Capítulo 1

Introdução a Rádios Definidos por *Software* com aplicações em GNU Radio

Wendley S. Silva, Jefferson Rayneres S. Cordeiro, Daniel F. Macedo, Marcos A. M. Vieira, Luiz F. M. Vieira, José Marcos S. Nogueira

Abstract

Software-defined radios (SDR) allow reducing the amount of communication functions implemented in hardware. Thus, the communication devices become more flexible and easy to be programmed. SDRs are used in experimental research in Computer Networks, as well as prototyping and fast deployment of new communication technologies. This course covers introductory concepts of SDR and GNU Radio platform. The course shows the basic concepts of wireless communications and digital transmissions, focusing on the data link layer. Practical applications of SDR, focused on experimental research on wireless networks, such as proposing new modules for GNU Radio, are demonstrated.

Resumo

Rádios definidos por software (SDR) permitem reduzir a quantidade de funções de comunicação implementadas em hardware. Desta forma, os dispositivos de comunicação se tornam mais flexíveis e fáceis de serem programados. SDRs são empregados na pesquisa experimental em Redes de Computadores, bem como na prototipagem e implementação rápida de novas tecnologias de comunicação. Este minicurso aborda os conceitos introdutórios de SDR e da plataforma GNU Radio. O curso apresenta os conceitos básicos de comunicações sem fio e transmissões digitais, focando nos aspectos relativos à camada de enlace. São demonstradas aplicações práticas de SDR, focadas na pesquisa experimental em redes sem fio, por exemplo como propor novos módulos no GNU Radio.

1.1. Introdução

De acordo com [Wireless Innovation Forum], os Rádios Definidos por *Software* (*Software-Defined Radios – SDR*) são um conjunto de tecnologias de *hardware* e *software*, onde algumas ou todas as funções operacionais do rádio são implementadas através de *software*

ou *firmware* modificável, que é executado em tecnologias de processamento programáveis (por exemplo, um FPGA, um CPU genérico). Por outro lado, em [Dillinger et al. 2003] o conceito SDR é um pouco diferente: SDR é um sistema de comunicação de rádio, onde os componentes que eram tipicamente implementados em *hardware* (por exemplo, filtros, amplificadores, moduladores/demoduladores, detectores *etc*.) agora são implementados em *software* utilizando um computador pessoal ou sistemas embarcados. O conceito utilizado neste trabalho é o apresentado em [Reis et al. 2012], em que SDR é um transceptor sem fio, onde módulos de *software* implementam as funcionalidades do transceptor. Estes módulos são executados em tempo real em microprocessadores genéricos, processadores de sinais digitais ou circuitos lógicos programáveis.

Os militares dos EUA foram os primeiros a empregar SDR, tentando fornecer rádios flexíveis para operações de grande escala [Dillinger et al. 2003]. O objetivo era garantir a interoperabilidade dos rádios militares com o de outras agências governamentais (bombeiros, polícia, agências de inteligência). Tal necessidade vem de uma razão prática, pois cada órgão realiza as suas compras de forma independente, e assim as tecnologias de comunicação empregadas podem ser incompatíveis do ponto de vista da frequência utilizada, tecnologia de modulação de sinais, dentre outros. Assim, o termo "rádio digital" foi cunhado para definir rádios que se adaptam a diferentes padrões operacionais.

Nesse contexto, Joseph Mitola propôs em sua tese de doutorado o conceito de rádios cognitivos, que são rádios que adaptam a sua operação com base em condições do ambiente, aprendendo com o comportamento passado e melhorando o seu funcionamento com o tempo [III 2000]. O conceito de rádio cognitivo é mais amplo que os rádios digitais, pois envolve mecanismos de inteligência artificial e aprendizado para identificar a condição do meio e propor uma configuração adequada a cada momento. Devido às altas demandas de processamento de rádio cognitivo, que eram impraticáveis na época, o conceito inicialmente não recebeu muita atenção.

Em 2002, o FCC (*Federal Communications Commission*), órgão regulador das telecomunicações nos EUA, reacendeu o interesse em rádios cognitivos. O FCC publicou um relatório sobre a utilização do espectro de rádio-frequência nos EUA [Force 2002], concluindo que o espectro eletromagnético é um recurso utilizado de forma ineficiente e escasso. O FCC descobriu que, apesar da maior parte do espectro já possuir um uso licenciado, os detentores da licença não usam o espectro todo o tempo e em todas as regiões geográficas onde a sua licença é válida. Assim, o FCC propôs a utilização oportunista do espectro, onde os usuários não licenciados poderiam explorar espectro vago sempre que o usuário licenciado não esteja a utilizando no momento.

Um aspecto fundamental para o sucesso dos rádios cognitivos são transceptores sem fio capazes de operar em múltiplas frequências, que ainda podem detectar a existência de usuários licenciados. Para tanto, o rádio deve ser capaz de "falar" diversas tecnologias de comunicação, seja para detectar os usuários licenciados, seja para comunicar de forma eficaz em frequências com características bem distintas. Além dos rádios cognitivos, o SDR permite ainda diversas aplicações.

1.1.1. Aplicações de SDR

Como veremos a seguir, os SDR são aplicáveis em diversos domínios, desde rádios universais de baixo custo a sistemas flexíveis que permitem a pesquisa experimental em Redes de Computadores e Telecomunicações. Primeiramente iremos discutir os usos comerciais da tecnologia.

1.1.1.1. Usos comerciais

Em 2011, o *Wireless Innovation Forum* encomendou um estudo para avaliar a taxa de adoção de SDR na indústria [Forum 2011]. Os resultados indicam que mais de 90% da infra-estrutura móvel naquele ano empregava tecnologias SDR de algum tipo. Para os mercados em que a interoperabilidade é um requisito obrigatório, como em aplicações militares e de segurança pública, eles descobriram que quase todos os transceptores e estações base empregam SDR. No entanto, dependendo da largura de banda do sinal, algum processamento ainda é realizado em *hardware*. Por exemplo, turbo códigos e correção de erro (*Forward Error Correction*) são freqüentemente implementados em *hardware*.

Rádios definidos por *software* também estão sendo utilizados para reduzir os custos para os operadores de telecomunicações. A *China Mobile* está promovendo a adoção de C-RAN [Chen and Duan 2011], que é uma torre de celular SDR onde todo o processamento de dados é feito em servidores em nuvem. Seu objetivo é a implementação de todas as funcionalidades via *software* executado em *data centers*, onde é mais fácil de atualizar as funções de rede. Ao mesmo tempo, esta arquitetura simplifica as torres celulares, reduzindo o consumo de energia.

Há também muitas empresas que exploram o mercado de SDRs para rádio amadores. Muitas plataformas são focadas em amadores ou *geeks* que querem desenvolver seus próprios receptores para aplicações como navegação marítima, rádio amador, dentre outros. Mesmo produtos comerciais que não foram destinados como SDR são reprogramados para este meio, uma vez que os usuários descobrem que um certo *hardware* ou placa realiza todo o processamento de sinais em *software* [Cass 2013]. Alguns grupos de rádio amador disponibilizaram receptores SDR ligados à Internet, onde os entusiastas podem sintonizar a frequência desejada em diferentes partes do mundo, sem a necessidade de comprar seus próprios SDRs[1].

Rádios definidos por *software* são agora uma realidade, como se vê pela quantidade de plataformas comerciais e gratuitas disponíveis no mercado [2]. Há uma lista extensa de módulos de *software* livre, e um bom suporte e integração com os ambientes de desenvolvimento (IDEs) mais populares do mercado. No entanto, as plataformas existentes ainda estão lutando para alcançar os requisitos de largura de banda e eficiência de energia necessários para padrões de alta velocidade, tais como LTE (*Long Term Evolution*, um dos protocolos de 4G) e o IEEE 802.11ac.

[1] Alguns exemplos são `http://websdr.org/` e `http://rsgb.org/main/operating/web-sdr-receiver/`.

[2] Ver `http://en.wikipedia.org/wiki/List_of_software-defined_radios` para uma lista de algumas das plataformas existentes.

