Instituto Superior de Engenharia do Porto

Acesso remoto via infra-estrutura de teste IEEE1149.1 a dispositivos lógicos programáveis

Valentim Peixoto de Sousa

Dissertação para a obtenção do grau de Mestre em

Engenharia Informática

Área de Especialização em

Arquitecturas, Sistemas e redes

Orientador: Professor Doutor Manuel Gradim de Oliveira Gericota

Co-orientador: Mestre Paulo Alexandre Duarte Ferreira

Júri:

Presidente:

Doutora Maria de Fátima Coutinho Rodrigues, Departamento Engenharia Informática/ISEP

Vogais:

Doutor José Manuel Magalhães Cruz, Departamento de Engenharia Informática/FEUP

Doutor Manuel Gradim de Oliveira Gericota, Departamento Engenharia Electrotécnica /ISEP

Mestre Paulo Alexandre Duarte Ferreira, Departamento Engenharia Informática/ISEP

Porto, Junho de 2011

Agradecimentos

A escrita de uma dissertação exige um esforço elevado. Embora uma dissertação seja um trabalho individual, não deixa de ser o resultado de vários contributos directos ou indirectos. Por este motivo, penso que todos os que contribuíram para o seu culminar merecem um sentimento de profunda gratidão.

Desta forma, gostaria de prorrogar os meus agradecimentos em primeiro lugar aos meus orientadores, Manuel Gericota e Paulo Ferreira, por toda a atenção, cuidado, disponibilidade e espírito crítico em todas as fases deste projecto, sendo incansáveis e rigorosos durante todo o período de desenvolvimento da dissertação.

Aos meus colegas de mestrado, em especial ao Pedro Teixeira, Rui Eusébio, Flávio Oliveira, Joaquim Teixeira, Sílvio Lopes, Armindo Felgueiras e Rui Felgueiras pela tolerância e concertação de esforços, que durante estes últimos anos me acompanharam neste duro percurso. Muito obrigado pela vossa ajuda e incentivos.

A uma pessoa muito especial que esteve sempre presente suportando os maus momentos, incentivando-me a não desanimar face às adversidades, à Sílvia Magalhães.

O meu muito obrigado à minha família, aos meus pais e meus irmãos, que sempre me apoiaram e são o grande incentivo para todo o meu esforço.

Por último, quero agradecer a todos aqueles que mesmo não estando referenciados contribuíram directa ou indirectamente para a realização deste trabalho.

Obrigado a todos.

Resumo

Actualmente verifica-se que a complexidade dos sistemas informáticos tem vindo a aumentar, fazendo parte das nossas ferramentas diárias de trabalho a utilização de sistemas informáticos e a utilização de serviços online. Neste âmbito, a internet obtém um papel de destaque junto das universidades, ao permitir que alunos e professores possam interagir mais facilmente. A internet e a educação baseada na Web vêm oferecer acesso remoto a qualquer informação independentemente da localização ou da hora. Como consequência, qualquer pessoa com uma ligação à internet, ao poder adquirir informações sobre um determinado tema junto dos maiores peritos, obtém vantagens significativas.

Os laboratórios remotos são uma solução muito valorizada no que toca a interligar tecnologia e recursos humanos em ambientes que podem estar afastados no tempo ou no espaço. A criação deste tipo de laboratórios e a sua utilidade real só é possível porque as tecnologias de comunicação emergentes têm contribuído de uma forma muito relevante para melhorar a sua disponibilização à distância. A necessidade de criação de laboratórios remotos torna-se imprescindível para pesquisas relacionadas com engenharia que envolvam a utilização de recursos escassos ou de grandes dimensões.

Apoiado neste conceito, desenvolveu-se um laboratório remoto para os alunos de engenharia que precisam de testar circuitos digitais numa carta de desenvolvimento de hardware configurável, permitindo a utilização deste recurso de uma forma mais eficiente.

O trabalho consistiu na criação de um laboratório remoto de baixo custo, com base em linguagens de programação *open source*, sendo utilizado como unidade de processamento um *router* da ASUS com o *firmware OpenWrt*. Este *firmware* é uma distribuição Linux para sistemas embutidos.

Este laboratório remoto permite o teste dos circuitos digitais numa carta de desenvolvimento de hardware configurável em tempo real, utilizando a interface JTAG.

O laboratório desenvolvido tem a particularidade de ter como unidade de processamento um *router*. A utilização do *router* como servidor é uma solução muito pouco usual na implementação de laboratórios remotos. Este *router*, quando comparado com um computador normal, apresenta uma capacidade de

processamento e memória muito inferior, embora os testes efectuados provassem que apresenta um desempenho muito adequado às expectativas.

Palavras-Chave: *OpenWrt*, JTAG, laboratórios remotos, pseudo-terminais.

Abstract

Currently, the complexity of computer systems is growing fast. They are, in conjunction with the online services they enable to offer, an indispensable part of our lives. In this context, internet plays a fundamental role at universities, allowing an easier interaction between students and teachers. Web-based education offers remote ways of accessing information without spatial or temporal constraints. As a consequence, any one with an internet connection may exchange information about a certain subject directly with an expert.

In that context, remote laboratories are an excellent solution to connect technology and human resources in environments that may be far away. Their viability is mainly due to the fast development of emergent communication technologies, and to its impact in diminishing distances.

Remote laboratories are an essential part of Web-based engineering lecturing, enabling future engineers to work with lab tools that otherwise would not be accessible to them. Furthermore, they allow sharing expensive resources among multiple community researchers.

During this work, a remote laboratory for the test of printed circuit boards and the programming/configuration of programmable logic devices and memories through a JTAG interface was developed. This laboratory allows students to gain access to a real development board through internet, from anywhere and at anytime. This allows an efficient, more flexible, use of the available resources, while increases the availability of laboratory resources beyond lab classes.

The developed remote laboratory is based on open source software and on a cheap router with OpenWrt firmware, which acts as a processing unity, one of the innovations of this work. This firmware is a Linux distribution targeted at embedded systems.

The router acts as a server, which is not a common solution in the remote laboratory implementation. When compared to a normal computer, the router has a lower processing and memory capacity. However, the tests proved that it has a very good performance, and is able to cope with the demands of the remote laboratory.

Keywords: OpenWrt, JTAG, remote laboratories, pseudo-terminal.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

CCI	Carta de circuito impresso
CI	Circuito integrado
CPLD	Complex Programmable Logic Devices
FPGA	Field-Programmable Gate Array
Glibc	GNU C Library
Glibc++	GNU C++ Library
GPL	General Public License
HDL	Hardware Description Language
IPC	Inter Process Communication
JTAG	Joint Test Action Group
LAMP	Linux, Apache, MySQL, PHP
LAN	Local Area Network
PC	Personal Computer
PHP	PHP: Hypertext Preprocessor
PLD	Programmable Logic Devices
RAM	Random Access Memory
ROM	Read-Only Memory
SDK	Software Development Kit
SQL	Structured Query Language
SOAP	Simple Object Access Protocol
SoC	System-on-a-Chip
SSH	Secure Shell
SVF	Serial Vector Format
TAP	Test Access Port
TCP	Transmission Datagram Protocol

TCK	Test Clock
TDI	Test Data Input
TDO	Test Data Output
TMS	Test Mode Select
TRST	Test Reset
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
VM	Virtual Machine
WWW	World Wide Web

1 Introdução

A presente dissertação expõe o trabalho desenvolvido no âmbito da Tese de Mestrado em Engenharia Informática. Nela está contido o desenvolvimento do tema, sendo descritos todos os passos do processo e devidamente justificadas as opções tomadas e respectiva sustentação teórica.

O tema da presente dissertação é o acesso remoto via infra-estrutura de teste IEEE1149.1 [IEEE Std 1149.1, 2001] a dispositivos lógicos programáveis. Os dispositivos lógicos programáveis são componentes de hardware cujas funcionalidades podem ser definidas pelo utilizador através de ficheiros específicos de programação. Estes ficheiros de programação contêm instruções que serão interpretadas com a finalidade de modificar o funcionamento dos dispositivos.

Ao longo do trabalho descrito nesta dissertação foi desenvolvida uma infra-estrutura que permite a programação remota, via Internet, de dispositivos lógicos programáveis e a sua verificação funcional, pós programação, através da interface definida na norma IEEE 1149.1. Esta pode ser utilizada não só para programar o dispositivo de hardware, mas igualmente para a realização de uma série de testes que permitem verificar a correcta implementação das funcionalidades pretendidas.

Para isso, foi criada uma infra-estrutura que permite a programação remota, dos dispositivos lógicos programáveis. Esta infra-estrutura englobou a criação de um cliente e um servidor utilizando *sockets* como meio de comunicação. O cliente fornece a interface gráfica pela qual o utilizador pode controlar as tarefas desempenhadas pelo servidor, enviando comandos e ficheiros de programação que serão usados para programar/testar o dispositivo lógico programável, que neste caso é uma *Field--Programmable Gate Array* (FPGA) ou uma *Complex Programmable Logic Device* (CPLD). O servidor interage com os dispositivos lógicos programáveis através da referida interface.

A lógica de negócio implementada no servidor foi adaptada para ser executada num *router*, que actua como unidade de processamento, tendo as aplicações necessárias para a interacção com os dispositivos a programar. A utilização de um *router* como servidor é uma solução pouco usual na implementação de laboratórios remotos, mas ficou demonstrado ser uma solução fiável e com um bom desempenho.

Com este trabalho pretende-se contribuir para o fomento do conhecimento na área desta dissertação.

1.1 Contextualização e objectivos

Os laboratórios remotos são cada vez mais uma aposta das instituições de ensino visto que proporcionam novas oportunidades e diferentes formas de aprendizagem aos seus alunos. A criação destes laboratórios resulta de uma inquietação em modernizar métodos de ensino e em facilitar o acesso à informação por parte dos alunos, sem nunca descurar a parte financeira, tentando minimizar os recursos dispendidos.

Aproveitando o forte desenvolvimento tecnológico da informática e das comunicações, as instituições de ensino procuram uma maior proximidade para com os seus educandos, oferecendo formas de aprendizagem mais versáteis e de melhor qualidade.

As novas formas de ensino procuram ir ao encontro das necessidades do aluno, proporcionando-lhe o acesso facilitado à informação, dando-lhe liberdade de acesso a qualquer altura e em qualquer lugar.

Nesse contexto, surgiu a ideia de criar um laboratório remoto de baixo custo que possibilitasse a programação ou o teste de dispositivos lógicos programáveis do tipo FPGA ou CPLD, através da interface definida na norma IEEE 1149.1 [IEEE Std 1149.1, 2001]. Esta interface pode ser utilizada não só para programar o dispositivo, mas igualmente para a realização de uma série de testes que permitem verificar a correcta implementação da funcionalidade pretendida.

A norma IEEE 1149.1 define os detalhes de acesso à infra-estrutura de teste de hardware, permitindo o teste estrutural da carta de circuito impresso sem necessidade de acesso físico aos pinos dos componentes ou às pistas. Esta norma define que cada circuito integrado deve possuir um conjunto de 4 pinos para acesso ao teste, o porto de teste, vulgarmente designado de porto JTAG, abreviatura de *Joint Test Action Group*, através dos quais é possível o acesso à infra-estrutura interna de teste e o controlo do seu funcionamento. Será através da porta JTAG que será efectuada a programação dos dispositivos lógicos programáveis e a sua verificação funcional.

Assim sendo, foi desenvolvido um laboratório remoto de baixo custo utilizando linguagens de programação *open source*.

Os objectivos específicos deste trabalho são:

- Estudar a norma IEEE 1149.1;
- Investigar qual o interesse dos laboratórios remotos;
- Investigar qual a plataforma/linguagem de programação que melhor se adequa à resolução do problema proposto;
- Analisar quais os programas que poderão ser utilizados e que melhor se adaptem ao esquema proposto para a resolução do problema;
- Criação de um laboratório de baixo custo que possibilite programar remotamente um dispositivo lógico programável.

Procurando satisfazer os pontos mencionados anteriormente e após uma vasta pesquisa, foi elaborado um laboratório remoto de baixo custo, com base em linguagens de programação *open source*, sendo utilizado como unidade de processamento um *router* da ASUS. Este laboratório remoto possibilita a programação dos dispositivos lógicos programáveis utilizando a interface JTAG.

Após o desenvolvimento do laboratório foram efectuados vários ensaios e foi verificada a viabilidade de utilização desta arquitectura, mostrando-se ser uma solução viável e com uma capacidade de processamento adequada às exigências do sistema.

1.2 Plano de trabalho

Foi elaborado um plano de modo a manter o trabalho organizado e com um fio condutor, cuja calendarização, tendo em conta os objectivos definidos, envolveu as seguintes etapas:

- Ambientação com o projecto proposto;
- Estudo do estado da arte;
- Investigação sobre o desenvolvimento de software para a plataforma *OpenWrt*;
- Elaboração de protótipos;
- Definição da arquitectura a adoptar para o sistema a desenvolver;
- Concepção e desenvolvimento do laboratório de experimentação remota;
- Início da elaboração do documento da dissertação;
- Fase de testes e validação do sistema;
- Conclusão do documento final da dissertação.

1.3 Estrutura do documento

Esta dissertação está dividida em seis capítulos.

Neste **primeiro capítulo** é feito o enquadramento da dissertação, são enumerados os objectivos que se pretenderam alcançar e é detalhado todo o planeamento do trabalho desenvolvido.

O **segundo capítulo** descreve toda a pesquisa efectuada sobre a tecnologia *open source*, sobre alguns laboratórios remotos já existentes e sobre a sua empregabilidade, uma descrição do que são FPGAs e CPLDs, da tecnologia JTAG, do *router* utilizado como unidade de processamento e do *firmware* utilizado no *router*.

No **terceiro capítulo** serão abordadas as opções técnicas e as escolhas efectuadas para o desenvolvimento do novo laboratório remoto.

O **quarto capítulo** aborda a arquitectura do novo laboratório remoto assim como todo o processo de desenvolvimento e as dificuldades que surgiram.

No **quinto capítulo** são demonstrados os resultados e confrontados com os objectivos propostos para esta dissertação.

Por último, são apresentadas as conclusões, no **sexto capítulo,** onde são também apontadas algumas considerações e propostas para trabalhos futuros.

2 Conceitos e estado da arte

Neste capítulo são explicitados alguns conceitos usados ao longo deste documento e apresentado um breve estado da arte como enquadramento e suporte às propostas e ao trabalho desenvolvido. A primeira subsecção versa o tema do software de acesso livre, enquanto nas subsecções seguintes os temas incidem numa demonstração de alguns laboratórios remotos já implementados, no que são FPGAs, no que são CPLDs e numa breve descrição da norma JTAG. Por último, aborda-se o *router* escolhido, como unidade de processamento, assim como o seu *firmware*.

2.1 O software Open Source

Muitas organizações utilizam infraestruturas baseadas em software de acesso livre considerado uma mais-valia em termos de custos de aquisição, tecnologia, fiabilidade e por possibilitarem soluções vantajosas em termos de custo de manutenção quando comparadas com soluções proprietárias.

Segundo um estudo apresentado pela Associação de Empresas de Software Open Source Portuguesas (ESOP), a Administração Central do Estado Português gastou no ano de 2009 cerca de 160 milhões de Euros em programas informáticos. Com base na análise dos valores apresentados, esta associação entende que é possível reduzir os custos de aquisição de software em 50% a 70%. Esta redução representaria uma diminuição anual dos custos de, pelo menos, 80 milhões de euros. Esta redução nos custos estaria sempre dependente da adopção de software *open source* e poderia contribuir para um investimento em áreas diferentes e que trariam uma melhoria significativa na balança comercial, uma vez que estaria a contribuir para a diminuição das importações decorrentes da aquisição de licenças de software de empresas internacionais, como são exemplo a Microsoft e a Oracle [ESOP, 2011].

O maior sitio web de software livre é o *SourceForge* [SourceForge, 2010], onde é possível encontrar milhares de projectos e pequenos utilitários para as mais variadas finalidades. Este é um sitio web visitado regularmente por programadores que procuram determinadas aplicações que possam ser utilizadas para complementar os seus projectos. O software de acesso livre é uma oportunidade tecnológica muito

poderosa, isto porque estas aplicações recebem influências e contribuições de programadores de todo o mundo. Os programadores que disponibilizam os seus projectos no *SourceForge* são dos mais variados pontos do globo, mas todos têm um ponto em comum: partilhar conhecimento.

As instituições de ensino têm adoptado soluções de software de acesso livre devido aos enormes benefícios que daí resultam, tais como a capacidade de um sistema para interagir e comunicar com outro, a possibilidade de visualizar e adaptar o código fonte e o facto de não serem necessárias licenças de utilização nas versões não comerciais [Hon et al., 2010],

Um dos grandes benefícios do software de acesso livre é a interoperabilidade. É possível encontrar várias ferramentas de desenvolvimento, utilitários e sistemas de base de dados de acesso livre tais como o Java [Java, 2011] e o MySQL [MySQL, 2011]. A interoperabilidade destes sistemas permite que estes possam funcionar independentemente do sistema operativo e possam facilmente interagir com outros sistemas abertos, ou até mesmo sistemas proprietários. Esta interoperabilidade deve-se ao facto da maioria dos projectos de acesso livre serem desenvolvidos em linguagem tais como o Java, C e C++.

Ao contrário do software proprietário, o software de acesso livre permite que o código seja examinado. Esta possibilidade permite uma quantidade infindável de vantagens tais como a verificação da existência de riscos de segurança que comprometam o bom funcionamento da Instituição, detectar o acesso não autorizado a informação e detectar erros de programação [Ven et al., 2008].

Quando se fala em software de acesso livre este é logo relacionado com software sem custos de utilização. A ideia do software de acesso livre é que este não necessita de licenças de utilização, sendo esta a variável principal para a maioria dos decisores no momento de adopção deste tipo de software. No entanto, isto não é de todo verdade, porque nem todo o software de acesso livre é gratuito. Existem algumas distribuições Linux para empresas que são pagas, como é o caso do *Red Hat Enterprise Linux* [RedHat, 2011] e do *SUSE Linux Enterprise Server* [Suse, 2011]. Estes produtos são baseados em distribuições gratuitas, mas incluem serviços adicionais, tais como as certificações Linux para determinado hardware, acesso a actualizações e suporte a determinados serviços, os quais comportam custos.

O software de acesso livre tem sido impulsionado por empresas com uma grande notoriedade tais como a IBM, Novell e a Sun, isto apenas para mencionar algumas das que têm contribuído para a distribuição, criação e suporte do software de acesso livre.

Teoricamente pode-se dizer que cada projecto tem o contributo de centenas ou até milhares de programadores, mas isso apenas é verdade em projectos de grande dimensão. A maioria dos projectos são pequenas aplicações desenvolvidas por apenas um programador ou por um número muito reduzido de colaboradores. Contudo, existem projectos que devido à sua dimensão e complexidade são desenvolvidos por um grande número de programadores, como é exemplo o Linux ou o Apache .

Quando se fala de software livre, tem de se falar inevitavelmente de Linux. O Linux é um fenómeno de popularidade e uma das maiores distribuições de software desenvolvidas através da contribuição de milhares de programadores voluntários. Este é um sistema operativo de elevada complexidade e bastante sofisticado, com várias distribuições e com uma enorme aplicabilidade.

Mais de um terço dos servidores Web são baseados em Linux, devido à sua fiabilidade. Pode-se encontrar Linux nas mais variadas aplicações, tornando a lista quase interminável, sendo de destacar [Weber, 2004]:

- Servidores Web;
- Sistemas de controlo de navegação em aviões;
- Automóveis;
- Máquinas de lavar roupa;
- *Routers*.

Um dos elementos vitais para o grande sucesso do Linux, e que pode ser aplicado à maioria dos sistemas *open source,* é a rapidez na detecção e correcção de erros, uma vez que existem milhares de programadores em todo o mundo que contribuem para tornar o processo muito mais rápido quando comparado com algumas aplicações proprietárias.

O processo de desenvolvimento e incremento de novas funcionalidades no sistema Linux passa por uma grande discussão, onde programadores partilham opiniões sobre novas implementações que irão funcionar e as que certamente não terão grande sucesso. Existindo, como é previsível, alguns pontos de discórdia, estes são mediados pelas equipas responsáveis por cada uma das distribuições. Cada uma destas distribuições tem um conjunto de pessoas responsáveis por aceitar/rejeitar as contribuições da comunidade. As implementações fornecidas pela comunidade e que sejam aceites passam a ser incluídas nas distribuições oficiais.

O desenvolvimento de software livre assenta num modelo colaborativo, isto porque a chave de todo o sucesso é baseada na participação de vários colaboradores.

Devido à participação de tantos programadores, com as suas contribuições, todos os dias aparecem novas aplicações com funcionalidades muito variadas. As distribuições de código aberto normalmente bifurcam em diferentes versões, isto porque as licenças deste tipo de software permitem que sejam incrementadas novas funcionalidades à distribuição original.

Sempre que se tenta comparar qual o modelo de desenvolvimento de software que apresenta melhores garantias de sucesso, o proprietário ou o software livre, as respostas não são consensuais. Isto porque depende do contexto em que o software é utilizado.

2.2 Laboratórios remotos

Nos últimos anos tem existido uma enorme preocupação em modernizar os métodos de ensino, facilitando o acesso à informação, mas tentando minimizar os recursos dispendidos.

As universidades modernas têm tentado proporcionar novas oportunidades de aprendizagem aos seus alunos, para que eles possam a qualquer altura e em qualquer lugar aceder à informação e alargar os seus conhecimentos. Esta é a chave de sucesso tanto para as universidades como para os seus alunos.

A maior mudança ocorrida nos métodos de ensino está ligada ao forte desenvolvimento tecnológico da informática e à enorme expansão da internet como meio de obtenção de informação e proximidade. Seguindo esta linha, o *e-learning*, ou ensino à distância, tem centrado as atenções por parte dos investigadores pois oferece enormes vantagens aos alunos, professores e instituições. O ensino à distância permite eliminar barreiras físicas no que se refere à intercomunicação de professores e alunos, ou mesmo entre estes e os laboratórios escolares. Com o uso da internet passou a ser possível aceder a qualquer recurso virtual, ou mesmo real, remotamente, minimizando tempo e custos e maximizando o tempo de funcionamento dos laboratórios.

Os laboratórios de investigação académicos requerem espaços físicos consideráveis, custos de manutenção elevados, bem como a instalação e manutenção de equipamentos e condições de trabalho tanto para os alunos como para os professores. Com a criação de laboratórios remotos existe a possibilidade de partilha de recursos e a minimização das infraestruturas que acolhem estes laboratórios, uma vez que não será necessária a presença física dos intervenientes.

Os laboratórios remotos, quando comparados com os laboratórios tradicionais, oferecem algumas vantagens aos seus utilizadores, das quais podem ser destacadas as seguintes:

- Redução de custos na manutenção dos equipamentos reais. Com a criação de laboratórios remotos não é necessário ter instalações físicas com capacidade para receber um enorme número de alunos, hardware suficiente para cada grupo de alunos testar as suas experiências, assim como todo o restante material de apoio (mesas, cadeiras, …).
- Valor pedagógico. Os laboratórios remotos devem incluir ferramentas que guiem os seus utilizadores no processo de utilização/aprendizagem, dando suporte a todas as operações para que este seja uma boa ferramenta de aquisição de conhecimento.
- Os equipamentos reais podem ser danificados pela má utilização por parte dos alunos. Os equipamentos podem ser danificados por utilizadores com pouca experiência. Os laboratórios remotos podem incluir formas de segurança para que utilizadores sem experiência sejam guiados para uma utilização estável e que não comprometa o hardware.

