UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM AUTOMAÇÃO E SISTEMAS

Roberto de Matos

ARQUITETURA DE CANAIS PARA RÁDIOS DEFINIDOS POR SOFTWARE DE MÚLTIPLAS CAMADAS

Florianópolis
2010

Roberto de Matos

ARQUITETURA DE CANAIS PARA RÁDIOS DEFINIDOS POR SOFTWARE DE MÚLTIPLAS CAMADAS

Trabalho apresentado ao Programa de Pós-Graduação em Automação e Sistemas do Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina como requisito parcial para obtenção do grau de Mestre em Engenharia de Automação e Sistemas.

Orientador:Leandro Buss Becker, Dr.
Co-orientador:Antônio Augusto M. Fröhlich, Dr.

Florianópolis
2010

Roberto de Matos

ARQUITETURA DE CANAIS PARA RÁDIOS DEFINIDOS POR SOFTWARE DE MÚLTIPLAS CAMADAS

Esta Dissertação de Mestrado foi julgada adequada para obtenção do Título de " Mestre" em Engenharia de Automação e Sistemas, Área de Concentração em Sistemas de Computação, e aprovada em sua forma final pelo Programa de Pós-Graduação em Automação e Sistemas da Universidade Federal de Santa Catarina.

Florianópolis, 17 de Setembro de 2010

_____

Prof. Jose Eduardo Ribeiro Cury, Dr.
Coordenador do Curso

Banca Examinadora:

_____

Prof. Leandro Buss Becker, Dr.
Orientador

_____

Prof. Antônio Augusto M. Fröhlich, Dr.
Co-orientador

_____

Prof. Jean Marie Alexandre Farines, Dr.
Universidade Federal de Santa Catarina – UFSC

_____

Prof. Eduardo Augusto Bezerra, Dr.
Universidade Federal de Santa Catarina – UFSC

_____

Prof. Edmar Candeia Gurjão, Dr.
Universidade Federal de Campina Grande – UFCG

Aos meus pais e à minha esposa.

Agradecimentos

Gostaria de aproveitar para agradecer as pessoas que de alguma forma apoiaram esse trabalho:

- Professor Leandro Buss Becker, pelo apoio incondicional em todos os momentos e pela transparência na sua orientação;
- Professor Antônio Augusto Fröhlich, pelas oportunidades dadas para a concretização deste trabalho;
- Todos os amigos do LISHA, em especial o Cleiber, João e Tiago, que muito contribuíram para este trabalho durante o projeto eSDR;
- Meu amigo Hugo Marcondes, pelo companheirismo durante a minha passagem pelo LISHA.

Resumo

Nos últimos anos, foram criados diversos padrões de redes sem fio para garantir o cumprimento de diferentes requisitos de alcance, vazão de dados, segurança e consumo de energia. Muitas vezes isso obriga a integração de componentes que suportam diversas redes sem fio para criação de equipamentos multipadrões. Essa solução tradicional, apesar de robusta, impõe uma série de limitações relacionadas com o espaço físico, consumo de energia e custo dos sistemas integrados, além da falta de flexibilidade para modificações. Uma alternativa a esse cenário tem sido o uso de Rádios Definidos por Software (SDR), os quais possuem a camada física completamente reconfigurável, permitindo flexibilidade em vários parâmetros de comunicação como faixa de freqüência, tipo de modulação e potência de transmissão. Atualmente, existem várias propostas para implementação de SDRs, destacando-se o GNU Radio em conjunto com a placa USRP como opção de baixo custo, que possibilita a criação de rádios funcionais a partir de modelos de alto nível utilizando computadores pessoais. Entretanto, existe uma lacuna no suporte nativo à implementação de múltiplas camadas físicas compartilhando a mesma interface física, o que causa um *overhead* maior que o desejado na criação de sistemas multipadrões. Para superar tal desafio, este trabalho apresenta a concepção de uma arquitetura de canais para múltiplas camadas físicas que se destaca por propor uma interface genérica que atende a todos os tipos de camadas físicas e pode ser suportada por diversos *hardwares* de SDR. Visando a melhor utilização dos recursos a arquitetura proposta permite o deslocamento do estágio de separação de múltiplos canais para o hardware sem a perda de flexibilidade. Para validar a arquitetura proposta foi desenvolvido um protótipo baseado no GNU Radio e USRP2, além das implementações de dois cenários de testes, com múltiplas camadas físicas iguais (IEEE802.15.4) e com múltiplas camadas diferentes (IEEE802.15.4 e IEEE802.11b). Os testes demonstraram uma melhora significativa no desempenho global do sistema e uma simplificação na interface com a camada física, uma vez que não há necessidade de configurar as variáveis relacionadas com os ajustes do próprio hardware.

Palavras-chave: Arquitetura, Rádios Definidos por Software, Múltiplas Camadas Físicas

Abstract

Over the last years many standards for wireless networks were created to support different applications requirements like range, throughput, security, power consumption, and others. Most communication equipments support a single standard, as multi-standard equipments are quite complex and expensive. Moreover, multi-standard equipments have several limitations related to physical space, energy consumption, and the cost of integrated systems, as well as a lack of flexibility for modifications. In this scenario Software Defined Radios (SDR) appear as a suitable alternative, as it has a completely configurable physical layer. This allows for flexibility of various communication parameters such as carrier frequency, modulation, and transmission power. Currently, there are several proposals for the implementation of SDRs. The GNU Radio and the USRP board must be highlighted as a low cost option, allowing the creation of functional radios from high-level models using personal computers. However, there is a gap in the native support to the implementation of multiple physical layers sharing the same physical interface, which causes an *overhead* greater than desired for the creation of multi-standard systems. To overcome this challenge, this work presents the so-called Channel Architecture for Multiple Physical Layers. This proposal consists in a generic interface that is compatible with all sorts of physical layers and that can be supported by several SDR *hardwares*. To optimize the resources utilization, the proposed architecture migrates the separation stage of multiple channels to the hardware without losing flexibility. To validate the proposed architecture it was developed a prototype based on a GNU radio and in the USRP2 board. The experimental evaluation consist of two test scenarios, with multiple similar physical layers (IEEE802.15.4), and with different physical layers (IEEE802.15.4 and IEEE802.11b). The performed experiments demonstrate that the proposed solution presents a significant improvement in the total performance of the system. Additionally it simplifies the interface with the physical layer, as there is no need to configure the variables related to the adjustments of the hardware itself.

Keywords: Architecture, Software Defined Radio, Multi Physical Layers

Lista de Figuras

Lista de Tabelas

Lista de Siglas

Lista de Equações

Sumário

CAPÍTULO 1

Introdução

1.1 Motivação

Os avanços da tecnologia possibilitaram sistemas computacionais cada vez menores, baratos e mais presentes em nossas vidas, modificando a maneira pela qual fazemos as atividades cotidianas. Um dos aspectos mais importantes e estudados na área é a forma de comunicação e interação dos vários equipamentos existentes. Atualmente, essa interação tem caminhado cada vez mais para o mundo sem fio, onde os equipamentos se comunicam por ondas de radio [12, 18]. Neste contexto, foram criados diversos protocolos e estruturas de comunicação. Para uma escolha correta do padrão de comunicação a ser utilizado é necessário levar em conta o domínio da aplicação, o qual delimita os requisitos com relação a quantidades de dados a serem transmitidos, imunidade a fatores externos, alcance, segurança e consumo de energia [24].

Considerando impossível a convergência de todas as redes sem fio para um único protocolo, a solução tem sido criar equipamentos capazes de se comunicarem utilizando padrões de redes, o que tradicionalmente tem sido alcançado integrando diversos componentes, cada um com compatibilidade para um tipo de rede [24]. Um dos exemplos mais diretos é o telefone celular (*smartphone*) que tem a capacidade de comunicação com a rede de telefonia celular (ex.: GSM , HSPA), com a rede sem fio doméstica (ex.: *802.11x*) e com redes pessoais (ex.: *bluethooth*), além disso, existem rádios utilizados somente para recepção, como os receptores de sinais de satélite para localização geográfica e receptores de programas de TV analógica e/ou digital.

Entretanto a implementação tradicional dos equipamentos de comunicação, baseada somente em hardware, apesar de robusta, impõem uma série de limitações. Cada elemento de hardware da cadeia de recepção ou transmissão exerce uma função específica, os componentes são projetados para operar em uma faixa de freqüência e de acordo com um padrão específico. Quando essa faixa de freqüência ou algum outro parâmetro do padrão muda, o rádio não consegue mais se comunicar nesse novo padrão. Para operar sob os novos padrões, todo o sistema tem que ser novamente projetado e os componentes de hardware substituídos. Outra limitação dentro do escopo de múltiplos padrões é a neces-

sidade de um rádio para cada serviço, por exemplo, um dispositivo que oferece conexão *802.11b* e *Bluetooth* pode possuir os mesmos componentes de hardware repetidos, apenas com configurações diferentes. Isso aumenta a necessidade de espaço físico e o consumo de energia, impondo dificuldades para integrar vários serviços em um único dispositivo [12, 18, 24].

Uma alternativa a esse cenário tem sido o uso de Rádios Definidos por Software (SDR - *Software Defined Radio*) [22], que antes eram direcionados somente para sistemas de alto desempenho e atualmente vem tomando mais espaço em sistemas com recursos limitados [1, 2]. Os SDRs possuem a camada física flexível, permitindo modificações em vários parâmetros de comunicação como faixa de freqüência, tipo de modulação, protocolo e potência de transmissão [25], minimizando as limitações dos rádios tradicionais. Além disso, como os componentes são implementados em software, eles podem ser reaproveitados em outras plataformas, testados e modificados facilmente [5].

Atualmente, existem várias propostas [5, 13, 2, 31, 19, 1] para implementação de SDRs, as quais tipicamente utilizam modelos de alto nível para definição do funcionamento do rádio e o conceito de distribuição de processamento por várias unidades heterogêneas, FPGA (*Field Programmable Gate Array*) , DSP (*Digital Signal Processor*) e GPP (*General Purpose Processor*) nas plataformas de uso específico ou no *host*. As diferenças são percebidas nas regras de criação dos blocos de processamento digital de sinais, conexão desses blocos e a maneira como o sinal é processado pela estrutura que forma o rádio. Um dos projetos mais populares da área é o GNU Radio [5], que juntamente com uma plataforma de baixo custo chamada de USRP (*Universal Software Radio Peripheral*) [13], possibilita a criação de rádios funcionais a partir de modelos de alto nível utilizando computadores pessoais.

Contudo, a arquitetura provida pela combinação de baixo custo USRP2 e GNU Radio não prevê suporte nativo a implementação de múltiplas camadas físicas compartilhando a mesma interface física, o que causa um custo maior que o desejado para esse tipo de implementação. Mesmo computadores pessoais modernos não conseguem processar mais de quatro canais em paralelo do protocolo IEEE 802.15.4, o qual é bastante utilizado em redes de sensores sem fio [10]. Assim um dos desafios que se apresenta é diminuir o custo da execução dos blocos em software utilizando arquiteturas que distribuam de maneira eficiente as tarefas entre as unidades de processamento heterogêneas (FPGA, DSP e GPP) sem a

perda de flexibilidade dos blocos de software [17].

Para superar esse desafio esta dissertação apresenta a concepção de uma arquitetura de canais para múltiplas camadas físicas. Mesmo usando o conceito já conhecido de desacoplamento de canal, a arquitetura apresentada se destaca por propor uma interface genérica que atende a todos os tipos de camadas físicas e pode ser suportada por diversos *hardware* de SDR. A arquitetura proposta permite o deslocamento do oneroso estágio para separação de múltiplos canais (translação de freqüência e decimação) integralmente para o hardware, possibilitando a alocação de vários canais em uma mesma interface física de forma transparente e garantindo uma distribuição mais eficiente das tarefas entre hardware (ex.:FPGA) e software (ex.: GPP).

Para validar a arquitetura proposta foi desenvolvido um protótipo baseado no GNU Radio e USRP2. Além das implementações de dois cenários de testes, com múltiplas camadas físicas iguais (IEEE 802.15.4) e com múltiplas camadas diferentes (IEEE802.15.4 e IEEE802.11b). Entretanto os conceitos utilizados sugerem que a utilização da arquitetura é possível em qualquer outro sistema empregado na construção de SDRs, uma vez que não possui dependências das estruturas do GNU Radio ou USRP2. Os testes demonstraram uma melhora significativa no desempenho global do sistema e uma simplificação na interface com a camada física, uma vez que não há necessidade de configurar as variáveis relacionadas com os ajustes do próprio hardware.

1.2 Objetivos

O principal objetivo deste trabalho é propor uma arquitetura para rádios de múltiplas camadas físicas, que simplifique a configuração e o uso em simultâneo de uma mesma interface física pelas implementações das camadas superiores.

A partir deste objetivo principal, são definidos os seguintes objetivos específicos:

- Analisar as cadeias básicas de processamento comum às camadas físicas.
- Propor a distribuição dos blocos de processamento de maneira eficiente entre as unidades de processamento heterogêneas (FPGA, DSP e GPP) sem a perda de flexibilidade dos blocos de software, garantindo o aproveitamento

máximo dos recursos do hardware.

– Propor uma arquitetura para a alocação e controle de canais, que abstraia a complexidade da configuração e os limites dos recursos do hardware.

– Testar e analisar a arquitetura proposta levando em consideração alguns parâmetros como desempenho e ocupação da FPGA.

1.3 Visão geral do texto

O próximo capítulo apresenta as tecnologias relacionadas com o desenvolvimento deste trabalho, onde inicialmente é apresentado a evolução do processador digital de sinal, bem como a fundamentação teórica da tecnologia *Software Defined Radio* e as estruturas internas do GNU Radio e da USRP.

O Capítulo 3 apresenta as principais características e fundamentos da arquitetura de múltiplas camadas físicas desenvolvida neste trabalho. Já o Capítulo 4 apresenta a implementação dessa arquitetura.

O Capítulo 5 apresenta a implementação de dois cenários de testes para validação da arquitetura proposta e uma avaliação de desempenho comparativa com a arquitetura original.

Finalmente são apresentadas no Capítulo 6 as conclusões e propostas de trabalhos futuros relacionados ao tema.

CAPÍTULO 2

Fundamentação Teórica

2.1 Processador Digital de Sinal

Várias tecnologias foram viabilizadas pelo avanço do processamento digital de sinais, que se originou na década de 40 com o refinamento de técnicas para o controle digital de sistemas, mas que somente avançou rapidamente no final da década de 70, com a evolução da microeletrônica e a criação dos Processadores Digitais de Sinais. Isto porque os DSPs contém estruturas dedicadas para melhorar o desempenho dessa classe de algoritmos. O primeiro grande sucesso comercial foi lançado em 1983 pela empresa Texas Instruments, o processador DSP TMS32010 trouxe várias inovações e um conjunto de instruções especiais, que provinha um grande desempenho para a época [23].

Uma das primeiras estruturas adicionadas como implementação em hardware para aplicações de processamento digital de sinais foi a operação MAC (Mutiplicador-Acumulador) , que consiste na multiplicação de dois números e soma do resultado com um valor contido no acumulador. O hardware dedicado possibilita que essa operação seja executada em apenas um ciclo de *clock*, enquanto que para processadores sem essa estrutura dedicada a mesma operação pode levar vários ciclos de *clock* na mesma freqüência. Outro circuito dedicado implementado em hardware é *Barrel Shifter*, que consiste em uma estrutura que pode fazer a rotação de um dado por um número de bits em um ciclo de *clock* [20, 14].