1.1.1.2. Usos acadêmicos

Dada a quantidade de plataformas comerciais e acadêmicas viáveis que são capazes de implementar 3G e rádios IEEE 802.11a/b/g/n e outras tecnologias, diversos grupos de pesquisa passaram a integrar SDRs na sua lista de equipamentos. Um aspecto positivo do SDR é que ele diminuiu o custo para a realização de pesquisas experimentais em tecnologias da camada física e enlace. Assim, experimentos que antes eram possíveis somente em laboratórios de grandes empresas, podem ser feitos em laboratórios de universidades a um custo razoável. Para dar uma importância do SDR na área de redes sem fio, somente a plataforma WARP da universidade de Rice contabilizou 59 artigos científicos que empregaram o seu *hardware* em 2014[3], apesar do seu alto preço (a partir de USD5.000 a unidade). Seguem abaixo algumas propostas recentes que foram possíveis devido ao uso de SDRs.

- **Uso de codificação de rede (*network coding*) para aumentar a capacidade.** Usando o conceito de codificação de rede, é possível enviar um pacote que é a combinação de vários pacotes em enlaces sem fio, aumentando a vazão global da rede. Os ganhos dependem do padrão de tráfego, que vão desde uma pequena percentagem até a ganhos de várias vezes [Katti et al. 2008]. O uso de múltiplas taxas de transmissão pode apresentar ganhos ainda maiores [Vieira et al. 2013].

- **Recuperação de quadros perdidos com o processamento de sinais.** Os rádios armazenam os sinais recebidos a partir de uma colisão, e tentam subtrair quadros recebidos corretamente (após uma retransmissão), permitindo muitas vezes obter o quadro envolvido em uma colisão sem que esse precise ser retransmitido. [Lin et al. 2008].

- **Códigos *Rateless*.** Na comunicação tradicional sem fio, os dados são transmitidos empregando uma certa modulação. Cada modulação requer um certo limiar *Signal to Interference plus Noise Ratio (SINR)* para a decodificação correta dos seus dados e, de modo a lidar com as variações no SINR, os protocolos existentes (por exemplo os protocolos WiFi e WiMax) usam mecanismos de troca automática de modulação. Códigos *Rateless* [Gudipati and Katti 2011, Perry et al. 2012, Shokrollahi 2006] utilizam uma quantidade variável de símbolos para codificar os dados. O transmissor envia os dados utilizando códigos diferentes, e o receptor usa esses códigos para identificar qual palavra é a mais provável que tenha sido empregada para gerar os sinais recebidos. Códigos *Rateless* exigem protocolos especiais da camada de enlace [Iannucci et al. 2012], que também podem ser testados usando SDR. O principal benefício de códigos *rateless* é que ele fornece uma largura de banda no enlace muito mais próxima da capacidade teórica de Shannon.

- **Rádios *full-duplex*.** Os rádios comerciais existentes são *half-duplex*, uma vez que uma antena de recepção seria saturada com os sinais transmitidos por outra antena adjacente. No entanto, pesquisas recentes empregando SDR e *hardware* dedicado conseguem obter níveis de cancelamento de ruído tais que permitem imple-

[3] http://warp.rice.edu/trac/wiki/about

mentar rádios *full duplex* compatíveis com as larguras de banda dos padrões IEEE 802.11b[Hong et al. 2012] e IEEE 802.11ac[Bharadia et al. 2013].

- **Separação de canais de controle e de dados.** Quando é necessário separar os quadros de controle dos quadros de dados em redes sem fio, a maioria das implementações empregam vários rádios. Em Flashback[Cidon et al. 2012], o rádio gera pulsos OFDM que criam um canal de controle separado, na mesma frequência utilizada pelo canal de dados. Este canal de controle permite que os nós enviem mensagens curtas de controle, sem comprometer a decodificação dos quadros de dados e sem reduzir a taxa de transferência do canal de dados.

- **MIMO cooperativo para WLANs empresariais.** OpenRF [Kumar et al. 2013] usa o conceito de vetores de codificação, a fim de executar a formação de feixes que permitem o alinhamento e o cancelamento de interferência em redes WLAN com múltiplos pontos de acesso. Com a ajuda de um controlador centralizado e um algoritmo de escalonamento local rodando em cada ponto de acesso, é possível explorar as capacidades de MIMO do padrão IEEE 802.11n para melhorar a SINR nas estações. Isto, por sua vez, reduz as variações em latência e aumenta a vazão da rede.

1.1.2. Relação entre os conceitos de SDN e SDR

As redes definidas por *software* (*Software-Defined Networks* – SDN) são uma tecnologia muito popular no mercado de redes atualmente, em que os algoritmos de controle são executados fora dos dispositivos de rede. Isto permite uma separação entre o plano de dados (que é responsável pelo encaminhamento e codificação dos dados na rede) e o plano de controle (que implementa as *políticas* que controlam o funcionamento do plano de dados). No entanto, em SDN, somente é possível modificar o funcionamento do plano de controle, enquanto o plano de dados permanece inalterado.

Assim, o SDR se apresenta como uma tecnologia complementar ao SDN, por permitir modificar o plano de dados em redes sem fio. Em um sistema misto SDN-SDR, seria possível desenvolver redes que empregam diversos mecanismos de transmissão de dados, que seriam controlados utilizando protocolos SDN. Enquanto o SDR se preocupa na adaptação da transmissão em cada enlace, os mecanismos de SDN garantiriam que todos os enlaces, e assim a rede como um todo, operem de forma coordenada e otimizada.

Em Sistemas Operacionais e gerenciamento de redes existe um conceito muito útil para explicar a interação entre SDN e SDR: o conceito de *política* e *mecanismo*. Os mecanismos definem algoritmos, procedimentos ou componentes que realizam uma determinada função, enquanto as políticas definem os parâmetros dos mecanismos, e como eles se relacionam. Por exemplo, para autenticação de usuários temos mecanismos como *servidor LDAP* e *contas locais*. Podemos construir políticas tais como: permitir autenticação de contas LDAP a qualquer momento, e contas locais somente em dias úteis e em horário comercial. Desta forma, podemos pensar que SDR irá fornecer os mecanismos para a construção de uma rede, enquanto SDN irá cuidar das políticas da rede, orquestrando as interações entre os diversos dispositivos sem fio.

1.1.3. Vantagens e desvantagens de SDR

Apesar de trazer enormes vantagens em relação à metodologia tradicional de desenvolvimento de placas de rádio-frequência, o SDR também possui desvantagens. Assim, quando utilizada em produtos comerciais, é importante avaliar a cada projeto se o uso de SDR é mais vantajoso que uma alternativa de circuito integrado puro. A seguir listamos algumas vantagens e desvantagens do uso de SDR.

- **Vantagem: dispositivos evolutivos**. Como a maioria das funcionalidades em SDR é implementada em *software*, o equipamento pode receber novas funcionalidades ou correções de *bugs* mesmo após a sua comercialização e entrada em operação. Por exemplo, se encontramos um *bug* em um módulo de demodulação do sinal de um telefone celular, a única solução é mudar o componente. O uso de camadas PHY/MAC programáveis permitiria correções. Além disso, o mesmo dispositivo pode ser atualizado sempre que novos padrões são liberados. Apesar dos dispositivos atuais já possuírem parte desta flexibilidade por utilizarem o conceito de *firmware*, em geral a totalidade ou grande parte da camada física dos dispositivos é implementada em *hardware*.

- **Vantagem: reuso de *hardware***. Cada país tem suas próprias regras no que diz respeito às comunicações, assim, grandes empresas são obrigadas a fabricar diferentes versões de um mesmo *hardware* a fim de obedecer a legislação de cada país (por exemplo, esquemas de modulação diferentes, formatos de mensagens, faixas de frequências *etc.*). Por exemplo, diferentes padrões de TV digital são adotados na Europa, EUA, Ásia e Brasil. Isso aumenta o custo, uma vez que os componentes são fabricados em uma quantidade menor. Enquanto isso, com *hardware* programável, as especificidades de cada região poderiam ser implementadas em *software*, criando uma única plataforma de *hardware*. Um "telefone mundial" exigiria menos *chips* e também seria menor: em vez de ter que possuir um chip por padrão, o chip poderia ser reprogramado sobre demanda.

- **Vantagem: protocolos de comunicação cientes da aplicação**. A comunicação TCP/IP, padrão nas redes, opera em um sistema de caixa preta. O uso de abstrações como mensagens, circuitos ou pacotes limita o conjunto de ações possíveis para operações básicas, como a filtragem por endereço e porta, estabelecendo parâmetros de QoS *etc*. Como a implementação tem de suportar fluxos genéricos, o equipamento deve utilizar soluções muito abrangentes, e desta forma não é possível empregar otimizações específicas de cada aplicação. Em muitas situações, redes sem fio são empregadas para um propósito específico, e são interligadas com redes tradicionais por um nó sorvedouro ou uma ponte. Assim, o trecho que é utilizado para um propósito específico poderia empregar protocolos otimizados para a aplicação, por exemplo utilizando os conceitos de redes centradas em dados ou identificadores geolocalizados.