Uma das mais-valias dos laboratórios remotos, ou sistemas de experimentação remota, é estarem disponíveis remotamente para os seus utilizadores através da internet (ou intranet) sem restrições nos horários e sem a necessidade dos utilizadores estarem fisicamente presentes. Estes laboratórios normalmente têm uma boa aceitação por parte dos seus utilizadores, muito devido à facilidade de utilização e às barreiras que são quebradas. Mas para que a aceitação seja mais facilitada tem que existir a preocupação de que estes laboratórios sejam o mais multiplataforma possível, isto para que os utilizadores de diferentes sistemas operativos tenham as mesmas condições de trabalho [Teixeira, 2010].

Existem alguns requisitos que os laboratórios remotos devem respeitar para que tenham sucesso:

- **Disponibilidade**: o sistema deve estar sempre disponível;
- **Portabilidade**: possibilidade de interagir com o laboratório remoto independentemente do sistema operativo que o utilizador esteja a usar;
- **Custo de manutenção reduzido**: uma boa opção para reduzir custos será utilizar tecnologia de acesso livre; desta forma é possível poupar dinheiro com as licenças de software;
- **Performance**: capacidade de aceitação de vários pedidos simultâneos, dando resposta a esses pedidos de uma forma quase instantânea.

Também na investigação, o facto dos laboratórios remotos estarem sempre disponíveis permite uma enorme flexibilidade nos horários de acesso por parte das equipas de investigação, sendo desta forma possível agilizar o trabalho a ser desenvolvido.

Ao desenvolver um laboratório de experimentação remota tem de existir uma tentativa de satisfazer as necessidades e preferências dos grupos de utilizadores finais do laboratório. Por isso, as interfaces têm que ser adaptadas, isto porque as preferências variam dependendo da área de formação, da idade dos utilizadores, do seu nível de conhecimento e dos contextos específicos onde serão usadas.

Alguns dos princípios que devem ser respeitados no desenvolvimento de interfaces para laboratórios de experimentação remota são [Cagiltay et al., 2010]:

- **Instruções de utilização para diferentes grupos de alunos**: para alunos com diferentes níveis de conhecimento as teorias e exemplos de experiências devem ser os mais adequados;
- **Acesso aos materiais de suporte e de ajuda**: os materiais de suporte não devem estar segundo uma determinada ordem e devem estar sempre acessíveis no momento em que o trabalho assim o exigir;
- **Distribuição dos conteúdos segundo níveis de capacidade**: os conteúdos devem ser desenvolvidos e distribuídos tendo em conta as capacidades dos grupos de utilizadores;
- **Disponibilizar exemplos**: os exercícios e experiências são muito importantes nas engenharias;
- **Resposta do sistema**: apresentação dos resultados de forma clara e objectiva.

A arquitectura típica dos sistemas de experimentação remota normalmente caracteriza-se por assentar numa arquitectura algo semelhante. Normalmente este tipo de laboratórios tem uma interface com o utilizador (tipicamente através de um *Web browser*), que comunica com um computador remoto (servidor onde está implementada a lógica da aplicação), que por sua vez comunica com os diferentes dispositivos. A interface com o utilizador deve ser bastante intuitiva e apresentar de forma clara os resultados obtidos. Para a implementação deste tipo de arquitectura são normalmente utilizadas várias linguagens de programação, sendo as mais tradicionais o Java e o PHP (o Java na implementação da lógica da aplicação; e o PHP na implementação das interfaces). A Figura 1 permite visualizar este tipo de arquitectura típica de uma forma bastante simplificada:

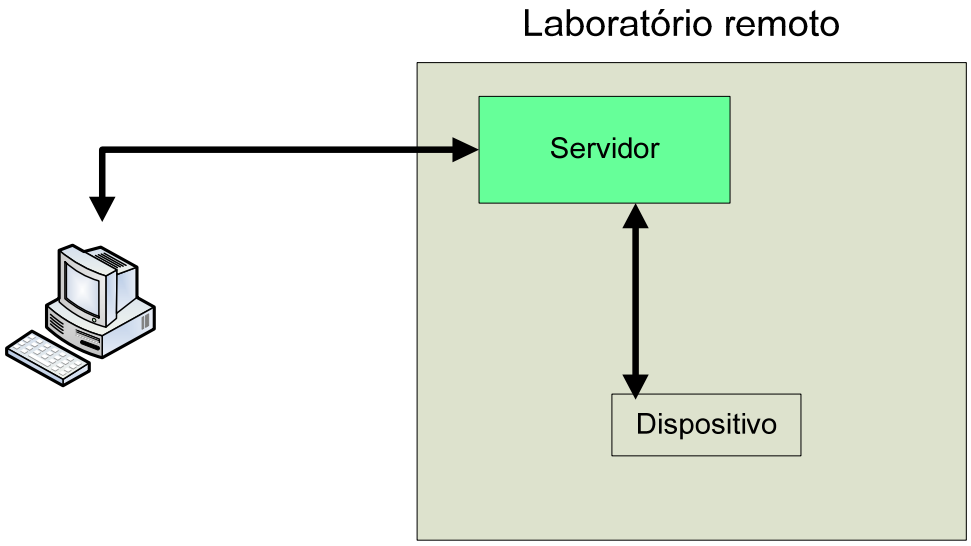

Figura 1- Arquitectura típica de um laboratório remoto

Existem inúmeros laboratórios remotos já implementados e em funcionamento e, segundo os seus intervenientes, com uma boa aceitação e resultados muito positivos.

Algumas instituições desenvolveram laboratórios próprios destinados a áreas específicas de ensino, como, por exemplo, a Universidade de Pisa, que desenvolveu um laboratório remoto para apoio ao ensino da robótica. Este projecto consiste num sistema distribuído para a aprendizagem à distância nas áreas da robótica. A arquitectura deste laboratório remoto está ilustrada na Figura 2.

Este laboratório remoto tanto pode ser usado como ambiente de simulação como para a realização de forma remota de experiências físicas reais. É de destacar na arquitectura deste laboratório o uso de linguagens de programação de acesso livre, tais como servidores Linux, *webservices* em Java e interfaces Java *applets*. Uma funcionalidade muito interessante é a possibilidade de acompanhar em tempo real os movimentos dos robôs através de *Webcams* de alta definição, o que possibilita a alunos e professores visualizar a experiência como se estivessem fisicamente no laboratório [Balestrino et al., 2009].

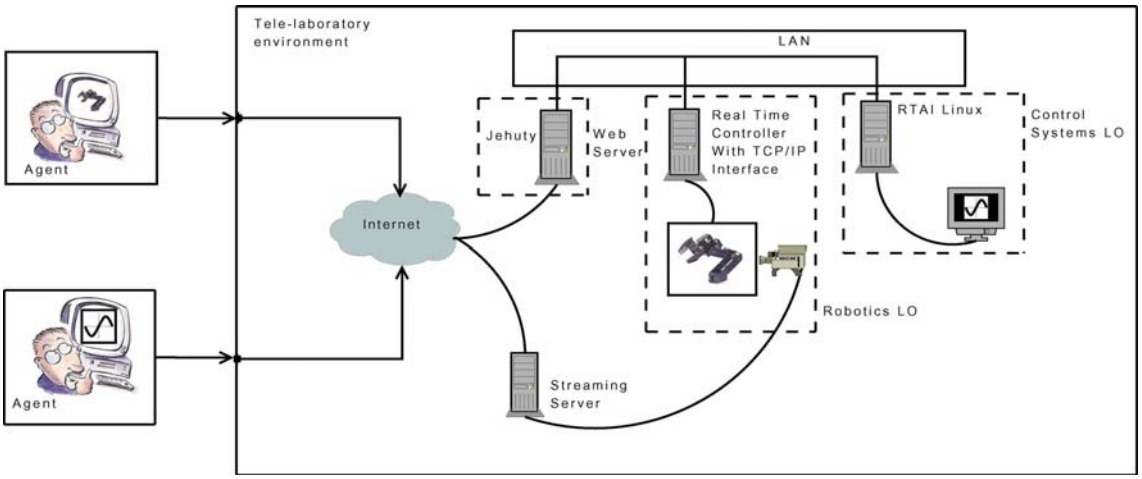

Figura 2 - Arquitectura do Telelaboratory [Balestrino et al., 2009]

Na Figura 3 é possível visualizar a arquitectura do laboratório remoto e-Lab implementado na Universidade de Bordéus. Este laboratório remoto foi implementado para dar apoio no ensino da electrónica.

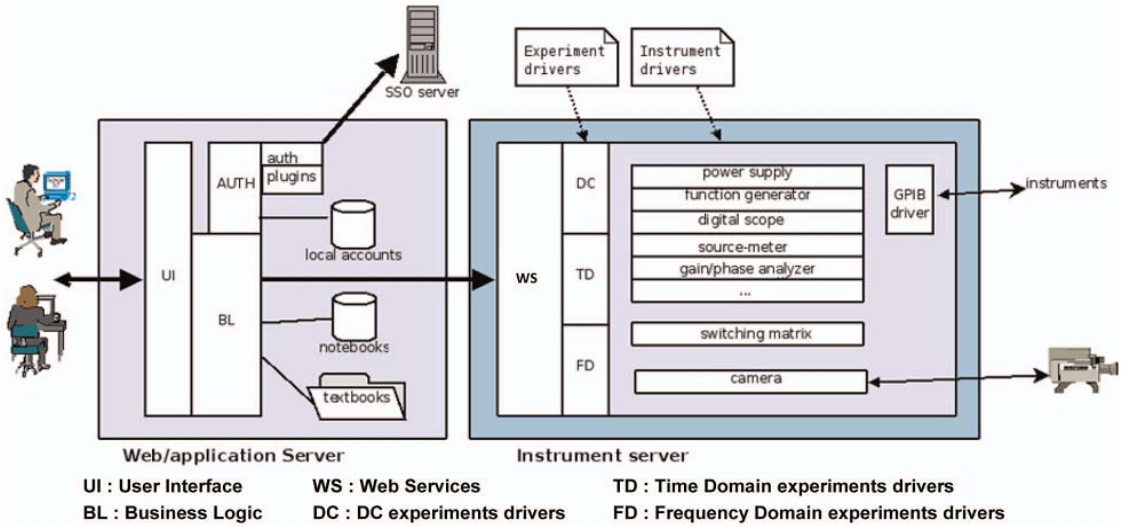

Figura 3 - Arquitectura do laboratório remoto e-Lab [Lewis et al., 2009]

Através deste laboratório é possível adquirir conhecimentos e efectuar alguns exercícios com circuitos integrados. É ainda possível a programação de hardware, como FPGAs oferecendo a partilha e reutilização de experiências já efectuadas pelos alunos.

Uma grande vantagem deste laboratório é a utilização de software de acesso livre, o que diminuiu drasticamente o custo de implementação desta solução, uma vez que não é necessário comprar licenças de utilização do software.

A interface com o utilizador é Web e a comunicação entre a camada intermédia e o servidor instrumental é baseado em SOAP (*Web Services*). O servidor instrumental utiliza software muito típico neste tipo de cenários para implementar a lógica de negócio: Linux, Apache, MySQL e PHP (LAMP)

Este laboratório tornou-se uma peça muito importante no ensino da electrónica na Universidade de Bordéus [Lewis et al., 2009].

Na Hosei University no Japão foi criado um laboratório remoto que possibilita a programação de FPGAs remotamente e a análise da operação da funcionalidade implementada, cuja arquitectura está representada na Figura 4.

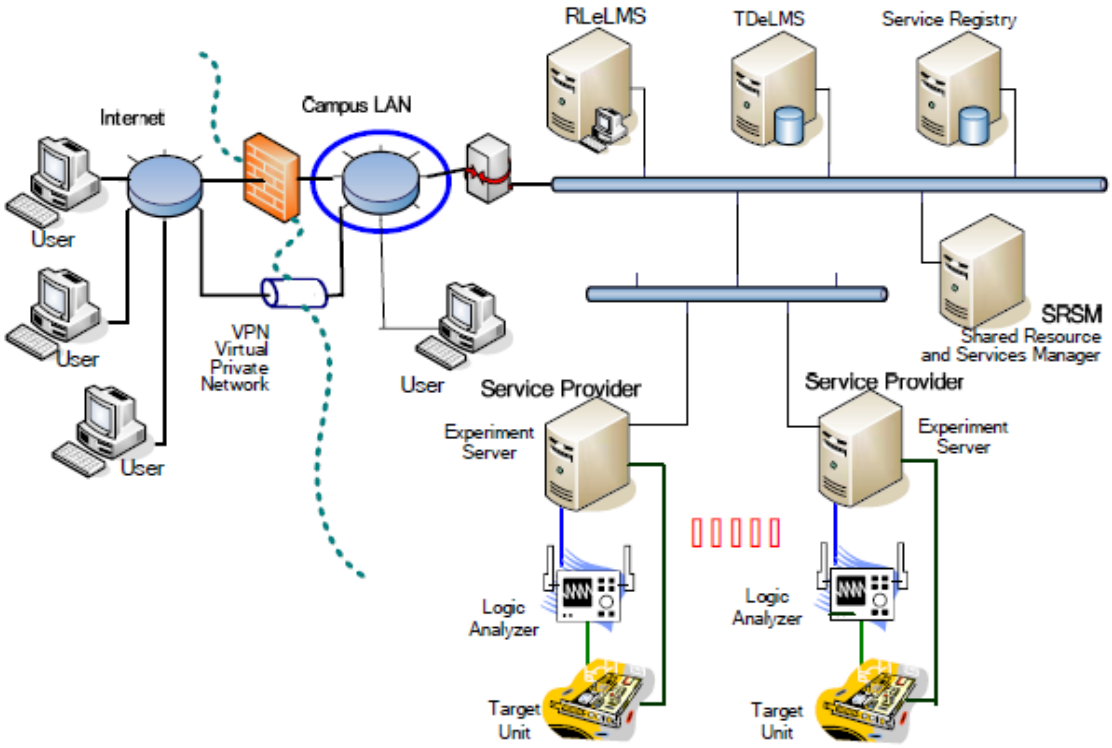

Figura 4 - Arquitectura do Shared-Lab [Fujii et al., 2005]

Como os equipamentos de programação e as unidades de teste são caras, é difícil às universidades terem uma unidade de teste para cada aluno. Sendo estas unidades em menor número que os alunos, cada aluno deve ocupar a unidade de teste, para programação e análise dos dados, por um período de tempo o mais reduzido possível. Geralmente o tempo de execução do teste é muito curto, sendo que a maior parte do tempo de ocupação destas unidades de teste é na preparação e na análise dos dados obtidos.

Da arquitectura apresentada na Figura 4, pode-se destacar o *Service Provider* cuja implementação é apresentada na Figura 5.

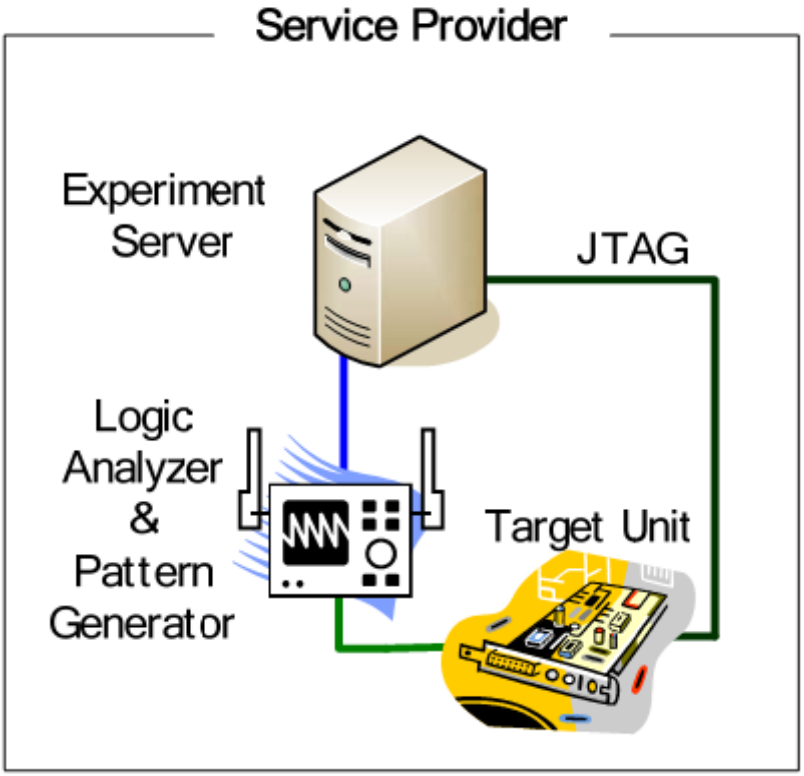

Figura 5 - Arquitectura do *Service Provider* [Fujii et al., 2005]

Nesta arquitectura, o servidor controla os equipamentos do laboratório remoto e efectua a programação do hardware através de uma interface JTAG. É ainda possível após a programação da FPGA analisar os resultados obtidos.

Com a criação deste laboratório é possível a partilha de equipamentos, maximizando a utilização do hardware e minimizando o tempo de espera para efectuar os testes [Fujii et al., 2005].

2.3 Os dispositivos lógicos programáveis do tipo FPGAs

As FPGAs são dispositivos que consistem em milhões de transístores interligáveis pelo utilizador, com a finalidade de implementarem funções lógicas. Estes dispositivos evoluíram de simples *Programmable Logic Devices* (PLD) para sistemas totalmente integrados SoCs (System-on-a-Chip), contendo microprocessadores e memória

embutida, ligados por caminhos altamente optimizados com uma grande capacidade de reconfiguração.

Um bom exemplo de FPGAs são as *Virtex FPGAs* da *Xilinx* [Xilinx, 2011] que contêm um grande número de blocos lógicos e de memória RAM reconfiguráveis através de uma interface JTAG.

A Figura 6 mostra uma FPGA Spartan 3E, a qual foi utilizada para testes durante o desenvolvimento deste trabalho.

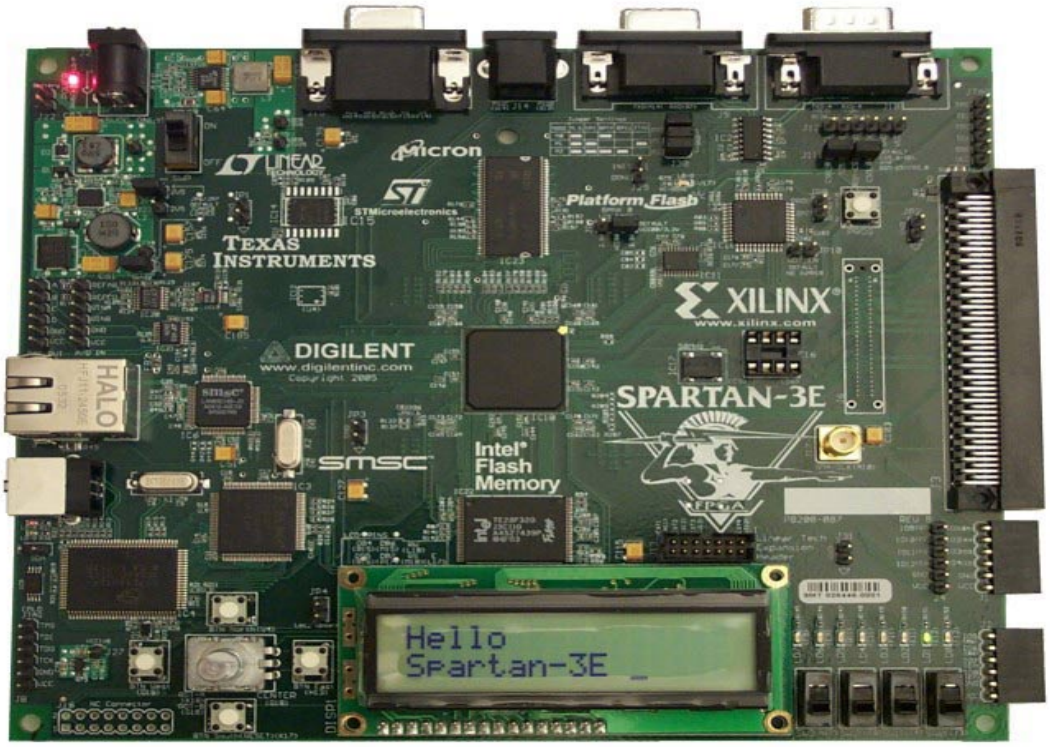

Figura 6 – Sistema de desenvolvimento com FPGA Spartan 3E da Xilinx

O grande benefício oferecido pelas FPGAs é a capacidade de reconfiguração dinâmica, a qual envolve a programação durante o tempo de execução, sem interrupção da funcionalidade, e a facilidade de teste aos seus componentes.

Com a possibilidade de reconfiguração existe a necessidade de uma interface que permita interagir com o dispositivo através de um computador, tendo sido aproveitada a interface JTAG, a qual possibilita uma interação rápida e facilitada com a FPGA [Tan et al., 2006].

Uma FPGA é um circuito integrado projectado para ser programado após o seu fabrico pelo cliente. A descrição da funcionalidade pretendida para a FPGA é normalmente

efectuada usando uma linguagem de descrição de hardware (*Hardware Description Language* - HDL) de alto nível.

As FPGAs contêm componentes programáveis chamados *logic blocks* e uma hierarquia de interconectores reconfiguráveis que permitem que os blocos sejam interligados e suportem novas funcionalidades. Os *logic blocks* podem ser configurados para executarem funções combinatórias ou sequenciais complexas, ou apenas lógica simples, como um *AND* ou um *XOR*.

Em vez de limitar o dispositivo a uma qualquer funcionalidade pré-determinada, uma FPGA permite que sejam programadas novas funcionalidades, adaptando o hardware a novas especificidades definidas pelo programador. Desta forma, é possível implementar qualquer funcionalidade lógica (possível de ser realizada) sempre que necessário, sendo uma grande vantagem para muitas aplicações [Altera, 2010].

Algumas das vantagens de usar uma FPGA:

- Prototipagem rápida;
- Menores custos;
- Capacidade de reprogramar uma determinada área para teste as vezes que assim se entender, sem que haja a necessidade de alocar recursos permanentes específicos para esse fim (útil para a correcção de erros).

As grandes empresas nesta área são a *Xilinx* e a *Altera* que detêm a maior parte da quota de mercado. Estas duas grandes empresas são separadas pela sua filosofia. No caso da *Xilinx* a filosofia assenta em fornecer todos os recursos possíveis mesmo que isso cause um incremento na complexidade. A filosofia da *Altera* é fornecer os recursos que a maioria dos utilizadores deseja, mantendo a facilidade de utilização dos seus dispositivos.

Para configurar uma FPGA é necessário fazer passar um *stream* de bits através de pinos reservados para este efeito, sendo a interface JTAG a melhor solução para efectuar esta tarefa, uma vez que não implica um acréscimo dos pinos ocupados, resultando antes do reaproveitamento do porto de teste previamente existente.

Existem diferentes formas de configurar uma FPGA [Tan et al., 2006b]:

- Utilizar um cabo para ligar o PC à FPGA e correr uma aplicação no PC capaz de fazer passar informação através do cabo para o dispositivo. Esta é a forma mais usual de configuração visto ser a mais rápida e fácil de utilizar;
- O uso de *boot-PROM* na placa, auto-configurando-se a FPGA automaticamente após a sua inicialização;

- O uso de um microcontrolador na placa com um *firmware* adequado capaz de enviar informação para a FPGA.

A configuração das FPGAs da *Xilinx* e da *Altera* funciona de uma forma muito idêntica. A principal diferença é ao nível do nome dos pinos e nos modos de operar que são nomeados de forma diferente. No entanto, as funcionalidades fornecidas são semelhantes.

A utilização do JTAG permite adicionalmente o controlo do estado de todos os pinos de entrada e saída da FPGA, possibilitando que esta seja testada ou programada através de comandos normalizados. Na Figura 7 pode-se verificar como é efectuada a interacção com uma FPGA utilizando a interface JTAG, sendo possível visualizar os pinos utilizados para enviar ou obter informação.