O *Barrel Shifter* pode ser implementado por um conjunto de multiplexadores cascateados, entretanto para possibilitar a adição de estruturas como *MAC* é preciso uma modificação profunda na arquitetura do processador, deixando de lado a implementação dos processadores tradicionais (ex.: *Von Neumann*) e utilizando as arquiteturas *Harvard* e suas variações. A arquitetura *Harvard* tem em sua definição a separação dos barramentos de dados e instruções que acessam memórias diferentes. Mesmo assim, para implementar a operação *MAC* em um ciclo de *clock* é preciso ler a instrução, os dois operadores armazenados na memória e executar a operação. Nesse sentido é preciso modificar a arquitetura Harvard e adicionar mais um barramento para acesso a uma terceira memória. [20, 14]

Outra função específica para otimizar a implementação de algoritmos de

processamento de sinal é o suporte ao "endereço circular", que permite o acesso a um bloco de dados seqüencialmente com auto incremento e retorno ao ponto inicial, o que evita a perda de ciclos de necessidade de instruções de incremento, verificação e salto. O que é muito utilizado para acesso aos coeficientes de filtros e *buffers* seqüenciais de dados. Com a mesma filosofia o suporte eficiente a *loop* são utilizados para executar repetidamente blocos de instrução sem perder ciclos de clock para incrementar e testar o contador de controle ou saltar para o ponto inicial do *loop* [14].

Com o aumento da exigência de desempenho das aplicações, novas filosofias arquiteturais foram sendo adicionadas a arquitetura original dos DSPs, como por exemplo SIMD (*Single Instruction, Multiple Data*) e VLIW (*Very Long Instruction Word*) . O conjunto de instruções SIMD, não é uma classe de arquitetura propriamente dita, mas uma técnica que pode ser adaptada a vários tipos de arquiteturas e tem como principal função utilizar múltiplos elementos de processamento para executar a mesma operação em um conjunto de dados diferentes simultaneamente, possibilitando a exploração do paralelismo no nível do dado [14].

VLIW é uma arquitetura projetada para explorar o paralelismo de instrução, uma vez que executa várias instruções em paralelo. Para isso é necessário que existam múltiplas unidades de execução para processar as instruções ao mesmo tempo, bem como múltiplos decodificadores de instrução, barramentos e registros. A possibilidade de paralelismo é definida em tempo do projeto, ou seja, o conjunto de instruções que será executado em paralelo é definido pelo usuário ou gerador automático de código, não podendo sofrer alterações durante a execução do programa [20, 14].

Essa evolução no suporte as aplicações de processamento digital de sinal possibilitaram que soluções projetadas com componentes fixos de hardware fossem substituídos por DSPs e software, o que facilita a correção de erros de projeto e facilidade de atualização. Além disso, as estruturas especiais (ex.: *MAC*, *Barrel Shifter*,etc.) desenvolvidas para processadores DSPs voltados ao mundo dos embarcados foram incluídos em processadores de uso geral (GPP), como por exemplo, as extensões MMX e SSEx da Intel e 3DNow da AMD. Com isso, o suporte ao processamento digital de sinais por GPPs fez surgir, por exemplo, os *"SoftModems"*, que são *modems* com o mínimo de hardware, projetado para usar os recursos do *host* para fazer parte da modulação e demodulação.

Na próxima seção será apresentado a trajetória dos rádios definidos por software, que se beneficia dessa evolução dos processadores específicos (DSP) e de processadores de uso geral (GPP) para execução dos algoritmos de processamento digital que formam os rádios completamente reconfiguráveis.

2.2 Rádio Definido por Software

Rádio Definido por Software (*Software Defined Radio* – SDR) é uma tecnologia emergente que tem sido pesquisada por mais de uma década e vem modificando a forma como os novos rádios de comunicação estão sendo projetados. O termo SDR foi adotado pela primeira vez por Joseph Mitola [22] para designar rádios com a capacidade de funcionamento em várias faixas de freqüência, tipo de modulação, protocolo e potência de transmissão, onde pelo menos 80% das funcionalidades são providas por software, podendo ser reconfiguráveis ou adaptáveis. O objetivo principal dessa tecnologia é conceber um rádio que virtualmente possa se comunicar utilizando qualquer novo padrão de rede sem fio apenas atualizando o software.

2.2.1 Rádio Convencional x SDR Ideal x SDR Real

Na Tabela 2.1 são apresentadas algumas diferenças entre redes cabeadas e sem fio relacionadas com o meio físico de comunicação. Por exemplo, em uma rede cabeada os nodos são conectados via linhas com propriedades físicas bem definidas e invariáveis durante o tempo, suas propriedades podem ser mapeadas no momento da concepção da rede, enquanto que as propriedades do meio RF são completamente variáveis e suscetíveis a interferências externas com maior facilidade. Além disso, o fio é compartilhado por todos os nós ligados a ele. Em contraste, as comunicações sem fio sofrem do problema do nó escondido, no qual dois nós detectam um terceiro, mas não se detectam entre si, aumentando o número de colisões [9].

Com o meio físico dinâmico somados a heterogeneidade de aplicações e cenários, a especificação e implementação de redes sem fio se torna uma tarefa complexa. Assim, para cada tipo de rede sem fio, é necessário criar uma camada física com uma faixa de canais, modulação e potências bem definidas. Isso

Tabela 2.1: *Diferenças básicas entre redes cabeadas e sem fio*

	Redes cabeadas	**Redes sem fio**
Meio Físico	Meio Estático	Meio RF Dinâmico
	Meio Controlável	Meio Incontrolável
	Meio Único Compartilhado	Nodos Escondidos

Fonte: [9]

permite que a rede tenha o alcance e largura de banda planejada, bem como, imunidade a ruídos e consumo de energia coerentes com o ambiente e os requisitos da aplicação. Todos esses aspectos levaram à criação de vários tipos de redes sem fio, as quais podem, em uma primeira aproximação, ser divididas em três grandes categorias [28]:

1. Interconexão de sistemas WPAN (*Wireless Personal Network*)
2. LANs sem fio (WLAN - *Wireless Local Area Network*)
3. WANs sem fio (WWAN - *Wireless Wide Area Network*)

Na primeira classe são incluídos os padrões utilizados para interconexão de sistemas computacionais de pequeno porte, os quais requisitam pequenas distâncias e taxa de dados limitada. Outra preocupação constante é manter o baixo consumo de energia, por se tratar normalmente de equipamentos alimentados por bateria. Essa classe também engloba a comunicação de pequenos sistemas autônomos que podem formar grandes redes em número de dispositivos, como por exemplo, as redes de sensores sem fio. Os padrões mais conhecidos dessa classe são o Bluetooth e o ZigBee (802.15.4).

A segunda classe é provavelmente a mais difundida de todas. Os padrões 802.11x tornaram popular a conexão de computadores pessoais às redes sem fio, possibilitando taxas de transferências aceitáveis, facilidade de uso e instalação. A adoção desses padrões foi rápida pela inviabilidade de estender uma rede local já estruturada por outras áreas onde não foi previsto cabeamento (ex.: prédios antigos), ou ainda onde o número de usuários é muito grande (ex.: Aeroportos). Outra vantagem desses padrões é a possibilidade do uso de configurações não hierárquicas para interconexão de computadores.

A última classe reúne as tecnologias de rede que abrangem uma grande área geográfica, com freqüência um país ou continente. A telefonia celular é um exemplo desse tipo de rede, a qual já passou por três gerações e atualmente se assemelha às WLANs, com a diferença da distância e da largura de banda. Enquanto as WLANs operam a velocidades de até 144Mbps (ex.: 802.11n) a uma distância de dezenas de metros, as WWANs operam a taxas muito menores (ex.: 7.2Mbps para dispositivos HSPA) a distâncias medidas em quilômetros.

Entretanto, cada vez mais, a interconexão desses sistemas vem acontecendo. Por exemplo, um cenário não raro é um celular recebendo *streams* de música por uma rede residencial sem fio 802.11g e enviando para um fone de ouvido conectado via *bluetooth*, ou ainda, captando sinais de um satélite para determinar sua posição geográfica e enviando essa informação via a rede de telefonia móvel para uma central de gerência de localização. Dessa forma, o aumento de funcionalidades em um único equipamento e a quantidade de padrões de redes sem fio aumentam a complexidade do projeto e geram dificuldade para alcançar os requisitos de tamanho, energia e preço. Já que para cada padrão é necessário um componente diferente que consome esses recursos.

Outro problema é que a arquitetura dos rádios tradicionais, baseada em hardware, apesar de robustas, impõe uma série de limitações de projeto. Cada elemento de hardware da cadeia de recepção ou transmissão exerce uma função específica, os componentes são projetados para operar em uma faixa fixa de freqüências e de acordo com um padrão específico. Quando essa faixa de freqüência ou algum parâmetro do padrão muda, o rádio não consegue mais codificar e decodificar as informações. Para operar sob os novos padrões, todo o sistema tem que ser novamente projetado e os componentes de hardware substituídos. A Figura 2.1 mostra o diagrama da cadeia de recepção de rádio convencional com as separações dos estágios RF (Radio Freqüência) , IF (*Intermediate frequency*) e banda base.

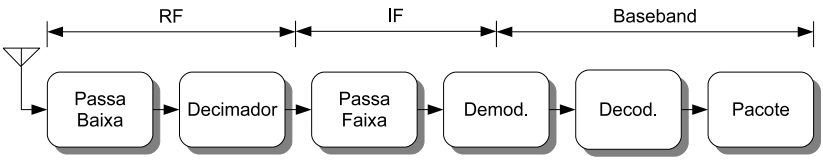


Figura 2.1: *Principais blocos de um Rádio Digital*

Além dos problemas da utilização de vários padrões em um mesmo equipamento e da falta de capacidade de atualização desses padrões nesse receptor, as redes sem fio tradicionais são projetadas para suportar condições de pior caso (*worst case*) podendo resultar em um desperdício do espectro, consumo desnecessário de energia e uso ineficiente da largura de banda. Isso porque, a natureza intrinsecamente dinâmica do meio modificam constantemente as condições dos canais, seja por obstruções momentâneas ou mudança de topologia da rede, afetando a forma de alcançar os requisitos de transmissão e recepção. [6]

Esses problemas podem ser solucionados utilizando a tecnologia de *Software Defined Radio* (Figura 2.2), que possui as seguintes capacidades:

- Integração de múltiplos padrões utilizando a mesma interface física.
- Atualização completa da camada física, o que possibilita a troca para novos padrões de comunicação.
- Adaptação dinâmica durante operação dependendo da situação do canal.

A tecnologia SDR rompe com o paradigma da arquitetura convencional de rádio buscando colocar o software o mais próximo possível da antena para filtrar, demodular e executar outros estágios da cadeia de recepção e transmissão. O SDR ideal elimina quase todo o hardware, mantendo somente o conversor analógico digital(ADC - Analog-to-Digital Converter) amostra o sinal de rádio da antena e o resto é feito por software. Entretanto, não existem ADCs rápidos o suficiente para amostrar toda a largura de banda utilizada. Assim, em um SDR real são necessárias mais algumas etapas em hardware, chamadas de *front-end RF*, para converter o sinal de interesse da alta freqüência para uma freqüência intermediária. A Figura 2.3 exemplifica a relação entre o rádio convencional e os SDRs ideal e real.

2.2.2 Evolução dos SDRs

Em 1991, Joseph Mitola, engenheiro da E-Systems, escreveu um artigo técnico sobre o desenvolvimento de um rádio reconfigurável para a Força Aérea e adotou pela primeira vez o termo SDR. Em 1995, como editor da revista *IEEE Communications*, ele escreveu um artigo sobre arquiteturas de SDR [22] e reuniu uma série de trabalhos tratando de conversão analógica digital de banda larga e

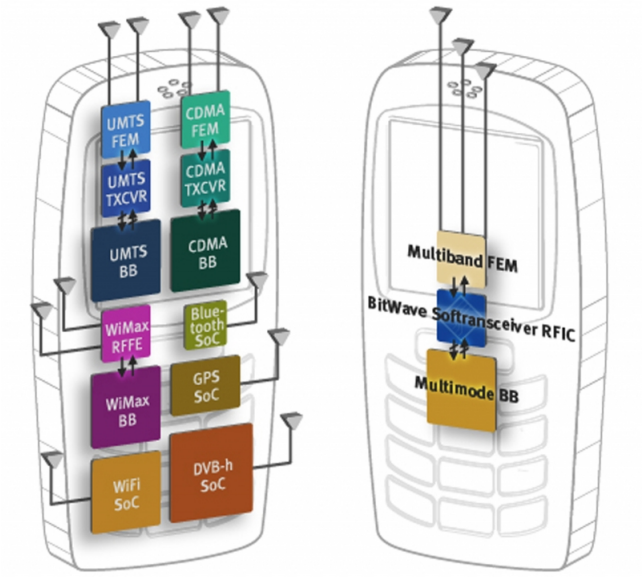
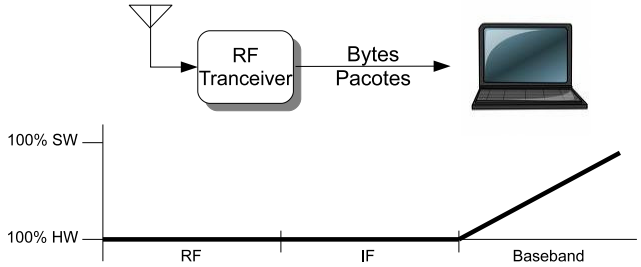


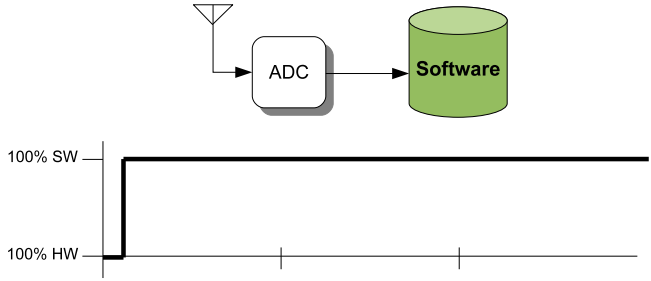
Figura 2.2: *Comparação entre celular atual e com blocos SDR.*
Fonte: pendente

processamento digital de sinais, aproximando várias pesquisas na área ao redor do mundo. Em 1996 Joseph Mitola foi convidado para presidência do Fórum SDR, cujo objetivo era prover uma padronização da nomenclatura, definições e terminologias relevantes relacionadas com toda a área de SDR, de forma a criar uma base de informações para decisões políticas, órgãos reguladores e de normatização.

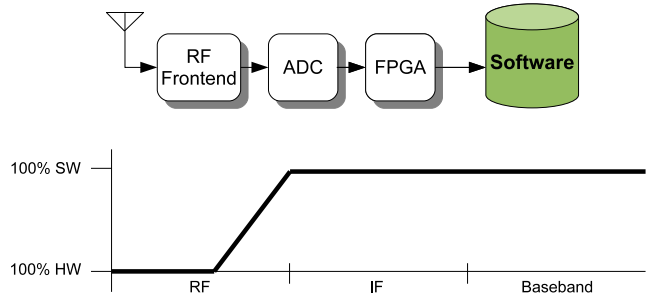
Em 1992, um dos laboratórios da Força Aérea Americana, o AFRL (*Air Force Rome Labs*) , e o (DARPA - *Defense Advanced Research Projects Agency*) juntam forças para começar o projeto *SPEAKeasy1* (SE1), com a meta de criar um rádio que operasse entre 2MHz e 2GHz, com modulações do Exército (VHF , FM e SINCGARS), Força Aérea (VHF AM), Marinha (VHF AM e HF SSB teleprinters) e satélites (microwave QAM), abrangendo todo o espectro de freqüência e modulações das forças armadas americanas. Apesar do tamanho, um *rack* de quase dois metros de altura, e da dificuldade de atualização o projeto alcançou



(a) Rádio Convencional.



(b) SDR Ideal.



(c) SDR Real.

Figura 2.3: *Comparação das perspectivas dos rádios.*

todos os objetivos propostos e é base arquitetural para SDR até hoje.

Após a demonstração bem sucedida do SE1 em 1995, a segunda fase do projeto foi iniciada (*SPEAKeasy2* - SE2) com o objetivo de tornar o sistema menor, mais leve e mais barato, além de permitir uma reconfiguração mais rápida e a capacidade de ser uma ponte entre dois rádios de diferentes tecnologias. O projeto inicialmente planejado para três anos demonstrou um protótipo em quinze meses com total cumprimento dos requisitos, de forma que o desenvolvimento foi interrompido e o equipamento foi colocado em produção com freqüência de operação relativamente limitada, de 4MHz até 400MHz apenas.