- **Vantagem: uso de níveis mais altos de abstração**. A maioria das arquiteturas de *hardware* programável fornece um conjunto de interfaces de programação (APIs), uma linguagem de programação de domínio específico (DSL) ou abstrações baseadas em componentes. Isso aumenta a independência do *hardware* em relação ao

software. Maiores níveis de abstração permitem que os compiladores, sistemas operacionais e *middlewares* possam executar métodos de otimização e verificação mais complexos, gerando um código melhor, mais rápido e mais confiável, e reduzindo o tempo de desenvolvimento.

- **Desvantagem: desempenho inferior.** Sistemas implementados totalmente em circuitos integrados tendem a superar os sistemas baseados em *software* em termos de desempenho devido a muitas razões. Um sistema baseado em *hardware* é otimizado para a tarefa em questão, entretanto um sistema baseado em *software* geralmente emprega um *hardware* genérico (por exemplo, uma CPU ARM ou uma FPGA), requerendo assim mais ciclos para executar as mesmas tarefas. Outro aspecto é que a mesma unidade de processamento de um dispositivo programável pode ser compartilhada entre muitas tarefas, de comunicação ou não, reduzindo a capacidade de processamento efetiva para a comunicação e gerando atrasos. Finalmente, a execução de muitas camadas de *software* (por exemplo, do sistema operacional, bibliotecas, *middlewares*) acrescenta um custo adicional de processamento, o que reduz o desempenho final. Para ilustrar, *switches* implementados em *software* rodando em CPUs tendem a processar 1 ou 2 ordens de grandeza menos pacotes por segundo que os *switches* baseados em *hardware* [Intel 2013].

- **Desvantagem: uso de energia superior e maior área de circuito.** Uma implementação programável pode exigir mais portas lógicas do que uma implementação de *hardware* completo. Esta pode ser uma limitação significativa para as redes programáveis em situações onde o tamanho do *chip* e o consumo de energia são restrições de projeto fundamentais, como por exemplo em dispositivos móveis. Entretanto, vale lembrar que a área de circuito depende muito das operações fornecidas pelo *hardware* e do seu desenho. A micro-programação, que nada mais é que uma forma de programação dos componentes de um micro-processador, permitiu uma redução da área das CPUs ao reduzir a complexidade dos seus componentes internos.

- **Desvantagem: problemas de interoperabilidade.** A proliferação de diferentes implementações pode exacerbar a interoperabilidade das redes diferentes. Por exemplo, um ponto de acesso pode ter que lidar com os pacotes com formatos desconhecidos, ou um ponto de acesso pode enfrentar uma rede com uma mistura de estações que implementam o 802.11 tradicional bem como outras implementações customizadas do mesmo.

- **Desvantagem: segurança**. Hoje em dia, uma parte ainda significativa da funcionalidade dos transceptores é implementada em *hardware*, de modo que é impossível modificá-la sem o acesso físico ao *hardware*. No entanto, como mais funcionalidades são movidas para *software*, as falhas de segurança podem ocorrer em níveis mais baixos dos protocolos, com um custo menor, e estas podem ser exploradas remotamente. Por exemplo, um *hacker* poderia inutilizar uma placa Wi-Fi, do outro lado do mundo, para transformá-la em um gerador de interferência.

1.2. Arquiteturas e plataformas de desenvolvimento

1.2.1. Tipos de SDR

A Figura 1.1 mostra a arquitetura básica de um SDR que opera tanto como um transmissor ou receptor. Nela, os ADCs (conversores analógico para digital), DACs (conversores de digital para analógico) e filtros de frequência são componentes que devem ser implementados em *hardware* (blocos em cor branca), enquanto todas as outras operações do transceptor poderiam ser realizadas por unidades de processamento programáveis (bloco em cor cinza escuro). A unidade de processamento pode empregar CPUs de propósito geral, processadores de sinais (DSP), FPGAs, ou até mesmo lógica discreta [Nagurney 2009]. Em muitas aplicações, o sinal de banda base (*baseband*) em si é digital, o que permite que os ADCs e DACs para a banda base, bem como os filtros associados, possam ser eliminados e os sinais digitais são alimentados diretamente para a unidade de processamento.

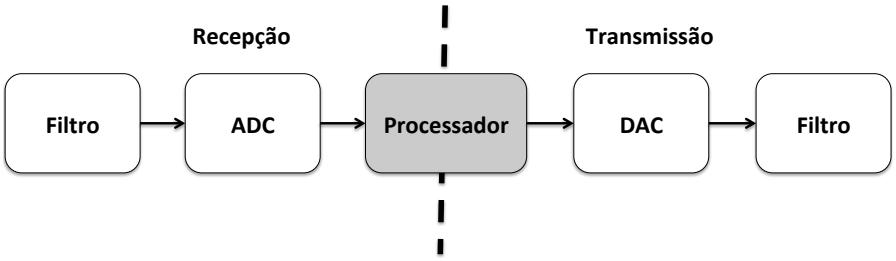

Figura 1.1. Arquitetura simplificada de um Rádio Definido por Software.

Para implementar SDR, existem duas arquiteturas principais. A solução mais simples é colocar vários *chipsets* que implementam diversos padrões de comunicação em um mesmo rádio e alternar entre eles conforme necessário, mas isso tem pouca flexibilidade. Outras técnicas atingem a flexibilidade usando configurações de *hardware* especializados, mas esses sistemas têm suas desvantagens. As duas principais arquiteturas utilizadas na prática são chamadas de *SDR Modal* e *SDR Reconfigurável*, e estão detalhadas abaixo.

1.2.1.1. SDR Modal

Arquiteturas de SDR Modal funcionam como um rádio com N implementações, que são alternadas sobre demanda. Esta arquitetura surgiu na década de 1990, quando as tecnologias celulares analógicas foram gradualmente substituídas por redes digitais. Nesta situação, eram necessários telefones que pudessem oferecer serviços digitais a maior parte do tempo, mas estes deveriam suportar o modo analógico em áreas onde a cobertura digital ainda não existia. A solução óbvia era combinar os *chipsets* analógicos e digitais no interior do aparelho, com *software* realizando o chaveamento entre eles. A Figura 1.2 mostra um exemplo de rádio modal, tendo duas tecnologias implementadas: TDMA e AMPS. Na figura, caixas cinzas indicam componentes baseados em *software*, enquanto que os componentes em caixas brancas indicam componentes em *hardware*. A abordagem de SDR Modal continua a ser uma alternativa para aplicações que necessitam de uma quantidade

limitada de flexibilidade, por exemplo, para redes celulares que fornecem conectividade 3G e 4G.

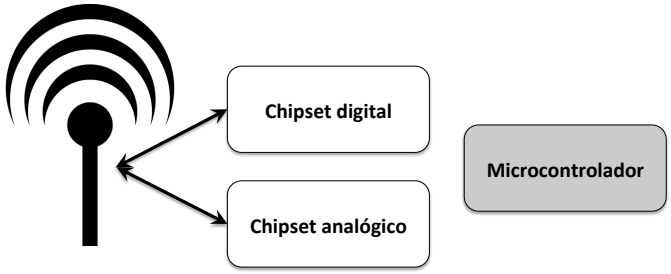

Figura 1.2. Exemplo de um SDR modal.

1.2.1.2. SDR Reconfigurável

Existem aplicações em que o SDR modal é uma solução inadequada, por exemplo na pesquisa experimental. Além disso, sistemas de SDR Modal não escalam bem com a quantidade de formas de onda a serem suportadas. Nestas situações, é mais interessante empregar *hardware* programável para fazer o processamento dos sinais. Desta forma, o *hardware* pode suportar uma quantidade ilimitada de padrões e técnicas de transmissão. Esta abordagem é frequentemente denominada SDR reconfigurável, porque o *hardware* pode ser configurado para qualquer aplicação. Como mostrado na Fig 1.3, o elemento programável pode ser composto por FPGAs, DSPs e/ou CPUs, que podem ser programados para realizar operações de processamento de sinais em alta velocidade. Novamente, componentes na cor branca indicam blocos de *hardware*, enquanto que blocos na cor cinza indicam *software*.