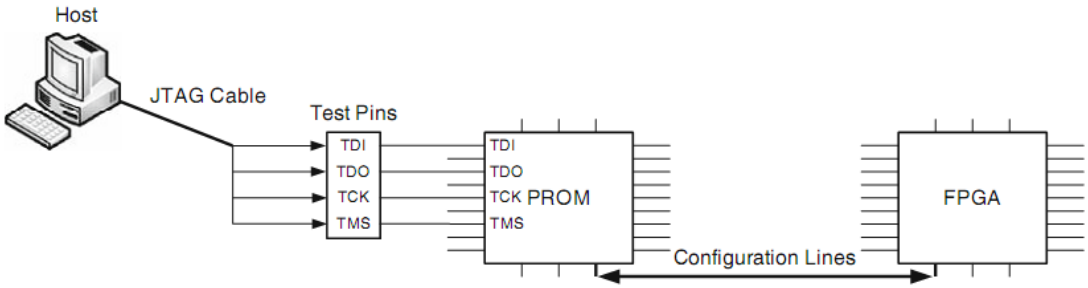

Figura 7 - Interação com uma FPGA utilizando a interface JTAG [Smith, 2010]

Para que seja possível a interacção com a FPGA através da interface JTAG é necessário ter um software e um cabo JTAG.

Para facilitar a interligação entre a interface JTAG e a FPGA, existem cabos que podem ser ligados à porta USB. O software JTAG assume especial importância porque é a interface pela qual o utilizador interage para efectuar a transferência da informação do PC para a FPGA.

Os fabricantes de FPGAs oferecem vários tipos de cabos e algumas soluções de software, as quais permitem a interacção com os dispositivos por estes fabricados.

No próximo subcapítulo é abordado com maior detalhe a tecnologia JTAG.

As FPGAs podem ser programadas para desempenharem funções simples de adição ou subtracção, ou funções muito mais complexas como a detecção e correcção de erros. As FPGAs estão presentes em sistemas de complexidade muito díspar como são exemplo:

- Os automóveis;

- Os radares;
- Os mísseis;
- Os telemóveis;
- Os computadores.

Os fabricantes de FPGAs, de que são exemplo a Xilinx e a Altera, partilham alguns conceitos básicos em termos de arquitectura dos seus dispositivos. De uma forma muito simplificada esta arquitectura consiste em três funcionalidades básicas: interfaces de entrada/saída, blocos básicos de construção e interligações.

Na Figura 8 está representada a arquitectura simplificada de uma FPGA. Através desta imagem é possível visualizar os blocos lógicos e a forma como estão interligados com outros blocos, os quais estão por sua vez ligados a interfaces de entrada/saída.

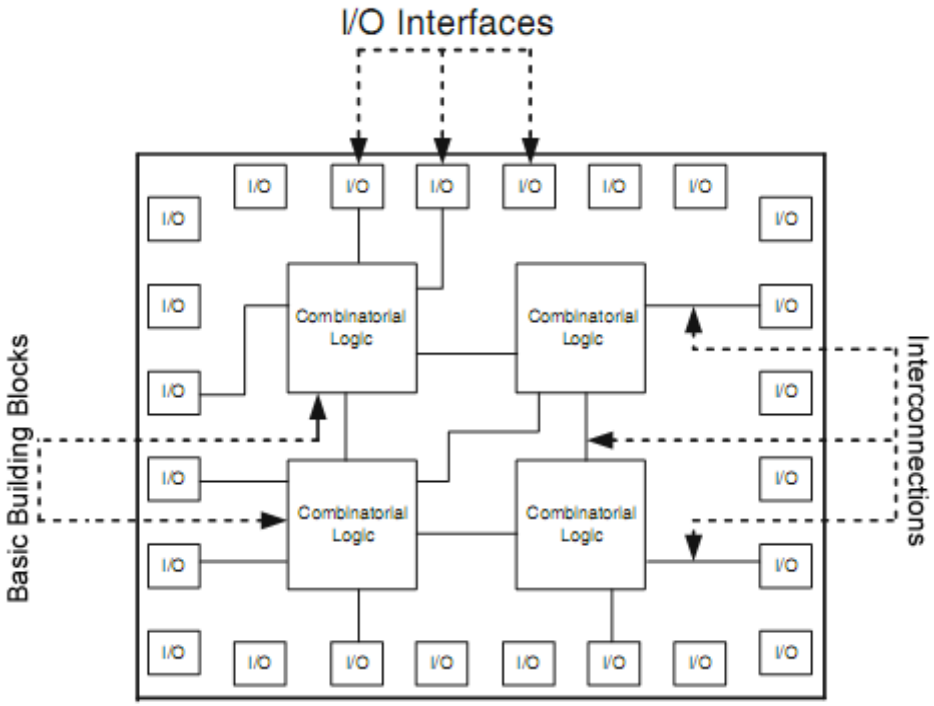

Figura 8 - Arquitectura genérica de uma FPGA [Smith, 2010]

As interfaces de entrada/saída são a forma de enviar informação da lógica interna para os dispositivos externos, e a partir dos quais os dados são recebidos das fontes externas.

2.4 Os dispositivos lógicos programáveis do tipo CPLDs

As CPLDs são dispositivos programáveis, cuja principal diferença relativamente às FPGAs é a sua arquitectura interna e a forma como são implementadas as várias funções lógicas.

A arquitectura das CPLDs é baseada em múltiplos PLDs, num único circuito integrado (CI), interligados através de canais de comunicação.

Cada fabricante de CPLDs, apresenta a sua arquitectura, contudo em termos gerais elas são algo semelhantes nos seguintes aspectos:

- Blocos funcionais;
- Blocos de entrada e saída de dados;
- Matriz de ligação dos vários blocos lógicos.

Na Figura 9 pode ser visualizada a arquitectura genérica de uma CPLD.

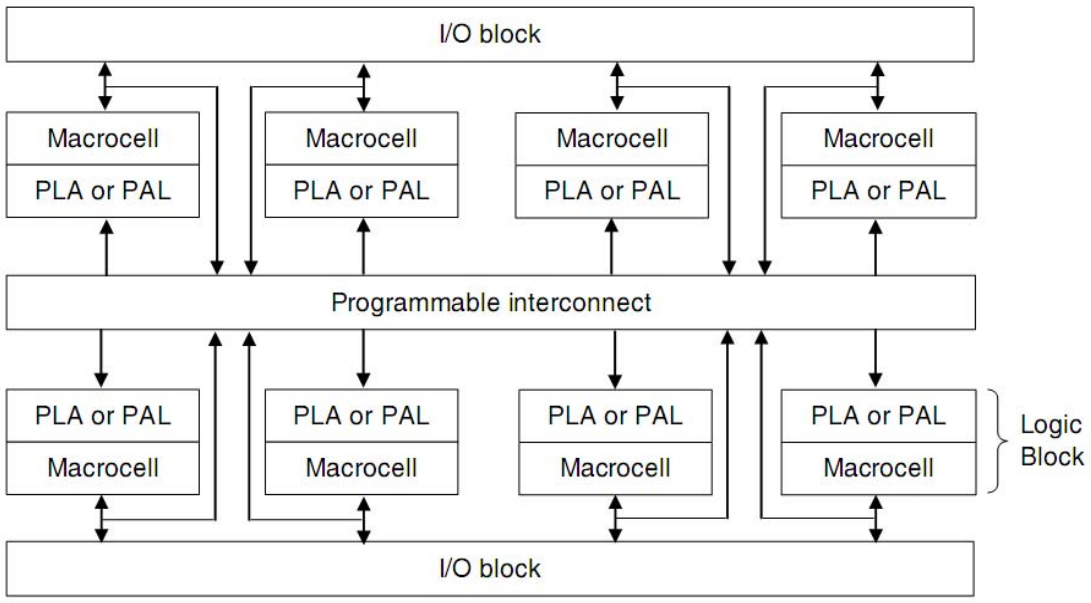

Figura 9 - Arquitectura genérica de uma CPLD (retirada de [Navadi, 2005])

Os sinais de entrada do bloco I/O são encaminhados para o respectivo bloco funcional através de um interconector reconfigurável que interliga todos os blocos. Algumas arquitecturas, para diminuírem o tempo que os sinais de entrada demoram a chegar aos blocos funcionais, implementam acessos directos a esses blocos, melhorando significativamente a performance da CPLD. A arquitectura das CPLDs apresenta um

largo número de PALs (Programable Array Logic) num único CI, sendo que cada um desses blocos inclui entre 8 e 16 macrocélulas que executam diferentes funcionalidades. Estes dispositivos programáveis podem ser interligados entre si formando uma cadeia programável através da interface JTAG [Corelis 2010b]. À semelhança das FPGAs, as CPLDs podem ser configuradas para executarem funções combinatórias ou sequenciais complexas ou para implementar apenas lógica simples como um AND ou um XOR.

As CPLDs podem ser utilizadas em diferentes aplicações, de que são exemplo:

- Controladores LAN (Local Area Network);
- Controladores de memória cache;
- Universal Asynchronous Receiver/Transmitter (UARTs).

Existem vários tipos de CPLDs que variam em termos de tensão de alimentação, interfaces, quantidade de memória e diferentes tipos de memória. As CPLDs podem conter memória ROM (Read-Only Memory) e RAM (Random Access Memory). Assim como as FPGAs, as CPLDs podem ser programadas ou reprogramadas através de um PC. Para isso, podem ser utilizadas as interfaces disponíveis tais como a porta série ou a interface JTAG. A interacção utilizando a porta JTAG exige a utilização de um software e de um cabo JTAG.

Na Figura 10 é possível visualizar uma placa de desenvolvimento contendo uma CPLD da XILINX, a qual foi utilizada para testes durante o desenvolvimento deste trabalho.

Para melhor conceber as diferenças/semelhanças entre as FPGAs e as CPLDs é necessário compreender a sua arquitectura interna, como esta arquitectura implementa as várias funções lógicas, e compreender as interfaces disponibilizadas para programar ou testar a placa.

As CPLDs têm algumas semelhanças com as FPGAs, tais como:

- Várias interfaces que permitem interagir com a placa permitindo a sua programação ou teste; destas interfaces destaca-se a JTAG;
- Vários componentes interligados por pistas lógicas;
- Possibilidade de reprogramar e testar as funcionalidades da placa.

Comparativamente, as FPGAs são mais versáteis que as CPLDs, porque permitem implementações mais complexas. As FPGAs contêm blocos lógicos mais complexos, de tamanho muito mais reduzido, mas em muito maior número, permitindo desta forma um maior número de blocos por CI.

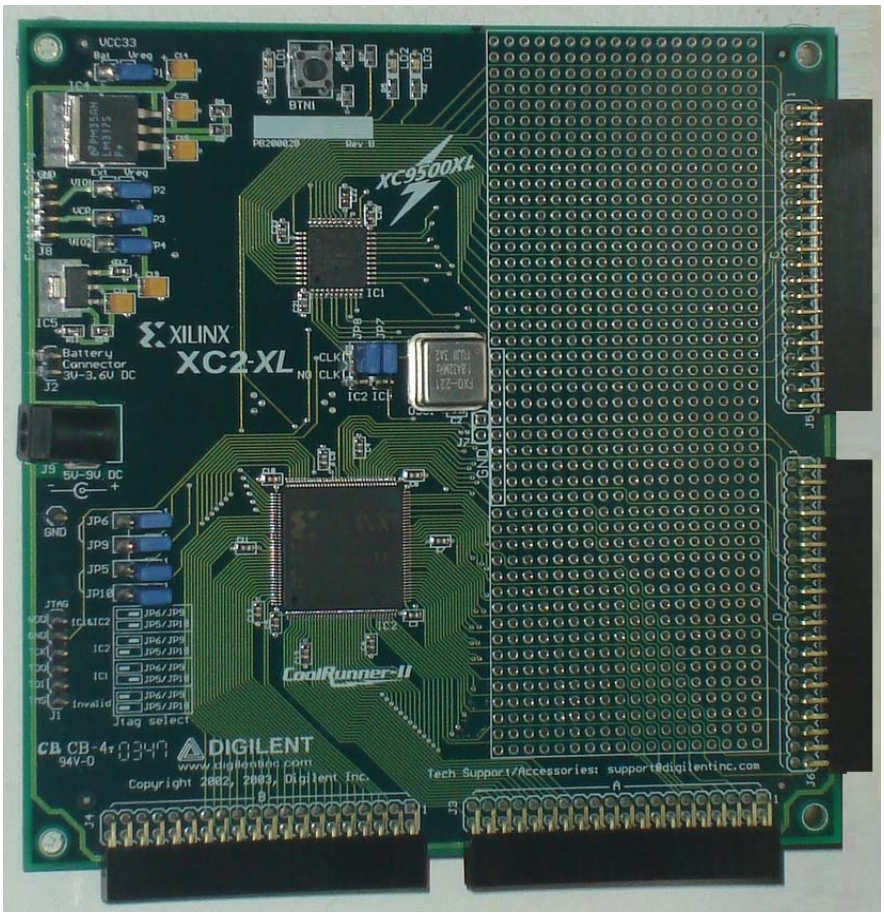

Figura 10 – Sistema de desenvolvimento com CPLD XC2-XL da Xilinx

2.5 JTAG

Uma enorme desvantagem da miniaturização dos circuitos integrados (CIs) é que o teste de cartas de circuito impresso (CCIs) se tornou mais complexo, devido à dificuldade física de acesso aos pinos dos componentes nela montados e às pistas que os interligam, em particular em CCIs de várias camadas. A montagem de componentes em ambos os lados de uma mesma CCI, a montagem superficial e os novos tipos de encapsulamento, em que os pinos se situam por baixo do próprio CI, complicou ainda mais o problema.

Para encontrar uma solução para estes problemas, um grupo de empresas decidiu juntar-se e formar um consórcio denominado JTAG. Estas empresas criaram um caderno de especificações para uma infra-estrutura de teste de hardware a ser incluída nos próprios componentes e que permite o teste estrutural da CCI sem necessidade de acesso físico aos pinos dos componentes ou às pistas, e que mais tarde resultou na norma IEEE 1149.1. Nesta norma são estabelecidos os detalhes de

acesso a essa infraestrutura, a sua implementação e o seu modo de funcionamento [Corelis, 2010]. A norma define que cada CI deve possuir um conjunto de 4 pinos de teste, o porto de acesso ao teste (TAP – *Test Access Port*), vulgarmente designado de porta JTAG, através da qual é possível o acesso à infraestrutura interna de teste e o controlo do seu funcionamento.

A especificação JTAG foi concebida inicialmente para permitir efectuar o teste estrutural de CCIs, mas o seu sucesso levou a que fosse reaproveitada para outros fins, tal como o teste funcional do próprio componente, para controlo de infraestruturas de auto-teste interno ou para *debugging* e diagnóstico de faltas. À tecnologia que permite testar os componentes de uma CCI, usando JTAG, dá-se o nome de *boundary scan* [Embedded, 2010].

A norma IEEE 1149.1 define uma infraestrutura de teste constituída pelo controlador do TAP, o porto de acesso à lógica de teste, o registo de *Boundary Scan*, o registo de *bypass*, o registo de instruções e o bloco de descodificação.

A interface para o exterior da infraestrutura de teste é constituída por quatro pinos de sinal. O pino de sinal TMS (*Test Mode Select*) e o TCK (*Test Clock*) servem para comandar e sincronizar a lógica de teste. O pino TDI (*Test Data In*) permite deslocar dados (em formato série) para o interior do CI e o pino TDO (*Test Data Out*) permite deslocar dados para o exterior do CI. Um quinto pino opcional TRST (*Test Reset*) permite fazer a inicialização assíncrona da lógica de teste [Alves, 1999].

De uma forma simplificada, os cinco pinos permitem:

- **TDI** (*Test Data Input*): Entrada série para os registos internos;
- **TDO** (*Test Data Output*): Saída série dos registos internos; quando em alta impedância tem como resposta um 1 lógico;
- **TCK** (*Test Clock*): Sinal do relógio de teste;
- **TMS** (*Test Mode Select*): Porta de controlo que define o modo de funcionamento;
- **TRST** (*Test Reset*): opcional, reinicializa o controlador do TAP.

Através da Figura 11 é possível visualizar como a informação é transferida através dos pinos da interface JTAG, criando um caminho lógico que interliga a interface aos dispositivos.

As CCIs com interface JTAG comunicam com o exterior através de uma porta JTAG igual à definida para os CIs. Os pinos desta porta estão interligados aos pinos dos vários CIs montados na CCI. A ligação interna entre o pino TDI (*Test Data Input*) e o pino TDO (*Test Data Output*) forma um caminho dedicado na periferia dos CIs (daí a

designação *boundary scan*), como se pode verificar na Figura 12 [Inaccess, 2010]. Este caminho possibilita o acesso virtual aos pinos, permitindo o controlo dos pinos de entrada e a análise dos pinos de saída.

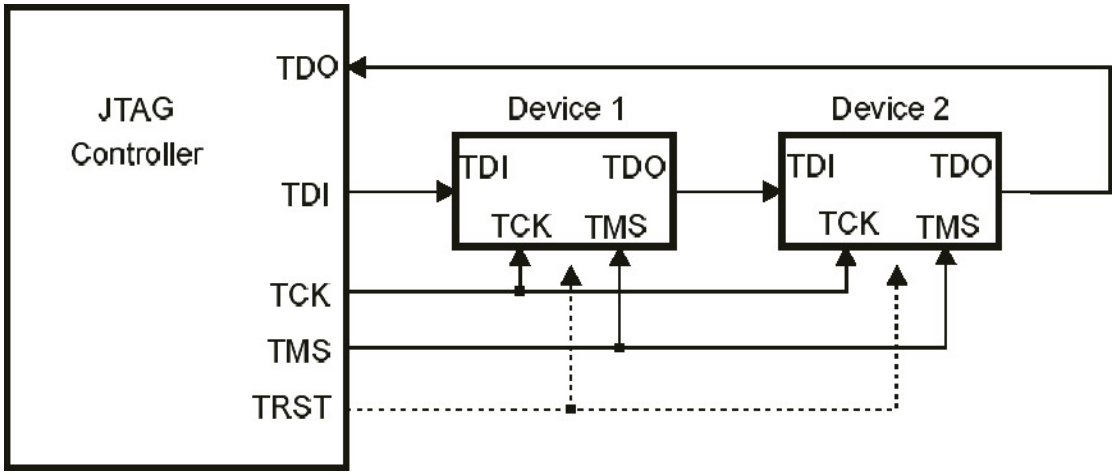

Figura 11 - Caminho criado pela interface JTAG

Durante o teste, os sinais de entrada/saída atravessam as células de *boundary scan* que constituem o denominado registo de *boundary scan*. Estas células podem ser configuradas para suportar testes externos de interligação entre CIs ou testes internos no próprio CI.

Nos últimos anos os dispositivos que incluem JTAG têm proliferado de uma forma muito sustentada. Quase todos os novos microprocessadores introduzidos no mercado incluem circuitos JTAG para o seu auto-diagnóstico, sendo que a maioria dos fabricantes de FPGAs ou CPLDs, como a *Altera* e a *Xilinx*, têm incorporado JTAG nos seus componentes. Essa incorporação possibilita a programação funcional destes dispositivos lógicos programáveis através da inclusão de um registo suplementar, o *configuration register*. A norma IEEE1532 [IEEE Std 1532, 2001] normaliza a forma como a geração do ficheiro de configuração e a programação dos dispositivos lógicos programáveis através da interface JTAG deve ser efectuada, compatibilizando ferramentas de desenvolvimento e dispositivos entre todos os fabricantes [JTAG Test, 2010].

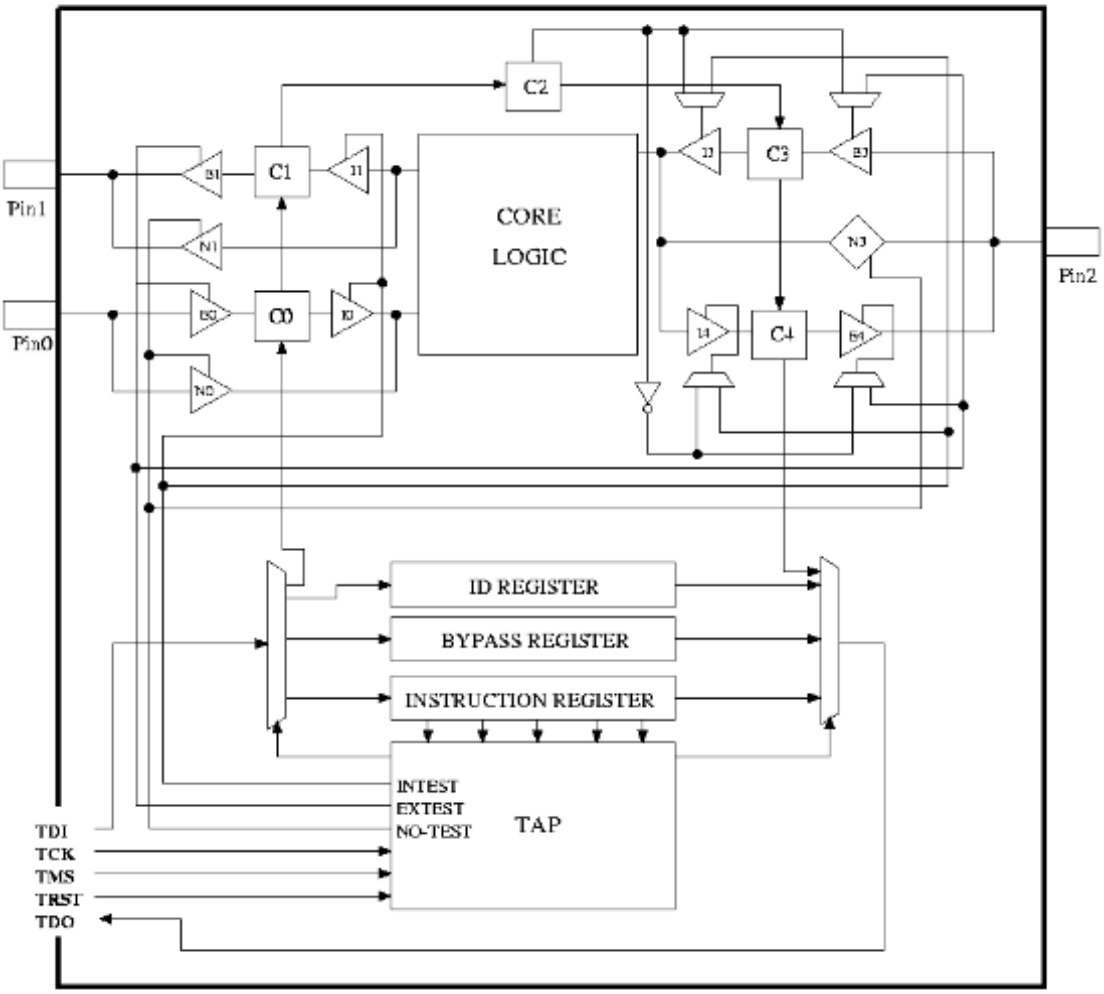

Figura 12 - Interligação dos pinos de sinal JTAG [InAccess, 2010]

2.6 Servidores Web baseados em *routers* e Linux (*OpenWrt*)

A ideia de transformar um *router* num servidor *Web*, divergindo completamente do papel original para que este equipamento foi criado, é bastante aliciante, isto porque se trata de um equipamento com capacidade de processamento, memória interna, interfaces de rede e portas USB às quais se pode ligar um conjunto muito alargado de periféricos.

Para além do baixo custo, o que torna a solução ainda mais convidativa, existe ainda a possibilidade de instalar, no *router*, *firmware* que possibilita a instalação de novos programas, dotando-o desta forma com novas funcionalidades.

O aparecimento de *firmware* para *routers* baseado em Linux teve um enorme crescimento a partir de 2003, quando a *Linksys* partilhou os seus desenvolvimentos

nesta área, uma vez que usava nos seus equipamentos software com licença GPL (*General Public License*).

As vantagens em utilizar *firmware* baseado em Linux são enormes, isto porque é possível adicionar novas funcionalidades ao equipamento, tirando o máximo proveito do hardware, e tendo todas as vantagens/versatilidade que o Linux oferece.