Uma das grandes contribuições do SE2 é a proposta de uma arquitetura aberta de software com interfaces bem definidas para cada módulo identificado a partir de uma análise de domínio das cadeias de recepção e transmissão dos rádios. Com a modularização fica mais fácil o reaproveitamento de código e a comunicação inter módulo, diminuindo a dependência de um processador central. Esse projeto também foi o pioneiro no uso de FPGAs para processamento digital de sinais.

Descendente direto dos projetos SPEAKeasy, o projeto JTRS (*Joint Tactical Radio System*) , começou em 1997 e foi financiado pelo Departamento de Defesa dos Estados Unidos (DoD - *Department of Defense*) . O objetivo principal e ambicioso foi a criação de uma geração de rádios flexíveis e interoperáveis para transmissão de dados, áudio e vídeo em um faixa de freqüência de 2MHz até 2GHz, além de incluir criptografia integrada e softwares para gerenciar a criação de MANETs (*Mobile Ad Hoc Networks*) banda larga. Com o escopo muito grande e cronograma irreal o projeto sofreu diversas reformulações e em 2005 foi feito um replanejamento para uma melhor delimitação do escopo e um cronograma mais realístico. A maior contribuição do JTRS foi a criação de um padrão amplamente aceito para desenvolvimento de SDRs, denominado SCA (*Software Communications Architecture*) , esse padrão faz uso de CORBA (*Common Object Request Broker Architecture*) suportado por sistemas operacionais POSIX (*Portable Operating System Interface*) para coordenar e integrar diversos módulos de software para criação do rádio.

Paralelamente aos esforços de padronizações para aplicações militares, em 1995 começa no MIT o projeto *SpectrumWare* [32], que desenvolveu uma abordagem que separa temporalmente os fluxos de amostras dos módulos de software, relaxando as restrições temporais sobre os algoritmos de processamento e a

sua execução [29]. O principal produto desse projeto foi a linguagem *Pspectra* [7] que serviu de base para uma iniciativa comercial, a empresa Vanu, Inc. [30], e uma iniciativa código aberto, o projeto GNU Radio [5].

A empresa Vanu Inc. fundada em 1998 é responsável pela primeira estação rádio base que integra as tecnologias GSM, iDEN (*Integrated Digital Enhanced Network*) e CDMA (*Code division multiple access*) funcionando em um único equipamento aprovado pela FCC (*Federal Communications Commission*) [27], com o grande diferencial de utilizar um SDR executando em plataformas de uso geral, o que possibilita o uso de tecnologias como virtualização.

O projeto GNU Radio foi iniciado com a doação do filantropo John Gilmore em 2001 e tem o intuito de criar um conjunto de software livre para desenvolvimento de SDRs em plataformas de uso geral (ex.: PCs). Com total reformulação em 2004, o GNU Radio não utiliza mais o código original do *Pspectra* e sim um sistema híbrido de *Python* e C++, que possibilita facilidade no uso e alto desempenho para processamento digital de sinais.

2.2.3 GNU Radio

GNU Radio é um conjunto de software livre para criação de SDRs utilizando computadores pessoais combinados com um hardware mínimo para as etapas de conversão analógica digital e *front-end RF*. O uso de plataformas de uso geral permite instalação e desenvolvimento facilitados, o que criou uma grande comunidade de desenvolvedores, propiciando um fortíssimo suporte para novas aplicações. Atualmente, o GNU Radio pode ser executado nos sistemas operacionais Linux, Windows e MAC OSX. [3, 4]

A biblioteca disponibilizada pelo GNU Radio possui blocos para processamento digital de sinal, desde funções básicas até algoritmos completos de filtros, (de)moduladores, (de)codificadores, FFTs entre outros. Além desses, também fazem parte da biblioteca blocos de acesso a arquivos e a *hardware* externos, como por exemplo, placas de som e alto-falantes. Todos esses blocos e mais os que podem ser criados livremente pelos usuários são combinados em grafos de fluxo (*flowgraphs*) para a criação de rádios implementados em software. A Figura 2.4 mostra a representação de um *flowgraph* que implementa um receptor FM faixa estreita.

O *flowgraph* abstrai o fluxo de dados, onde o processamento de sinais é

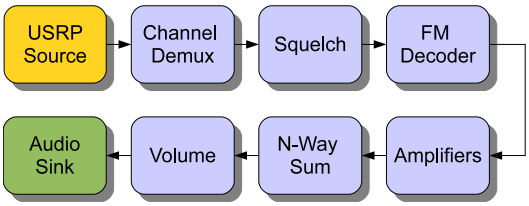


Figura 2.4: *Representação de um* flowgraph *típico do GNU Radio.*

executado pelos blocos e as conexões (vértices) caracterizam a ligação e seqüência desse processamento. A facilidade de uso e a alta performance são alcançadas pela natureza híbrida do GNU Radio, os blocos que exigem desempenho são implementados em C++, enquanto os *flowgraphs* e as interfaces com o usuário são implementados em *Python*.

A estrutura interna do GNU Radio é formada basicamente por quatro componentes: **Blocos de Processamento**; **Controlador de *flowgraph***; **Buffer de Dados**; e **Escalonador** [11].

Blocos de Processamento são os componentes que efetivamente atuam sobre a *stream* e podem ser divididos em três classes: Normal, Fonte (*Source*) e Coletor (*Sink*). A maioria dos blocos é do tipo Normal, os quais possuem entradas, saídas e são responsáveis pelo processamento do sinal nas fases intermediárias do *flowgraph*.

Os blocos Fontes possuem somente saída e sempre começam o *flowgraph*, esses blocos podem ser geradores de sinais sintéticos ou captadores de sinais físicos. A primeira classe pode ser representada pelos blocos *Noise Source*, geradores de diversos tipos de ruídos baseados em geradores de números pseudo-aleatórios; *Signal Source* produz uma forma de onda (ex.: senoidal, co-senoidal ou constante) com uma determinada freqüência e amplitude; ou ainda o *File Source* que lê de um arquivo a *stream* que deverá ser injetada no próximo bloco. Como exemplos de blocos de fontes de sinais físicos podem ser citados o *Audio Source* que captura os dados de audio da entrada *line-in* do PC a uma taxa programável e o *USRP source* que configura e lê o *RX chain* da placa USRP. [16]

Os blocos Coletores possuem somente entrada e consomem a *stream* processada. Exemplos dessa classe de blocos são *Vector sinks* ou *File sinks*, os quais escrevem os dados recebidos em um vetor ou arquivo respectivamente. Os blocos

coletores também servem de interface com o mundo real, como por exemplo, o *Audio sink*, que envia a uma taxa programável para placa de som a *stream* já processada, e *USRP sink* que configura e escreve no *TX chain* da placa USRP, a qual faz a conversão digital analógica e envia via interface de rádio pré-definida.

Todos os blocos de processamento possuem assinaturas de entrada e/ou de saída (*I/O signature*), métodos *work* e *forecast*. Para cada bloco são definidas uma assinatura de entrada e outra de saída, cada assinatura define o número mínimo e máximo de conexões, além do tamanho do tipo de dados que o bloco trabalhará. O método *work* atua sobre os dados de entrada e produz os dados de saída, ou seja, esse método é o núcleo das funções de processamento do bloco. O método *forecast* ajuda o escalonador decidir quando o método *work* deve ser chamado, sendo utilizado para estimar a quantidade de dados de entrada para produzir uma saída. Os dois parâmetros do método *forecast* são o número de itens de saída para produzir cada saída e um vetor de inteiros que salva o número de dados de entrada requeridos.

O **Controlador de *Flowgraph*** (CF) é responsável pela abstração do fluxo de dados, ou seja, a seqüência de como o sinal é processado pelos blocos e as conexões entre eles. O método *connect* especifica como as *streams* de saída serão conectadas às entradas dos próximos blocos, sendo utilizado para construção do *flowgraph*, que automaticamente será criado para processar o stream de dados. Na Figura 2.5 é apresentada a descrição em Python de um *flowgraph* para geração do tom de discagem. A Figura 2.6 apresenta uma representação gráfica do *flowgraph* gerado a partir dessa descrição em *Python*.

O primeiro passo é a criação do *flowgraph* (linha 10). Depois são gerados dois blocos do tipo *Signal Source*, para geração de duas ondas senoidais com 350Hz e 440Hz, e um bloco *Audio Sink* para conduzir o sinal gerado até a placa de som. Depois de criar e configurar os blocos, é utilizado o método *connect* para interligar os blocos *src0* e *src1* em portas diferentes do bloco *dst*. Com isso, as duas ondas senoidais se sobrepõem e no alto-falante pode ser escutado o tom de discagem de uma linha telefônica padrão.

O processo detalhado para criação do *flowgraph* é omitida do usuário, onde a função mais complexa é a alocação dos ***Buffer* de Dados** entre os blocos. Essa função de alocação define o tamanho dos *buffers* considerando a taxa relativa de consumo e produção e os tamanhos dos dados de entrada e saída. Os *Buffers* de Dados são alocados em cada saída e entrada dos blocos de processamento,

```
1  #!/usr/bin/env python
2
3  from gnuradio import gr
4  from gnuradio import audio
5
6  def build_graph ():
7      sampling_freq = 48000
8      ampl = 0.1
9
10     fg = gr.flow_graph ()
11     src0 = gr.sig_source_f (sampling_freq, gr.GR_SIN_WAVE, 350, ampl)
12     src1 = gr.sig_source_f (sampling_freq, gr.GR_SIN_WAVE, 440, ampl)
13     dst = audio.sink (sampling_freq)
14     fg.connect ((src0, 0), (dst, 0))
15     fg.connect ((src1, 0), (dst, 1))
16
17     return fg
18
19 if __name__ == '__main__':
20     fg = build_graph ()
21     fg.start ()
22     raw_input ('Press Enter to quit: ')
23     fg.stop ()
```

Figura 2.5: *Programa gerador de tom de discagem.*

Fonte: [5]

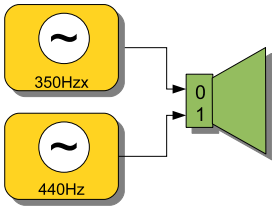


Figura 2.6: *Representação gráfica do gerador de tom de discagem.*

simbolicamente eles são representados pelas arestas no *flowgraph*. Cada *buffer* é implementado como uma FIFO que possibilitam múltiplas portas de leitura e uma única porta de escrita. [16]

O **Escalonador** é responsável por movimentar os dados pelo *flowgraph*,

passando repetidas vezes por cada bloco, ele verifica se há dados suficientes na entrada e espaço suficiente na saída. Se esses requisitos forem satisfeitos o método *work* do bloco é chamado. Para implementar as funções de *start*, *stop* e *wait*, o Escalonador utiliza as funções de controle de *thread* do próprio Python. Apesar de atualmente o Escalonador ser executado como mono *thread*, futuras versões do GNU Radio irão implementar um Escalonador *thread-per-block*, ou seja, cada bloco possui seu executor, que analisa os requisitos e determina quando o método *work* deve ser chamado. [16]

2.2.4 USRP

A USRP (*Universal Software Radio Peripheral*) é uma plataforma aberta, de baixo custo e extremamente flexível desenvolvida sob medida por Matt Ettus [13] para o projeto GNU Radio, o que a fez ganhar rapidamente uma grande comunidade de desenvolvedores em todo o mundo. Sendo basicamente composta por ADCs, DACs, uma FPGA, slots para placas filhas e uma interface de comunicação, a qual conecta o GNU Radio ao mundo RF. O principal objetivo da USRP é permitir aos desenvolvedores a criação de SDRs de baixo custo e com um mínimo esforço inicial. A Figura 2.7 mostra um diagrama de blocos clássico da USRP, retirado de [13], e mostrado em inúmeras publicações da área.

A USRP possui slots para placas "filhas" de recepção e transmissão, as quais são a interface com o mundo físico e onde são implementados os *front end RF*. A função dessa primeira etapa é converter as freqüências da portadora de interesse para uma freqüência intermediária (IF) possibilitando a digitalização da onda pelo ADC na recepção, o caminho inverso é feito para transmissão. Existe uma variedade de placas filhas, que trabalham com diferentes faixas de freqüência e cobrem todo o espectro livre.

Após a conversão da faixa de interesse do espectro eletromagnético, a filosofia da USRP é fazer todo o processamento específico referente a camada física, como modulação e demodulação, no processador do Host, permitindo flexibilidade total para implementação de qualquer protocolo de comunicação. Enquanto todas as operações de propósito geral, que não dependem do protocolo e exigem alta velocidade por se localizarem no começo/final da recepção/transmissão, como por exemplo, o decimador ou interpolador, são implementados na FPGA.

Sendo o principal ponto de configuração da plataforma, a FPGA desem-

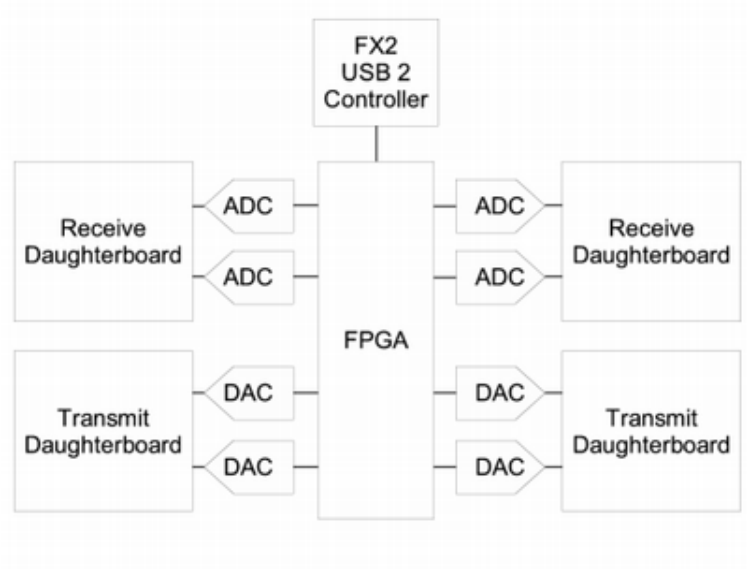


Figura 2.7: *Diagrama de blocos da USRP.*
Fonte: [5]

penha um papel importante no sistema. Basicamente, sua função é implementar os algoritmos matemáticos de alto desempenho e reduzir a taxa de dados para o envio das amostras da onda digitalizada até o host. A principal estrutura dentro da FPGA é o DDC (*Digital Down Converter*) , mostrado em detalhes na Figura 2.8, ele permite a seleção da parte ou partes de interesse do espectro digitalizado, conversão para banda base e decimação. Sendo equivalente ao processo executado pelo *front end RF* (placas filhas), com a diferença do processo ser totalmente digital nessa etapa. Essa função implementada digitalmente permite mais flexibilidade através de uma gama de configurações, como por exemplo, a mudança da freqüência central, possibilitando ao sistema a troca de canal instantaneamente.

Existem duas versões da placa USRP, totalmente compatíveis com as placas filhas, a Tabela 2.2 mostra as principais diferenças. O principal problema da USRP1 é a interface de conexão com o PC, o padrão USB2 tem uma taxa máxima teórica de 60MB/s, limitando na prática transferências a 32MB/s. Como

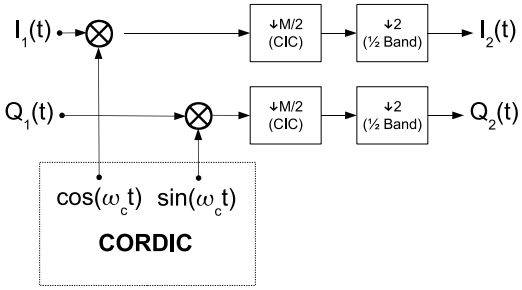


Figura 2.8: *Implementação do DDC na FPGA da USRP*
Fonte: [5]

cada amostra complexa é composta por quatro bytes, 2 bytes do canal I e 2 bytes do canal Q, a USB2 limita a transferência de oito milhões de amostras complexas por segundo (8MS/s), o que permite uma janela de amostragem de 8MHz. Enquanto o ADC pode amostrar uma janela de aproximadamente 32MHz e o DAC pode gerar sinais dentro de uma janela de 60MHz. Isso caracteriza uma grande limitação na flexibilidade dos SDRs implementados com a USRP [5, 13].