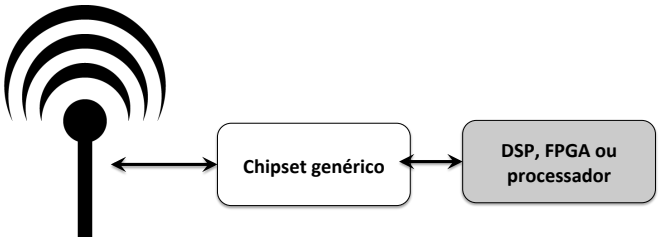

Figura 1.3. Exemplo de um SDR reconfigurável.

1.2.2. Plataformas de SDR

Nesta seção iremos apresentar algumas plataformas existentes de SDR. Esta lista não é exaustiva, e foi compilada tendo em vista apresentar o aspecto de possibilidades de projeto e capacidades das plataformas de SDR atual.

1.2.2.1. USRP

A *Universal Software Radio Peripheral* (USRP) é um *framework* para o desenvolvimento de rádios digitais, proporcionando uma infra-estrutura completa para o processamento de sinais. O sistema caracteriza-se por sua alta flexibilidade e custo-benefício [Ettus Research a]. No USRP, mais de uma antena pode ser ligada ao dispositivo, em função das exigências do usuário. É possível, por exemplo, ligar três antenas com o objetivo de transmitir diferentes estações de rádio FM. Também é possível realizar o procedimento inverso, ou seja, receber várias frequências ao mesmo tempo. Outra possibilidade é a ligação simultânea de antenas para transmitir e receber sinais usando MIMO. USRPs são normalmente programados com o *software* GNU Radio, simplificando o desenvolvimento de novas soluções SDR através da reutilização de funcionalidades existentes.

As placas filhas (*daughter-boards*) implementam o *front-end* de rádio frequência do USRP. Estas placas executam a conversão digital analógica, definindo assim a potência e frequência dos sinais emitidos e recebidos. O USRP trabalha com potências de sinal entre 50 e 200 mW. Finalmente, o USRP permite a ligação de várias placas filhas simultaneamente. Uma placa USRP popular é a Ettus Research N210 [Ettus Research b], que inclui conversores ADC e DAC, um FPGA, uma interface *Ethernet Gigabit* para se comunicar com o computador, e uma interface serial de alta velocidade para a integração com outras placas, permitindo o desenvolvimento de sistemas MIMO. O FPGA pode realizar a totalidade ou uma parte do processamento de sinal digital, permitindo, se for configurado, transferir o processamento para outros dispositivos de computação, tais como um computador, via Ethernet ou USB.

Visualizamos na Figura 1.4 a placa Ettus B210, que possui como principais características:

- É o primeiro dispositivo USRP totalmente integrado de dois canais com cobertura de RF contínua de 70 MHz a 6 GHz, *i.e.*, não há necessidade de instalação de placas-filhas;
- *Full-Duplex*, operação MIMO (2 Tx e 2 Rx) com até 56 MHz de largura de banda em tempo real (61.44MS/s quadratura);
- conectividade rápida com USB 3.0 através do controlador Cypress EZ-USB FX3;
- suporte ao GNU Radio e OpenBTS através de *drivers* de código-fonte aberto (UHD - *USRP Hardware Driver*);
- FPGA Xilinx Spartan 6 XC6SLX150;
- prototipagem utilizando o dispositivo interno AD9361 RFIC.

Este equipamento permite executar uma ampla gama de aplicações, incluindo: FM e transmissão de TV, celular, GPS, WiFi, ISM, ZigBee *etc*. A arquitetura do USRP B210 pode ser melhor compreendida observando a Figura 1.5. Temos como principais elementos as interfaces USB 3.0 e RF Frontend, o FPGA Xilinx Sparta 6 e o sistema de

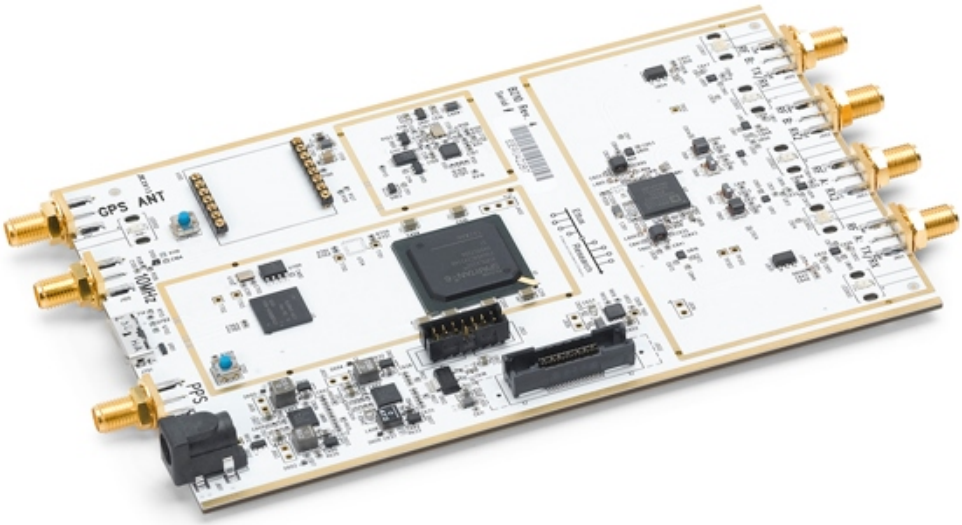

Figura 1.4. Placa USRP Ettus B210 [Research 2015]

clock. A placa é alimentada por uma fonte externa de energia DC de 6V [Research 2015]. Opcionalmente, podemos inserir um módulo GPSDO (*GPS-Disciplined Oscillator Kit*), apresentado na Figura 1.5, para aplicações que necessitem de temporização com precisão de nanosegundos.

1.2.2.2. SORA

SORA [Tan et al. 2009] é um pseudônimo para *Microsoft Research Software Radio*, que é a plataforma SDR desenvolvida pela Microsoft Research (MSR) Asia em Beijng. SORA combina o desempenho e a fidelidade da SDR baseada em *hardware* com a programação e flexibilidade de um processador de propósito geral (*Generic Purpose Processor* – GPP). SORA é uma plataforma SDR que permite o desenvolvimento de novas tecnologias em redes sem fio sem a necessidade de *hardware* específico, ou seja, usando apenas um computador pessoal. Assim, o *hardware* SORA possui somente um decodificador de banda base, e todo o processamento é feito por uma CPU x86. No entanto, o desenvolvimento de uma solução em PCs traz algumas dificuldades, tais como a necessidade de uma ligação de alta taxa de transferência para transferir o sinal digital para a memória principal.

Além disso, o processamento de sinal na camada física requer quantidades significativas de processamento, e a camada física e protocolos MAC exigem temporização rigorosa e baixa latência. SORA usa técnicas tanto de *hardware* quanto de *software* para superar estes desafios. Como uma solução para a alta taxa de transferência de dados, a MSR emprega uma placa de controle de rádio (*Radio Control Board* – RCB) para a transmissão e recepção de sinal, que é conectada ao PC por um barramento *PCI Express* de baixa latência. Com este barramento, o RCB pode suportar até 16.7Gbps (modo PCI-e 8x) com latências inferiores a um microssegundo. Esta capacidade satisfaz os requisitos de tempo de padrões de alta velocidade, como o IEEE 802.11g.