2.6.1 *Router*

O *router* utilizado neste projecto é o Asus WL500g Premium v2, ilustrado na Figura 13. A escolha recaiu sobre este modelo devido às suas características se encaixarem nos requisitos do projecto, nomeadamente uma boa capacidade de processamento e um espaço de memória para a instalação de novos programas. Este espaço de memória, quando comparado com um PC é bastante reduzido, mas é suficiente para incluir novas funcionalidades no *router*, tornando-o mais versátil.

Figura 13 - *Router* Asus WL500g Premium v2

Os critérios que levaram à escolha do *router* Asus WL500g Premium v2 foram:

- As características de hardware (processador e memória);
- A facilidade de aquisição;
- O número de portas USB;

- O baixo custo;
- A possibilidade de instalar um *firmware* de acesso livre.

A maior dificuldade quando se procura incluir novas funcionalidades num *router* é que estes tendem a ser equipados com reduzidas quantidades de memória, muito provavelmente numa tentativa por parte dos fabricantes em reduzir custos.

Quando comparado um *router* com um PC normal, pode-se enumerar algumas vantagens e desvantagens.

Vantagens:

- Custo;
- Tamanho;
- Consumo energético.

Desvantagens:

- Performance;
- Interface com o utilizador;
- Periféricos, apenas USB.

Para a implementação de um laboratório remoto de baixo custo, objectivo deste trabalho, poder-se-ía ter escolhido desenvolver ou até mesmo criar um sistema embutido dedicado. Quando comparada a opção *router* versus sistema embutido dedicado, a primeira apresenta as seguintes vantagens e desvantagens.

Vantagens:

- Custo, baixo para pequenas quantidades;
- Rapidez de obtenção;
- Facilidade de programação.

Desvantagens:

- Mais caro para grandes quantidades;
- Modelos em permanente actualização.

2.6.2 Descrição do hardware do *router*

Na Tabela 1 [ASUS, 2010] são apresentadas as principais características do *router* usado, as quais são suficientes para implementar uma rede de computadores de pequenas dimensões. Como se pode verificar na Tabela 1, o ASUS WL500g Premium v2 usa um processador BCM5354 da Broadcom. Este processador é um MIPS [MIPS, 2011.] a 32-bit que funciona a uma frequência de 240 MHz neste *router*. A memória

interna é de 32 MB, divididos por dois CIs DDR SDRAM fabricados pela Hynix [xbit, 2010].

Tabela 1 - Especificações do ASUS wl-500g Premium

Parâmetro	Características
Protocolos	IEEE802.11b, IEEE802.11g, IEEE802.11d, IEEE802.3, 802.3u, 802.1X (Autenticação de Segurança), preparada para 802.11i (Segurança por WPA2), Segurança 802.11e (QoS wireless), IPv4
Velocidade de transferência	802.11g: 6, 9, 12, 18, 24, 36, 48, 54Mbps 802.11b: 1, 2, 5,5, 11Mbps
Portas USB	2 portas USB 2.0
Segurança Wi-Fi	64/128-bit WEP, WPA-PSK, WPA2-PSK, WPA-Pessoal, WPA-Empresarial, WPA2-Empresarial, Radius com 802.1x
WAN	1 ligação RJ-45
LAN	4 ligações RJ-45
Dimensões	215 x 160 x 42 mm
Processador	Broadcom BCM5354 a 240 MHz
Arquitectura	MIPS
Memória RAM	32 MB
Consumo máximo	12,5 W
Memória Flash	8MB
Alguns dos sistemas suportados	*OpenWrt*, X-WRT, DD-WRT, OLEG

O *firmware* genérico que está presente neste *router* não permite que sejam desenvolvidas novas aplicações e não permite a utilização do hardware segundo as necessidades específicas do utilizador. Desta forma, para tornar o *router* num sistema mais versátil, é necessário instalar um *firmware* que permita, por exemplo, ligações

com a consola do sistema operativo, envio de ficheiros e instalação de novos programas.

A arquitectura do ASUS WL 500g Premium v2 é capaz de executar várias distribuições diferentes de Linux com diferentes funcionalidades instaladas e diferentes interfaces com o utilizador, dos quais se pode destacar o *OpenWrt* (que será descrito no próximo subcapítulo). Com um sistema como o *OpenWrt*, este *router* torna-se numa plataforma muito versátil e fácil de utilizar. Logo, este *router* é uma boa opção para ser usado como uma unidade de processamento, tem um bom desempenho e algumas opções diferentes de ligação que podem ser usadas para interagir com outros dispositivos.

A existência das duas portas USB é um atributo muito interessante pois permite a ligação de periféricos de uma forma muito facilitada e, consequentemente, a expansão da plataforma.

2.7 OpenWrt

O *OpenWrt* foi criado em Janeiro de 2004 por uma equipa de programadores que tinha como principal objectivo desenvolver um sistema baseado em Linux com um conjunto de funcionalidades muito reduzidas, que fosse possível instalar num *router*, que fizesse a gestão das interfaces de rede, e que possibilitasse ao utilizador os meios necessários para personalizar cada instalação para as suas necessidades.

Em 2005, com a entrada de novos programadores para a equipa de desenvolvimento, foi possível lançar a primeira versão experimental deste sistema. A sua primeira versão ficou conhecida como *OpenWrt* "*White Russian*". Esta versão teve um enorme sucesso sendo utilizada em aplicações como Freifunk-Firmware ou Sip@Home [OpenWrt, 2010]. O desenvolvimento do White Russian terminou na versão 0.9 em 30 de Janeiro de 2007. A Tabela 2 foi elaborada para mostrar as várias versões do *OpenWrt*.

As plataformas suportadas por este sistema são IA-32, x86-64, ARM, MIPS, MIPSel, PowerPC, SPARC e PA-RISC, o que faz com que seja compatível com vários modelos das marcas mais populares de *routers* como são o caso do Asus, Linksys, D-Link e Thomson.

A utilização da distribuição Linux *OpenWrt* permite que o sistema desenvolvido possa ser utilizado não só no *router* Asus WL 500, mas também nos modelos compatíveis com o *firmware OpenWrt*. A utilização deste *firmware* resolve o problema de

actualização do software para suportar outros *routers* mais recentes, quando o WL 500 já não for comercializado.

Tabela 2 - Versões do *OpenWrt*

Versão	Data
Versão experimental	02-02-2005
White Russian RC1	25-06-2005
White Russian RC2	18-07-2005
White Russian RC3	14-09-2005
White Russian RC4	23-11-2005
White Russian RC5	27-03-2006
White Russian RC6	08-11-2006
White Russian 0.9	03-02-2007
Kamikaze 7.06	01-06-2007
Kamikaze 7.07	26-07-2007
Kamikaze 7.09	30-09-2007
Kamikaze 8.09	19-02-2009
Backfire 10.03	06-08-2010

O *OpenWrt* usa uma arquitectura simples de onde se destacam duas ferramentas muito comuns na gestão de sistemas embutidos, como é o caso da biblioteca C uClibc, Busybox (que é um programa que fornece algumas ferramentas padrão dos sistemas Linux) e uma consola que permite interpretar comandos. A Figura 14 permite visualizar esta arquitectura.

Como o *router* tem uma capacidade de memória muito reduzida, não foi possível à equipa que desenvolveu o *OpenWrt* incluir as bibliotecas padrão de desenvolvimento em C como o GCC (*GNU Compiler Collection*) e glibc (*GNU C Library*). Em vez destas bibliotecas o *OpenWrt* vem com a biblioteca uClibc instalada, própria para sistemas embutidos baseados em Linux. A uClibc é uma biblioteca, para a linguagem C, muito pequena mas que suporta quase todas as funcionalidades desenvolvidas com a glibc. Por norma, os sistemas embutidos utilizam a biblioteca uClibc em conjunto com o

Busybox para criar sistemas que ocupem muito pouco espaço em memória e que necessitem de recursos muito reduzidos para o seu funcionamento normal.

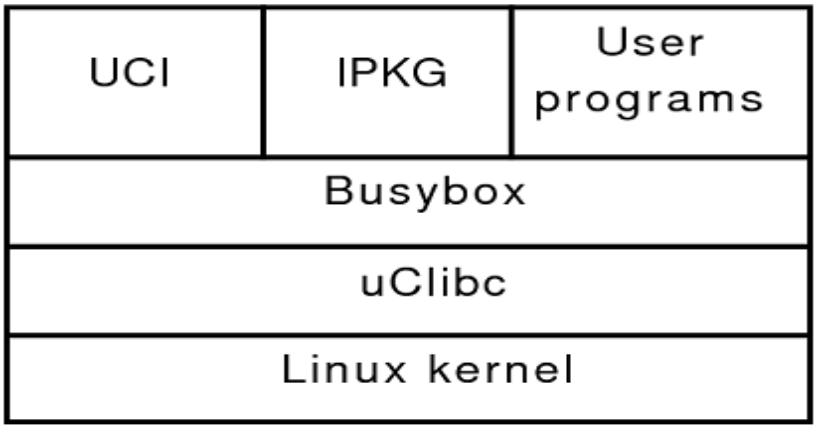

Figura 14 – Arquitectura de software do *OpenWrt* [Fainelli, 2008]

O *Busybox* é um componente chave na maioria dos sistemas embutidos, pois fornece uma implementação, embora que bastante reduzida, das funcionalidades da linha de comandos dos sistemas Linux. Em Unix os "comandos", tais como o cd, rm e mv, são executáveis que devem existir no sistema de ficheiros. O *Busybox* aglutina num único utilitário todas as funcionalidades básicas de muitos dos comandos Unix, daí a importância deste utilitário no sistema *OpenWrt*.

Para o utilizador o *Busybox* funciona da mesma forma que os utilitários utilizados em Unix (cd, rm, mv, …) isto porque produz os mesmos resultados. O funcionamento do *Busybox* é baseado em *links* simbólicos para o executável com o nome desejado (cd, rm, mv, …). Quando o *Busybox* arranca sabe como foi chamado e mediante a forma como é chamado irá desempenhar as respectivas funcionalidades (cd, rm, mv, …). Desta forma tem-se apenas um executável e os respectivos *links* para implementar as mesmas funcionalidades de vários executáveis, permitindo um melhor aproveitamento da memoria disponivel.

Este sistema é uma distribuição Linux minimalista (com funcionalidades reduzidas), que permite a interacção através da linha de comandos ou através de uma interface Web.

O *OpenWrt* é um sistema bastante versátil, pois permite que sejam instalados de uma forma fácil novas funcionalidades através do seu sistema de gestão de programas, o IPKG (Itsy Package Manager). O IPKG é um gestor de *packages* que permite instalar programas localmente e através da internet. Este gestor permite instalar *packages* do

tipo "*.ipkg*", criadas para serem utilizadas em sistemas embutidos. Uma das vantagens em utilizar um gestor de *packages* como o IPKG é a agilidade que estes gestores permitem na instalação/gestão de novos *packages*. Durante a instalação do software, se este necessitar de determinadas dependências, o gestor de *packages* consegue através da informação contida em cada um dos *packages* que está a instalar verificar as suas dependências, fazendo o *download* dos respectivos repositórios.

Na Tabela 3 pode ver-se as utilizações mais comuns do IPKG.

Tabela 3 - Como utilizar o IPKG

Comando	Descrição
ipkg update	Descarrega a lista dos *packages* disponíveis
ipkg list	Disponibiliza a lista de *packages*
ipkg install <nome>	Instala o *package* <nome>
ipkg remove <nome>	Remove o *package* <nome>

Com a instalação de novos programas pode-se adicionar novas funcionalidades, conferindo uma maior liberdade e versatilidade na configuração do sistema. Os programas podem ser removidos para libertar espaço em memória uma vez que esta é bastante reduzida. Num sistema *OpenWrt* os *packages* são do tipo .ipk e podem ser criados através da SDK de desenvolvimento fornecida para cada uma das versões deste sistema.

A grande vantagem em utilizar *packages* é que estes podem ser geridos por um gestor de pacotes como é o caso do OPKG usado nas versões mais recentes do *OpenWrt*. O OPKG é um gestor de pacotes utilizado em sistemas com recursos muito reduzidos, como é o caso do *router* ASUS utilizado.

Os gestores de pacotes normalmente incluem a possibilidade de efectuarem toda a gestão dos *packages* instalados, permitindo também a instalação de novos programas.

A instalação de novas funcionalidades através do gestor de *packages* pode ser efectuada localmente ou remotamente. Ao instalar novos programas, o gestor instala também todas as dependências necessárias para o seu bom funcionamento, numa única transacção.

No repositório oficial é possível encontrar algumas dezenas de programas, com funcionalidades muito variadas [OpenWrt, 2011]. Estes programas são fornecidos pela enorme comunidade de programadores que dão o seu contributo, fazendo com que esta plataforma seja muito popular.

Para instalar o *OpenWrt* no *router* existem duas opções: através da interface Web, como é possível visualizar através da Figura 15, ou em alternativa através da consola do Linux utilizando o cliente *tftp,* que permite enviar o *firmware* para o *router*.

O *OpenWrt* é um sistema muito prático e versátil, sendo uma excelente solução para os utilizadores que necessitem de funcionalidades avançadas activas num *router wireless.*

Figura 15 - Interface Web do *OpenWrt*

3 Soluções técnicas e opções

Antes de iniciar o desenvolvimento do laboratório remoto foi necessário avaliar algumas opções que permitissem desenvolver este laboratório com as tecnologias mais adequadas, tendo em conta os requisitos especificados inicialmente. Neste capítulo serão justificadas as opções mais relevantes em termos de tecnologias escolhidas.

Para uma melhor compreensão da arquitectura do laboratório remoto, esta pode ser visualizada através da Figura 16, onde se pode ver as diferentes camadas que compõem o laboratório desenvolvido.

Figura 16 - Arquitectura simplificada do laboratório remoto

Actualmente, as soluções de software são desenvolvidas em camadas distintas que permitem uma maior separação e compreensão das funcionalidades. Para isso, foram criadas duas camadas: interface (*front-end*) e o *middleware* (lógica de negócio).

A interface será responsável por permitir/disponibilizar a forma pela qual o utilizador comunica com o laboratório. Será a partir da interface que o utilizador manuseia o laboratório e comunica com o *middleware.*

O *middleware* foi desenvolvido para ser executado num *router* (ASUS WL500), permitindo desta forma ter uma unidade de processamento de baixo custo e com uma capacidade de processamento adequada às necessidades. O *middleware* será responsável por enviar valores de teste/configuração para a FPGA ou CPLD e receber os respectivos resultados. O teste/configuração da FPGA/CPLD será efectuada pelo *software* URJTAG, que será apresentado numa subsecção posterior. Para executar o URJTAG no *router* ASUS foi necessário efectuar algumas alterações ao seu código fonte e foi ainda necessário compilá-lo para poder ser executado no *router*.

Nos subcapítulos que se seguem são justificadas as opções tomadas em termos de tecnologias utilizadas, confrontando-as com as opções possíveis para cada uma das camadas.

3.1 Concepção e implementação do Servidor

Figura 17 - Arquitectura simplificada do laboratório remoto: middleware

Com a instalação do *OpenWrt* no *router* WL500g Premium v2, o software desenvolvido funciona num ambiente baseado em Linux, embora este ambiente seja algo diferente e com algumas especificidades, como é exemplo a memória RAM bastante reduzida (cerca de 32 MB), a arquitectura do processador e a sua capacidade de processamento. Como o *router* tem um espaço de memória muito reduzido, não é possível incluir bibliotecas de programação como é o caso do Glibc (*GNU C Library*) ou do Glibc++ (*GNU C++ Library*).

A uClibc é uma biblioteca nativa do *OpenWrt* que foi criada para os sistemas embutidos, como é o caso dos *routers*, e que permite utilizar programas desenvolvidos na linguagem de programação C. Esta biblioteca é muito reduzida mas suporta quase todas as funcionalidades criadas com GCC. Contudo, para desenvolver aplicações compatíveis com o *OpenWrt*, é necessário criar um mecanismo que compile a aplicação num outro sistema, sendo que após essa compilação será necessário instalar os executáveis no *OpenWrt*. O desenvolvimento de software compatível com o *OpenWrt* é efectuado através da SDK, correspondente à versão do *OpenWrt* utilizada.

O código para ser instalado no *router* foi todo desenvolvido utilizando a linguagem de programação C. Esta linguagem de programação permite uma enorme portabilidade e permite poupar mais espaço de memória comparativamente com a outra alternativa que seria o C++. Em termos de comparação com a linguagem C++ é ainda de destacar que as aplicações desenvolvidas em C são mais rápidas na compilação e durante o processamento.

A linguagem C foi originalmente criada por Dennis Ritchie nos laboratórios da Bell, entre 1969 e 1973, para sistemas Unix, sendo a mais utilizada no desenvolvimento de sistemas embutidos de recursos limitados. O sucesso desta linguagem de

programação está relacionado com a facilidade de criar compiladores para uma determinada arquitectura. Daí ser possível utilizá-la na grande maioria dos processadores. Os programas desenvolvidos em C permitem uma enorme portabilidade uma vez que existem compiladores e bibliotecas em C para uma grande maioria de processadores.

A portabilidade tem sido um dos factores que fazem com que esta linguagem seja muito utilizada, sendo as alterações necessárias ao código fonte muito reduzidas. Outra grande vantagem desta linguagem, quando comparada com linguagens consideradas de alto nível, é o facto de permitir, por exemplo, o acesso directo a determinada posição de memória, o que em muitos casos pode ser bastante útil.

3.1.1 URJTAG

O software URJTAG está instalado no *router* e tem como finalidade interagir com a FPGA ou a CPLD através da interface JTAG.

O URJTAG é um software que permite através da interface JTAG o acesso à infraestrutura de teste das FPGAs, definida na norma IEEE 1149.1. O URJTAG é um software livre, com licença GNU (*General Public License*), ou seja, é possível alterar este programa adaptando-o às necessidades específicas de cada utilizador.

O URJTAG foi criado com base no projecto OpenWince desenvolvido por Marcel Telka [Openwince, 2011].

Este software tem suporte para vários cabos de ligação, paralelos ou USB. Alguns dos cabos suportados são:

Porta paralela:

- Arcom JTAG;
- Altera ByteBlaster;
- Altera ByteBlaster II;
- Altera ByteBlasterMV Parallel Port Download Cable;
- Xilinx DLC5 JTAG Parallel Cable III.

Porta USB:

- Altera USB-Blaster;
- Xilinx Plataform USB Cable;
- Amontec JTAG;
- Olimex ARM-USB-JTAG.

A utilização do comando *help cable* permite visualizar todos os cabos suportados pela versão actual do URJTAG.

Para utilizar o URJTAG primeiro é necessário conectar, através do adaptador JTAG, o dispositivo que se pretende testar, executando depois o comando *JTAG* para iniciar o URJTAG, o qual produz a seguinte saída:

```
WARNING: UrJTAG may damage your hardware!
Type "quit" to exit, "help" for help.
```

Os comandos básicos para a utilização deste software são:

- **help**, disponibiliza a ajuda;
- **cable**, permite configurar os parâmetros do cabo a utilizar. Este geralmente é o primeiro comando a ser executado;
- **print**, permite visualizar o caminho JTAG e as instruções activas;
- **svf**, executa comandos svf a partir de um ficheiro;
- **quit**, permite sair do programa terminando a sua consola;
- **detect**, detecta dispositivos ligados através da interface JTAG.

Os comandos apresentados anteriormente são apenas alguns dos comandos disponibilizados. A listagem completa de comandos suportados pelo URJTAG pode ser obtida através do comando *help*. É ainda possível obter informação detalhada de um determinado comando utilizando o comando help, ex. *help cable*. [URJTAG, 2010]

Após alguns testes efectuados este foi o software escolhido para interagir com a CCI. A escolha deste programa deveu-se ao facto de ser possível a programação do CI através de ficheiros svf [Xilinx, 2010], assim como a possibilidade de interacção através da linha de comandos. Como a unidade de processamento é um *router*, existe a necessidade de que o programa que interage com a CCI seja o mais simples possível devido às limitações de memória e capacidade de processamento que o *router* impõe.

3.1.2 Pseudo-terminais

A interacção entre o Servidor em C e o URJTAG é feita utilizando pseudo-terminais.

Tradicionalmente os dados de entrada de um processo são obtidos a partir do teclado e os resultados genéricos (*stdout* e *stderr*) são direccionados para o ecrã.

De uma forma simplificada, a Figura 18 permite visualizar a forma tradicional de passagem de dados através do teclado para o processo e do processo para a interface com o utilizador (monitor).

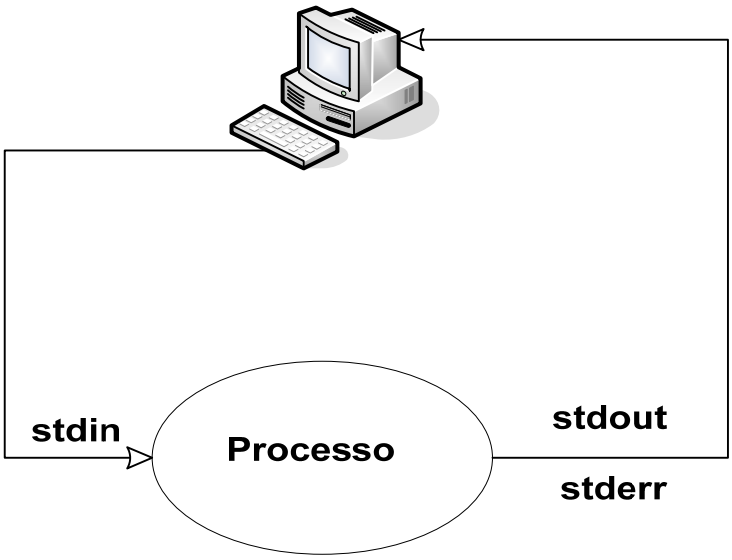

Figura 18 - Entrada e saída de informação de um processo

Alguns sistemas operativos, como, por exemplo, o Linux, permitem redireccionar as entradas e os resultados. Esta funcionalidade tem como intuito obter a informação de uma fonte diferente do teclado e direccionar a informação produzida para um destino diferente do monitor. Esta funcionalidade é bastante útil pois permite que o processo escreva para um determinado resultado ou leia de uma determinada entrada sem que para isso conheça os detalhes do dispositivo. Desta forma é possível correr programas que obtenham a informação de entrada de diferentes fontes, como o teclado, um ficheiro ou até mesmo de um outro processo (exemplo *pipes*).

Os pseudo-terminais servem para se controlar correctamente a partir de um programa, um outro programa que só foi preparado para ser controlado a partir de um terminal. Um exemplo de um programa cuja interacção é efectuada a partir do terminal é o URJTAG.

Os pseudo-terminais não são terminais reais mas sim um tipo de terminais abstractos que consistem num par de dispositivos virtuais, que proporcionam um canal de comunicação bidireccional. Um dos pontos extremos do canal de comunicação é chamado de *master* e o outro ponto extremo é denominado *slave.*

O *slave* proporciona uma interface com um comportamento em tudo igual a um terminal clássico. Desta forma, um programa que execute através do terminal pode ser executado a partir do ponto de comunicação *slave* do pseudo-terminal, sendo controlado pelo ponto de comunicação *master.*

Tudo o que seja escrito no *master* pode ser passado para o processo que está a ser executo no *slave* e tudo o que seja escrito no *slave* pode ser lido no *master*. Desta forma, o processo que esteja a ser executado no *master* pode enviar informação para o processo que esteja a ser executado no *slave.* O funcionamento dos pseudo-terminais pode ser visto como um canal bidireccional entre dois processos, que permite o envio de informação num extremo e a recepção do resultado no outro extremo do canal de comunicação [Niu et al., 2009].

A Figura 19 descreve o funcionamento da comunicação entre processos utilizando pseudo-terminais.

