Resett is ACTIVE-LOW */
74          gpio_out_RE_Reset.data = 0x00000003;
75          /* Int variable stores data of channel */
76
77          /* xgpio_ioctl_data struct */
78          struct xgpio_ioctl_data gpio_in_data_empty;
79          /* Struct for gpio_in_data_empty */
80          memset(&gpio_in_data_empty, 0, sizeof(gpio_in_data_empty));
81          /* Clean gpio_in_data_empty struct */
82          gpio_in_data_empty.chan = 1;                                   /*
                The gpio channel */
83          gpio_in_data_empty.mask = 0x000001FF;
84          /* Gpio mask, that sets channel data pins. In this case,
85          there are 9 data pins in channel, the first 8, that are
86          connected to FIFO Data Out, and the last one (MSB), that
87          is connected to FIFO Empty Flag */
88
89
90          /* Important: before we write we must set the proper data direction
                .
91          The data direction is 0 on boot (all write), but may have changed
                for
92          whatever reason.
```

```
93          Doing a write will set it only AFTER the operation, so it's no good
                .
94          The correct way to set the direction is using TRISTATE which sets
                the
95          data direction without reading. */
96
97          /* Set gpio_out_RE_Reset as TRISTATE */
98          if(0 > (ioctl(fd, XGPIO_TRISTATE, &gpio_out_RE_Reset)))
99          /* Check if Ioctl command, that sets gpio_out_RE_Reset as tristate,
100          was done sucessfully */
101         {
102                 printf("\nioctl TRISTATE failed\n");
103                 close(fd);
104                 return errno;
105         }
106
107         /* Set gpio_in_data_empty as input */
108         ioctl(fd, XGPIO_IN, &gpio_in_data_empty);
109         /* Ioctl command, that sets gpio_in_data_empty as input.
110         It updates gpio_in_data_empty.data with data input value  */
111
112         while(1)
113         {
114                 /* Check if FIFO is empty. The FIFO Empty Flag is the
115                 MSB of gpio_in_data_empty.data */
116                 if(gpio_in_data_empty.data > 0xFF)
117                 /* gpio_in_data_empty.data greater than 0XFF means FIFO
118                 Empty Flag is 1, so gpio_in_data_empty.data is 0x1FF */
119                 {
120
121
122                         if(flag)
123                         {
124                                 data_int[n-1] = data_int[n-1] -  0x00000100
                                        ;
125                                 /* Take FIFO Empty Flag out of real data */
126                                 flag = false;
127                         }
128
129                         printf("\nFIFO is Empty\n\n");
130                         printf("\nPress enter to still read, type E to exit
                                .\n");
131                         fgets(q,2,stdin);
```

```
132                     if((!strcmp(q,"e"))||(!strcmp(q,"E")))
133                     {
134                             if (pFile!=NULL)
135                             {
136                                     /* Write data_int in file */
137                                     for(i = 0; i < n;i++)   /* Run all
                                            the data_int array */
138                                     {
139                                             sprintf(data, "%02X\n",
                                                    data_int[i]);
140                                             fputs (data,pFile);     /*
                                                    Put the char data in
                                                    file */
141                                             printf("\nThe data 0x%s
                                                    read from FIFO will be
                                                    written
142                                             in the file %s\n\n",data,
                                                    path);
143                                     }
144                             }
145                             else
146                             {
147                                     printf("\nIt is not possible to
                                            create %s\n\n",path);
148                                     close(pFile);
149                                     close(fd);
150                             }
151                             close(pFile);
152                             printf("\nThe file %s was closed\n\n",path)
                                    ;
153                             close(fd);
154                             return 0;
155                     }
156                     /* Set Ioctl command to check if FIFO is empty */
157                     ioctl(fd, XGPIO_IN, &gpio_in_data_empty);
158                     /* Ioctl command, that sets gpio_in_data_empty as
                            input.