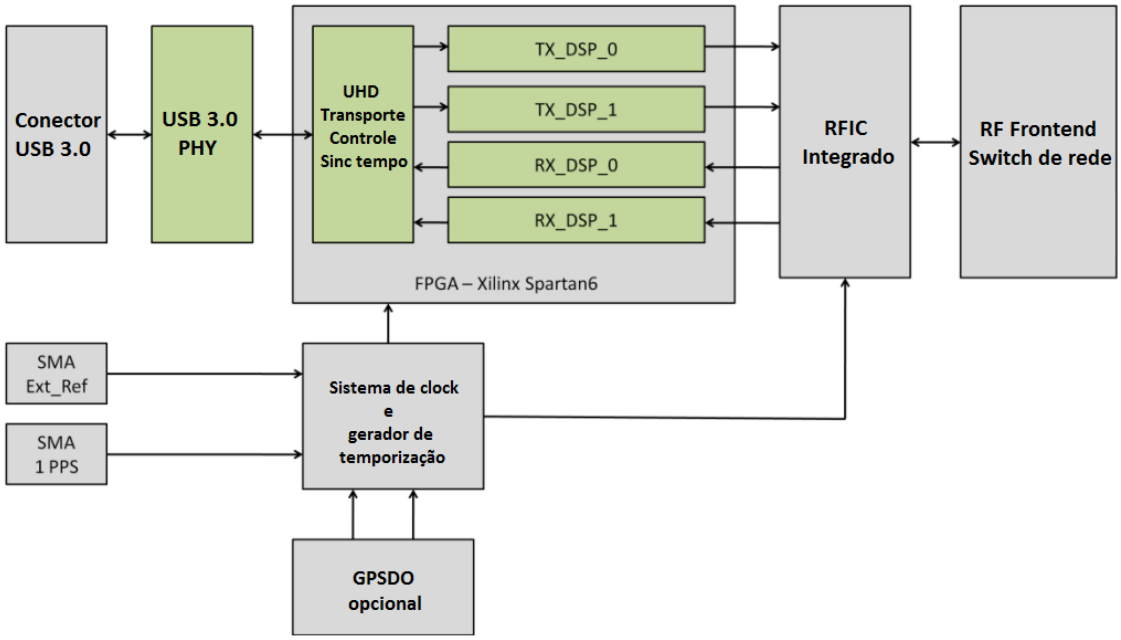

Figura 1.5. Esquema da arquitetura do USRP Ettus B210 [Research 2015]

Os requisitos de processamento exigidos por uma SDR são possíveis no SORA através do uso de instruções SIMD (*Single Instruction Multiple Data*) existentes nas CPUs x86, que aceleram o processamento de sinais na camada física. Além disso, o SORA requer um novo serviço de *kernel*, a dedicação de núcleos, que aloca um núcleo do processador exclusivamente para tarefas de SDR. Outro truque curioso da plataforma é a implementação de mensagens de confirmação. Como estas mensagens devem ser enviadas logo após a recepção do quadro de dados, elas são pré-computadas em paralelo à recepção dos dados, independentemente do quadro de dados ter sido decodificado corretamente.

1.2.2.3. SODA

SODA é uma implementação SDR de baixo consumo [Lin et al. 2006]. Ela foi desenvolvida como um processador de propósito especial, projetado para as operações mais comuns necessárias para a implementação de PHY e camadas MAC em *software*. Como conseqüência, SODA emprega várias unidades de processamento paralelo, com pouca ou quase nenhuma inter-comunicação.

O projeto do SODA foi baseado em uma análise cuidadosa do fluxo de dados e operações realizadas em um transceptor sem fio. Esta análise mostrou que as operações comuns do transceptor não exigem interação com outras operações em execução. A única comunicação entre elas ocorre quando uma determinada operação for concluída, e a sua saída deve ser encaminhada para a próxima operação, formando um *pipeline*. Este *pipeline* também influenciou o modelo de execução do SORA, pois cada tarefa é estaticamente atribuída a uma unidade de processamento. O número de unidades de processamento,

bem como o tamanho das instruções SIMD foi definida de acordo com simulações, que identificaram a quantidade de níveis de *pipeline* que minimizava o consumo de energia da arquitetura. Finalmente, a arquitetura também emprega um processador ARM para o processamento de protocolos da camada MAC, que requerem um conjunto de instruções mais genérico.

SODA foi empregada para implementar dois padrões importantes de redes sem fio: W-CDMA e 802.11a. Em ambos os casos, SODA apresentou vazão (*throughput*) suficiente para a boa execução dos protocolos. Por fim, os autores avaliaram o consumo de energia de SODA usando um sintetizador, mostrando que esta arquitetura poderia atender às restrições de energia de dispositivos móveis se produzido usando métodos de fabricação baseados na tecnologia de 90 nm ou 65 nm.

1.2.2.4. FLAVIA e soft-MACs

Existem diversas plataformas para desenvolvimento de protocolos MAC em *software*, que são conhecidas na literatura como *soft MACs*. Nestas plataformas, o programador possui uma camada física fixa, e o mesmo implementa somente a sub-camada MAC da camada de enlace. Os *soft MACs*, apesar de mais limitados que um SDR completo, são mais fáceis e amigáveis de programar. Um exemplo muito interessante de *soft MAC* é o FLAVIA, que será descrito a seguir.

O FLAVIA executa aplicações SDR em placas sem fio de baixo custo, permitindo o fácil desenvolvimento de novos protocolos MAC [Tinnirello et al. 2012]. A plataforma emprega um *firmware modificado*, que é carregado em transceptores que empregam um *chipset* bem conhecido e popular no mercado. O *firmware* implementa uma máquina de estados finitos baseada em eventos, onde os eventos são situações comuns em protocolos sem fio: início de um intervalo de contenção, colisão, recepção de um quadro de confirmação *etc*. O projeto FLAVIA está limitado ao protocolo MAC em 2.4GHz, uma vez que não é possível implementar novos módulos para processamento e modulação do sinal, ou mudar a frequência de operação.

No FLAVIA, os usuários podem criar novos protocolos sem fio utilizando os eventos pré-definidos. Assim, ele oferece uma linguagem de programação de propósito específico e de nível de abstração superior ao das outras plataformas. Ele é de fácil programação e possui uma interface gráfica básica, onde o usuário constrói o seu código rapidamente. A limitação da plataforma é a falta de extensibilidade, uma vez que apenas os estados e eventos fornecidos pelos autores são suportados. Além disso, não é possível criar programas mais complexos, uma vez que a plataforma tem um número limitado de registros e operações condicionais.

FLAVIA foi implementado em um *chipset* descontinuado, de forma que não é mais possível comprar placas para PCs. Entretanto, este *chipset* ainda pode ser encontrado em alguns roteadores IEEE 802.11g disponíveis no mercado por até R$300,00. Esses roteadores podem ser programados usando distribuições Linux baseadas em ARM, como OpenWRT [Ope]. O OpenFWWF (*Open Firmware for Wi-Fi*) [Gringoli and Nava] é um projeto *open source* gerado a partir de FLAVIA, que divulgou as especificações da memória e registradores usados no *firmware*, bem como o seu código fonte. Entretanto,

os desenvolvedores estão pouco ativos, e poucas atualizações foram realizadas no código depois que o projeto se separou do FLAVIA.

1.2.2.5. Outras plataformas

Além das plataformas citadas acima, existem muitas outras plataformas de pesquisa e comerciais de SDR. Uma lista mais completa pode ser encontrada na página Wikipedia de SDR[4]. Uma plataforma importante do ponto de vista da pesquisa é o WARP [Murphy et al. 2006][Amiri et al. 2007], pois é muito empregada nos EUA e em outros países na pesquisa experimental em redes de computadores e telecomunicações. Entretanto, as placas são muito custosas, partindo de USD 5.000 a unidade. Outra plataforma interessante é o *chip* RTL2832U. Este *chip* é muito empregado por placas de TV digital, mas hobbistas descobriram que o mesmo pode ser utilizado como uma ótima plataforma SDR de baixo custo, custando apenas 40 dólares a unidade.

1.2.2.6. Comparativo e evolução das plataformas

Como vimos nas plataformas estudadas, existe uma gama muito grande de arquiteturas e formas de programação. De um lado, possuímos plataformas em que todo o processamento é realizado em CPUs x86 ou ARM. Nestas, a curva de aprendizado é pequena, pois os programadores utilizam linguagens populares no mercado. Entretanto, a programação de protocolos com alta largura de banda requer diversos cuidados, como o uso de código de montagem em partes críticas de código e otimizações para a arquitetura.

Uma forma de aliviar esta limitação é o uso de arquiteturas que empregam FPGAs e DSPs para tarefas de baixa latência e alta vazão, e processadores genéricos para tarefas mais lentas, como protocolos MAC. Além disso, o uso de *hardware* para processamento da banda base diminui a largura de banda necessária no barramento entre a CPU e as FPGAs/DSPs. Um exemplo é o USRP, onde o usuário pode executar parte ou todo o seu código em FPGAs.

Finalmente, da mesma forma que o mercado de PCs desenvolveu as GPUs para o processamento gráfico e mais recentemente estas foram adotadas para processamento científico, propostas como o SODA partiram para o uso de processadores de propósito específico no SDR. O benefício de processadores de propósito específico é uma maior velocidade, menor área de *chip* e menor consumo de energia. Entretanto, a programação para estas plataformas é mais complexa, pois pode requerer compiladores e linguagens apropriados.