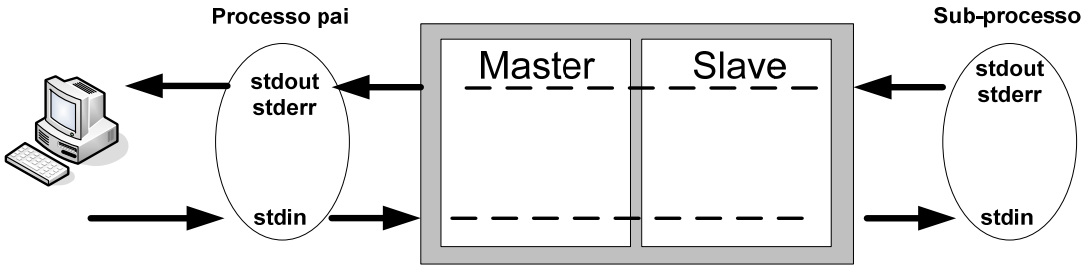

Figura 19 - Descrição do funcionamento do pseudo-terminal

Os pseudo-terminais são bastante úteis quando é necessário controlar um programa a partir de um outro programa. A utilização dos pseudo-terminais é iniciada pelo processo pai, ou seja, o processo que envia informação para o sub-processo. O sub--processo por sua vez executa/trata a informação recebida e retorna o resultado para o processo pai.

Assim sendo, tem-se sempre dois processos: o processo pai que lê a informação do teclado e a redirecciona para o *master* do pseudo-terminal; o processo filho para onde a entrada e saída de informação é redireccionada pelo *slave*.

3.2 Concepção e implementação do cliente

Definidas as tecnologias utilizadas na camada intermédia, passou-se à escolha da plataforma que melhor se adequa ao desenvolvimento da interface com o utilizador.

A Figura 20 apresenta a arquitectura simplificada do laboratório remoto onde é destacada a interface.

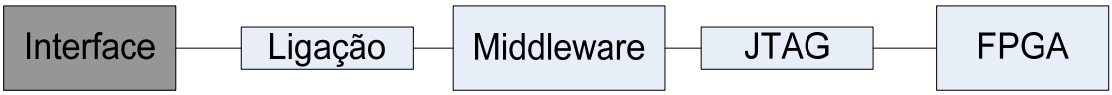

Figura 20 - Arquitectura simplificada do laboratório remoto: interface

Através de uma breve pesquisa foi possível concluir que existe um grande número de plataformas capazes de serem utilizadas na implementação de interfaces para *Desktop*. No entanto, são duas as que se destacam pela sua popularidade: Java e C#.

No que diz respeito à utilização do C# para o desenvolvimento da interface, deve-se ter em conta a necessidade de instalar a *framework* do .NET. Esta linguagem de programação foi lançada em Janeiro de 2002, encontrando-se numa fase bastante madura, com um crescimento de utilizadores muito grande, o que lhe confere o estatuto de uma das linguagens de programação mais utilizadas no mundo. Através desta linguagem é possível criar aplicações para *Desktop*, Web e para dispositivos móveis. A comunicação com outras aplicações pode ser garantida através de Web Services, *firmware*, pedidos http ou *remoting*.

Como ponto negativo, pode-se destacar que a linguagem C# é propriedade da Microsoft. Em termos de portabilidade, já é possível executar aplicações desenvolvidas em C# em várias plataformas, tais como Linux, Solaris e Mac OS. A portabilidade do C# só é possível devido ao projecto *mono* [Mono, 2011]. O *mono* é uma plataforma projectada para permitir aos programadores desenvolver facilmente aplicações multi-plataforma. Este projecto tem como intuito reduzir as barreiras no desenvolvimento de aplicações. O *mono* é uma implementação *open source* da Microsoft.Net baseada nos padrões ECMA [ECMA, 2011] e *Common Language Runtime* para C#. Este projecto tem uma comunidade muito activa e entusiasta que o tem impulsionado para níveis de popularidade muito interessantes. Algumas das chaves do sucesso deste projecto são:

- Multi-plataforma;
- Permite executar aplicações ASP .Net e Windows forms;
- É um projecto *open source.*

Através do projecto *mono* existe a garantia de portabilidade desta linguagem, contudo esta portabilidade fica dependente dos desenvolvimentos do projecto *mono* e da disponibilidade dos seus entusiastas em lhe dar continuidade.

A linguagem de programação Java é a mais utilizada pelos programadores em todo o mundo. Este estatuto deve-se muito à sua portabilidade, uma vez que os programas

desenvolvidos com esta linguagem podem ser executados em qualquer plataforma que tenha instalada a *Java Virtual Machine*. A máquina virtual é responsável pela tradução das instruções de alto nível em "código máquina". Uma vez compilado o código em java, este pode ser executado em qualquer plataforma, desde que tenha instalada a máquina virtual, sem que seja preciso voltar a compilar o programa. A máquina virtual Java é responsável por oferecer um nível de abstracção que permite a execução do programa independentemente do hardware ou do sistema operativo. Esta é uma linguagem de programação orientada a objectos, com uma sintaxe simples (o que facilita a escrita e leitura do código desenvolvido), robusta e de enorme portabilidade.

A Tabela 4 permite comparar as principais características das linguagens de programação C# e Java.

Tabela 4 - Cruzamento das características das linguagens Java e C#

Característica	**C#**	**Java**
Linguagem	Proprietária	Open Source
Portabilidade	Sim	Sim
Rapidez	Boa	Boa
Robustez	Grande	Grande
Sockets	TCP e UDP	TCP e UDP
Remoting	WebOrb, AMF .Net, FluorineFx	WebOrb, Red5, BlazeDS, GraniteDS
Web Services	SOAP e REST	SOAP e REST

Da análise da tabela 4 conclui-se que as duas linguagens de programação são muito semelhantes. Decidiu-se optar pela linguagem Java devido ao facto desta ser *open source* e por ser multiplataforma, o que lhe confere portabilidade.

3.3 Comunicação entre a interface e o middleware

Figura 21 - Arquitectura simplificada do laboratório remoto: ligação

A escolha da tecnologia de comunicação entre a interface e a camada intermédia foi condicionada pelas linguagens de programação utilizadas na camada intermédia, C ou C++, devido às restrições impostas pela utilização do *router* como unidade de processamento. Uma vez que a escolha recaíu sobre a linguagem de programação C para o *middleware*, para fazer a ligação entre a interface e o *middleware* optou-se pela utilização de *sockets*.

Embora esta escolha tenha sido condicionada, é de destacar que os *sockets* são uma excelente escolha, porque permite uma comunicação bastante eficiente em termos de rapidez e de fiabilidade. Mediante as necessidades de comunicação entre a interface e o *middleware* esta é uma boa solução isto porque se adapta perfeitamente às necessidades deste sistema.

Os *sockets* são um método de comunicação IPC (*Inter Process Communication*) que permite que a informação seja trocada entre aplicações, estando estas na mesma máquina ou em máquinas diferentes (ligadas através de uma rede de comunicação). A informação escrita no *socket* por uma aplicação numa determinada máquina, pode ser lida por outra aplicação numa máquina diferente.

A primeira implementação da API de *sockets* apareceu em 1983, sendo suportada por vários sistemas operativos como o Unix, Linux e Windows. Os pontos fortes dos *sockets* são:

- A rapidez na transferência de dados;
- Baixa latência;
- Escalabilidade;
- Eficácia.

Através da Figura 22 é possível visualizar, de uma forma muito simplificada, a comunicação entre duas aplicações em máquinas diferentes, utilizando *sockets*. Nela pode ver-se a relação entre aplicação, *sockets* e protocolos de comunicação.

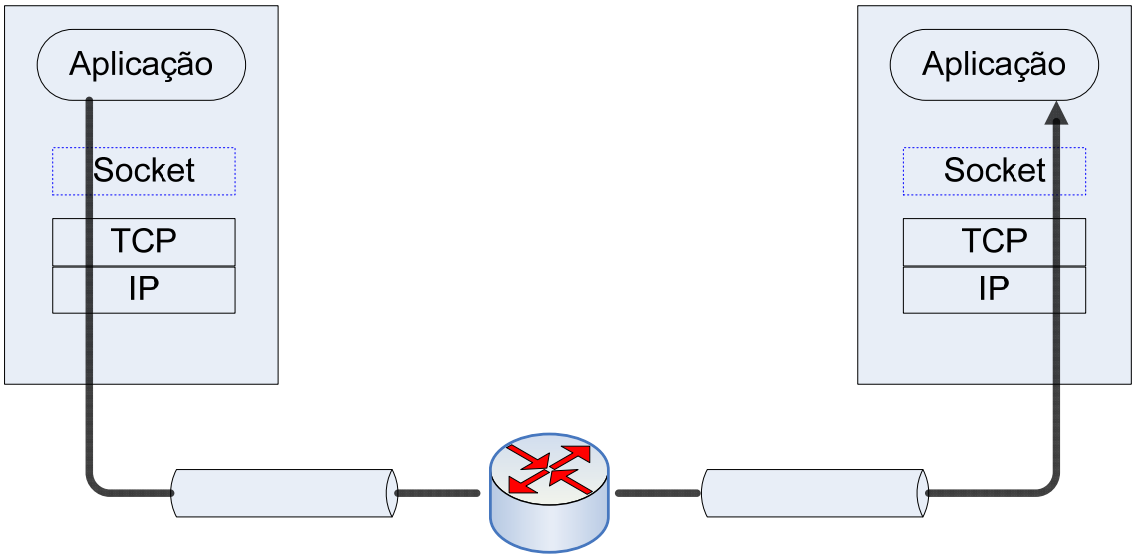

Figura 22 - Comunicação entre duas aplicações usando *sockets* [Donahoo, 2009]

A informação que é passada através dos canais de comunicação, é uma sequência de *bytes* que é construída para ser interpretada por programas. A esta sequência de *bytes* dá-se o nome de pacotes.

Para que a troca de informação entre diferentes máquinas seja possível, são necessários protocolos de comunicação. Os protocolos de comunicação contêm toda a informação necessária para que os pacotes possam ser interpretados e para que possam ser encaminhados para o seu destino.

Um dos protocolos é o protocolo TCP/IP, o qual engloba um conjunto de outros protocolos dos quais é de destacar o TCP (*Transmission Control Protocol*), IP (*Internet Protocol*) e UDP (*User Datagram Protocol*).

O protocolo IP permite que cada pacote seja tratado e entregue pela rede de forma independente, numa analogia com as cartas que são entregues através dos correios, onde é necessário indicar o endereço do destinatário e o do remetente da informação.

O protocolo TCP é um protocolo orientado à conexão, garantindo a ordem dos pacotes durante a comunicação, e detectando e recuperando pacotes perdidos, duplicados e outros erros que possam decorrer da utilização do protocolo IP. Este protocolo é normalmente mais lento que o UDP, mas, no entanto, fornece um serviço confiável, garantindo que a mensagem será entregue.

O protocolo UDP tem uma grande vantagem quando comparado com o TCP: é mais simples. Este protocolo fornece um serviço não confiável, uma vez que não garante que a mensagem será entregue ao destinatário, assim como não garante a ordem dos

pacotes enviados. Os dois protocolos (UDP/TCP) funcionam sobre o protocolo IP [Donahoo, 2009].

É através destes protocolos de comunicação que é efectuada a passagem de informação de uma aplicação para outra, utilizando *sockets*.

Existem dois tipos de *sockets*: *stream* e *datagram.* A diferentes tipos de *sockets* correspondem diferentes tipos de protocolos de comunicação. Os *sockets* do tipo *stream* utilizam o protocolo TCP, garantindo que a mensagem chega ao destino. Os *sockets* do tipo *datagram* utilizam o protocolo UDP. Numa implementação utilizando UDP não existe a garantia de um serviço confiável mas existe um maior desempenho na troca de informação. A utilização de *sockets* do tipo *datagram* normalmente é implementada quando a perda de alguns pacotes não compromete o bom funcionamento do sistema.

Uma vez que o TCP oferece mais vantagens em termos de fiabilidade na troca de mensagens, optou-se por construir a comunicação utilizando *sockets* TCP.

3.4 Comunicação com a FPGA através da interface JTAG

A interface JTAG foi utilizada para permitir efectuar o teste e a programação das CCIs.

A Figura 23 apresenta a arquitectura simplificada do laboratório remoto onde é destacada a interface JTAG.

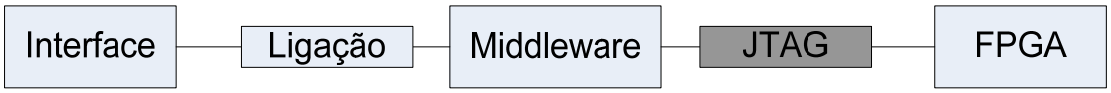

Figura 23 - Arquitectura simplificada do laboratório remoto: JTAG

A ligação entre o *router* e a CCI será efectuada utilizando uma das portas USB do *router*. Para isso será utilizado um cabo USB. Foram testados dois cabos: um da Xilinx, modelo DLC9G; e um outro da Altera, o UsbBlaster [JTAG Cables, 2010].

Para que seja possível a programação da CCI serão utilizados ficheiros .svf que contêm as instruções que serão interpretadas pelo programa URJTAG e passadas para a CCI através da interface JTAG.

4 Descrição da implementação

Neste capítulo é apresentada uma descrição da solução implementada.

Aborda-se inicialmente a arquitectura do novo laboratório remoto, prosseguindo-se com a descrição, de uma forma muito sintetizada, do processo de implementação.

4.1 Arquitectura da solução

Nos capítulos anteriores foram justificadas as opções tomadas em termos de hardware e de software para o desenvolvimento do laboratório remoto.

Os laboratórios tradicionais que envolvem o teste de dispositivos como as FPGAs envolvem espaços físicos com estações de trabalho (normalmente computadores) ligadas a uma única FPGA, fazendo com que, em cada laboratório tradicional, exista um conjunto de recursos que só podem ser utilizados presencialmente. Nos laboratórios tradicionais é necessária a presença física dos alunos, exigindo condições logísticas adequadas.

A utilização de plataformas de simulação tem sido uma outra solução para a escassez dos recursos, contudo não faculta aos alunos uma compreensão adequada e aprofundada de todos os conceitos e dificuldades que a manipulação física dos equipamentos oferece.

O grande objectivo por parte das organizações é a tentativa de maximizar os recursos e minimizar os custos logísticos e de manutenção. É neste cenário que os laboratórios remotos têm um grande destaque, pois oferecem a possibilidade aos alunos destas organizações de interagirem com os laboratórios por um período mais alargado de tempo e sem que estes necessitem de grandes condições em termos logísticos.

Tendo em conta o que foi descrito ao longo dos capítulos anteriores, a necessidade de aceder a dispositivos lógicos programáveis para programação e teste da sua funcionalidade, levou a que fosse implementada uma solução de baixo custo mas sem nunca descurar o desempenho. Para isso foi criada uma solução que, tirando partido das características de diferentes linguagens de programação, melhor se adaptasse à funcionalidade e ao ambiente em que iria operar.

A arquitectura da nova solução está visível na Figura 24, onde se pode destacar a lógica da aplicação que terá que operar na plataforma *OpenWrt* a correr no *router* ASUS.

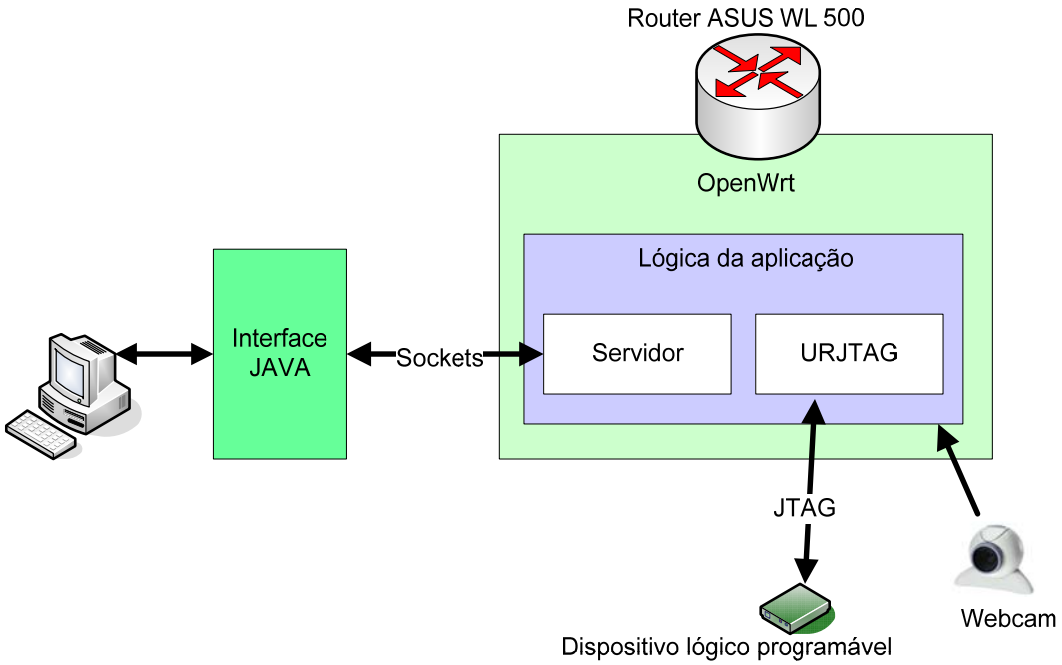

Figura 24 - Arquitectura do novo laboratório remoto

Com esta arquitectura pretende-se solucionar o problema proposto, permitindo o acesso remoto via infraestrutura de teste IEEE1149.1 a dispositivos lógicos programáveis.

O utilizador interage com o laboratório remoto através da camada de apresentação, desenvolvida em Java, e através da qual é possível enviar comandos e ficheiros com a finalidade de testar e programar esses dispositivos. Esta interface comunica através de *sockets* com o *middleware*, ou seja, com um *router* que é a unidade de processamento da arquitectura apresentada.

A lógica da aplicação reúne duas aplicações:

- Uma aplicação Servidor desenvolvida na linguagem C;
- Uma versão adaptada do URJTAG.

Ambas as aplicações foram desenvolvidas na linguagem de programação C.

O servidor tem como principal funcionalidade o envio de informação para o URJTAG, sendo este último o responsável pelo teste da CCI. A Figura 25 permite uma melhor

compreensão da ligação entre estas duas aplicações. A comunicação entre o Servidor e o URJTAG é exequível devido à utilização de pseudo-terminais.

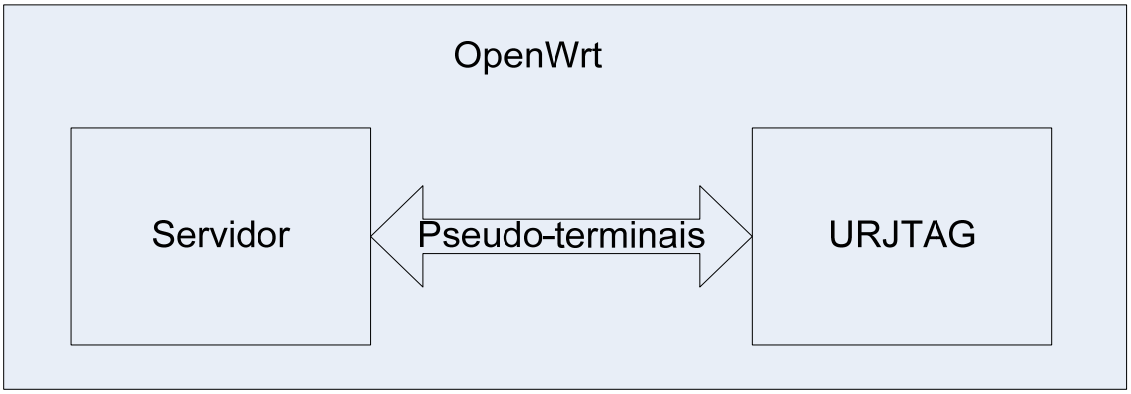

Figura 25 - Comunicação entre aplicações instaladas no *router*

4.2 Implementação da solução

Neste subcapítulo é descrito o processo que conduziu à criação do laboratório remoto de baixo custo.

São descritas a instalação do *firmware OpenWrt* no *router*, o sistema de desenvolvimento do *OpenWrt* e a utilização da SDK (*Software Development Kit*) do *OpenWrt*, que permitiu o desenvolvimento de software para esta plataforma. É igualmente descrito o trabalho realizado no desenvolvimento da interface Java, através da qual o utilizador interage com o laboratório remoto. É ainda apresentada uma descrição do desenvolvimento do Servidor em C e a utilização da API dos pseudo--terminais. Por último, é apresentada uma breve descrição das dificuldades sentidas durante o processo de implementação da solução proposta.

4.2.1 Instalação do *OpenWrt*

Um dos primeiros passos para a implementação do laboratório remoto de baixo custo foi a escolha e instalação de uma versão do *OpenWrt* no *router* Asus WL 500.

Foram testadas duas versões do *OpenWrt*. A primeira foi a Kamikaze, mas foram encontradas algumas dificuldades, nomeadamente durante a utilização da SDK, isto porque esta é algo limitada, o que dificulta a sua utilização. Outra foi a dificuldade em

encontrar algumas bibliotecas compatíveis com esta versão, tornando o trabalho algo moroso e difícil. Na sua falta, ainda se tentou compilar e instalar algumas dessas bibliotecas, mas o trabalho mostrou-se infrutífero.

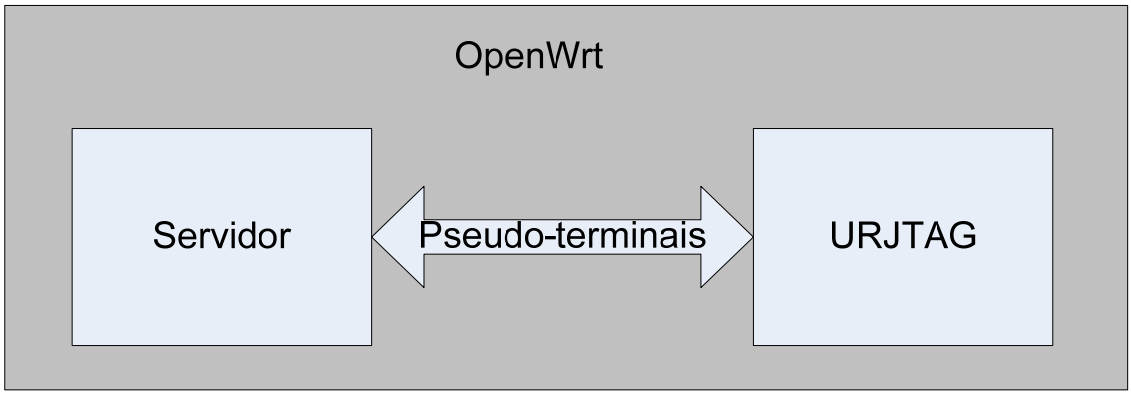

Figura 26 - Arquitectura simplificada do novo laboratorio remote: *OpenWrt*

Uma vez que a tentativa de utilização do Kamikaze falhou, e após um estudo mais cuidadoso das opções e das necessidades que este trabalho apresentava, a versão escolhida do *OpenWrt* recaiu sobre o *Backfire* com o *kernel* 2.6.

Para instalar o *OpenWrt* é possível utilizar o TFTP (*Trivial File Transfer Protocol*) ou a interface Web do *router*, sendo necessário colocar o *router* em modo de diagnóstico.

Para que a instalação seja bem sucedida, a sequência de passos a ser executada é a seguinte [OpenWrt, 2010c]:

- Desligar o cabo de energia do *router*;
- Manter pressionado o botão de *restore*;
- Ligar o cabo de energia do *router*;
- Esperar até que a luz de energia fique intermitente;
- Enviar através de TFTP a imagem do *OpenWrt* para o *router*.