Tabela 2.2: *Diferenças das Versões da USRP*

	USRP1	**USRP2**
Interface	USB 2.0	Gigabit Ethernet
FPGA	Altera EP1C12	Xilinx Spartan3 2K
RF Bandwidth to/from host	8MHz @ 16bits	25MHz @ 16bits
Cost	700	1400
ADC Samples	12-bit,64 MS/s	14-bit,100 MS/s
DAC Samples Daughterboard	14-bit, 128 MS/s	16-bit, 400 MS/s
capacity	2 TX, 2 RX	1 TX, 1 RX
SRAM	None	1 Megabyte
Power	6V, 3A	6V, 3A

Fonte: [5, 13]

As melhorias com relação a interface na USRP2 foram alcançadas substituindo a interface USB por uma interface Gigabit Ethernet, com uma capacidade de transferência de 125MB/s, permitindo uma janela de amostragem de 25MHz. Outras melhorias foram o aumento das taxas de amostragens do ADC e do DAC

e uma FPGA mais rápida e maior. Por outro lado, a capacidade de placas filhas funcionando simultaneamente caiu pela metade e o preço aumentou duas vezes.

Devido ao aumento da FPGA na USRP2, mais funções podem ser implementadas no hardware, o que combina a flexibilidade da reconfiguração e a performance requerida por algumas aplicações. Além disso, outra mudança importante ocorreu na lógica de controle, a USRP1 espalha essa função entre a FPGA e o controlador da USB, como por exemplo, o ajuste das placas filhas não pode ser efetuado pela lógica interna da FPGA, o controlador USB usa o barramento I2C (*Inter-Integrated Circuit*) , o qual não é conectado na FPGA. Por outro lado, a USRP2 centraliza todas as funções de controle no processador *softcore* aeMB sintetizado dentro da FPGA, com exceção da própria configuração, que é feita por um CPLD externo.

A segunda versão não torna a primeira completamente obsoleta para aplicações com o GNU Radio. Entretanto, a USRP2 é mais flexível, poderosa e pode ser utilizada para pesquisas mais profundas no futuro.

CAPÍTULO 3

Arquitetura para Múltiplas Camadas Físicas

Este capítulo apresenta as principais características e fundamentos da arquitetura para múltiplas camadas físicas desenvolvida neste trabalho.

3.1 Fatores Dinâmicos e Estáticos em Redes sem Fio

Fatores dinâmicos e estáticos influenciam no projeto e funcionamento da camada física de uma rede sem fio. Em linhas gerais os fatores dinâmicos estão relacionados com diferentes requisitos das aplicações, ambiente RF e a taxa de consumo de recursos do sistema. Por outro lado, os fatores estáticos têm relação com os limites do hardware e com a regulamentação do país para esta classe de dispositivos de comunicação. A vantagem dos sistemas baseados na tecnologia SDR é que esses fatores podem servir como agentes modificadores, ou seja, dependendo da relação entre eles o sistema pode adaptar ou modificar a sua camada física.

Apesar de não esgotar todos os fatores estáticos e dinâmicos pertencentes ao domínio de camadas físicas de redes sem fio, a Tabela 3.1 apresenta alguns desses fatores classificados em quatro classes [9]. A primeira classe, chamada *Requisitos da Aplicação*, especifica a camada física ou o comportamento momentâneo dela, isso porque com a flexibilidade na camada mais inferior do sistema pode-se operar modificações baseadas nas necessidades variáveis das aplicações. Em outras palavras, quando a taxa de dados requisitada pela aplicação for menor, o sistema pode se adaptar para usar uma camada física mais simples e economizar energia.

A classe *Ambiente RF* agrupa os principais fatores relacionados com o meio físico de redes sem fio, o qual apresenta uma natureza altamente dinâmica. A observação desses fatores permite explorar características relacionadas com as condições do canal para adaptar a camada física e aumentar o desempenho da rede. Por exemplo, em canais com maior interferência há necessidade da utilização de algoritmos mais robustos de correção de erros (FEC - *Forward Error*

Tabela 3.1: *Fatores que afetam a determinação da camada física de redes sem fio*

Tipo	Classe	Fatores
Dinâmicos	Requisitos da Aplicação	Taxa de dados
		Latência
		Taxa de erros (FER)
		Confirmação
		Distância
		Nível de Segurança
	Ambiente RF	Estado do Canal
		Congestionamento
		Topologia
		Infra-estrutura Disponível
		Relação sinal ruído (SNR)
		Bit Error (BER)
		Interferência
	Recursos	Espectro alocado
		Quantidade de Bateria
		Ocupação da CPU
Estáticos	Limitações do sistema	Sensibilidade do Rádio
		Poder de Processamento
		Faixa de sintonia
		Máxima potência de transmissão

Fonte: [9]

Correction) , diminuindo a banda efetiva, mas garantindo a conectividade.

A terceira classe lista os fatores dinâmicos referentes às quantidades de *Recursos* alocados ao sistema ou que podem ser consumidos. Essas quantidades são variáveis durante a execução do sistema e influenciam a camada física, que deve se adaptar para não exceder os limites. Por exemplo, em uma mesma faixa de espectro de freqüência alocado ao sistema para comunicação pode-se transferir dados a taxas maiores utilizando modulações mais elaboradas, as quais podem consumir mais recursos de energia e processamento. Entretanto, se existe uma faixa maior do espectro livre, a modulação pode ser simplificada mantendo a mesma taxa de dados e comprometendo menos os recursos do hardware.

A última classe, denominada *Limitações do sistema*, agrupa os fatores estáticos relacionados com os limites do hardware e regulamentação imposta por agências reguladoras. Não se pode implementar uma camada física que requisite uma potência acima do permitido ou do alcançado pelo hardware disponível, ou

ainda, que exija uma capacidade de processamento maior do que o disponibilizado pelo sistema. Essa classe, bem como todos os outros fatores estáticos, são limites impostos durante a fase de projeto e devem ser alinhados com os requisitos do sistema para que não sejam ultrapassados.

Na próxima seção é apresentada a arquitetura proposta, a qual trabalha diretamente com os fatores estáticos e os fatores dinâmicos da classe dos *Recursos*. Isto acontece uma vez que o controle proposto para alocação de canais faz a análise das limitações do sistema e do espectro já alocado por outras camadas físicas.

3.2 Arquitetura Proposta

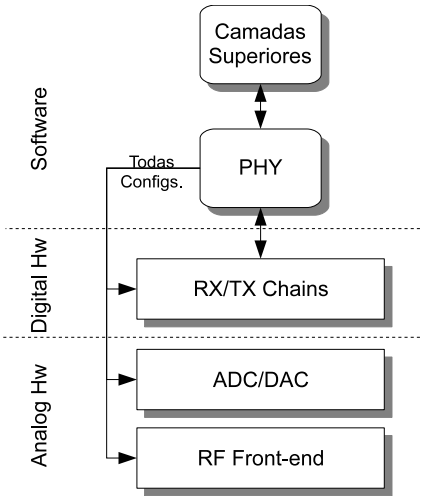
Nas arquiteturas tradicionais de SDR é comum que a relação entre a interface (hardware) e a camada física (PHY em software) seja de um para um. Isto mais por uma questão da herança de implementação dos rádios tradicionais do que por necessidade, uma vez que não existia separação entre camada física e interface. Por exemplo, a USRP exporta somente um canal por interface, o qual deve ser configurado para possuir as características esperadas pela implementação da camada física, o que inutiliza o resto do espectro capturado pela interface para outras camadas físicas.

A Figura 3.1 apresenta um modelo de implementação de uma camada física utilizando a arquitetura tradicional da USRP. Os métodos providos pela USRP exigem que o PHY configure todos os estágios da interface, composta pelo RX/TX *Chains*, ADC/DAC e *RF Front-end*. Além de configurações complexas para selecionar um canal com freqüência central e largura bem definidas, o resto do espectro amostrado pelo ADC é desperdiçado. Isso porque não há mecanismos para outro PHY compartilhar a interface.

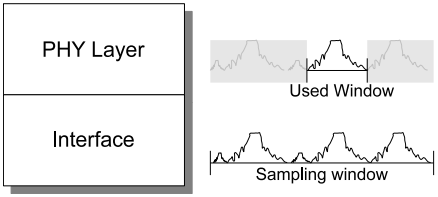
Uma alternativa para resolver o problema da má utilização do espectro capturado é fazer em software a separação de cada canal. Isso é possível configurando a plataforma de hardware para envio de todo o espectro de interesse e adicionando uma camada de software responsável por "fatiar" a janela amostrada para cada PHY do sistema [10]. Entretanto, essa abordagem é altamente onerosa para o processamento do *host* e deve ser feito sob medida para cada conjunto de camadas físicas. Além disso, como os parâmetros de configuração do hardware

são inferidos a partir das características dos canais, cada mudança exige um esforço grande de reconfiguração, que pode variar com o tipo de hardware ou com o novo conjunto de canais alocados.

A Figura 3.2 exemplifica uma implementação seguindo tal filosofia [10]. Para cada PHY é adicionado um conjunto composto de blocos de "Translação de Freqüência" e decimação/interpolação (DDC/DUC) que recebem a janela amostrada e retiram somente o canal de interesse. Além desses blocos implementa-



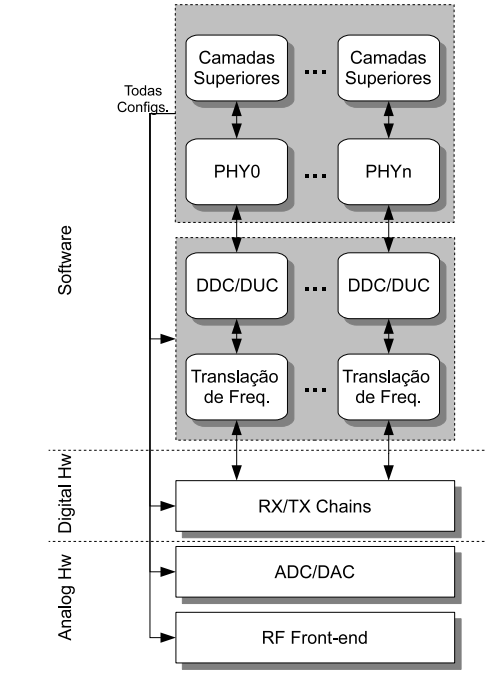
(a) Diagrama de Blocos.



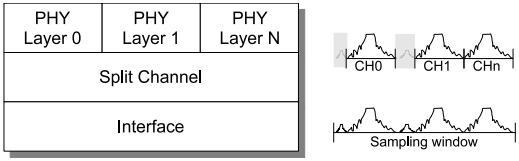
(b) Desperdício do espectro capturado pela interface física.

Figura 3.1: *Implementação tradicional das camadas físicas*

rem algoritmos complexos para separação de canais, a janela amostrada completa representa uma grande quantidade de dados, que deve ser processada em paralelo para cada PHY/canal da implementação. O que limita consideravelmente o número de canais por falta de capacidade do *host*.



(a) Diagrama de Blocos.



(b) Múltiplas camadas utilizando uma única interface física.

Figura 3.2: *Abordagem em software altamente onerosa para separação de canais.*

Com uma análise mais cuidadosa percebe-se que todas as camadas físicas já utilizam o conceito de canal, o qual possui apenas dois parâmetros: freqüência central e largura de banda. Padrões de comunicação sem fio (ex.: 802.11, 802.15.4) normalmente fixam a largura do canal e utilizam identificadores numéricos para referenciá-los, os quais podem ser facilmente traduzidos utilizando uma tabela de padronização. Portanto, o desacoplamento do canal da implementação da camada física é simples e adiciona flexibilidade no desenvolvimento de aplicações com múltiplas camadas físicas.

Baseado nisso, esse trabalho propõe uma arquitetura com uma interface simples e única para alocação de canais independentes, os quais podem ser reconfigurados a qualquer momento no processo de comunicação, respeitando os, aqui classificados, fatores dinâmicos e estáticos. Uma visão geral da arquitetura é apresentada na Figura 3.3, onde se destacam a adição dos blocos de controle e GoC (*Grupo de Canais*) . Com essa configuração, a arquitetura proposta permite que a dependência da camada física (*PHY*) passe a ser um canal ou um grupo de canais e não mais a interface física completa. Assim, proporcionando um melhor aproveitamento da janela de amostragem com a extração de vários canais para camadas físicas que estão funcionando simultaneamente, sem onerar o *host*, uma vez que todo o processamento é executado no hardware.

A Figura 3.3 mostra o bloco de controle funcionando como interface simplificada de configuração para a implementação de múltiplas camadas físicas (*PHY0 ... PHYn*). Cada PHY precisa apenas solicitar a alocação de seu canal ou canais de interesse (freqüência central e largura de banda) e o bloco de controle, baseado nos fatores estáticos que limitam a atuação do hardware, infere as configurações do hardware analógico, dos *chains* e do GoC. Por sua vez, o GoC é formado pelas estruturas de cada canal, que é detalhada na Figura 3.4. Essa estrutura implementa o seletor de canal, que recebe as configurações (*config.*) do bloco de controle, e o conjunto de registradores, que armazenam as medidas realizadas na fatia do espectro selecionada e podem ser lidos pelo bloco de controle.

Nas próximas subseções serão abordados em mais detalhes os blocos *Estrutura do canal* e *Controle*.

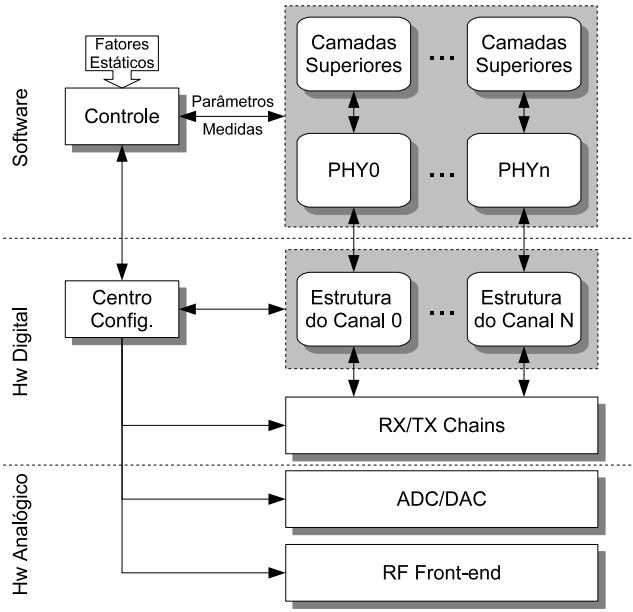


Figura 3.3: *Arquitetura de múltiplos canais.*

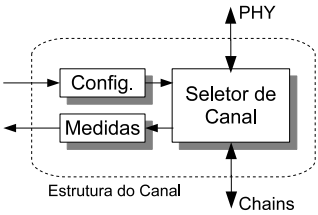


Figura 3.4: *Componentes da estrutura do canal.*

3.2.1 Estrutura do Canal

O bloco de *Estrutura do Canal* é formado pelo *Seletor de Canal*, que recebe as configurações (*config.*) do bloco de controle, e o conjunto de registradores

que armazenam as medidas executadas diretamente no hardware. A proposta da arquitetura é prover o suporte a leitura, por intermédio do bloco de *Controle*, dessas medidas para análise dos fatores dinâmicos da classe *Ambiente RF* na fatia do espectro selecionada pelo *Seletor de Canal*, estando fora do escopo do trabalho a implementação dos blocos que fazem tais medidas.

O suporte a esse tipo de medida foi adicionado à arquitetura pela importância que esse tipo de informação possui em um SDR, uma vez que a flexibilidade da camada física pode permitir a adaptação e otimização da performance do sistema [8, 21]. Sendo necessárias, para isso, medições desses fatores dinâmicos para nortear as decisões e verificar o quanto um ajuste alterou o cenário de comunicação. Essas otimizações podem ser automáticas, feitas sob medida pela própria aplicação ou por níveis superiores dos protocolos (ex.: Sub-camada MAC) de forma que as métricas alimentem funções de custo que devem ser maximizadas ou minimizadas para a operação ideal do rádio. Em um caso ideal, podemos encontrar uma função que converge a um objetivo único, cuja maximização ou minimização corresponde à melhor configuração. No entanto, conforme discutido em [15], os sistemas de comunicação têm requisitos complexos que não podem ser agrupados em função de um objetivo único, especialmente se os requisitos da aplicação ou o meio de comunicação são dinâmicos.