A Figura 1.6 apresenta uma comparação entre as arquiteturas discutidas. Os valores do eixo das ordenadas não são mostrados pois não foi feita uma comparação quantitativa, mas qualitativa, baseada na vivência dos autores com algumas das arquiteturas e na sua descrição técnica.

[4] `http://en.wikipedia.org/wiki/List_of_software-defined_radios`

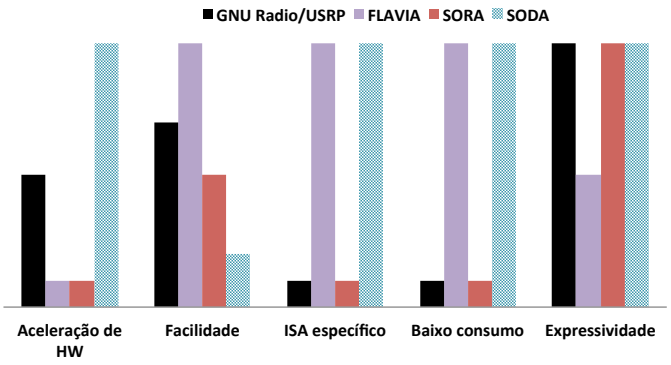

Figura 1.6. Comparação entre as arquiteturas discutidas.

1.3. Fundamentos de transmissão digital

Nesta seção será apresentada uma visão geral de vários conceitos-chave de transmissão digital de dados, iniciando com uma compreensão de como os dados binários podem ser usados para manipular as características físicas de uma onda eletromagnética, e em seguida discutimos as principais técnicas de modulação para rádios móveis.

1.3.1. Elementos fundamentais

Chama-se de transmissão de dados digitais a transferência física de dados (uma sequência de bits ou um sinal analógico digitalizado) ao longo de um canal de comunicação ponto-a-ponto ou ponto-a-multiponto. Exemplos desses canais são os fios de cobre, fibras ópticas, canais de comunicação sem fios, meios de armazenamento e barramentos de computadores [Clark 1983].

As transmissões modernas envolvem basicamente dois tipos básicos de comunicação, que podem ser sumarizados da seguinte forma: *broadcasting*, ou radiodifusão, que envolve o uso de um único transmissor e diversos receptores; e comunicações ponto-a-ponto, nos quais as comunicações são realizadas por meio de enlaces diretos entre um transmissor e um único receptor [Haykin and Moher 2006].

Enquanto a transmissão analógica é a transferência de um sinal analógico que varia continuamente ao longo de um canal analógico, comunicações digitais são a transferência de mensagens discretas ao longo de um canal digital ou analógico. As mensagens são representadas tanto por uma sequência de impulsos por meio de códigos de linha (transmissão de banda base), como por um conjunto limitado de diferentes formas de onda contínua (banda passante de transmissão). Este processo é conhecido como modulação digital. A modulação e a demodulação da banda passante correspondente é realizada por equipamento denominado *modem* (MOdulador-DEModulador). De acordo com a definição mais comum de sinal digital, os dois sinais de banda base e de banda passante representando fluxos de bit são considerados como transmissão digital, enquanto uma definição alternativa apenas considera como o sinal de banda base digital, e transmissão de banda passante dos dados digitais como uma forma de conversão digital-analógica [Sklar 2001].

Um transceptor digital é essencialmente responsável pela tradução entre uma sequên-

cia de dados digitais, representados por bits, em ondas eletromagnéticas com características físicas que representam unicamente esses bits. Algumas características físicas das ondas utilizadas para representar dados digitais incluem a amplitude, fase e frequência da onda, de forma que diferentes combinações de bits representam diferentes níveis de amplitude ou valores de fase ou de frequência, ou ainda por um alterações simultâneas de duas ou mais destas características da onda [Wyglinski and Pu 2013].

Contudo, em uma transmissão digital há muito mais a ser feito do que apenas realizar um mapeamento entre bits e formas de ondas, conforme ilustrado na Figura 1.7. Essa ilustração mostra uma representação genérica de um transceptor digital, visualizando os vários blocos funcionais que constituem um sistema de comunicação digital, como por exemplo, os blocos de *modulação* e *demodulação*, que representam o mapeamento entre bits e características das ondas eletromagnéticas. Adicionalmente, observa-se os blocos de *codificação de fonte* e *decodificação de fonte* que lidam com a remoção de informações redundantes dos dados binários; os blocos *codificação do canal* e *decodificação do canal*, que introduzem uma quantidade controlada de informação redundante para proteger a transmissão de erros potenciais; e os blocos *RF Front-end*, que trabalham na conversão das ondas bases para as frequências da portadora.

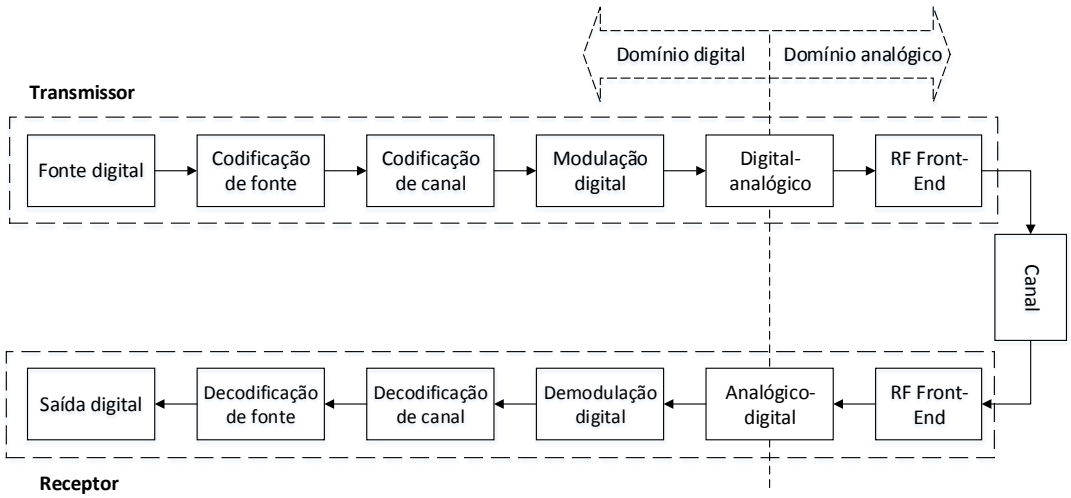

Figura 1.7. Representação genérica de um transceptor de comunicação digital.

Uma das principais razões para que um sistema de comunicação digital precise de todos esse blocos é devido ao *canal*. Se este fosse um meio ideal onde as ondas eletromagnéticas do transmissor fossem claramente enviadas e recebidas pelo receptor sem qualquer tipo de distorção, interferência ou ruídos, então o projeto de um sistema de comunicação digital poderia ser trivial. O que acontece na realidade é que o sinal ao chegar nos transmissores se encontra alterado, devido às distorções do meio e outras fontes de interferência e ruído. Isto pode potencialmente afetar a correta recepção da onda interceptada pelo receptor. Para evitar tais problemas é que utilizamos sistemas de correção e detecção de erros em diversos níveis da transmissão digital.

1.3.1.1. Codificação de fonte

Um dos principais objetivos de qualquer sistema de comunicação é realizar uma comunicação eficiente e confiável entre um transmissor e um receptor através de um meio. Dessa forma, o ideal seria remover toda a informação redundante de uma transmissão a fim de minimizar a quantidade de informação que necessita ser enviada através do canal, o que pode resultar em menor tempo para transmissão, menor uso de recursos computacionais ou de microprocessadores, e economia de energia elétrica. Em suma, diz-se que a *codificação de fonte* é um mecanismo projetado para remover as informações redundantes para facilitar comunicações mais eficientes [Wyglinski and Pu 2013].

O modo como esse mecanismo opera é tomando uma sequência de símbolos de origem u e mapeando-os para uma sequência correspondente de símbolos codificados v, sendo $v_i \in v$ de forma mais aleatória possível, e os componentes de v não são correlacionados [Wyglinski and Pu 2013]. Estas duas regras de seleção de símbolo são empregadas devido ao ruído e interferência: quanto mais diferentes forem os símbolos codificados, mais fácil é distingui-los um do outro, e assim mais fácil fica identificar um símbolo mesmo na presença de ruído.