Com o *router* em modo diagnóstico é possível enviar a imagem do *OpenWrt* usando os seguintes comandos:

```
tftp 192.168.1.1
binary
rexmt 1
timeout 60
trace
tftp> put OpenWrt-brcm47xx-squashfs.trx
```

Após esse envio é necessário esperar cerca de 6 minutos, sendo que este processo não pode ser interrompido pois danificaria *firmware* do *router*. Após este processo estar concluído, é possível comunicar com o *router* através de Telnet no IP 192.168.1.1. Uma vez ligado ao *router,* pode ser definida a senha *root* para ser utilizada numa ligação SSH (Secure Shell) através do comando:

```
passwd root
```

Em alternativa, é possível efectuar a ligação ao *router* através da interface Web do Backfire, sendo necessário apenas no Web browser indicar o IP do *router*.

Uma vez estabelecida a ligação com o *router* através de Telnet ou SSH, é possível listar, remover e adicionar novos programas ao *router*, através do comando *opkg*.

Para a criação do laboratório remoto foi necessário adaptar o software e as respectivas bibliotecas para serem executadas no *Backfire*. Para isso foi necessário utilizar a SDK disponibilizada para esta versão do *OpenWrt*.

Para que o *Backfire* detecte as portas USB foi necessário adicionar os seguintes *packages*:

- Usbutils;
- Libusb;
- kmod-usb-core;
- kmod-usb-ohci;
- kmod-usb-storage;
- kmod-usb2.

Estes *packages* são importantes porque a interface JTAG estará ligada ao *router* através da porta USB, bem como a WebCam que permite visualizar em tempo real o estado da experiência.

4.2.2 Sistema de desenvolvimento do *OpenWrt*

O sistema de desenvolvimento disponibilizado pelo *OpenWrt* é um conjunto de ficheiros *makefile* e *scripts* que permitem compilar *packages* que podem ser executados em sistemas embutidos.

Através do sistema disponibilizado pelo *OpenWrt* é possível através de um PC normal com um processador x86 ou x64 (por exemplo), produzir *packages* que serão executados em arquitecturas diferentes, por exemplo, em processadores ARM ou MIPS.

Os ficheiros *makefile* do *OpenWrt*, utilizados para compilar os *packages,* têm uma sintaxe muito própria, exigindo que se tenha um bom conhecimento de toda a arquitectura deste sistema para que se consiga determinar as instruções necessárias à compilação do *package*.

Para melhor compreender as dificuldades sentidas na utilização do *OpenWrt*, convém referir que o sistema disponibilizado para compilar os *packages* é uma série de ficheiros de configuração e *scripts,* distribuídos por uma arquitectura de pastas, que descarregam da internet as dependências necessárias e compilam tudo num *package.* No caso das dependências não existirem no repositório oficial, será necessário desenvolver os *scripts* que permitem criar a dependência e adicioná-la à pasta correcta na estrutura do *OpenWrt*, para que esta possa ser utilizada durante o processo de compilação.

Neste sistema de compilação não existem ficheiros binários - tudo é compilado em tempo real. A grande vantagem da não existência de ficheiros binários é a possibilidade de, apenas mudando um *template*, se poder alterar qualquer etapa do processo, exigindo contudo um maior conhecimento de todo o processo de compilação e um maior esforço na criação de um ficheiro que contenha todas as instruções necessárias à criação do *package*.

Ao ser disponibilizada a *source* do *OpenWrt* permite-se que este não esteja preso a nenhuma versão de determinado sistema, conferindo-lhe uma enorme versatilidade.

A estrutura de pastas do sistema de desenvolvimento do *OpenWrt* assenta em quatro directorias base:

- **Toolchain**: nesta directoria estão as ferramentas (compilador, biblioteca C, …) que serão utilizadas para construir o package. O resultado desta compilação dará origem a duas novas directorias, toolchain_build, que é um directório temporário utilizado para construir o conjunto de ferramentas para uma arquitectura específica, e staging_dir, onde serão colocadas as ferramentas específicas da arquitectura;
- **Target**: nesta directoria estão todos os itens que são específicos de uma determinada plataforma. É nesta directoria que está a informação de configuração do kernel para a plataforma escolhida. De realçar a directoria “target/image” que contém a informação necessária de como construir um package para uma determinada plataforma;
- **Package**: nesta directoria estão todos os packages que serão compilados. É nesta directoria que são incluídas as bibliotecas que serão utilizadas no processo de compilação;

- **Bin:** nesta directoria podem ser encontrados os packages após a compilação. Antes de serem colocados os ficheiros .ipk neste directório ainda passam pelo directório temporário “build_dir” . O directorio “build_dir” permite construir os packages, os quais, no caso de tudo correr dentro do previsto, serão copiados para a directoria “package”.

No sistema *OpenWrt* existem duas formas de construir *packages*: usando a SDK ou utilizando o *OpenWrt* “total”. Ambas as soluções produzem o mesmo resultado final, embora a utilização da SDK seja mais simples e mais rápida na compilação.

É através do *OpenWrt* “total” que é construída a SDK para a plataforma pretendida. Tanto a compilação dos *packages* como a compilação da SDK têm de ser configuradas através do comando `make menuconfig`. Através deste menu é possível seleccionar a plataforma desejada, quais as ferramentas pretendidas para criar os *packages* e quais os *packages*. Após sair deste menu de configuração, é necessário guardar as configurações definidas e executar o comando `make` para construir todo o sistema mediante as especificidades definidas.

A utilização da SDK será descrita com algum detalhe no próximo subcapítulo.

4.2.3 Utilização da SDK do *OpenWrt*

O *Backfire* oferece uma SDK que contém todas as ferramentas, incluindo os compiladores, necessários para o desenvolvimento de *packages* compatíveis com esta versão do *OpenWrt*. Foi através desta SDK que foi feita a adaptação do URJTAG e do servidor em C, possibilitando o teste da CCI.

O primeiro passo para desenvolver *packages* para o *OpenWrt* é obter as ferramentas de desenvolvimento compatíveis com a versão do *OpenWrt* que será utilizado no *router*. Para armazenar em disco estas ferramentas é necessário ter cerca de 3 Gb de espaço livre. Para obter as ferramentas do *Backfire* pode ser utilizado o seguinte comando:

```
svn co svn://svn.OpenWrt.org/OpenWrt/branches/backfire
```

Para criar os *packages* do laboratório remoto foi necessário adicionar algumas bibliotecas às ferramentas de desenvolvimento, como, por exemplo, a *libusb*. Os seguintes comandos mostram como podem ser adicionadas novas bibliotecas, por exemplo, adicionar a biblioteca *libusb*:

```
./scripts/feeds update
./scripts/feeds install libusb
```

A biblioteca *libftdi* não existe no repositório oficial, não sendo possível a sua inclusão através dos comandos anteriormente mencionados. Para que fosse possível adicionar às ferramentas de desenvolvimento do *OpenWrt* a libftdi foi necessário criar um ficheiro *makefile* com as instruções necessárias para descarregar, compilar e configurar esta biblioteca. Após a criação do *makefile* foi necessário criar na directoria *package* uma sudirectoria com o nome *libftdi* ficando nesta subdirectoria o ficheiro *makefile* com as instruções para a criação da referida biblioteca. Este ficheiro *makefile* pode ser consultado no ANEXO I. Para uma melhor compreensão, esta estrutura de pastas, pode ser visualizada através da Figura 27.

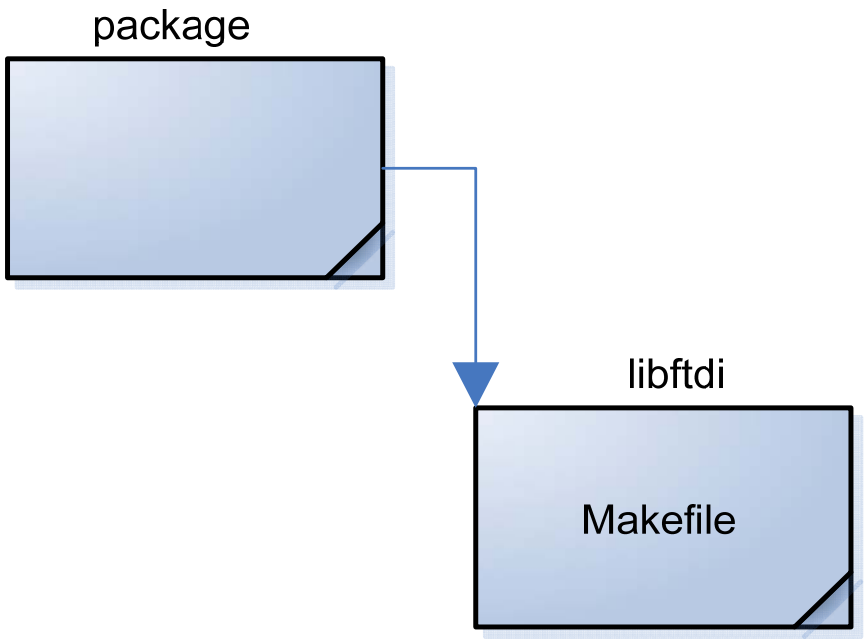

Figura 27 - Estrutura de pastas para a biblioteca *libftdi*

A inclusão das bibliotecas *libftdi* e *libusb*, é fundamental pois no momento de compilação do URJTAG, se estas não forem detectadas não será possível a utilização dos cabos USB. Estas duas bibliotecas têm de ser adicionadas às ferramentas de desenvolvimento do *OpenWrt*, não sendo suficiente compilar as bibliotecas e instala--las posteriormente.

Após adicionar todas as bibliotecas necessárias para o desenvolvimento dos *packages,* é necessário executar o comando `make menuconfig` para iniciar a configuração que levará à criação da SDK com as especificidades do sistema. Este menu de configuração pode ser visualizado na Figura 28.

Durante a configuração da SDK é possível escolher várias opções que permitem a criação e o bom funcionamento dos *packages*. Uma das opções mais relevantes é a

escolha da arquitectura do sistema para o qual se desenvolveu os *packages.* Através da interface mencionada anteriormente é ainda possível definir os *packages* que serão necessários na criação do novo *package,* assim como algumas opções de Kernel e as bibliotecas que serão utilizadas. O *menuconfig* permite definir através de uma interface gráfica todas as opções necessárias para a criação do novo *package.* Desta interface podem destacar as seguintes opções:

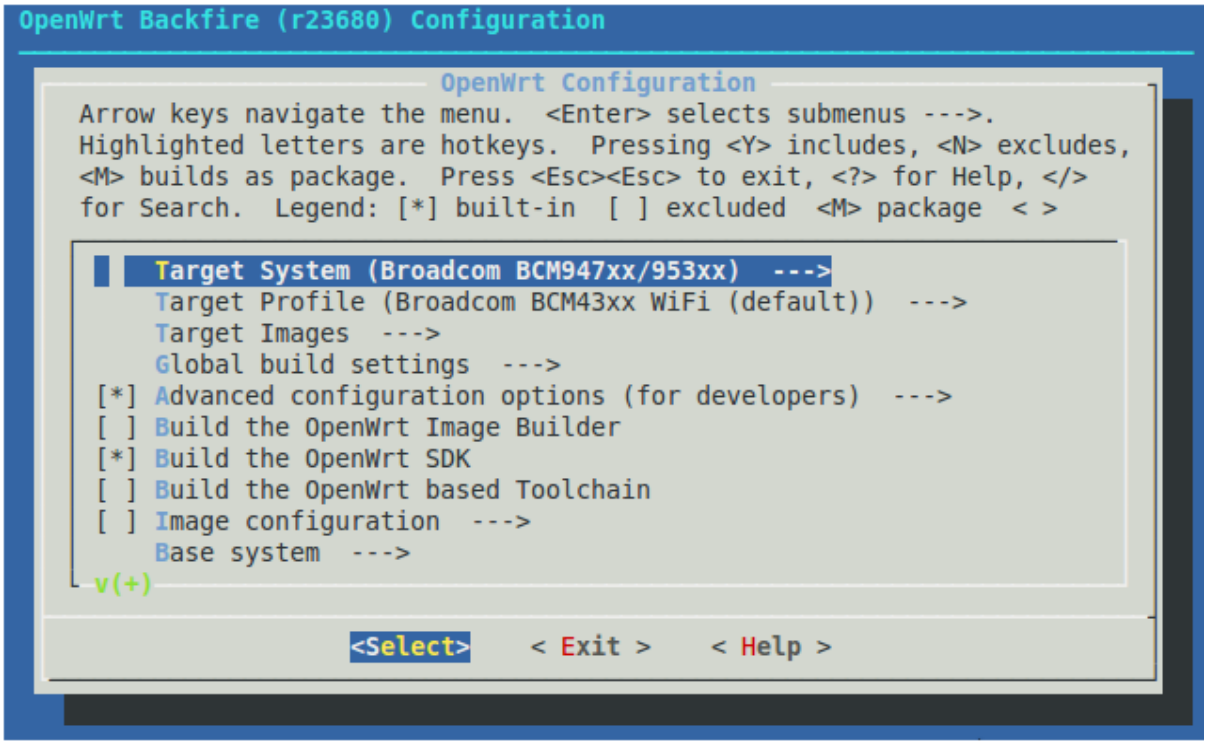

Figura 28 - Menu de configuração da SDK do *Backfire*

- ***Target system***, Através de uma lista de plataformas suportadas, permite seleccionar a plataforma para a qual se pretende criar o *package*. Uma vez que o *Backfire* escolhido foi o *brcm47xx* com o *kernel* 2.6, o *target system* compatível com esta arquitectura é o *Broadcom BCM947xx/953xx;*
- ***Package selection***, Permite seleccionar os *packages* que serão construídos. Nesta lista devem ser incluídos os *packages default;*
- ***Kernel modules,*** Permite especificar alguns *drivers*, sendo usado tipicamente para definir módulos USB e interfaces de rede;
- ***Libraries,*** permite escolher as bibliotecas necessárias ao bom funcionamento do software. No caso do URJTAG é necessário especificar duas bibliotecas de software, a *libftdi* e a *libusb*. Estas duas bibliotecas possibilitam a utilização de cabos USB. Durante o desenvolvimento desta solução utilizou-se dois cabos, o

DLC9G da *Xilinx* e o UsbBlaster da *Altera,* sendo que ambos necessitam da biblioteca *libusb*, e no caso do UsbBlaster é necessária também a utilização da biblioteca *libftdi*. A escolha das bibliotecas é fundamental antes de compilar o URJTAG, isto porque ao compilar se não forem detectadas estas duas bibliotecas, não será possível a utilização dos referidos cabos.

Após especificar todas as configurações necessárias é necessário utilizar o seguinte comando:

```
make V=99
```

O comando *make* permite configurar, compilar e efectuar o *download* das ferramentas seleccionadas. É nesta fase que é gerada a SDK e todos os *packages* especificados aquando da configuração da ferramenta de desenvolvimento para o *Backfire*. Todo o software gerado nesta fase, incluindo as bibliotecas, pode ser encontrado na directoria "bin/brcm/"

O sistema *OpenWrt* é basicamente um conjunto de *Makefiles* que configura, transfere e compila software mediante determinadas opções.

Para criar *packages* com a SDK do *OpenWrt* é necessário estar algo familiarizado com os ficheiros *Makefile* do Unix e os ficheiros *Makefile* do *OpenWrt*, sendo ainda necessária a criação de uma estrutura de pastas específica onde é colocado o código fonte do projecto e os ficheiros *Makefile*.

A estrutura de pastas é criada dentro da pasta "*package*" da SDK, onde é necessário criar uma pasta com o nome do projecto e, dentro dessa, uma outra pasta de nome "*src*".

A disposição das pastas explicada nos parágrafos anteriores pode ser consultada mais em pormenor na Figura 29, que contém a estrutura das pastas, mostrando também a localização dos ficheiros *Makefile*. Como se pode visualizar, todas as pastas foram criadas dentro da directoria do projecto.

Na directoria *package* deverão existir tantas subdirectorias quanto o número de *packages* que se pretende compilar. O *makefile* existente nesta directoria tem como finalidade controlar a compilação de todas as subdirectorias, sendo que para isso é criada uma lista com a informação das ferramentas que serão desenvolvidas.

Na directoria do projecto fica o ficheiro "*Makefile* especial". Este ficheiro *Makefile* é bastante importante pois é o responsável por descrever todo o processo de criação do *package*. Na directoria "*src*" deve ser colocado o código fonte e o ficheiro *Makefile* que permite compilar esse mesmo código fonte.

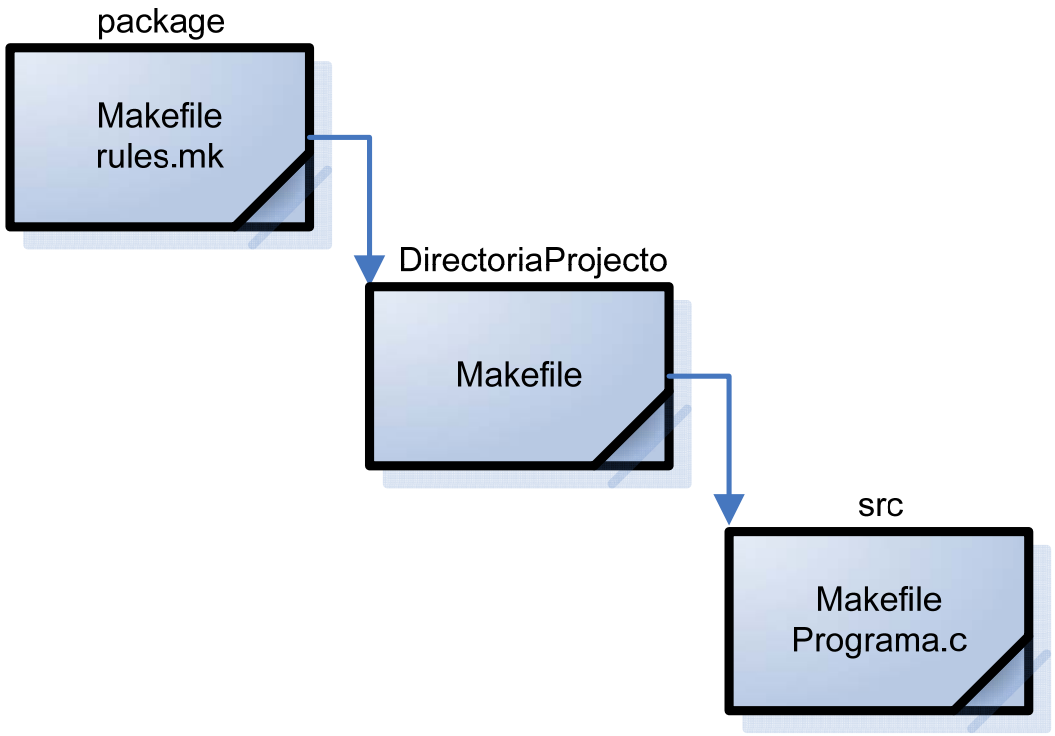

Figura 29 - Estrutura de pastas para criar *packages* usando a SDK do *OpenWrt*

A sintaxe do ficheiro "*Makefile* especial" difere consoante a versão da SDK do *OpenWrt* que se utiliza e nada tem a ver com os ficheiros *Makefile* do Unix. A escrita deste ficheiro requer alguns cuidados como os espaçamentos, as indentações e os espaços em branco no final das linhas que não estejam comentadas.

O ficheiro "*Makefile* especial" é usado para descrever todo o processo de criação/adaptação do programa para que este possa ser executado no *OpenWrt*. A compreensão deste ficheiro é fundamental para o desenvolvimento de programas para o *OpenWrt*. Na Figura 30 pode ser visualizada parte do ficheiro *makefile* que permite compilar o programa URJTAG, através das ferramentas de desenvolvimento do *OpenWrt*. Este mesmo ficheiro pode ser consultado no ANEXO II.

As variáveis usadas neste ficheiro são [OpenWrt, 2010b]:

- **PKG_NAME**, permite definir o nome do package;
- **PKG_VERSION**, permite definir a versão do package;
- **PKG_SOURCE**, permite definir o nome do código fonte que se pretende compilar;

- **PKG_SOURCE_URL**, nesta variável é definido o URL do código fonte, isto se se pretender descarregar o código fonte de um servidor;
- **PKG_MD5SUM**, esta variável define o *checksum* que valida se o código fonte foi descarregado correctamente;
- **PKG_BUILD_DIR**, esta variável define para onde é descarregado o código fonte antes de ser descompactado;
- **PKG_CAT**, esta variável define como descompactar o código fonte, no caso deste ser descarregado a partir de um servidor (zcat, bzcat, unzip);

```
PKG_NAME:=urjtag
PKG_VERSION:=0.10
PKG_RELEASE:=1
PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz
PKG_SOURCE_URL:=http://ignum.dl.sourceforge.net/project/$(PKG_NAME)
PKG_MD5SUM:=f7d1236a1e3ed2cf37cff1987f046195
PKG_CAT:=zcat
PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)-$(PKG_VERSION)
PKG_INSTALL_DIR:=$(PKG_BUILD_DIR)/ipkg-install
include $(INCLUDE_DIR)/package.mk
define Package/urjtag
    SECTION:=none
    CATEGORY:=Utilities
    DEPENDS:=+libusb +libftdi
    TITLE:=Program to control JTAG cables and devices
    URL:=http://www.urjtag.org
endef
define Package/urjtag/description
    A programa to talk to various JTAG cables and devices
endef
define Build/Prepare
    $(call Build/Prepare/Default)
    chmod -R u+w $(PKG_BUILD_DIR)
endef
```

Figura 30 - Ficheiro *Makefile* do programa URJTAG

As próximas variáveis são utilizadas para definir a informação do *package*:

- **CATEGORY**, em que categoria o programa se insere;
- **TITLE**, pequena descrição do programa;
- **DESCRIPTION**, descrição mais detalhada do programa;
- **URL**, informação sobre onde se pode obter o programa original;

- **DEPENDS**, informação sobre os programas que têm que estar instalados antes deste.

Os ficheiros *makefile* permitem criar automaticamente os *packages* .ipk que serão instalados no *router*. Para uma melhor compreensão será apresentada a linha do ficheiro m*akefile* que permite criar o *package* .ipk:

```
$(eval $(call BuildPackage,urjtag))
```

Após a criação da estrutura de pastas e dos respectivos ficheiros m*akefile* é utilizado o seguinte comando para compilar os programas URJTAG e o servidor em *sockets*:

```
make V=99
```

O ficheiro makefile que permite compilar o servidor em *sockets* pode ser consultado no ANEXO III.

Da compilação dos programas, URJTAG e do servidor em *sockets*, resultam dois ficheiros do tipo *.ipk* na directoria "*bin/brcm47xx/packages*" dentro da pasta da SDK do *Backfire*. Estes dois ficheiros podem ser copiados para o *router* através do comando *scp*, podendo ser utilizado da seguinte forma:

```
scp urjtag.ipk 192.168.1.1@root
```

Após efectuar a cópia dos *packages* para o *router*, estes são instalados através do utilitário *opk.*

O programa URJTAG permite interagir com as CCIs através da porta série, sendo possível a utilização desta porta para programar/testar a CCI. Para o sistema desenvolvido, a utilização da porta série não é necessária porque não se utiliza esta funcionalidade. Por isso foram excluídas as ferramentas que possibilitavam esta interacção.

Para que o URJTAG detectasse os cabos USB, foi necessário adicionar a biblioteca *libusb* e *libftdi* à SDK. Através do *menuconfig* da SDK é possível activar as bibliotecas que são necessárias para a compilação do programa. É de realçar a importância da selecção das referidas bibliotecas, porque no momento de compilação do *package* se estas não existirem, o *package* será criado, mas contudo não será possível a utilização dos cabos USB.

O software de compilação do *OpenWrt* assume especial importância uma vez que faz o "strip" dos ficheiros e compila tudo para que estes ocupem o menor espaço possível. Para que os *packages* ocupem o menor espaço possível é removido o máximo de informação de "debugging" e "linking" dos executáveis. Esta redução de espaço assume especial destaque, uma vez que o pretendido era instalar estas aplicações num *router* que tem apenas 32 MB de memória RAM.