O rápido crescimento da complexidade para otimizar diversas métricas é notável. Cada métrica tem relações únicas com outras métricas, entretanto ajustes diferentes afetam as métricas de diferentes formas. Por exemplo, o aumento da taxa de dados com a manutenção da taxa de sinalização melhora a eficiência espectral, porém causa aumento na complexidade computacional e no consumo de energia. A Figura 3.5 apresenta algumas dessas relações, onde a direção da seta indica que a otimização do parâmetro da fonte afeta o parâmetro do alvo. Atualmente, vários trabalhos tentam definir essas relações por completo e criar as funções de otimização para serem utilizadas em sistemas automáticos de decisão e aprendizagem [15, 9, 6].

A proposta da arquitetura aqui apresentada é permitir a leitura de medidas relacionadas com o canal sem depender da implementação de uma camada física (PHY), o que possibilitaria a seleção de uma implementação que melhor se adapte à situação do canal. Possíveis medições que interessariam nesse sentido são apresentadas na Tabela 3.2. As métricas mais óbvias são a relação sinal-ruído (SNR - *Signal-to-Noise Ratio*) e RSSI (*Received Signal Strength Indication*) .

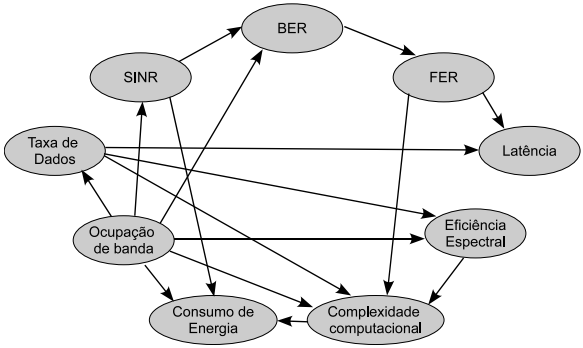


Figura 3.5: *Grafo indicativo das relações de dependência entre dois parâmetros. Fonte: [15]*

Tabela 3.2: *Métricas relacionadas com a camada física.*

Camadas	Métrica
PHY	SNR
	SINR
	RSSI
	Path loss
	Fading statistics
	Doppler spread
	Delay spread
	Multipath profile
	Noise power
	Interference power
	Peak-to-average power ratio
	Error vector magnitude
	Spectral efficiency

Fonte: [15]

Entretanto, é importante salientar que a implementação dos blocos de processamento digital de sinal que provêem essas medidas estão fora do escopo do presente trabalho.

3.2.2 Bloco de Controle

Os parâmetros de controle de cada canal são a freqüência central e a largura de banda. Entretanto, como discutido na Seção 3.1, existem fatores estáticos e dinâmicos que limitam esses parâmetros e devem ser levados em consideração no projeto. A limitação dos recursos do sistema aparece como fator estático e a alocação de canais concorrentes que consomem esses recursos é o fator dinâmico.

Os recursos do sistema que limitam a configuração da freqüência central e largura de banda dos canais são:

Capacidade do *RF Front-End*: A função do RF Front-End é deslocar uma janela de interesse para uma freqüência intermediária (IF), onde o sinal é amostrado por ADCs e o inverso para a transmissão. Cada RF Front-End possui uma faixa de freqüência de trabalho com limites máximo e mínimo.

Janela de amostragem: Após o deslocamento para IF uma faixa do espectro é amostrado pelo ADC, esse por sua vez tem limitações ditadas pelo teorema de *Nyquist-Shannon*, onde a freqüência de amostragem tem que ser o dobro da freqüência de interesse. Ou seja, para uma janela de 100MHz o ADC deve trabalhar amostrando a 200MHz.

Interface de comunicação: O hardware transmite os canais selecionados para uma unidade de processamento onde será executado o software que define o comportamento da camada física. Se o processador estiver ligado via interfaces de alta velocidade, esse limite será maior que a freqüência de trabalho do ADC, tendo pouca influência no sistema. Entretanto, não é raro a utilização de interfaces USB ou Giga Ethernet, limitando a transferência de espectros com janelas de 8MHz ou 25MHz, respectivamente.

As equações (3.1), (3.2), (3.3) e (3.4), apresentam a relação entre a freqüência central (f_{cx}) e a largura de banda (L_{cx}) de um canal qualquer com os limites impostos pela capacidade do *RF Front-End* (f_{FEmin} e f_{FEmax}), pela largura da janela de amostragem do ADC (L_{max}) e pela capacidade da interface de comunicação (LI_{max}). Para simplificar, o formato de representação das amostras é abstraído e todos os parâmetros são estipulados em *Hertz* (Hz). A Tabela 3.3 resume os parâmetros utilizados nas equações.

$$f_{FEmin} \leq f_{cx} - \frac{L_{cx}}{2} \tag{3.1}$$

$$f_{cx} + \frac{L_{cx}}{2} \leq f_{FEmax} \tag{3.2}$$

$$L_{cx} \leq L_{max} \tag{3.3}$$

$$L_{cx} \leq LI_{max} \tag{3.4}$$

Tabela 3.3: *Características do canal e limitantes do recurso do sistema.*

Parâmetro	Variável
Freqüência central de um canal x	f_{cx}
Largura de banda de um canal x	L_{cx}
Limites mínimo e máximo das freqüências de trabalho do *RF Front-End*	f_{FEmin} e f_{FEmax}
Largura máxima da janela de amostragem do ADC	L_{max}
Largura máxima da janela transmitida pela interface de comunicação	LI_{max}

Além dos fatores estáticos existem a concorrência dos recursos pelos canais, ou seja, dinamicamente os canais concorrem por fatias do espectro e a soma dos recursos alocados não podem ultrapassar os limites estabelecidos do sistema. Assim a alocação de todos os canais gera uma janela que possui uma freqüência inferior B_i e um freqüência superior (B_s) que tem relação com os canais de menor (f_{cmenor} e L_{cmenor}) e maior freqüência (f_{cmaior} e L_{cmaior}), respectivamente. As equações (3.5) e (3.6) demonstram essa relação.

$$B_i = f_{cmenor} - \frac{L_{cmenor}}{2} \tag{3.5}$$

$$B_s = f_{cmaior} + \frac{L_{cmaior}}{2} \tag{3.6}$$

Após calcular as freqüências inferior e superior da janela formada pela alocação dos canais os limites impostos pela capacidade do *RF Front-End* (f_{FEmin}

e f_{FEmax}) e pela largura da janela de amostragem do ADC (L_{max}) devem ser verificados. As equações (3.7), (3.8) e (3.9) apresentam essa relação.

$$B_i \geq f_{FEmin} \tag{3.7}$$

$$B_s \leq f_{FEmax} \tag{3.8}$$

$$B_s - B_i \leq L_{max} \tag{3.9}$$

Finalmente, após amostrados, cada canal gera uma quantidade de dados proporcional a sua largura de banda L_c. Assim, a somatória da largura de banda de todos os canais alocados deve ser menor ou igual a capacidade de transmissão da interface de comunicação (equação (3.10)).

$$\sum_{i=0}^{n} L_{ci} \leq LI_{max} \tag{3.10}$$

O Algoritmo 1 apresenta o pseudocódigo para uma das possíveis implementações da verificação dos recursos do sistema e alocação de uma lista de canais seguindo as equações descritas acima.

Algoritmo 1 Verificação dos recursos do sistema para alocação de uma lista de canais.

```
l_ca = 0
for i := 1 to n_channels do
  if f_FEmin ≤ f_c[i] ≤ f_FEmax then
    l_ca = l_ca + l_c[i]
    v[ ].insere_ordenado(f_c − L_c/2)
    v[ ].insere_ordenado(f_c + L_c/2)
    if (v[ ].max() − v[ ].min() > L_max) or (l_ca > LI_max) then
      error("Impossivel alocar todos os canais")
    end if
  end if
end for
```

3.3 Resumo da Proposta

A arquitetura proposta emprega o conceito de múltiplos canais entre a interface física (hardware) e as camadas físicas (PHYs) implementadas em software. A mudança de paradigma de "manipulação do espectro" para "múltiplos canais" simplifica a interação com o hardware e permite que a dependência da camada física (*PHY*) passe a ser um canal ou um grupo de canais e não mais o hardware como um todo, permitindo o compartilhamento da mesma janela amostrada pelo ADC por várias camadas físicas. Essa simplificação tornou transparente várias configurações do hardware, uma vez que a partir dos canais solicitados (freqüências centrais e larguras de banda) ao bloco de controle é possível inferir todos os parâmetros de configuração. Além disso, a proposta não diminui em nada a flexibilidade no projeto ou em tempo de execução se comparada com a arquitetura tradicional.

Outro benefício proporcionado pelo conceito de canal é a possibilidade da adição de uma estrutura de separação de canais no hardware (GoC), o que diminui drasticamente a ocupação do processador de uso geral do sistema (*host*), uma vez que o paralelismo intrínseco existente na separação de diversos canais, que exige o processamento concomitante do mesmo grupo de dados, e a alta quantidade de amostras por segundo, proveniente das fases iniciais do rádio, são características onerosas para implementações em software e são beneficiados pela implementação paralela do hardware.

No próximo capítulo serão abordados os aspectos de implementação da arquitetura e a comparação da arquitetura tradicional e proposta em dois cenários de aplicação.

CAPÍTULO 4

Implementação da Arquitetura Proposta

Este capítulo descreve a implementação da arquitetura para múltiplas camadas físicas utilizando a USRP2 e o GNU Radio.

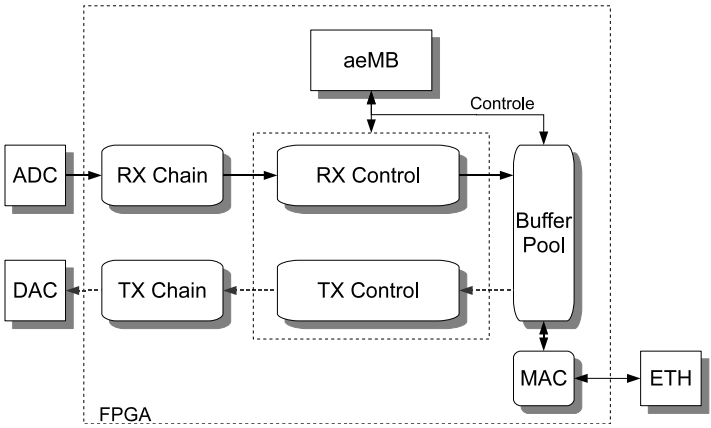
4.1 Modificações USRP2

Como apresentado na Seção 2.2.4 existem duas versões da placa USRP. Para as modificações e os experimentos realizados neste trabalho foi escolhida a segunda versão da placa, devido a maior resolução do ADC e do DAC, 14-bits 100 MS/s (complexo) e 16-bits 400 MS/s respectivamente, possibilitando uma lagura da janela de amostragem de até 100MHz. A interface Gigabit Ethernet permite que o *host* processe instantaneamente uma janela de até 25MHz. Além disso, a maior capacidade da FPGA garante o espaço necessário para implementação da arquitetura proposta.

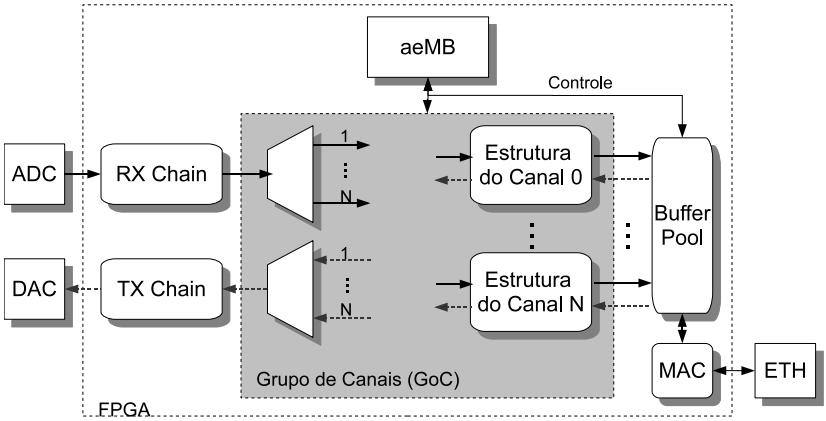
Foram realizadas duas classes de modificação na FPGA da USRP2 original: Modificações no hardware reconfigurável e na lógica de controle. As modificações de hardware incluem a adição de um bloco GoC (Grupo de Canais) e a ampliação do sistema de *buffers* (*Buffer Pool*) das cadeias de recepção e transmissão. Já a lógica de controle, realizada pelo processador *softcore* aeMB, sofreu modificações na execução do roteamento das amostras dentro do sistema para suportar mais canais. As Figuras 4.1(a) e 4.1(b) apresentam, respectivamente, a arquitetura original e a implementação da arquitetura proposta.

A adição do bloco GoC permite que a janela do espectro de freqüência capturada pelo *RX chain* ou modificada pelo *TX chain* sejam "fatiados" em canais com freqüências centrais (FC) e larguras distintas. Cada um desses parâmetros pode ser configurado pelo *host* mantendo a flexibilidade necessária para essa etapa do processamento executada pelo bloco *Seletor de Canal* dentro da *Estrutura do Canal*, que é detalhado na Figura 4.2.

Todos os canais compartilham os mesmos *RX* e *TX chains*, sendo necessária a adição de um demultiplexador e um multiplexador para fazer a interface entre os canais e os *chains*. Para o *RX chain* o demultiplexador é bastante simples e sua única função é o roteamento das amostras para a entrada de cada canal, não



(a) Fluxo original das amostras na USRP2



(b) Implementação da arquitetura para múltiplas camadas físicas.

Figura 4.1: *Arquiteturas original e proposta.*

havendo necessidade de processamento das amostras. Por sua vez, o multiplexador para o *TX Chain* é um somador que pega as várias fatias processadas pelos canais e monta uma janela completa antes do envio ao conversor digital analógico.

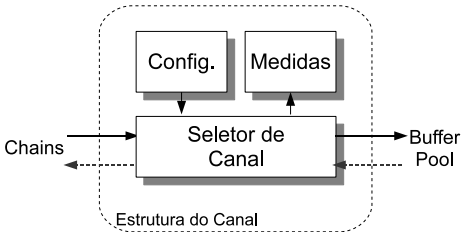


Figura 4.2: *Estrutura interna de cada canal.*

Para apoio ao funcionamento do GoC, outro bloco que sofreu modificações foi o *Buffer Pool*, que consiste em um conjunto de *buffers* circulares que podem ser acessados através de uma interface FIFO. Em sua configuração original o *Buffer Pool* possui 4 portas de leitura e 4 portas de escrita, que podem ser associadas pelo microcontrolador embarcado *softcore aeMB* a qualquer um dos 8 buffers (2 KB cada um). Originalmente as portas de leituras estão alocadas ao canal de expansão serial (SERDES - Serializer/Deserializer) , *TX control* e Ethernet. Enquanto as portas de escrita estão alocadas ao SERDES, *RX Control* e Ethernet. Os dados de todos os *buffers* podem ser acessados via barramento *Wishbone* pelo microcontrolador *aeMB*, os quais originalmente são distribuídos igualmente entre a Ethernet, RX, TX e saída de expansão serial (SERDES).

As modificações do *Buffer Pool* incluem a parametrização do número de portas e quantidade de *buffers* a partir do número de canais. Cada canal exige uma porta de leitura e escrita e um par de *buffers*, os quais serão gerenciados pelo *aeMB*. Junto com a lógica consumida por cada canal, a quantidade de memória é limitada somente pelos recursos da FPGA em uso. Ou seja, a adição de mais canais exige que os requisitos de lógica e memória sejam atendidos concomitantemente.