1.3.1.2. Codificação de canal

Para proteger uma transmissão digital da possibilidade de uma informação ser corrompida, é necessário introduzir alguns níveis de redundância controlada para reverter os efeitos dos dados corrompidos. Consequentemente, a *codificação de canal* é projetada para corrigir os erros de transmissão em um canal introduzindo redundância controlada nos dados de transmissão. Vale salientar que essa redundância é especificamente projetada para combater os efeitos de erros de bit na transmissão (*i.e.*, a redundância possui uma estrutura específica que é conhecida tanto pelo emissor como pelo receptor) [Wyglinski and Pu 2013].

De modo geral, a codificação de canal opera da seguinte forma: cada vetor da saída da codificação de fonte de tamanho K, chamado v_l, onde $l = 1, 2, ..., 2^k$, é associado a uma única palavra-chave tal que o vetor $v_l = (101010...)$ esteja associado a uma única palavra-chave $c_l \in \mathbb{C}$ de tamanho N, onde $\mathbb{C}$ é um livro-chave (i.e., a taxa de código é igual a k/N). O mapeamento do vetor para uma entrada no livro chave segue uma função bijetora, ou seja, existe um mapeamento um para um e reversível entre a entrada e a saída da codificação. Além disso, a quantidade de bits da palavra-chave em relação ao tamanho do vetor, a razão k/N, indica a taxa de redundância da codificação de canal: quanto mais próxima de 1, menor será a quantidade de bits redundantes. Ou seja, para canais onde haja pouca interferência e ruído, podemos empregar menos redundância.

1.3.2. Técnicas de modulação

Chama-se de *modulação* o processo de codificação das informações a partir de uma fonte de mensagem de uma forma adequada para transmissão. O sinal da banda de passagem é chamado sinal *modulado*, e o sinal da mensagem na banda base é chamado sinal *modulante* [Rappaport 2001]. Podemos ter modulação variando-se a amplitude, a fase e a

frequência de uma portadora de alta frequência. Além disso, podemos combinar duas ou mais das características acima para modular a portadora, de forma a carregar ainda mais informação. Ao processo de extrair a mensagem de banda base da portadora de modo que ela possa ser processada e interpretada pelo receptor (também chamado de *sink*), dá-se o nome de *demodulação*.

Nesta subseção serão discutidas brevemente algumas técnicas de modulação como FM (Frequência modulada), AM (Amplitude modulada), PAM (*Pulse-amplitude modulation*), QAM (*Quadrature amplitude modulation*) e OFDM (*Orthogonal Frequency-Division Multiplexing*), utilizadas em diversos sistemas de comunicação móvel. Ainda que os sistemas digitais ofereçam amplos benefícios e já estejam sendo utilizados para substituir os sistemas analógicos convencionais, os sistemas análogicos serão brevemente discutidos pois ainda estão em utilização e por terem sidos empregados nos principais sistemas de comunicação de primeira geração.

1.3.2.1. Amplitude modulada - AM

Na modulação por amplitude, a amplitude de um sinal de portadora de alta frequência é variada em conformidade com a amplitude instantânea do sinal da mensagem modulante. A frequência e a fase da portadora são mantidas constantes. Matematicamente, é uma aplicação direta da propriedade de deslocamentos em frequências da transformada de Fourier, assim como da propriedade da convolução. Se $c(t) = A_c \cos(2\pi f_c t)$ é o sinal da portadora e $m(t)$ é o sinal da mensagem modulante, o sinal de AM pode ser representado como

$$s_{AM}(t) = A_c[1 + m(t)]\cos(2\pi f_c t) \tag{1}$$

AM é formalmente definida como um processo no qual a amplitude da onda portadora $c(t)$ é variada sobre um valor médio, de forma linear com o sinal de mensagem $m(t)$ [Haykin and Moher 2006].

O *índice de modulação k* de um sinal AM é definido como a razão entre a amplitude de pico do sinal da mensagem e a amplitude de pico do sinal da portadora. Para um sinal modulante senoidal $m(t) = (\frac{A_m}{A_c})\cos(2\pi f_m t)$, o índice de modulação é dado por

$$k = \frac{A_m}{A_c} \tag{2}$$

Este índice normalmente é expresso como uma porcentagem e é chamado de *modulação percentual*.

1.3.2.2. Frequência modulada - FM

A técnica de modulação analógica mais popular utilizada nos sistemas de rádio móvel é a frequência modulada (FM) [Rappaport 2001]. A modulação FM transmite informações através de uma portadora variando a sua frequência instantânea, de acordo com o sinal

modulante da mensagem. Salienta-se que a amplitude é mantida constante nesta técnica. Dessa forma, os sinais de FM tem toda a informação na fase ou na frequência da portadora, o que oferece uma melhoria não-linear e muito rápida na qualidade da recepção quando um certo nível mínimo do sinal recebido, chamado *patamar FM*, é alcançado.

FM oferece diversas vantagens em relação à amplitude modulada (AM), tornando-a uma melhor opção para muitas aplicações de rádio móvel, como uma melhor imunidade ao ruído em comparação com a amplitude modulada. Essa menor suscetibilidade ao ruído atmosférico e de impulso se deve ao fato desses ruídos afetarem a amplitude, causando rápidas flutuações nestes níveis do sinal recebido, contudo, como as variações de amplitude não transportam informações no sinal FM, o ruído intermitente não afeta o desempenho do sistema FM (desde que o sinal recebido esteja acima do patamar de FM). Além disso, os receptores FM apresentam uma característica conhecida como *efeito de captura*, resultado direto da melhoria rápida e não-linear na qualidade da recepção para um aumento na potência recebida. Esse efeito ocorre da seguinte maneira: se existirem dois ou mais sinais de FM emitidos na mesma frequência e ambos estiverem disponíveis ao receptor FM, este irá responder (demodular) ao sinal de maior potência e ignorar os menores (os restantes).

Algumas desvantagens dos sistemas FM são que eles exigem uma ampla faixa de frequência no meio de transmissão (geralmente, várias vezes a necessária para AM); e os equipamentos transmissor e receptor são mais complexos do que aqueles utilizados por AM.

1.3.2.3. Pulse-amplitude modulation - PAM

Nas modulações de ondas contínuas estudadas anteriormente, um parâmetro de uma onda portadora senoidal é variado continuamente de acordo com o sinal de mensagem. Isso está em contraste direto com modulação por pulso. Neste esquema de modulação, alguns parâmetros de um trem de pulso são variados de acordo com o sinal de mensagem. Neste contexto, pode-se distinguir duas famílias de modulação por pulso: *modulação por pulso analógica* e *modulação por pulso digital*, dependendo de como a modulação é realizada. Na modulação por pulso analógica, um trem de pulsos periódicos é usado como a onda portadora, e uma característica de cada pulso (por exemplo, amplitude, duração ou posição) é variada de maneira contínua, em conformidade com a amostragem correspondente do sinal de mensagem [Haykin and Moher 2006].

Dessa forma, na modulação por pulso analógica, a informação é transmitida basicamente em forma analógica, mas a transmissão é *carregada* em intervalos de tempo discretos. Na modulação por pulsos digitais, por outro lado, o sinal de mensagem é representado por uma forma de onda que é discreta tanto em tempo como em amplitude, permitindo assim a sua transmissão na forma digital como uma sequência de pulsos codificados. A utilização de pulsos codificados para a transmissão de sinais portadores de informação para analógicos representa um ingrediente básico na aplicação das comunicações digitais. Esta seção pode ser vista, portanto, como a transição das comunicações analógicas para as digitais no nosso estudo dos fundamentos da transmissão digital [Wyglinski and Pu 2013].

1.3.2.4. Quadrature amplitude modulation - QAM

Visando aumentar a transmissão de bits por segundo, foi criada a técnica QAM, método para codificar dados digitais em um sinal analógico através de modulação em que duas componentes diferentes são combinadas em um único sinal através de modulação ortogonal destas duas componentes, evitando assim a interferência; daí o termo *quadratura*. A técnica empregada consiste na combinação da modulação por amplitude (AM) com modulação por fase (PSK, do inglês *Phase-shift keying*) para criar uma constelação de pontos de sinal, cada qual representando uma combinação exclusiva de bits. É bastante utilizada em TV digital e outros sistemas que necessitam de alta taxa de transferência de informação.

A Figura 1.8 mostra o diagrama de constelação da QAM 16-ária, sendo Q e I representações dos sinais modulantes (fase e amplitude). A constelação consiste em uma trama quadrada de pontos de sinal.