4.2.4 Interface

No desenvolvimento da interface teve-se como cuidados especiais garantir a portabilidade e utilizar tecnologia *open source* (foi desenvolvida em Java) por proporcionar, um conjunto de funcionalidades básicas (uma vez que a lógica de negócio está instalada num *router*) e uma interacção com o utilizador bastante simplificada.

No sentido de permitir as funcionalidades necessárias para que o laboratório remoto funcione dentro do previsto, definiram-se os seguintes requisitos funcionais para a interface:

- Envio de ficheiros (Serial Vector Format – SVF);
- Listagem dos ficheiros enviados;
- Remoção de ficheiros;
- Envio de comandos e recepção do respectivo resultado.

A possibilidade de transferência de ficheiros do tipo SVF possibilita que a FPGA ou CPLD seja programada através deste ficheiro.

Nos ficheiros SVF são representadas as operações JTAG de forma textual, usando uma sintaxe e um conjunto de regras simples que possibilitam aos programadores escrever as novas funcionalidades que pretendem ver executadas pela lógica de teste. É neste ficheiro que são escritos os comandos que serão interpretados pelo URJTAG. O URJTAG tem a capacidade de ler e interpretar o conteúdo do ficheiro e gerar as instruções que serão enviadas através da interface JTAG e que possibilitarão o teste ou reconfiguração das novas funcionalidades do dispositivo.

O ficheiro SVF é um meio de troca de informação que beneficia de processos de normalização, sendo suportado por vários fabricantes, como é o caso da *Altera, Xilinx, Lattice, Cypress, Actel*.

Para permitir a comunicação entre a interface Java e o Servidor desenvolvido em C foram utilizados *sockets*. A Figura 31 permite visualizar essa comunicação.

Os *sockets* permitem a comunicação entre duas aplicações estando elas instaladas na mesma máquina ou em máquinas diferentes.

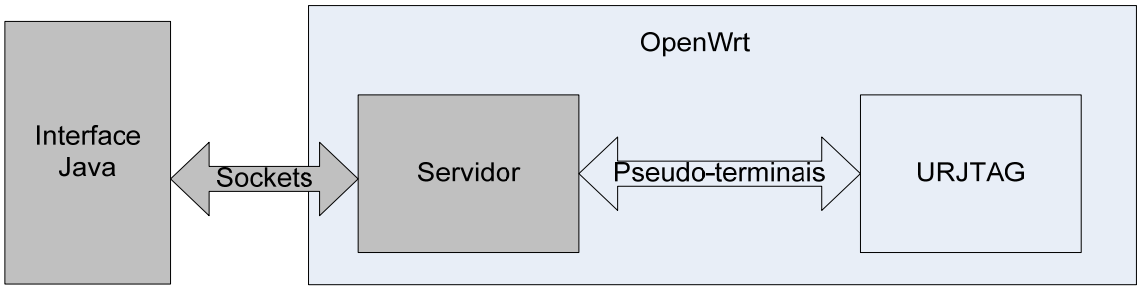

Figura 31 - Comunicação entre a interface e o servidor

Um *socket* existe apenas dentro de um domínio de comunicação, podendo ser identificado através do formato do endereço IP. O domínio mais frequente para a comunicação entre aplicações em máquinas diferentes é o *AF_INET* (IPv4). No entanto, começa a ganhar algum destaque o *AF_INET6* (IPv6). Através da Tabela 5 é possível comparar as principais diferenças entre os tipos de domínios que podem ser utilizados em *sockets*.

Tabela 5 - Tipos de domínios utilizados em *sockets* e as suas principais diferenças

Dominio	Comunicação	Comunicação entre aplicações	Estrutura
AF_UNIX	Kernel	Na mesma máquina	sockaddr_un
AF_INET	IPv4	Através de IPv4	sockaddr_in
AF_INET6	IPv6	Através de IPv6	sockaddr_in6

Existem dois tipos de *sockets*: *stream (SOCK_STREAM) e datagram (SOCK_DGRAM).* Na Tabela 6 podem visualizar-se as principais diferenças entre estes dois tipos.

Numa comunicação como a que se implementou, para se colocar as duas aplicações (interface Java e o Servidor em C) a comunicar, são necessários dois passos essenciais:

- Cada uma das aplicações tem de criar um *socket*, sendo o *socket* o responsável por permitir a comunicação entre aplicações;
- O *socket* do Servidor tem de ser registado num endereço conhecido pela aplicação cliente, para que esta consiga estabelecer a ligação com ele.

Tabela 6 - Tipos de *sockets* e suas principais características

Característica	Tipo de *socket*	
	Datagram	**Stream**
Bidireccional		X
Orientado à conexão		X
Entrega da informação garantida		X
Orientado à mensagem	X	

A criação do *socket* é efectuada utilizando a função *socket(),* que devolve um descritor de ficheiro usado para referenciar o *socket* em futuras chamadas:

```
int sock = socket(AF_INET, SOCK_STREAM, 0);
```

Na implementação efectuada utilizam-se *sockets* do tipo *stream* que são indicados na criação do *socket* através do parâmetro `SOCK_STREAM`. Este tipo de *socket* é muito semelhante à utilização de *pipes*, isto porque os *pipes* também permitem uma comunicação bidireccional entre duas aplicações. A grande diferença é que a comunicação utilizando *sockets* pode interligar duas máquinas através de uma rede. Os *sockets* do tipo *stream* funcionam aos "pares", isto porque são orientados à conexão, garantindo que os dados são entregues no destino, sendo este o motivo pelo qual se escolheu *stream sockets* (comunicação fiável).

Para que seja possível o *socket* aceitar ligações é necessário utilizar a função *listen():*

```
listen(sock, 5);
```

As ligações das aplicações cliente são aceites através da função *accept()*, que devolve um descritor de ficheiro para o novo *socket* que acabou de efectuar a conexão:

```
newSock = accept(sock, (struct sockaddr *) &from, (void *) &ad1);
```

De uma forma resumida, a interface em Java cria um cliente *socket* do tipo *stream*, através da função *socket()*, e estabelece a comunicação com a aplicação Servidor, instalada no *router*, utilizando a função *connect()*. Assim que a ligação seja efectuada com sucesso, é possível enviar informação nas duas direcções, utilizando a função *read()* e a função *write().* A comunicação pode ser terminada utilizando a função *close()*.

4.2.5 Servidor e Pseudo-terminais

Os pseudo-terminais (também conhecidos como pty) são um par de dispositivos virtuais que permitem uma comunicação bidireccional entre dois processos. Estes dispositivos virtuais disponibilizam um canal IPC. Este tipo de tecnologia assume especial relevância sempre que é necessário colocar dois processos a comunicar directamente um com o outro.

Os pseudo-terminais criam um canal de comunicação em que os dois pontos extremos do canal criado são vulgarmente conhecidos como *master* e *slave*.

O lado *slave* do canal criado pelos pseudo-terminais fornece uma interface que funciona exactamente como um terminal clássico. Assim sendo, um processo que esteja à espera de informação do terminal, pode receber essa mesma informação do *slave*, sendo controlado por outro programa ligado ao *master*. Desta forma, tudo o que seja escrito no *master* será lido pelo processo ligado ao *slave*, e tudo o que seja escrito no *slave* será lido pelo processo ligado ao *master*.

Como se pode visualizar na Figura 32, pretendeu-se colocar a aplicação Servidor a comunicar com o programa URJTAG.

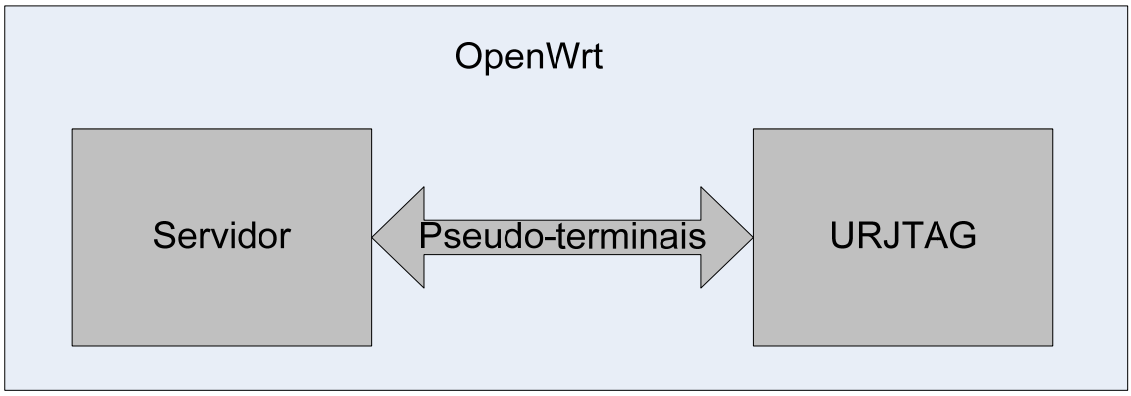

Figura 32 - Arquitectura simplificada do novo laboratório remoto: pseudo-terminais

O servidor será o responsável por enviar informação para o URJTAG, o qual por sua vez comunica pela interface JTAG com a CCI.

Como o URJTAG tem um funcionamento controlado pelo terminal, foi criado um programa Servidor que comunica com o URJTAG através de pseudo-terminais. Assim sendo, ao lado *master* do canal está ligado o Servidor e ao lado *slave* está ligado o URJTAG.

A Figura 33 ilustra como é utilizado o pty no novo laboratório remoto e como cada aplicação interage possibilitando o funcionamento do laboratório.

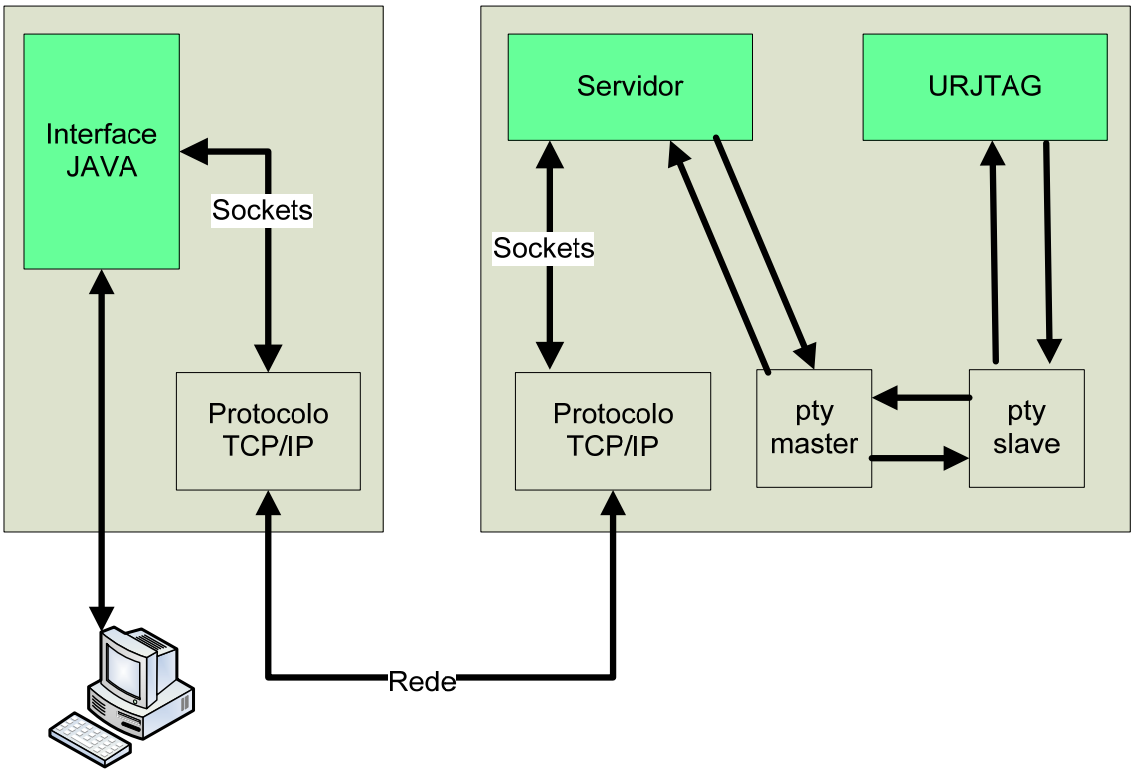

Figura 33 – Interação entre aplicações no laboratório remoto desenvolvido

Uma aplicação que utilize pseudo-terminais tipicamente assegura os seguintes passos:

- A aplicação que implementa os pseudo-terminais abre o dispositivo *master* do canal de comunicação;
- A aplicação cria um processo filho através da função *fork(),* sendo este processo responsável pelas seguintes enumerações:
    - É efectuada uma chamada à função *setsid().* Esta função permite que o processo filho perca o controlo do terminal.
    - É aberto o *slave* do canal de comunicação e este assume o controlo do terminal.
    - Através da função *dup()* são direccionadas as entradas, os resultados e os erros para o descritor correspondente ao *slave*.
    - Por fim, é feita uma chamada a uma função *exec(),* neste caso *execlp(),* para iniciar o programa que está interligado com o *slave*.

Assim sendo, e detalhando um pouco mais a implementação, o primeiro passo para criar o pty é através da função *posix_openpt()*, a qual permite criar um descritor de ficheiro que identifica o dispositivo.

```
fdm = posix_openpt(O_RDWR);
```

O parâmetro *O_RDWR* permite abrir o dispositivo para leitura e escrita. A variável *fdm* será o identificador do pty criado. O funcionamento da função *posix_openpt()* pode ser comparada à função *open()*, isto porque o funcionamento é baseado em descritores de ficheiros. Existe um limite de pseudo-terminais que podem ser criados, limite esse controlado por uma opção de configuração de kernel (CONFIG_UNIX98_PTYS), cujo valor por omissão é 256, mas que pode ir até 2048.

Após a criação do pty, foram utilizadas as seguintes funções:

- `grantpt(fdm)`, a qual permite alterar o domínio e as permissões do *slave* que está referenciado pelo descritor do *master*;
- `unlockpt(fdm)`, a qual é utilizada antes de qualquer tentativa de leitura/escrita no *slave*, e que permite que sejam efectuadas as inicializações necessárias (por exemplo: alteração das permissões através da função *grantpt*), antes que algum processo obtenha permissões para utilizar o *slave*;
- `open(ptsname(fdm), O_RDWR)`, a qual, após desbloquear o *slave* através da função *unlokpt,* permite criar um descritor que identifica o *slave* e através do qual é possível efectuar a comunicação nesta extremidade do canal de comunicação.

A execução do programa URJTAG foi efectuada através da função:

```
execlp("jtag", "jtag", NULL);
```

Como é normal nas funções *exec,* a função *execlp* substitui a imagem do processo actual pela imagem do novo processo. Desta forma é possível executar um novo programa com o identificador do processo que efectuou a chamada da função *exec,* sobrepondo a imagem do processo original.

4.2.6 Principais dificuldades sentidas durante o desenvolvimento do laboratório remoto

Durante o desenvolvimento deste laboratório remoto foram sentidas algumas dificuldades, tais como:

- A falta de experiência na programação em ambiente Unix;

- Dificuldade em utilizar os ficheiros *makefile*, para a criação dos *packages* que seriam instalados no *router*, isto porque os *makefiles* do *OpenWrt* nada têm a ver com os *makefiles* normais, exigindo o conhecimento prévio da sua sintaxe assim como o conhecimento do sistema *OpenWrt*;
- Dificuldade em compilar os *packages* utilizando a SDK do *OpenWrt*.

Durante a criação dos *packages* que seriam instalados no *router*, foram sentidas algumas dificuldades porque os ficheiros *makefile* do *OpenWrt* têm uma sintaxe muito própria, exigindo que se tenha um bom conhecimento de toda a arquitectura deste sistema. Devido às especificidades de cada programa, a criação do *makefile* não é trivial, isto porque estes ficheiros têm de conter todas as instruções necessárias para que o programa seja compilado e o respectivo *package* criado.

A compilação dos *packages* não foi uma tarefa fácil e exigiu bastante esforço. Um exemplo das dificuldades sentidas foi a não existência de todas as bibliotecas necessárias no repositório oficial do *OpenWrt*. Devido à falta das bibliotecas, foram testadas outras alternativas, tais como o teste de diferentes cabos que não precisariam dessas bibliotecas. Uma vez que tais alternativas não se mostraram ser viáveis, foi necessário pesquisar a forma de criar novas bibliotecas e as adicionar ao *OpenWrt*.

A utilização das ferramentas de desenvolvimento do *OpenWrt* exigiu bastante estudo que levou à compreensão de como toda a estrutura do sistema se interliga/funciona. Uma vez que as ferramentas de desenvolvimento do *OpenWrt* não têm ficheiros binários, mas sim um conjunto de *scripts* e ficheiros *makefile* que coordenam todas as operações, foi necessário compreender toda a interligação e funcionamento destes ficheiros, para que fosse possível a sua utilização.

A versatilidade apresentada pelo *OpenWrt* em não utilizar ficheiros binários, faz com que este não esteja dependente de versões de determinado sistema. Contudo, esta mesma versatilidade complica a sua utilização pelos utilizadores menos experientes.

5 Demonstração de resultados

De forma a validar o trabalho exposto e as decisões tomadas, foram efectuados alguns testes funcionais ao laboratório. Para isso, foram efectuadas algumas experiências que normalmente são aplicadas no teste de CCIs em laboratórios tradicionais.

O objectivo desta demonstração é provar que, apesar dos recursos reduzidos, é possível programar através da interface JTAG uma CCI remotamente.

5.1 A experiência

A implementação do laboratório remoto de baixo custo descrita nos capítulos anteriores levou a que fossem efectuados vários testes de funcionamento para validar o trabalho realizado.

A informação introduzida no interface é transmitida via Ethernet para o *router*, o qual processa e devolve o resultado da operação para a interface. Este sistema tem a particularidade de não efectuar qualquer tipo de pré-processamento da informação recebida: ela é apresentada exactamente como é recebida. Após a conexão entre a interface e o *router* ser estabelecida é inicializado do lado do *router* uma nova instância do programa URJTAG. Nesta fase, a interface apresenta a informação inicial do URJTAG, significando que a conexão foi estabelecida com sucesso e o sistema está a funcionar dentro do previsto. Em caso de falha na conexão são apresentadas mensagens com a descrição do erro que originou essa falha. Como já foi referido nos capítulos anteriores a comunicação entre a interface e o *router* é efectuada através de *sockets* TCP/IP. Uma vez que a comunicação é sequencial, para cada pedido é produzida uma resposta.

Durante o desenvolvimento do laboratório remoto proposto foram utilizadas duas placas para efectuar os respectivos testes de funcionamento. As placas utilizadas foram uma baseada numa FPGA e outra numa CPLD. Para esta demonstração foi utilizada a CPLD.

A CPLD está ligada ao *router* através da interface JTAG, utilizando o cabo UsbBlaster. Este cabo teve que ser modificado porque a extremidade que permite ligar à interface JTAG era originalmente diferente. A Figura 34 mostra o cabo UsbBlaster, conectado com a CPLD através da interface JTAG.

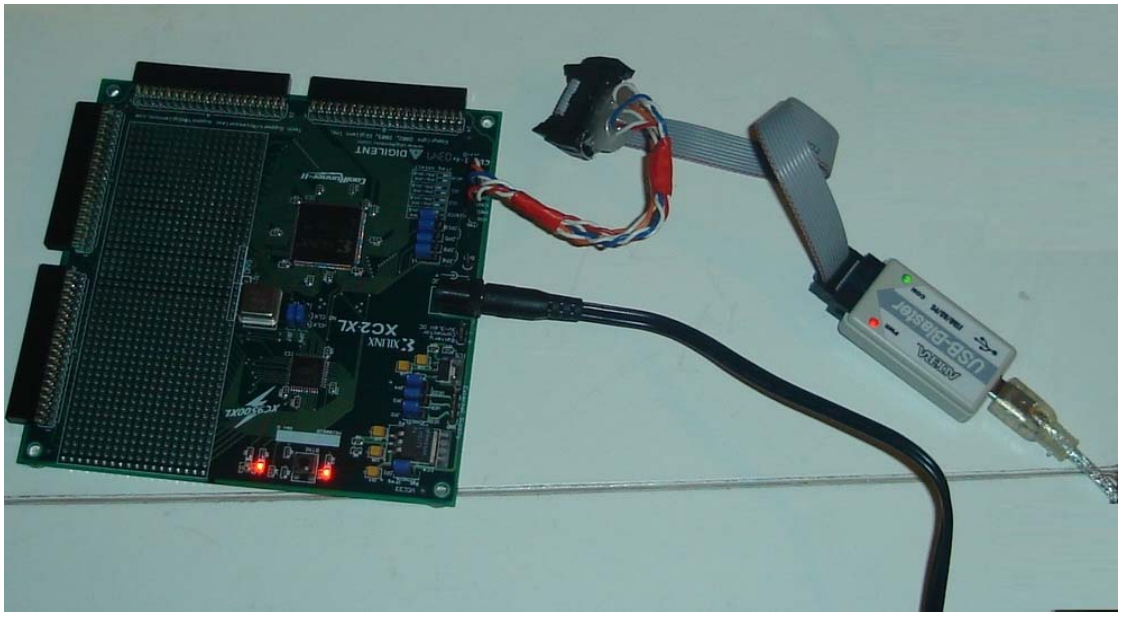

Figura 34 - Cabo UsbBlaster (ligado a uma placa com CPLDs)

O primeiro passo foi possibilitar a escrita da informação através da porta USB onde está ligado o cabo UsbBlaster. Para isso foi criado um *script* que detecta em que porta está ligado o cabo e dá as permissões necessárias para que a comunicação possa ser efectuada. A Figura 35 permite visualizar o script que foi criado com essa finalidade (este script também pode ser consultado no ANEXO IV).

```
#!/bin/sh
#set -x
LINE=`lsusb | grep  "09fb\:6001" `
#echo $LINE
if [ -n "$LINE" ]
then
echo "Update "
#echo "$LINE"
BUS=`echo $LINE | cut -f2 -d" " `
DEVICE=`echo $LINE | cut -f4 -d" "  | tr -d ":" `

#echo $BUS
#echo $DEVICE

TARGET="/proc/bus/usb/$BUS/$DEVICE"

#echo $TARGET

chmod 666 $TARGET
```

Figura 35 – Script para possibilitar a escrita através da porta USB

Para o cabo DLC9G da Xilinx, outro dos cabos testados durante o desenvolvimento deste sistema, é necessário utilizar o seguinte comando que possibilita carregar os *drivers* para que a comunicação seja possível. Este comando varia dependendo da porta USB onde o cabo está ligado:

```
/sbin/fxload -t fx2 -I /usr/share/xusb_emb.hex –D /proc/usb/002/003
```

Para facilitar o carregamento dos *drivers* para o cabo DLC9G foi criado um *script* que pode ser consultado no ANEXO V.

Foi igualmente utilizada uma *webcam* com a finalidade de captar o estado dos LEDs da CPLD. Esta webcam está ligada ao *router* através da porta USB.

O primeiro passo para utilizar a *webcam* no *router* é a instalação dos *drivers* mais adequados. Através do comando `dmesg` é possível visualizar os dispositivos ligados ao *router*. A Figura 36 permite visualizar o resultado da execução do comando `dmesg` onde é possível identificar a presença da *webcam* ligada à porta USB.