A outra classe de modificações exige uma alteração na lógica de controle implementada no *firmware* do aeMB para gerenciar múltiplos canais. Para isso é necessário mudar a interpretação do *frame* padrão de comunicação entre a USRP2 e o *host*, que se localiza no *Data Payload* do *frame* ethernet, ambos mostrados na Figura 4.3. O *frame* padrão é composto pelos cabeçalhos *Transport Header* e *Fixed Header*, utilizados para controle da comunicação. O cabeçalho *Transport Header* é utilizado para fazer o controle de fluxo e detecção dos pacotes perdi-

dos. Enquanto o cabeçalho *Fixed Header* serve para roteamento das amostras e controles realizados internamente pelo *aeMB*. Abaixo a descrição original dos campos disponíveis:

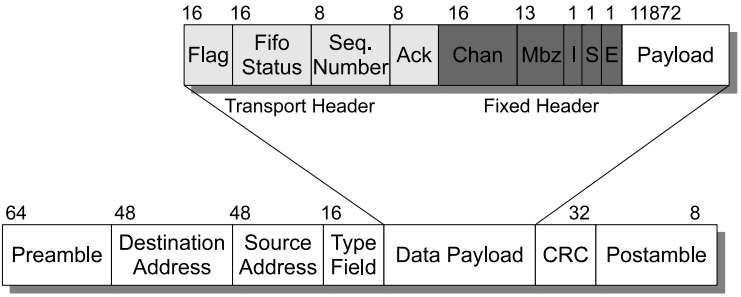


Figura 4.3: *Formato do frame de comunicação entre a USRP2 e o* host.

- **Flags:** deve ser zero, mas futuramente pode ser utilizado por um canal;
- **Fifo status:** espaço livre no *buffer* de recepção;
- **Seq. Number:** número de seqüência deste pacote;
- **Ack:** número de seqüência do próximo pacote esperado.
- **Chan:** este campo configura o tipo do *frame*, que pode ser controle (valor 0xF1) ou amostras (valor 0), outros valores são suportados para expansões futuras.
- **Mbz:** deve ser zero;
- **I:** se ligado esse bit informa que o *payload* do *frame* é de amostras;
- **S:** se ligado esse bit inicia o modo *burst* de transmissão;
- **E:** se ligado esse bit finaliza o modo *burst* de transmissão;
- **Timestamp:** basicamente especifica quando os dados foram recebidos ou quando devem ser transmitidos.
- **Payload:** dados de controle ou amostras.

Como descrito acima, no cabeçalho *Fixed Header* já existe o campo *Chan* que indica se os dados são referentes a controle ($Chan = 0xF1$) ou amostras ($Chan \neq 0xF1$). A modificação no código do *aeMB* estendeu a semântica desse campo para reconhecer/indicar de qual canal as amostras pertencem. Ou seja, quando os pacotes vindos do *host* são adicionados no *Buffer Pool* pelo *Ethernet MAC* é gerado uma interrupção no *aeMB*, que além da verificação do tipo de pacote (controle ou amostras) ele também verifica para qual canal as amostras devem ser roteadas. No fluxo oposto, as amostras vindas de um canal específico são adicionados no *Buffer Pool*, onde o *aeMB* indica o número do canal correspondente no campo *Chan* durante a montagem do pacote.

A adição de vários canais gera uma disputa pela interface de comunicação da USRP2 com o *host* (ethernet). Considerando que todos os canais funcionem a mesma taxa de amostras por segundo, a solução mais simples é uma divisão da banda da interface igualmente entre todos os canais. Entretanto essa solução limita o funcionamento de canais com larguras distintas, o que impossibilitaria o funcionamento de camadas físicas com largura de canais diferente ou exigiria processamento adicional para adequação do menor canal. Assim, a solução definida foi um escalonamento entre os canais que leva em consideração a largura de cada canal. Ou seja, o canal que precisa de maior banda envia mais *frames* de amostras. A verificação da alocação de banda acontece na alocação dos canais como descrito na seção 3.2.

Além disso, as modificações no código do *aeMB* ainda contaram com a extensão da interpretação dos pacotes de controle que são utilizados para escrever e ler nos registradores que armazenam as configurações e as medidas relacionadas com cada canal.

4.2 Modificações GNU Radio

A comunicação entre o GNU Radio e USRP2 se dá por meio de uma interface Gigabit Ethernet. Para receber e enviar frames Ethernet, o GNU Radio utiliza *raw socket* [1] da API de *sockets*, o qual permite que os pacotes de rede sejam tratados por aplicações em espaço de usuário sem a interferência do *kernel*. Na Figura 4.4 é apresentada uma comparação ilustrativa das camadas de redes

[1] Modalidade de *socket* que não possui o encapsulamento dos protocolos de rede.

tradicionais (esq.) e do acesso direto feito pela modalidade de *raw socket* (dir.).

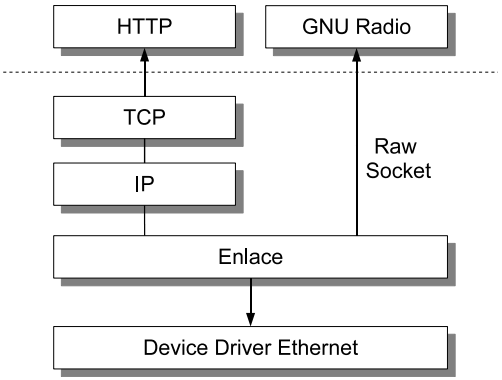


Figura 4.4: *Comunicação do GNU Radio com a USRP2.*

Normalmente quando utilizamos a API de *sockets* enviamos mensagens TCP/IP, porém a USRP2 envia frames Ethernet sem os cabeçalhos IP. Em condições normais o kernel ao receber este tipo de frame não consegue tratá-lo e acaba descartando. Neste caso, o GNU Radio cria um *raw socket* e configura um filtro para que os frames com endereço MAC da USRP2 sejam acessados diretamente na camada de enlace pela aplicação.

Para realizar a recepção dos frames o GNU Radio utiliza o recurso PACKET_MMAP. Este recurso permite mapear em user space os buffers de recepção dos frames, assim o acesso é direto sem a necessidade de chamadas de sistema, minimizando as cópias de pacotes. O GNU Radio realiza a leitura dos buffers compartilhados por meio de uma thread, ao receber o frame a thread o identifica e dependendo do tipo, controle ou amostras, chama o tratador correto.

Para abstrair esses detalhes de implementação o GNU Radio tem um bloco *Source* e um bloco *Sink* para tratar as amostras vindas ou enviadas para USRP2. Dessa forma, boa parte das modificações realizadas para alocação, controle e leitura dos canais foram concentradas nesses blocos. Além disso, foi criado o bloco *channels* (Figura 4.5) que exporta as interfaces de cada canal em pares de RX e TX, e ainda, todos os métodos de controle. Esse bloco também é responsável por fazer o gerenciamento dos recursos disponíveis do espectro, apresentados na Seção 3.2. Com isso, os PHYs possuem uma interface única de comunicação e

configuração com os canais, não dependendo mais do hardware que implementa a estrutura física.

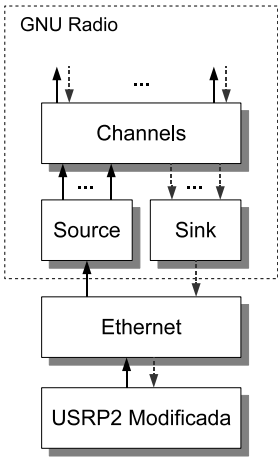


Figura 4.5: *Diagrama dos blocos projetados para o GNU Radio.*

4.3 Resumo das Modificações

Para a implementação da arquitetura proposta foram necessárias modificações na FPGA, tanto no HDL (*Hardware Description Language*) quanto no *firmware* do processador *soft-core aeMB*. No HDL foi adicionado o bloco GoC e ampliado o *Buffer Pool*, que são parametrizáveis pelo número de canais no momento da sintetização do hardware reconfigurável. No *firmware* do *aeMB* foi implementado suporte a múltiplos canais, escalonamento do envio de amostras à múltiplas taxas e expansão na escrita e leitura dos registradores de cada canal.

Também foram necessárias modificações no GNU Radio, onde foram adaptados os blocos *sink* e *source* para suporte a múltiplos canais e implementado o bloco *channels*. Os dois primeiros fazem a interface com a USRP2 e possuem configurações detalhadas das estruturas de hardware. Enquanto o bloco *channels* implementa a interface simplificada dos canais, que inclui o bloco controle apresentado na Seção 3.2.

No próximo capítulo essas modificações são utilizadas nos cenários de aplicação para comparação entre implementações com a arquitetura original e a proposta neste trabalho.

CAPÍTULO 5

Avaliação da Arquitetura Proposta

Este capítulo apresenta a implementação de dois cenários de aplicação utilizando a arquitetura original e a arquitetura proposta neste trabalho, bem como a avaliação comparativa dos testes realizados.

5.1 Cenários de aplicação

A avaliação proposta foi dividida em dois cenários de aplicação: múltiplos canais simétricos e não simétricos. Para cada classe de testes foram utilizadas uma implementação contendo os componentes originais do GNU Radio e uma implementação equivalente utilizando a arquitetura proposta. Para análise e comparação de desempenho das implementações foi utilizado um ambiente de teste, composto por um PC com processador Intel QuadCore de 2.83 GHz, com 4GB de RAM e rodando Ubuntu 9.10 com o kernel 2.6.31-19. A placa de rede gigabit utilizada para interface com a USRP2 foi a Broadcom BCM5755 integrada. Nas próximas subseções cada cenário será apresentado com detalhes.

5.1.1 Múltiplos Canais Simétricos

Múltiplos canais simétricos é um cenário onde todos os canais possuem a mesma largura e conseqüentemente a mesma quantidade de dados capturados do meio, sendo bastante comum quando há necessidade da demodulação em paralelo de vários canais. O exemplo mais comum são os analisadores de rede sem fio (*Sniffers*), onde é preciso analisar vários canais ao mesmo tempo a procura de informações de interesse. Outro exemplo, são as *bridges* entre redes sem fio de mesma tecnologia, os quais organizam a rede sem fio utilizando múltiplos canais em paralelo.

Para esse cenário foi utilizado a implementação da camada física que demodula o padrão IEEE 802.15.4 da UCLA [26, 10] seguindo os componentes tradicionais do GNU Radio e uma implementação equivalente utilizando a arquitetura de canais proposta. O receptor multicanal demodula paralelamente quatro canais que, por limitações da arquitetura original, devem ser vizinhos.

O ambiente de experimentações montado é composto por 4 motes [1] MicaZ, uma USRP2 e um *host*, como mostrado na Figura 5.1. Os motes MicaZ utilizam o rádio CC2420 com a implementação do padrão 802.15.4. Eles enviam mensagens contendo apenas seu identificador, os quais servem apenas para *debug*, uma vez que o intervalo e o sincronismo não fazem diferença, pois a ocupação do canal não influência no desempenho dos testes. Cada implementação foi executada com prioridade "tempo real" e testada durante 300 segundos com dados de performance sendo lidos em intervalos de 2 segundos.

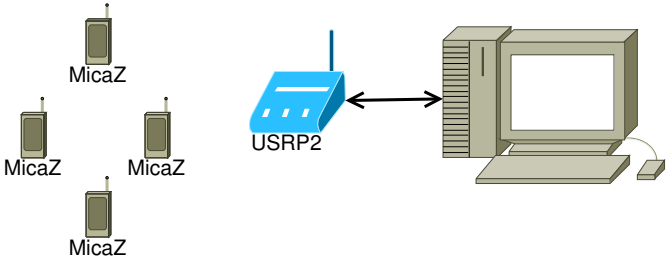
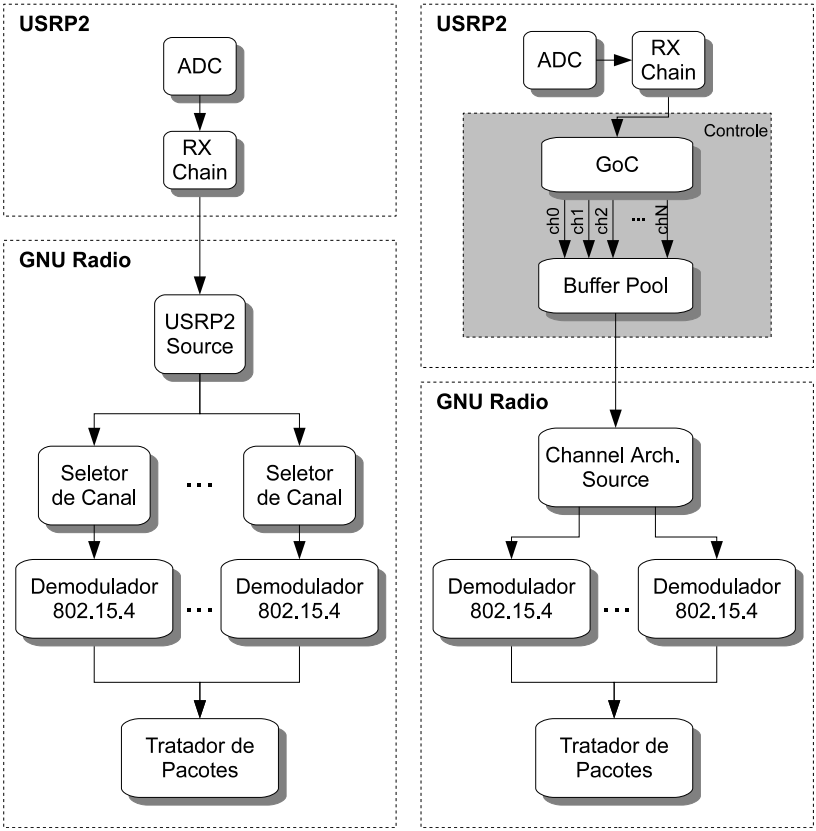


Figura 5.1: *Ambiente de testes para Canais Simétricos.*

O primeiro teste foi executado utilizando a implementação tradicional, que possui a estrutura mostrada na Figura 5.2(a). A predominância dos blocos em software e o paralelismo das funções como "Translação e DDC", dentro do bloco *Channel Select*, a altas taxas de amostras por segundo são conseqüência da falta de suporte pelo hardware para múltiplos canais. Sendo necessário o uso da abordagem de software apresentada na Seção 3.2.

A Figura 5.2(b) mostra a estrutura do segundo teste, desenvolvida utilizando a arquitetura de canais proposta. O deslocamento do processamento dos canais para o hardware reconfigurável permite que a janela do espectro de freqüência capturada pelo *RX chain* seja "fatiada" em canais com freqüências centrais (FC) e larguras específicas. Cada um desses parâmetros pode ser configurado pelo *host*, não diminuindo em nada a flexibilidade alcançada pelos blocos de software utilizados no primeiro teste e permitindo o mesmo nível de configuração em tempo de execução. Utilizando essa estrutura, cada camada física pode escolher livremente as freqüência de trabalho para um ou vários canais.

[1] Mote é o nome dado ao nó de uma rede de sensores sem fio.



(a) Arquitetura Tradicional (b) Arquitetura Proposta.

Figura 5.2: *Implementação Canais Simétricos.*

5.1.2 Múltiplos Canais Assimétricos

O cenário de *Múltiplos Canais Assimétricos* é o mais comum nos sistemas modernos, onde tecnologias sem fio heterogêneas são agrupadas nos equipamentos mais diversos. Esses diferentes padrões possuem suas características únicas e

se baseiam em canais de larguras variadas, o que gera fluxo de dados (amostras por segundo) desiguais para cada camada física, sendo necessário o tratamento de forma transparente pela arquitetura proposta. O exemplo mais direto de um sistema desse é um *smartphone* que agrupa tecnologias como *bluetooth*, *802.11x* e o próprio rádio da comunicação celular. Outro exemplo comum são os *gateways* residenciais entre tecnologias sem fio (ex.: 3G e 802.11x).

Para esse cenário foram utilizadas as camadas físicas do padrão 802.11b e do padrão 802.15.4, que possuem requisitos de largura de canal diferentes. Como no cenário anterior, foram utilizadas uma implementação seguindo os componentes tradicionais do GNU Radio e uma implementação equivalente utilizando a arquitetura proposta. Sendo que esse teste busca validar não somente o ganho de performance, mas o funcionamento da arquitetura com requisitos de canais não simétricos e todas as suas implicações.