159                     It updates gpio_in_data_empty.data with data input
                            value   */
160
161             }
162
163             else    /* If FIFO is not empty, it can be read */
```

```
164                 {
165
166                         flag = true;
167
168                         gpio_out_RE_Reset.data = 0x00000003;
169                         /* Set gpio_out_RE_Reset.data as 0b11 */
170
171                         /* Active FIFO Read Enable */
172                         ioctl(fd, XGPIO_OUT, &gpio_out_RE_Reset);
173                         /* Ioctl command, that sets gpio_out_RE_Reset as
                                output.
174                         It puts the gpio_out_RE_Reset.data at output, */
175
176                         gpio_out_RE_Reset.data = 0x00000002;
177                         /* Set gpio_out_RE_Reset.data as 0b10 */
178
179                         /* Disable FIFO Read Enable */
180                         ioctl(fd, XGPIO_OUT, &gpio_out_RE_Reset);
181                         /* Ioctl command, that sets gpio_out_RE_Reset as
                                output.
182                         It puts the gpio_out_RE_Reset.data at output, */
183
184                         /* Read Data from FIFO data out */
185                         ioctl(fd, XGPIO_IN, &gpio_in_data_empty);
186                         /* Ioctl command, that sets gpio_in_data_empty as
                                input.
187                         It updates gpio_in_data_empty.data with data input
                                value */
188
189                         data_int[n] = gpio_in_data_empty.data;
190                         /* Put data read from FIFO in data_int array */
191                         n++;
192                 }
193         }
194
195         close(pFile);   /* Close the pointer to File */
196
197         close(fd);      /* Close the file descriptor of GPIO_DEV */
198
199         return 0;
200 }
```

Referências Bibliográficas

BOA. *Boa Webserver*. 23 feb. 2005. Disponível em: <http://www.boa.org/>. Acesso em: 18 mai. 2010.

IEEE. *IEEE 1149.1*. 2001.

JTA - TELNET/SSH FOR THE JAVA(TM) PLATFORM. *Installing the Applet*. 15 set. 2006. Disponível em: <http://javassh.org/space/Installing+the+Applet>. Acesso em: 1 jun. 2010.

LWIP. *A Lightweight TCP/IP stack*. 2010. Disponível em: <http://savannah.nongnu.org/projects/lwip/>. Acesso em: 19 fev. 2010.

MEYER-BAESE, U. *Digital Signal Processing with Field Programmable Gate Arrays*. 2. ed. [S.l.]: Springer, 2007.

OPENCORES. *MB-Lite: Overview*. 2010. Disponível em: <http://opencores.org/project,mblite>. Acesso em: 15 fev. 2010.

OPENCORES. *Wishbone High Performance Z80: Overview*. 2010. Disponível em: <http://opencores.org/project,wb_z80>. Acesso em: 15 fev. 2010.

OPENCORES. *ZPU: the worlds smallest 32 bit CPU with GCC toolchain: Overview*. 2010. Disponível em: <http://opencores.org/project,zpu>. Acesso em: 15 fev. 2010.

PETALINUX. *PetaLinux User Guide*. 2010. Disponível em: <http://www.petalogix.com/univ/documentation/userguide>. Acesso em: 16 mar. 2010.

PETALOGIX. *Adding or Creating New Applications*. 2010. Disponível em: <http://www.petalogix.com/resources/documentation/petalinux/userguide/Customising/UserApps>. Acesso em: 17 mar. 2010.

PETALOGIX. *Downloading Files*. 2010. Disponível em: <http://www.petalogix.com/resources/documentation/petalinux/userguide/Bootloaders/UBoot/UBDownload>. Acesso em: 17 mar. 2010.

PETALOGIX. *Introducing PetaLinux SDK: The industry standard for Embedded Linux on Xilinx MicroBlaze*. 2010. Disponível em: <http://www.petalogix.com/products/petalinux/files/petalinux-sdk-brochure.pdf>. Acesso em: 22 fev. 2010.