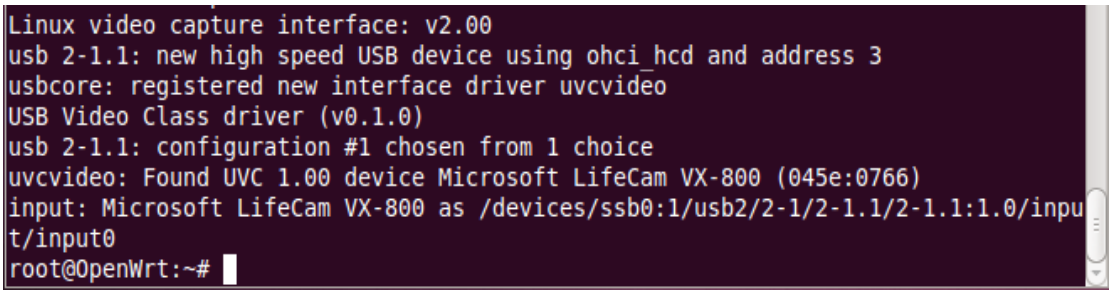

Figura 36 - Visualizar dispositivos ligados ao *router* através do comando dmesg

Uma vez detectada a *webcam*, é necessário instalar as bibliotecas necessárias para que seja possível a captura de imagem através deste dispositivo. Estas podem ser instaladas através do seguinte comando:

```
opkg install kmod-video-core kmod-video-uvc mjpg-streamer
```

Após a correcta instalação das bibliotecas, é necessário inicializar a captura de imagem através do seguinte comando:

```
mjpg_streamer -i "input_uvc.so -d /dev/video0 -y" -o "output_http.so"
```

Na Figura 37 é possível visualizar o resultado do comando que permite a captura de imagem através da *webcam* ligada ao *router*.

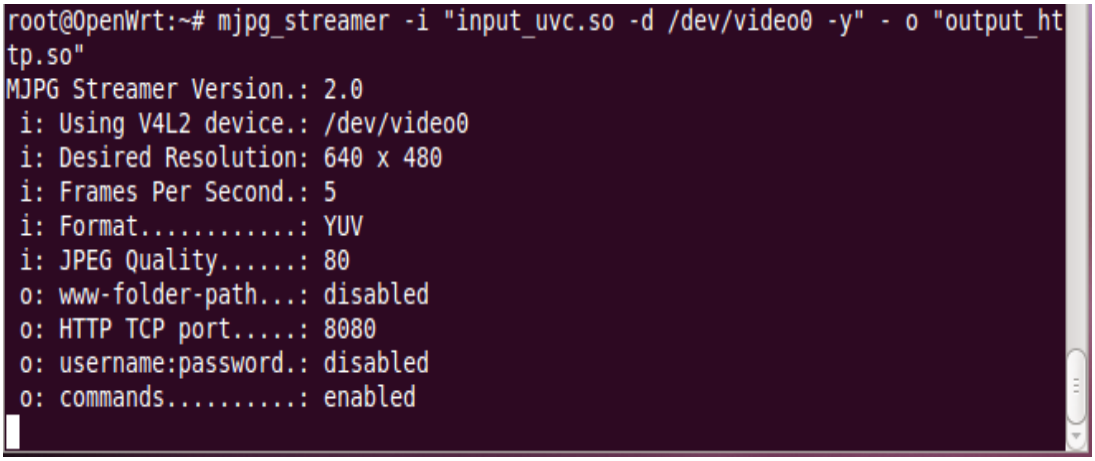

Figura 37 - Inicialização da captura de imagem através da webcam ligada ao *router*

Para que ao ligar o *router* o laboratório remoto estivesse logo disponível, foi efectuada uma automatização da configuração. Para isso foi editado o ficheiro de configuração que é executado no fim do arranque do *router*. Este ficheiro pode ser encontrado em `/etc/rc.local` e normalmente tem o conteúdo mostrado na Figura 38:

```
# Put your custom commands here that should be executed once
# the system init finished. By default this file does nothing.

exit0
```

Figura 38 - Conteúdo do ficheiro de arranque do *router*

Neste ficheiro foi adicionado o script de configuração do cabo USB e o script que permite o arranque do servidor. Sempre que o *router* for inicializado, é executada a configuração inicial em *background* do novo laboratório remoto.

Na Figura 39 está representado o novo laboratório remoto e os dispositivos a ele associados.

Figura 39 - Laboratório remoto desenvolvido

Para a demonstração de resultados foram criados dois ficheiros .svf que possibilitam reprogramar a CPLD com diferentes funcionalidades. Estes dois ficheiros têm de ser enviados previamente para o *router*. A programação de novas funcionalidades na CPLD utilizando os ficheiros svf exige os seguintes comandos pela ordem apresentada:

```
cable usbblaster
```

```
detect
part 1
svf cpld01.svf
```

A interface apresentada para este laboratório poderá não ser a mais adequada para apresentar aos alunos, por se tratar de um protótipo bastante simplificado.

Foram concluídos todos os passos necessários para que seja possível uma interacção com o laboratório. Os resultados retornados ao utilizador são apresentados de forma clara e semelhante à utilização do programa URJTAG localmente.

6 Conclusão

Neste capítulo é resumido todo o trabalho efectuado ao longo deste projecto e que foi descrito nos capítulos anteriores.

Nos próximos subcapítulos será efectuada uma comparação entre os objectivos propostos e os resultados alcançados. Por último, será apresentada uma breve descrição de algumas soluções que poderiam tornar o sistema apresentado mais funcional.

6.1 Objectivos alcançados

Este trabalho teve como objectivo implementar um laboratório remoto de baixo custo que permitisse através da interface JTAG programar dispositivos lógicos programáveis ou testar CCIs. Com o intuito de manter os custos controlados, todo o desenvolvimento assentou na tecnologia *open source* e na utilização de um *router* como unidade de processamento.

No que diz respeito à interface criada, esta é bastante simples, permitindo contudo efectuar a interacção com o laboratório remoto, extraindo todas as potencialidades da solução apresentada.

Para chegar a estes resultados, a arquitectura deste laboratório foi dividida em camadas. A interface foi desenvolvida em Java e comunica com a camada de negócio através de *sockets*. A camada de negócio foi instalada num *router* com o *firmware OpenWrt*, possibilitando desta forma ter uma unidade de processamento de muito baixo custo e com uma capacidade de processamento muito satisfatória. Para que a camada de negócio comunique com a CCI é utilizada a interface JTAG, possibilitando testar a CCI e programar qualquer dispositivo lógico programável nela presente.

O laboratório remoto apresentado cumpre os objectivos específicos que foram propostos inicialmente. A escolha do *router* para unidade de processamento assim como a escolha de tecnologia *open source* possibilitaram manter os custos muito controlados, sem descurar o bom funcionamento da solução apresentada.

O programa URJTAG foi uma escolha acertada, visto que a sua simplicidade permite que este possa ser executado no *router* sem gastar demasiados recursos, não comprometendo o funcionamento do laboratório remoto.

6.2 Trabalho futuro

Ainda no contexto deste trabalho e com o intuito de prosseguir com a sua evolução, seria importante tentar melhorar a interface possibilitando uma interacção com o laboratório mais amigável e, consequentemente, mais produtiva e versátil. As alterações na interface devem conduzir a que esta de alguma forma se assemelhe ao ambiente real que o utilizador iria encontrar se estivesse em contacto com um laboratório tradicional, procurando diminuir a sua tradicional resistência em utilizar laboratórios remotos.

A arquitectura da aplicação permite que as alterações na interface do laboratório apresentado não sejam muito dispendiosas.

No seguimento de sugestões para um trabalho futuro, seria interessante implementar um sistema de ajuda que facilitasse o trabalho dos utilizadores, dando-lhes mais meios de adquirir informação.

Seria de igual forma importante guardar os testes efectuados às CCIs para que estes possam ser reutilizados por outros intervenientes.

6.3 Considerações finais

Em suma, após avaliar todos os aspectos referidos ao longo dos vários capítulos desta dissertação, é possível distinguir este trabalho pela aptidão em oferecer uma solução de muito baixo custo na criação de um laboratório remoto, apresentando um sistema distribuído com uma capacidade de processamento bastante aceitável. Neste trabalho pode ser realçada a utilização de um *router* como unidade de processamento, algo que não é muito vulgar na implementação de laboratórios remotos.

Ao longo dos vários capítulos desta dissertação foram apresentados todos os passos que levaram à criação deste laboratório, bem como algumas diferenças entre os laboratórios remotos e os laboratórios tradicionais, ficando visível que a grande diferença é apenas física, tendo o utilizador uma forma diferente de aceder ao laboratório e de ver apresentados os resultados.

Este trabalho permitiu o desenvolvimento de competências na área das tecnologias *open source* e no desenvolvimento de aplicações para sistemas embutidos, algo que nunca foi abordado durante a parte lectiva.

A utilização do *OpenWrt* foi uma experiência muito enriquecedora, porque possibilitou compreender o seu funcionamento e como utilizar as suas ferramentas de desenvolvimento.

Foi ainda possível adquirir conhecimentos em áreas mais ligadas à electrónica, como é o caso da utilização da interface JTAG para o teste de uma CCI e programação de componentes programáveis via essa interface, compreender o que são e para que servem as FPGAs e CPLDs.

Um aspecto muito positivo deste trabalho foi a utilização de um *router* como unidade de processamento, o que constitui uma utilização não habitual deste dispositivo.

Referências

Altera, 2010. “FPGA”. Altera Corporation. http://www.altera.com/products/fpga.html [acedido em 31 de Janeiro de 2010]

Alves, R. G., 1999. Projecto para o Teste e Depuração com base nas Arquitecturas 1149.1 e P1149.4. Faculdade de Engenharia da Universidade do Porto. Abril 1999. Dissertação para a obtenção do grau de Doutor.

ASUS, 2010. "WL-500gp v2". ASUS. http://usa.asus.com/product.aspx?P_ID=jTcQKIRXgNPLo54b&templete=2 [acedido em 18 de Novembro de 2010].

Balestrino, A.; Caiti, A.; Crisostomi, E.; “From Remote Experiments to Web-Based Learning Objects: An Advanced Telelaboratory for Robotics and Control Systems”. University of Pisa, Italia, 2009

Cagiltay E. N.; Aydin E.; Kara A.; “Principles for the Design of a Remote Laboratory: A Case Study on ERRL”. Atilim University, Ankara, Turkey, 2010

Corelis, 2010. “JTAG Tutorial”. Corelis an EWA Company. http://www.corelis.com/education/JTAG_Tutorial.htm [acedido em 12 de janeiro de 2010]

Corelis, 2010b. “ Boundary-Scan for PCB Interconnect Testing and In-System Programming of CPLDs and Flash Memories”. Corelis an EWA Company. http://www.corelis.com/whitepapers/Boundary-Scan_Whitepaper.pdf [acedido em 19 de Janeiro de 2010]

Donahoo, Michael J.; Calvert, Kenneth L. “Practical Guide For Programmers”. Elsevier Inc, 2009, 196 p., ISBN 978-0-12-374540-8.

ECMA, 2011. “Ecma international”. Ecma international. http://www.ecma-international.org/ [acedido em 31 de Maio de 2011]

Embedded, 2010. "Introduction to JTAG". Embedded Systems Design. http://www.embedded.com/story/OEG20021028S0049 [acedido em 23 de Fevereiro de 2010]

ESOP, 2011. “Associação de Empresas de Software Open Source Portuguesas”. ESOP. http://www.esop.pt/debate-sobre-o-papel-do-software-open-source-no-crescimento-da-economia/ [acedido em 01 de Junho de 2011]

Fainelli, Florian, 2008. "The OpenWrt Embedded development framework". January 22, 2008.

Fujii, N.; Koike, N.; “A Time-sharing Remote Laboratory for Hardware Design and Experiment with Shared Resources and Service Management”. Graduate Sch. of Comput. & Inf. Sci., Hosei Univ., Japan, 2005

Hon, Mun-Wai; Russell, Greg; Welch, Michael; “Open Source Software Considerations for Law Enforcement”, Noblis, 2010.

IEEE Std 1149.1, 2001. IEEE Standard Test Access Port and Boundary-Scan Architecture. New York: IEEE Standards Board, July, 2001, 208 p. ISBN 0-7381-2945-3

IEEE Std 1532 2001. IEEE Standard for In-System Configuration of Programmable Devices. New York: IEEE Standards Board, December, 2001, 139 p. ISBN 0-7381-3102-4.

InAccess, 2010. “inAccess Networks's JTAG tools”. InAccess Networks. http://www.inaccessnetworks.com/projects/ianjtag/jtag-intro.html [acedido em 24 de Fevereiro de 2010].

Java, 2011. “Java + You”. Java. http://www.java.com/en/ [acedido em 07 de Março de 2011].

JTAG Cables, 2010 “JTAG Programmer - Unbuffered JTAG Cable”. JTAG Cables. http://jtagcables.com/jtag-programmer [acedido em 02 de Fevereiro de 2010]

JTAG Test, 2010. “JTAG Pinouts”. Secons company. http://www.jtagtest.com/pinouts/ [acedido em 09 de Fevereiro de 2010]

Lewis, N.; Billaud, M.; Geoffroy, D.; Cazenave, P.; Zimmer, T.; “A Distance Measurement Platform Dedicated to Electrical Engineering”. Bordeaux University, França, 2009.

MIPS, 2011. “MIPS Technologies”. MIPS Technologies, Inc. http://www.mips.com/ [Acedido em 15 de Maio de 2011].

MySQL, 2011. “MySQL the world´s most popular open source database”. MySQL. http:// http://www.mysql.com/, [acedido em 13 de Maio de 2011].

Mono, 2011. “Cross platform, open source .NET development framework”. Mono-project. http://www.mono-project.com/Main_Page [acedido em 31 de Maio de 2011]

Navadi, Zainalabedin; "Digital Desing and Implementation with Field Programmable Devices". Springer Science, 2005, 293 p., ISBN 1-4020-8011-5

Niu, Y.; Li, J.; Li, D.; "The Application of Linux Pseudo-terminal Device in the L2TP," iita, vol. 1, pp.326-328, 2009 Third International Symposium on Intelligent Information Technology Application, 2009

Openwince, 2011. “The openwince project”. SourceForge. http://openwince.sourceforge.net/ [acedido em 15 de Maio de 2011].

OpenWrt, 2010. "About OpenWrt". OpenWrt Wireless Freedom. http://wiki.OpenWrt.org/oldwiki/OpenWrtDocs/About [acedido em 08 de Novembro de 2010].

OpenWrt, 2010b. " Creating your own packages". OpenWrt Wireless Freedom. http://wiki.OpenWrt.org/doc/devel/packages#creating.packages.for.kernel.modules [acedido em 10 de Novembro de 2010].

OpenWrt, 2010c. “Installing OpenWrt via TFTP”. OpenWrt Wireless Freedom.

http://wiki.OpenWrt.org/doc/howto/tftp [acedido em 06 de Dezembro de 2010].

OpenWrt, 2011. “Downloads”. OpenWrt Wireless Freedom.

http://downloads.OpenWrt.org/ [acedido em 14 de Maio de 2011].

RedHat, 2011. “Red Hat Enterprise Linux”. Red Hat. http://www.redhat.com/ [acedido em 07 de Março de 2011]

Smith, Gina R.; “FPGAs 101 Everyting you need to know to get started”. Elsevier Inc, 2010, 229 p. , ISBN 978-1-85617-706-1

SourceForge, 2010. “Find, Create, and Publish Open Source software for free”. Sourceforge. http://sourceforge.net/ [acedido em 12 de Maio de 2011].

Suse, 2011. “SUSE Linux Enterprise Server”. Novell. http://www.novell.com/products/server/ [acedido em 07 de Março de 2011]

Tan, H.; DeMara, R. F.; Thakkar, A. J.; Ejnioui, A.; Sattler, J. D.; "Complexity and Performance Tradeoffs with FPGA Partial Reconfiguration Interfaces", submitted to The 13th Reconfigurable Architectures Workshop (RAW'06), Rhodes, Greece, April, 2006.

Tan, H.; DeMara, R. F.; Thakkar, A. J.; Ejnioui, A.; Sattler, J. D.; "Complexity and Performance Evaluation of Two Partial Reconfiguration Interfaces on FPGAs: a Case Study," in Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'06). Las Vegas, Nevada, U.S.A, 2006b.

Teixeira, Pedro, 2010. “Construção de interfaces em Flex para sistemas de experimentação remota”. Instituto Superior de Engenharia do Porto. 2010. Tese de Mestrado.

UrJTAG, 2010. “UrJtag Universal Jtag library, server and tools”. UrJtag. http://urjtag.sourceforge.net//book/_general.html#_interfaces [acedido em 19 de Janeiro de 2010].

Ven, K.; Verelst, I.; Mannaert, H.; “Should You Adopt Open Source Software?”. Univ. of Antwerp, Antuérpia, 2008.

Weber, Steven; "The Success of Open Source". Harvard University Press, 2004, 312 p., ISBN 0-674-01292-5

Xbit, 2010. “ASUS WL500g Premium Wireless Internet Router Review”. Xbit Laboratories. http://www.xbitlabs.com/articles/networking/display/asus-wl500g-premium_4.html [acedido em 18 de Novembro de 2010].

Xilinx, 2010. “SVF and XSVF File Formats for Xilinx Devices”. Xilinx Company. http://japan.xilinx.com/support/documentation/application_notes/xapp503.pdf [acedido em 08 de Fevereiro de 2010]

Xilinx, 2011. “FPGA, CPLD, and EPP Solutions from Xilinx.”. Xilinx Company. http://www.xilinx.com/ [acedido em 13 de Maio de 2011]

ANEXOS

ANEXO I – Makefile libftdi

```
#
# Copyright (C) 2006-2008 OpenWrt.org
#
# This is free software, licensed under the GNU General Public License v2.
# See /LICENSE for more information.
#
include $(TOPDIR)/rules.mk
PKG_NAME:=libftdi
PKG_VERSION:=0.18
PKG_RELEASE:=1
PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz
PKG_SOURCE_URL:=http://www.intra2net.com/en/developer/libftdi/download/
PKG_MD5SUM:=916f65fa68d154621fc0cf1f405f2726
PKG_FIXUP:=libtool
PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)-$(PKG_VERSION)
include $(INCLUDE_DIR)/package.mk
PKG_INSTALL=1
define Package/libftdi
        SECTION:=libs
        CATEGORY:=Libraries
        DEPENDS:=+libftdi
        TITLE:=Library to control and program the FTDI USB controller
        URL:=http://www.intra2net.com/en/developer/libftdi/
endef
define Build/Configure
        $(call Build/Configure/Default, \
                --enable-shared \
```

```
		--enable-static \
	)
endef
TARGET_CFLAGS += $(FPIC)
define Build/InstallDev
	$(INSTALL_DIR) $(2)/bin
	$(INSTALL_BIN) \
		$(PKG_INSTALL_DIR)/usr/bin/libftdi-config \
		$(2)/bin/
	$(SED) \
		's,^\(prefix\|exec_prefix\)=.*,\1=$(STAGING_DIR)/usr,g' \
		$(2)/bin/libftdi-config

	$(INSTALL_DIR) $(1)/usr/include
	$(INSTALL_DATA) \
		$(PKG_INSTALL_DIR)/usr/include/ftdi.h \
		$(1)/usr/include/

	$(INSTALL_DIR) $(1)/usr/lib
	$(CP) \
		$(PKG_INSTALL_DIR)/usr/lib/libftdi*.{la,so*,a} \
		$(1)/usr/lib/

	$(INSTALL_DIR) $(1)/usr/lib/pkgconfig
	$(INSTALL_DATA) \
		$(PKG_INSTALL_DIR)/usr/lib/pkgconfig/libftdi.pc \
		$(1)/usr/lib/pkgconfig/
endef
define Package/libftdi/install
	$(INSTALL_DIR) $(1)/usr/lib
```

```
	$(CP) \
		$(PKG_INSTALL_DIR)/usr/lib/libftdi*.so* \
		$(1)/usr/lib/
endef
$(eval $(call BuildPackage,libftdi))
```

ANEXO II – Makefile UrJtag

```
#
# Copyright (C) 2006 OpenWrt.org
#
# This is free software, licensed under the GNU General Public License v2.
# See /LICENSE for more information.
#

include $(TOPDIR)/rules.mk
PKG_NAME:=urjtag
PKG_VERSION:=0.10
PKG_RELEASE:=1
PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz
PKG_SOURCE_URL:=http://ignum.dl.sourceforge.net/project/$(PKG_NAME)/$(PKG_
NAME)/$(PKG_VERSION)/
PKG_MD5SUM:=f7d1236a1e3ed2cf37cff1987f046195
PKG_CAT:=zcat
PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)-$(PKG_VERSION)
PKG_INSTALL_DIR:=$(PKG_BUILD_DIR)/ipkg-install

include $(INCLUDE_DIR)/package.mk
define Package/urjtag
        SECTION:=none
        CATEGORY:=Utilities
        DEPENDS:=+libusb +libftdi
        TITLE:=Program to control JTAG cables and devices
        URL:=http://www.urjtag.org
endef
```

```
define Package/urjtag/description
	A programa to talk to various JTAG cables and devices
endef

define Build/Prepare
	$(call Build/Prepare/Default)
	chmod -R u+w $(PKG_BUILD_DIR)
endef

define Build/Configure
	$(call Build/Configure/Default, \
		--enable-lowlevel=libftdi \
	)
endef

define Build/Compile
	$(MAKE) -C $(PKG_BUILD_DIR)\
	$(TARGET_CONFIGURE_OPTS) \
	OFLAGS="$(TARGET_CFLAGS)" \
	DESTDIR="$(PKG_INSTALL_DIR)" \
	install
endef


define Package/urjtag/install
	$(INSTALL_DIR) $(1)/usr/bin/
	$(INSTALL_BIN) $(PKG_INSTALL_DIR)/usr/bin/jtag $(1)/usr/bin/
	$(INSTALL_BIN) $(PKG_INSTALL_DIR)/usr/bin/bsdl2jtag $(1)/usr/bin/
	$(INSTALL_DIR) $(1)/usr/share/urjtag
```

```
	$(CP) -rv $(PKG_INSTALL_DIR)/usr/share/urjtag/* $(1)/usr/share/urjtag
endef
$(eval $(call BuildPackage,urjtag))
```

ANEXO III – Makefile ServidorJTAG

```
#
# Copyright (C) 2006 OpenWrt.org
#
# This is free software, licensed under the GNU General Public License v2.
# See /LICENSE for more information.
#
include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/package.mk

#Nome do pacote.
#Deve ser igual ao nome da pasta do pacote.
PKG_Name:=servidorJTAG

#Release:
PKG_RELEASE:=1

PKG_BUILD_DIR := $(BUILD_DIR)/$(PKG_NAME)

define Package/servidorJTAG
	SECTION:=utils
	CATEGORY:=Utilities
	TITLE:=ServidorJTAG
endef

define Package/servidorJTAG/description
	Pacote servidorJTAG.
endef
```

```
define Build/Prepare
	mkdir -p $(PKG_BUILD_DIR)
	$(CP) ./src/* $(PKG_BUILD_DIR)/
endef

define Package/servidorJTAG/install
	$(INSTALL_DIR) $(1)/bin
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/servidorJTAG $(1)/bin/
endef

$(eval $(call BuildPackage,servidorJTAG))
```

ANEXO IV – Script para atribuir permissões de escrita na porta USB onde está ligado o cabo JTAG

```
#!/bin/sh
#set -x
LINE=`lsusb | grep  "09fb\:6001" `
#echo $LINE
if [ -n "$LINE" ]
then
echo "Update"
#echo "$LINE"
BUS=`echo $LINE | cut -f2 -d" " `
DEVICE=`echo $LINE | cut -f4 -d" "  | tr -d ":" `

#echo $BUS
#echo $DEVICE
TARGET="/proc/bus/usb/$BUS/$DEVICE"

#echo $TARGET
chmod 666 $TARGET
```

ANEXO V – Script para carregar os drivers do cabo DLC9G da Xilinx

```
#!/bin/sh
#set -x
LINE=`lsusb | grep  "03fd\:0007" `
#echo $LINE
if [ -n "$LINE" ]
then
echo "Update firmware"
#echo "$LINE"
BUS=`echo $LINE | cut -f2 -d" " `
DEVICE=`echo $LINE | cut -f4 -d" "  | tr -d ":" `

#echo $BUS
#echo $DEVICE

TARGET="/proc/bus/usb/$BUS/$DEVICE"

#echo $TARGET
/sbin/fxload  -t fx2 -I /usr/share/xusb_emb.hex -D $TARGET
```