Conforme exibido na Figura 5.3, o ambiente de experimentações foi composto por uma USRP2, um MicaZ, um roteador 802.11b e um computador com as configurações já especificadas. O roteador foi utilizado para transmitir mensagens no padrão 802.11b enquanto o MicaZ transmitiu mensagens no padrão 802.15.4. A USRP2 foi utilizada para compor um receptor múltiplo 802.11b/802.15.4 nos testes das duas implementações, as quais foram executadas com prioridade "tempo real" durante 300 segundos com dados de performance sendo lidos em intervalos de 2 segundos. Devido a proximidade dos dois transmissores, foram selecionados canais não sobrepostos para evitar destruição das mensagens do padrão 802.15.4.

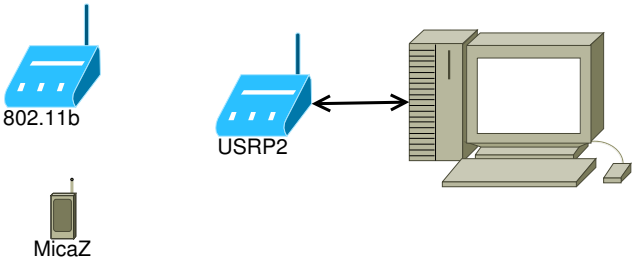
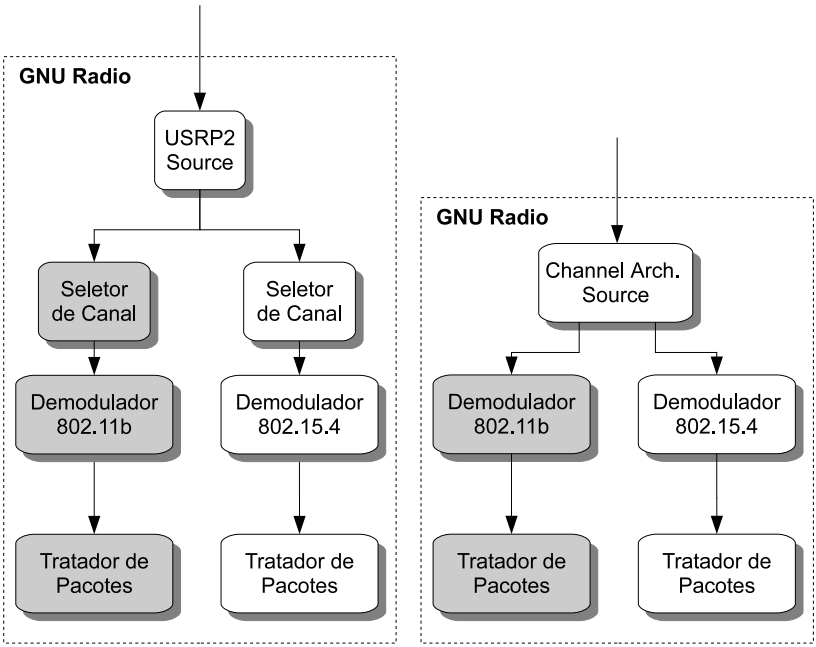


Figura 5.3: *Ambiente de testes para Canais Assimétricos.*

As Figuras 5.4(a) e 5.4(b) mostram as duas implementações testadas: Tradicional e com arquitetura proposta, respectivamente. Como esperado, a imple-

mentação original possui mais blocos em software para executar as tarefas de separação de canais, que no caso da arquitetura proposta são implementadas no hardware. O recebimento dos dois padrões de rede sem fio demonstram a habilidade da arquitetura de abstrair as diferentes taxas do fluxos de dados gerados por canais de diferentes larguras e a mesma flexibilidade alcançada pelos blocos de software utilizados no primeiro teste.



(a) Arquitetura Tradicional (b) Arquitetura Proposta.

Figura 5.4: *Implementação Canais Assimétricos.*

5.2 Avaliação dos Resultados

Esta seção apresenta os resultados obtidos através de experimentos realizados e é organizado de forma a agrupar os aspectos de análise. A performance

foi analisada de forma comparativa, com e sem a arquitetura proposta, para cada implementação dos cenários propostos e com relação ao aumento do número de canais. Outro aspecto abordado é o consumo dos recursos da FPGA para adição da estrutura de canais, com a exemplificação do máximo de canais possíveis em alguns dispositivos comerciais.

5.2.1 Análise de Desempenho

Para avaliar os experimentos foi desenvolvido um bloco (*bench_graph*) para análise de desempenho de implementações do GNU Radio. Esse bloco utiliza o *mpstat* do pacote *sysstat* do Linux, que apresenta a ocupação de cada processador disponível na máquina ou a média global do sistema. A porcentagem de uso da CPU é separado pelo *mpstat* em sete categorias: *user*, *nice*, *system*, *iowait*, *irq*, *soft* e *idle*. A duas primeiras categorias (*user* e *nice*) apresentam a porcentagem do uso da CPU em espaço de usuário com aplicações, sendo que a segunda categoria separa o que é executado com prioridade "*nice*". As categorias *System*, *iowait*, *irq* e *soft* apresentam métricas relacionadas com a porcentagem do uso da CPU pelo kernel, requisição de disco, interrupções e interrupções de software, respectivamente. E a última categoria (*idle*) apresenta o tempo em que CPU fica inativa.

Para simplificar a apresentação dos resultados a saída do *mpstat* foi agrupada em três categorias *USR*, *SYS* e *IDLE*. A *USR* agrupa as duas primeiras categorias (*user* e *nice*) e representa de forma geral o gasto de CPU pelas implementações dos cenários propostos. A *SYS* agrupa os valores do consumo de processamento das tarefas do sistema operacional (*System*, *iowait*, *irq* e *soft*). Por fim, o *IDLE* é o valor direto retirado das medições com *mpstat*.

Para a análise foi utilizado o valor da média global gerada pelo *mpstat*, que representa melhor o consumo dos recursos do sistema por cada implementação. Como descrito anteriormente, todas as medições foram feitas com intervalos de 2 segundos e os testes foram executadas com prioridade "tempo real" durante 300 segundos (150 medições por teste), o que foi suficiente uma vez que as implementações não apresentam grandes oscilações no processamento dos fluxos de amostra, os quais são constantes durante todo o teste.

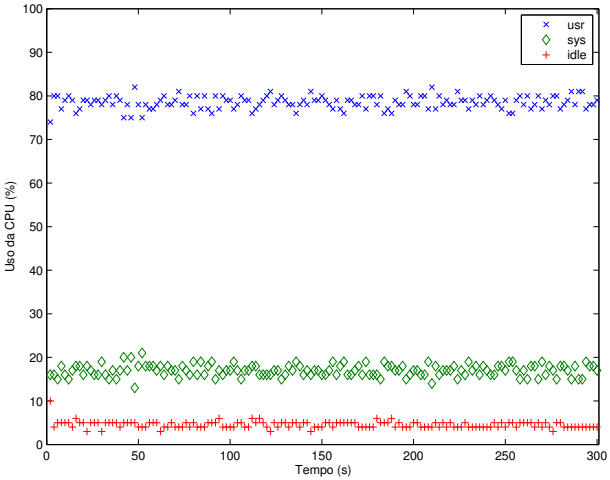
As Figuras 5.5 e 5.6 apresentam os resultados da análise de desempenho para os cenários de *Canais Simétricos* e *Canais Assimétricos*, respectivamente.

Os ambientes de testes foram montados como descrito na Seção 5.1 e as medições de cada uma das categorias (*USR*, *SYS* e *IDLE*) foram plotadas. Como esperado, as implementações tradicionais mostraram desempenho bastante inferior se comparado à arquitetura proposta. Isto pode ser observado pela curva *USR* e *SYS* que somadas mostram uma ocupação média maior que 85% nas duas implementações com a arquitetura original, o que torna o computador praticamente sem responsividade nenhuma a outras possíveis tarefas.

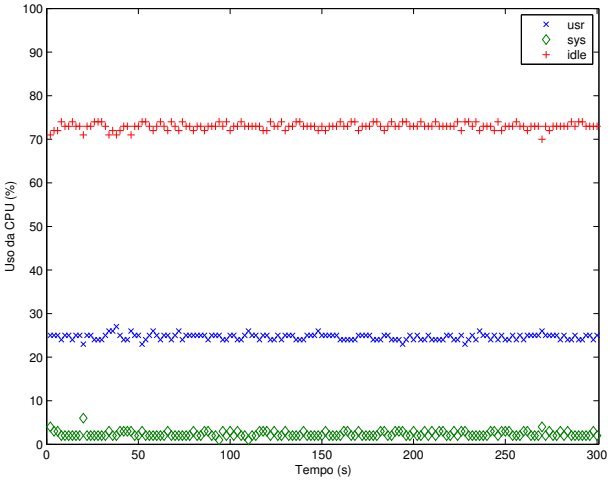
A Figura 5.7 apresenta as médias de ocupação da CPU no primeiro cenário de teste – Canais Simétricos. A ocupação é dividia em *USR* e *SYS* para a implementação com a arquitetura proposta, respectivamente 24,6% e 2,4%. E também para implementação tradicional, respectivamente 78,5% e 17%. A diminuição de 68,7% de ocupação da CPU por tarefas em espaço de usuário quando a arquitetura de canais é utilizada mostra um ganho de performance significativo. Além disso, a diminuição do fluxo de dados, que possuem uma parte tratada diretamente no *hardware*, possibilita uma diminuição da ocupação da CPU pelas tarefas do sistema de 85,9%.

Como esperado isso se repete no cenário de canais assimétricos (Figura 5.8), com uma ocupação média utilizando a arquitetura proposta de 11% e 5,4% para as categorias *USR* e *SYS* e de 56,6% e 31,7% para a implementação com a arquitetura tradicional. Nesse caso a ocupação em espaço de usuário diminuiu 80,6%, enquanto que as tarefas do sistema ocuparam 83% a menos.

Verificou-se também o aumento da ocupação da CPU com relação ao aumento do número de canais. Para isso foram utilizadas as implementações tradicional e proposta para canais simétricos. Foram executadas 150 medições para cada número de canais medindo a média de ocupação total e a soma das médias de ocupação *USR* e *SYS*. A Figura 5.9 apresenta essa relação para as implementações tradicional e com a arquitetura proposta. Com apenas um canal o desempenho é praticamente o mesmo para as duas implementações, uma vez que até a arquitetura tradicional oferece suporte a um canal por interface. O problema está no aumento de canais, pois a implementação tradicional utiliza blocos de software que executam funções similares de alto desempenho para separação de canais, onerando o *host* de forma a diminuir significativamente o número de canais.

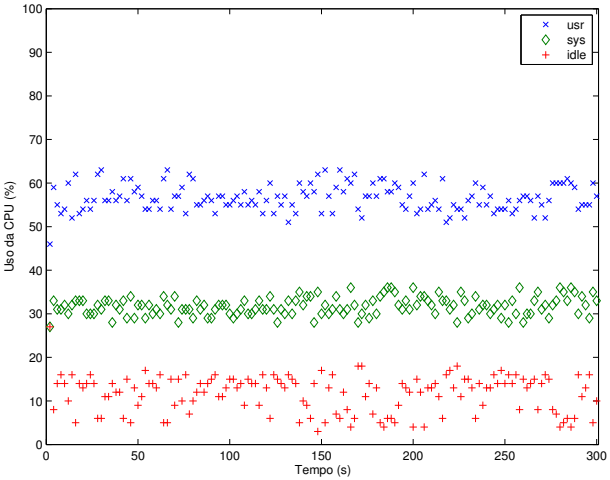


(a) Implementação Tradicional

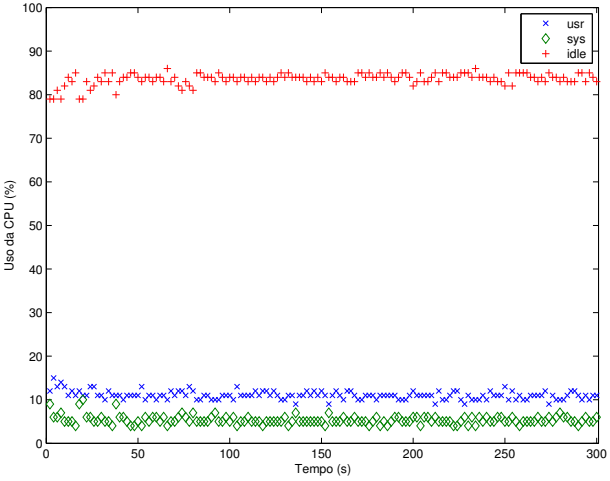


(b) Implementação da arquitetura proposta.

Figura 5.5: *Ocupação da CPU no Cenário de Canais Simétricos.*



(a) Implementação Tradicional



(b) Implementação da arquitetura proposta.

Figura 5.6: *Ocupação da CPU no Cenário de Canais Assimétricos.*

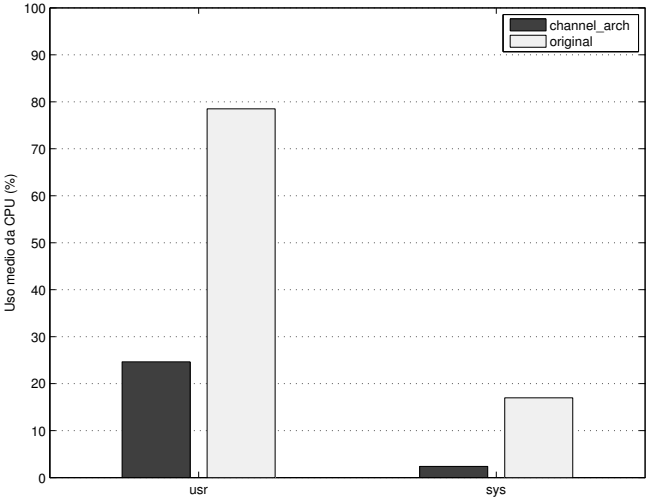


Figura 5.7: *Ocupação média da CPU no Cenário de Canais Simétricos.*

5.2.2 Consumo dos Recursos da FPGA

Na Tabela 5.1 é apresentado o custo relativo ao aumento do número de canais implementados na FPGA para a arquitetura proposta. Os custos são separados em *Slices*, *Flip-Flops*, *LUTs* e multiplicadores em hardware (*18x18 Mult*). O consumo leva em consideração a adição da "estrutura de canal" (Figura 4.2) e o aumento do *buffer pool* para suportar o fluxo das amostras. O aumento de canais tem um consumo praticamente linear, por não adicionar mais do que a lógica necessária para o processamento do canal.

Tabela 5.1: *Consumo de recursos por quantidade de canais adicionado.*

N° canais	Slices	Slices FF	4-input LUTs	18x18 Mult
1	13	9	10	20
2	25	19	21	40
3	38	28	32	60
4	51	38	43	80
5	64	47	53	100

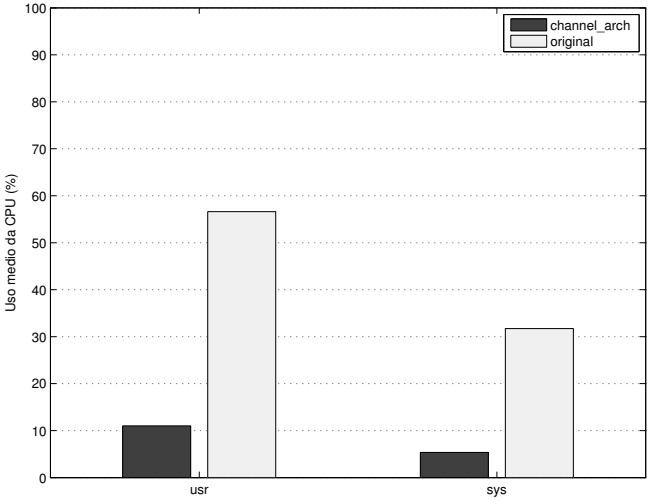


Figura 5.8: *Ocupação média da CPU no Cenário de Canais Assimétricos.*

A Tabela 5.2 mostra o número máximo possível de canais para quatro modelos diferentes de FPGA da *Xilinx*: XC3S2000, XC3SD3400A, XC6SLX100 e XC6VSX315T. Um dos maiores limitantes para a adição de múltiplos canais é o número de multiplicadores em hardware *18x18 Mult*, os quais são intensamente utilizados para processamento de sinais. Devido a isso, fabricantes de FPGA criam linhas específicas para processamento digital de sinais, como por exemplo a famílias *Spartan-3A DSP* e *Virtex-6 SXT*.