PETALOGIX. *PetaLinux Bootloader Solutions*. 2010. Disponível em: <http://www.petalogix.com/resources/documentation/petalinux/userguide/Bootloaders>. Acesso em: 17 mar. 2010.

PETALOGIX. *PetaLinux Tools*. 2010. Disponível em: <http://www.petalogix.com/resources/documentation/petalinux/userguide/petalinux-tools/PetaLinuxTools>. Acesso em: 17 mar. 2010.

PETALOGIX. *Supported FPGA and CPU families*. 2010. Disponível em: <http://www.petalogix.com/about/supported-fpga-and-cpu-families>. Acesso em: 14 mar. 2010.

PETALOGIX. *Using the Xilinx 16550 UART*. 2010. Disponível em: <http://www.petalogix.com/resources/documentation/petalinux/userguide/AdvancedTopics/Using16550Uart>. Acesso em: 17 mar. 2010.

PETALOGIX. *Welcome to Petalogix*. 2010. Disponível em: <http://www.petalogix.com/>. Acesso em: 22 fev. 2010.

RAD INKS (PVT). *The Rad FTP Applet: Online Demo*. 2010. Disponível em: <http://radinks.net/ftp/applet/demo.php>. Acesso em: 1 jun. 2010.

SASS, R.; SCHMIDT, A. G. *Embedded Systems Design with Platform FPGAs*. 1. ed. [S.l.]: Morgan Kaufmann, 2010.

TANENBAUM, A. S. *Computer networks*. 4. ed. [S.l.]: Prentice Hall, 2002.

UIP. *Main Page*. 2010. Disponível em: <http://www.sics.se/~adam/uip/index.php/Main_Page>. Acesso em: 18 fev. 2010.

UIP. *Porting uIP*. 6 ago. 2007. Disponível em: <http://www.sics.se/~adam/uip/index.php/Porting_uIP>. Acesso em: 18 fev. 2010.

UNIVERSITY OF QUEENSLAND. *VHDL IP Stack*. Brisbane, Australia., 16 jul. 2001. Disponível em: <http://www.itee.uq.edu.au/~peters/xsvboard/stack/stack.htm>. Acesso em: 11 fev. 2010.

WIKIPEDIA. *Soft microprocessor*. 13 fev. 2010. Disponível em: <http://en.wikipedia.org/wiki/Softcore_processor>. Acesso em: 14 fev. 2010.

WIKIPEDIA. *Internet Protocol Suite*. 16 jun. 2010. Disponível em: <http://en.wikipedia.org/wiki/Internet_Protocol_Suite>. Acesso em: 18 jun. 2010.

WIKIPEDIA. *Boundary Scan*. 21 may. 2010. Disponível em: <http://en.wikipedia.org/wiki/Boundary_scan>. Acesso em: 30 jul. 2010.

WIKIPEDIA. *Ethernet*. 25 jul. 2010. Disponível em: <http://en.wikipedia.org/wiki/Ethernet>. Acesso em: 30 jul. 2010.

WIKIPEDIA. *Field-programmable gate array*. 30 jul. 2010. Disponível em: <http://en.wikipedia.org/wiki/Field-programmable_gate_array>. Acesso em: 31 jul. 2010.

WIKIPEDIA. *Joint Test Action Group*. 30 jul. 2010. Disponível em: <http://en.wikipedia.org/wiki/Joint_Test_Action_Group>. Acesso em: 31 jul. 2010.

WIKIPEDIA. *Programmable Interrupt Controller*. 7 jul. 2010. Disponível em: <http://en.wikipedia.org/wiki/Programmable_Interrupt_Controller>. Acesso em: 30 jul. 2010.

XILINX. *MicroBlaze Processor Reference Guide*. 26 out. 2009. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf>. Acesso em: 2 mar. 2010.

YOUTUBE. *Xilinx EDK Tutorial: Integrating EDK and ISE Projects*. 6 feb. 2009. Disponível em: <http://www.youtube.com/watch?v=R5wZ89BcBPo>. Acesso em: 13 abr. 2010.