Tabela 5.2: *Número de canais suportado pelos modelos de FPGA.*

FPGA	Famíla	N° canais
XC3S2000	Spartan-3	4
XC3SD3400A	Spartan-3A DSP	8
XC6slx100	Spartan-6 LX	10
XC6VSX315T	Virtex-6 SXT	70

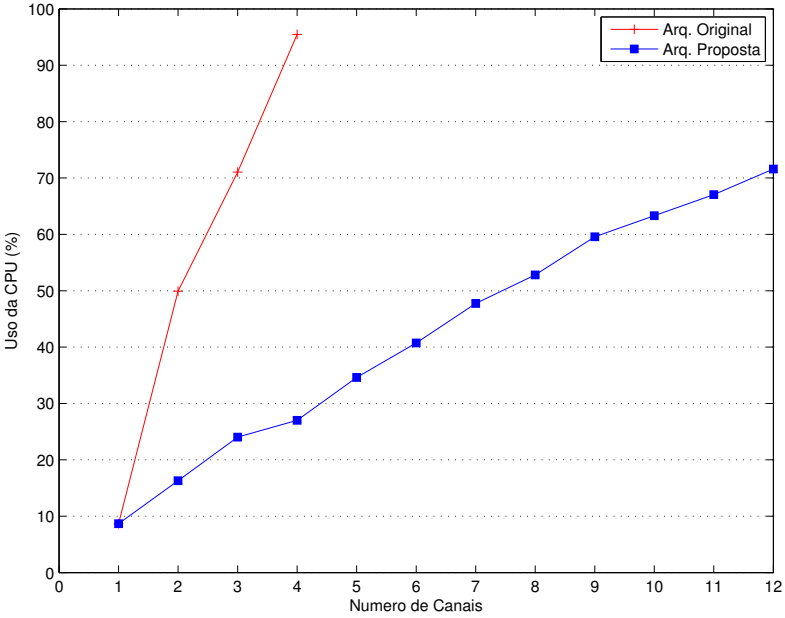


Figura 5.9: *Ocupação da CPU vs. Número de Canais.*

5.2.3 Análise

A principal vantagem da arquitetura proposta perante o sistema originalmente implementado é a capacidade do gerenciamento de canais diretamente no hardware sem a perda da flexibilidade para as implementações de SDR. Isso diminuiu drasticamente a ocupação do processador de uso geral do sistema (*host*), uma vez que o paralelismo intrínseco existente na separação de diversos canais, que exige o processamento concomitante do mesmo grupo de dados, e a alta quantidade de amostras por segundo, proveniente das fases iniciais do rádio, são características onerosas para implementações em software e foram beneficiados pela implementação paralela do hardware.

Outra vantagem é o melhor aproveitamento dos limites impostos pelo hardware da janela de interesse. Como apresentado na Seção 3.2 o hardware

apresenta dois limites, *Janela de Amostragem* e *Interface de Comunicação*. No caso da plataforma utilizada para desenvolvimento do trabalho esses limites são 100MHz e 25 MHz, respectivamente. Ou seja, apesar da interface física conseguir amostrar 100MHz o software recebe apenas uma janela de 25MHz devido ao gargalo da interface de comunicação do *host*.

Além disso, a inferência de todas as configurações do hardware baseados no conjunto de freqüências centrais (F_c) e larguras do canais (L_c) solicitados, simplificam a interface e facilitam a criação de sistemas de múltiplas camadas. Com uma interface de mais alto nível, as camadas físicas podem ser projetas como blocos auto contidos que compõem um repositório e podem ser trocadas a partir de um motor de cognição do sistema para adequação do canal de comunicação aos requisitos da aplicação.

No caso da arquitetura tradicional, esse gargalo é agravado devido a necessidade da janela enviada ao *host* ser contínua, uma vez que há somente o conceito de um canal por interface física. Essa característica causa dois tipos de desperdícios, exemplificados na Figura 5.10, o desperdício de dados amostrados pelo ADC e o desperdício da banda de comunicação do *host*.

Nesse exemplo existem quatro canais de interesse distribuídos na janela de 100MHz amostrada pelo ADC, na arquitetura tradicional apenas dois canais podem ser enviados para o *host*, *ch1* e *ch2* ou *ch3* e *ch4*, ou seja, o hardware tem a capacidade de captura dos quatro canais, mas por limitações da arquitetura isso não é aproveitado. Além disso, a parte hachurada mostra o desperdício de banda da interface com amostras que serão descartadas, uma vez que não fazem parte dos canais de interesse.

A Figura 5.11 apresenta o mesmo cenário com a arquitetura proposta, onde a procura por canais pode acontecer em toda a janela do ADC e multiplexadas no canal de comunicação. Além disso, não há envio de amostras não representativas ao *host*.

Analisando um exemplo mais prático, o padrão mundial IEEE 802.15.4 prevê o uso de 16 canais em uma janela de freqüência de 2400MHz até 2483.5MHz. Cada canal tem uma largura de 2MHz e um espaço entre canais de 3MHz. Com a implementação original uma janela de 25MHz é enviada ao *host* sem processamento de canais prévio, ou seja, as amostras referentes aos espaços do espectro sem informação ocupam banda do canal de comunicação. Assim, podemos considerar que cada canal na verdade ocupa 5MHz e pela Equação (3.10) chega-

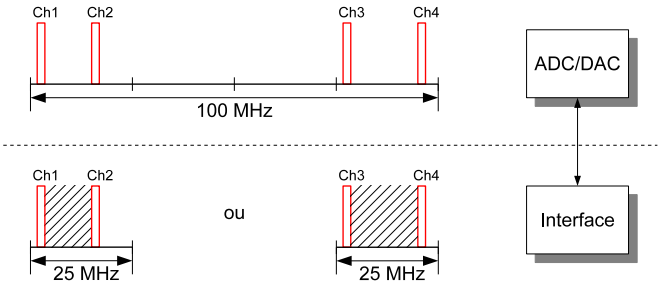


Figura 5.10: *Desperdício das capacidades do hardware pela arquitetura tradicional.*

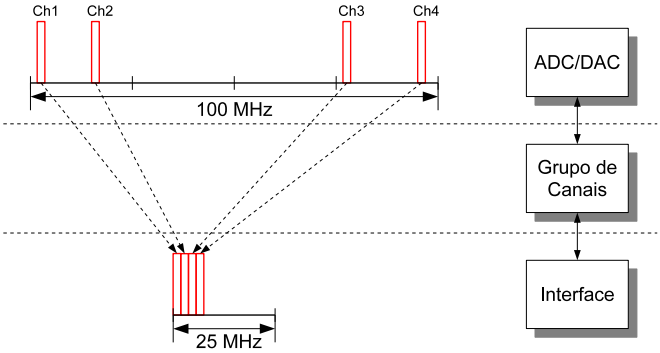


Figura 5.11: *Resolução do desperdício pela arquitetura proposta.*

mos a um número máximo de apenas 5 canais e ainda com a limitação de serem consecutivos. O aproveitamento da janela de 25MHz é de apenas 10MHz de informações úteis (40%). Com a arquitetura proposta podemos programar cada canal separadamente enviando apenas 2MHz de informação por canal para o *host*. Permitindo assim, até 12 canais sem a necessidade de serem consecutivos, além disso, o aproveitamento da janela de 25MHz sobe para 96%.

Assim, a diminuição do uso da CPU, a interface simplificada, o melhor aproveitamento dos escassos recursos de comunicação e a manutenção da flexibilidade exigida por sistemas SDR figuram como vantagens da arquitetura proposta.

CAPÍTULO 6

Conclusões

Os engenheiros de sistemas computacionais tem enfrentado vários desafios para projetar equipamentos de comunicação menores, mais baratos, versáteis e integrados as atividades cotidianas. A comunicação sem fio tem se tornado peça essencial nesse contexto, onde a crescente incompatibilidade entre os padrões se deve pelas limitações impostas pelo domínio das aplicações. Considerando impossível a convergência de todas as redes sem fio para um único padrão, a solução tem sido a integração de diversos componentes para dar a versatilidade exigida pelos equipamentos atuais. Entretanto, a arquitetura dos rádios tradicionais, apesar de robusta, impõem uma série de limitações devido a inflexibilidade intrínseca do hardware. Nesse contexto, o uso de SDRs (Rádios Definidos por Software) aparecem como solução interessante por possuírem a camada física completamente reconfigurável, permitindo alta flexibilidade para criação e atualização dos padrões de comunicação. Entretanto, existe uma lacuna no suporte nativo à implementação de múltiplas camadas físicas compartilhando a mesma interface física, o que causa um *overhead* maior que o desejado na criação de sistemas multipadrões.

O presente trabalho analisou os problemas recorrentes na implementação de rádios com múltiplas camadas e propôs uma arquitetura para SDRs com o conceito de desacoplamento de canais da camada física. Mesmo sendo um conceito já conhecido, a arquitetura apresentada se destaca por propor uma interface genérica que atende a todos os tipos de camadas físicas e pode ser suportada por diversos *hardware* de SDR. A arquitetura proposta permitiu o deslocamento das fases com alto consumo de processamento, que necessitam altas taxas de amostragem, para o hardware reconfigurável. O que possibilitou a alocação de vários canais em uma mesma interface física de forma transparente e garantiu uma distribuição mais eficiente das tarefas entre hardware (ex.:FPGA) e software (ex.: GPP). A arquitetura foi implementada utilizando o próprio GNU Radio e a USRP2, onde foi detectado o problema, entretanto os conceitos utilizados sugerem que sua utilização é possível em outros *frameworks* para construção de SDRs, uma vez que não possui dependência das estruturas do GNU Radio ou USRP2. Os testes comparativos entre a implementação tradicional e a arquitetura proposta demonstraram ganhos significativos no aproveitamento dos recursos do sistema, relacionados mais especificamente ao desempenho e as interfaces de comunicação. Além de uma simplificação na interface de configuração, que tornou transparente vários

parâmetros do hardware.

A principal proposta para trabalhos futuros é a resolução dos problemas relacionados com a latência de comunicação entre o momento em que o sinal é pela antena e as amostras chegam nos blocos de software. Um menor atraso possibilitaria uma implementação mais fiel dos protocolos baseados em detecção da portadora (ex.: CSMA). Outro trabalho futuro é uma análise de domínio das camadas físicas a fim da criação de uma interface padronizada para acesso das camadas superiores (ex.: MAC). Isso facilitaria o provimento de serviços, como troca de padrão de comunicação, baseados nos requisitos da aplicação, como por exemplo, largura de banda, energia e imunidade a ruído. Outra iniciativa, que inclusive já está em curso, é a criação do projeto *Embedded Software Defined Radio* (eSDR), que visa diminuir ainda mais o *overhead* dos sistemas atuais a fim de permitir que equipamentos embarcados que utilizem a flexibilidade dos rádios definidos por software. Os resultados deste projeto deverão ser abertos e disponibilizados para acesso público na Internet.

Referências Bibliográficas

[1] Bryan Ackland, Dipankar Raychaudhuri, Michael Bushnell, Christopher Rose, and Ivan Seskar. High performance cognitive radio platform with integrated physical and network layer capabilities. Technical report, http://www.winlab.rutgers.edu/pub/docs/NeTS-ProWiN1.pdf, Jul 2005.

[2] Philip Balister, Tom Tsou, and Jeffrey H. Reed. Software defined radio on small form factor systems, Mar 2007.

[3] Eric Blossom. Exploring gnu radio. Technical report, http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html, Jul 2006.

[4] Eric Blossom. How to write a signal processing block. Technical report, http://www.gnu.org/software/gnuradio/doc/howto-write-a-block.html, Jul 2006.

[5] Eric Blossom. Gnu radio. http://www.gnu.org/software/gnuradio, 2009.

[6] V. Bose, R. Hu, and R. Morris. Dynamic physical layers for wireless networks using software radio. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 4:2045–2048, 2001.

[7] Vanu Bose, Michael Ismert, Matt Welborn, and John Guttag. Virtual radios. *IEEE Journal on Selected Areas in Communications*, 17(596-602):1090–1097, April 1999.

[8] D. D. Braga and E. C. Gurjão. Implementation of an adaptive modulation system using software defined radio. In *8th International Information and Telecommunication Technologies Symposium*, pages 224–227, dezembro 2009.

[9] J. Chapin and K. Allain. Waveforms for dynamic wireless networks: Is layering a good idea. In *2004 Software Defined Radio Technical Conference*, Nov 2004.

[10] Leslie Choong. Multi-channel ieee 802.15.4 packet capture using software defined radio. Technical report, Mar 2009.

[11] Rahul Dhar, Gesly George, Amit Malani, and Peter Steenkiste. Supporting integrated mac and phy software development for the usrp sdr. In *Networking Technologies for Software Defined Radio Networks, 2006. SDR '06.1st IEEE Workshop on*, pages 68–77, Sept. 2006.

[12] Tad James Dreier. Design Environment for Rapid Prototyping of Software Defined Radio. Master's thesis, University of California, Los Angeles, 2006.

[13] Matt Ettus. Universal software radio peripheral. http://www.ettus.com/, 2009.

[14] J. Eyre and J. Bier. The evolution of dsp processors. *Signal Processing Magazine, IEEE*, 17(2):43–51, mar. 2000.

[15] Bruce Fette. *Cognitive Radio Technology*. Elsevier, Massachusetts, USA, 2006.

[16] JNL Research Group. Sdr documentation. University of Notre Dame https://radioware.nd.edu/documentation, 2009.

[17] A. Haghighat. A review on essentials and technical challenges of software defined radio. In *MILCOM 2002. Proceedings*, volume 1, pages 377–382, oct. 2002.

[18] H. Harada. Software defined cognitive radio prototype toward imt-advanced wireless communication systems. In *Radio and Wireless Symposium, 2007 IEEE*, pages 7–10, jan. 2007.

[19] KUAR. Kansas university agile radio. https://agileradio.ittc.ku.edu/, 2009.

[20] Sen M. Kuo and Woon-Seng S. Gan. *Digital Signal Processors: Architectures, Implementations, and Applications*. Prentice Hall, Massachusetts, USA, 2004.

[21] A. V. G. Menezes, D. D. Braga, and E. C. Gurjão. Automatic modulation recognition using software defined radio. In *8th International Information and Telecommunication Technologies Symposium*, pages 228–231, dezembro 2009.

[22] J. Mitola. The software radio architecture. *Communications Magazine, IEEE*, 33(5):26–38, May 1995.

[23] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Prentice Hall, New Jersey, USA, 1975.

[24] Erik L. Org, Russel J. Cyr, Geoff Dawe, John Kilpatrick, and Tim Counihan. Software defined radio different architectures for different applications. In *SDR Forum Technical Conference 2007*, 2007.

[25] Jeffrey Reed. *Software radio: a modern approach to radio engineering*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2002.

[26] Thomas Schmid, Tad Dreier, and Mani B. Srivastava. Software Radio Implementation of Shortrange Wireless Standards for Sensor Networking. In *Conference On Embedded Networked Sensor Systems*, 2006.

[27] Thomas Schmid, Oussama Sekkat, and Mani B. Srivastava. An experimental study of network performance impact of increased latency in software defined radios. In *WinTECH '07: Proceedings of the the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 59–66, New York, NY, USA, 2007. ACM.

[28] Andrew S. Tanenbaum. *Redes de Computadores*. Campus, 2003.

[29] David L. Tennenhouse and Vanu G. Bose. The spectrumware approach to wireless signal processing. *Wireless Network Journal*, 2, 1996.

[30] VANU. Vanu, inc. http://www.vanu.com/, 2009.

[31] WARP. Rice university wireless open-access research platform (warp). http://warp.rice.edu, 2009.

[32] Matt Welborn, Sunil Rao, Ripal Nathuji, Rattapoom Tuchinda, John Ankcorn, Steve Garland, and John Guttag. Spectrumware project. http://nms.csail.mit.edu/projects/spectrumware/, 2009.