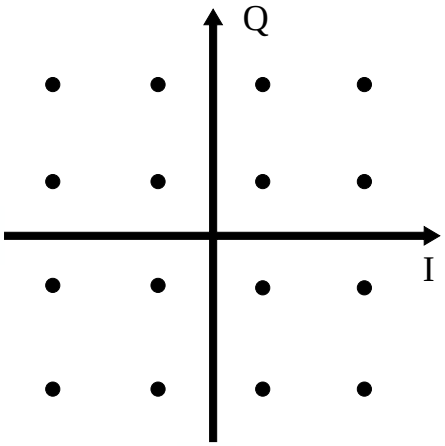

Figura 1.8. Diagrama de constelação de um conjunto de sinais QAM M-ário (M=16)

A forma geral de um sinal QAM-M-ário pode ser definida como

$$S_i(t) = \sqrt{\frac{2E_min}{T_s}} a_i cos(2\pi f_c t) + \sqrt{\frac{2E_min}{T_s}} b_i sen(2\pi f_c t) \tag{3}$$

$$0 \leq t \leq T; i = 1,2,...,M$$

onde E_{min} é a energia do sinal com a amplitude mais baixa, e a_i e b_i são um par de inteiros independentes escolhidos de acordo com o local do ponto de sinal. Observa-se que a QAM M-ária não possui energia constante por símbolo nem a distância constante entre os estados de símbolos possíveis.

1.3.2.5. Orthogonal Frequency-Division Multiplexing - OFDM

Proposto em 1968, a primeira aplicação da técnica OFDM foi apenas em 1985 [Cimini 1985] e surgiu como uma evolução da técnica convencional de Multiplexação por Divisão de Frequência - FDM (*Frequency Division Multiplexing*) onde, ao invés de utilizar-se bandas de guarda para a separação das subportadoras na recepção do sinal, trabalha-se com uma particular sobreposição espectral de subportadoras [Pinto and de Albuquerque 2002].

A técnica OFDM consiste na transmissão paralela de dados em diversas subportadoras com modulação QAM ou PSK e taxas de transmissão por subportadora tão baixas quanto maior o número destas empregadas. A redução na taxa de transmissão aumenta tanto a duração dos símbolos presentes em cada subportadora, como a diminuição na sensibilidade à seletividade em frequência. A geração direta e a demodulação do sinal OFDM, em princípio, requerem conjuntos de osciladores coerentes, resultando em uma implementação complexa e cara; entretanto, esses processos de modulação e demodulação podem ser executados de forma mais simples utilizando-se, respectivamente, algoritmos IFFT (*Inverse Fast Fourier Transform*) e FFT (*Fast Fourier Transform*) [Pinto and de Albuquerque 2002].

Como principais vantagens do uso do OFDM tem-se a elevada eficiência espectral, imunidade contra multi-percursos e filtragem de ruído simples, resultando, por exemplo, em alta velocidade de transmissão de dados [Haykin and Moher 2006].

1.4. Introdução ao GNU Radio

Nesta seção serão apresentados exemplos práticos com o GNU Radio a fim de permitir que o participante desenvolva sua própria pesquisa. As principais aplicações demonstrativas, como demodulação de sinais FM, comunicação entre nós sensores e desenvolvimento de um novo módulo, serão descritas passo a passo, e possuem nível de complexidade incremental. Apresentamos um passo a passo para suas corretas execuções. O objetivo deste mini-curso é mostrar rapidamente as funcionalidades e capacidades do GNU Radio, assim leitores que tenham interesse de compreender a fundo as interfaces gráficas e a criação de modificação de blocos devem procurar tutoriais na Internet. Existe uma grande comunidade de tutoriais e guias na Web e no Youtube [Radio 2015a], [Radio 2015c] [Balint 2015] .

Em todas as aplicações apresentadas a seguir são utilizados rádios USRP da Ettus, modelos B100 e B210, associados com o aplicativo GNU Radio. Finalmente, no término da seção iremos apresentar dicas práticas no uso do GNU Radio e USRP, e discutir outras plataformas de software compatíveis com o hardware USRP que tem sido empregadas na pesquisa em redes de computadores.

1.4.1. GNU Radio

O GNU Radio é um conjunto de ferramentas de código aberto que fornece um ambiente de desenvolvimento e blocos de processamento para implementar rádios definidos por *software*. Ele se integra com placas USRP através de *drivers* [GNU Radio 2013]. As aplicações GNU Radio são desenvolvidas utilizando Python, que é usado para conectar os blocos básicos de processamento. Esses blocos são desenvolvidos em C++ por mo-

tivos de desempenho. Há também Ambientes de Desenvolvimento Integrado, tais como o GNU Radio Companion (GRC), que permite o desenvolvimento de aplicações usando uma interface gráfica de usuário.

Cada bloco no GNU Radio tem associadas entradas e saídas que podem ser ligadas a vários tipos de fluxos de dados. A ligação de diferentes blocos cria um fluxo que implementa os passos de processamento de um dado sistema de comunicações. Além das entradas e saídas, cada bloco tem um conjunto específico de parâmetros que definem o seu comportamento. O GNU Radio fornece centenas de blocos para os usos mais comuns, tais como operações matemáticas, conversão de tipo de dados, modulação e demodulação, entre outros. Além disso, novos blocos podem ser criados com base nas necessidades do usuário. Um ponto que deve ser destacado é o fato de que o GNU Radio é um projeto *open source*, permitindo a alteração do código de blocos existentes se necessário. O GNU Radio tem uma comunidade de apoio muito ativa, tornando ainda mais fácil de aprender e desenvolver usando a plataforma. No entanto, o suporte para alguns protocolos e técnicas mais avanças de modulação é ainda muito limitado, como é o caso de WiFi [Bloessl et al. 2013b], Bluetooth, e ZigBee, como veremos rapidamente nos estudos de caso apresentados a seguir.

O *software* de GNU Radio pode ser utilizado sem placas SDR, por exemplo para processar sinais de um arquivo no computador, ou empregando uma plataforma SDR compatível. Para tanto, a plataforma SDR deve fornecer um componente GNU Radio chamado UHD (USRP *Hardware Driver*), que é um *driver* para o *hardware*. Por exemplo, as placas USRP da Ettus, mencionadas anteriormente, possuem drivers UHD para todas as suas versões.

A instalação do GNU Radio pode ocorrer por pelo menos três formas. Descrevemos brevemente cada uma delas:

- ***Script build-gnuradio***. Caso haja interesse por uma instalação que utilize o código-fonte sempre atualizado e que solucione dependências automaticamente, é preferível instalar o GNU Radio a partir dos arquivos fonte. Para isso, usuários do Fedora e Ubuntu contam com um *script* que realiza a instalação completamente, sendo recomendado para a maioria dos usuários. Trata-se de um *script* fornecido por Marcus Leech [Leech 2015] para sistemas Fedora e Ubuntu recentes. Com uma janela de terminal aberta, siga ao diretório que você deseja que os arquivos fonte sejam armazenados e execute o comando:

  ```
  $wget http://www.sbrac.org/files/build-gnuradio
  chmod a+x
  ./build-gnuradio.
  ```

  Esse procedimento baixa o instalador (*build-gnuradio*) e o torna executável, baixando e instalando também todas as dependências, o UHD e o sistema GNU Radio via Git; assim, automaticamente será instalada a versão mais recente do *master branch*. O procedimento segue executando o *make*, e instalando o sistema. Salienta-se que muito disso é realizado de forma transparente ao usuário, logo, é provável

que nenhuma novidade surja na tela por alguns minutos. Em muitos casos, simplesmente executando o *script*, ele fará tudo o que for necessário para obter e deixar o GNU Radio executando a partir dos arquivos fonte, mantendo-os no seu disco disponíveis para futuras modificações. Dessa forma, ele combina a flexibilidade de instalar a partir dos arquivos fonte com a facilidade de usar binários.

- **Binários pré-compilados**. Dessa forma a instalação é prática, mas nem sempre é instalada a versão mais recente. Também é comum usuários terem problemas com dependências não resolvidas corretamente. O processo de instalação no Ubuntu é simples, basta executar no terminal `$ apt-get install gnuradio`. Na distribuição Fedora, execute `$ yum install gnuradio`.

- **PyBOMBS (*Python Build Overlay Managed Bundle System*)**. Trata-se de uma nova forma de instalar o sistema GNU Radio e outros módulos, inclusive aqueles *out-of-tree*, ou seja, aqueles que não fazem parte dos projetos nativos do GNU Radio. No terminal basta executar:

  ```
  git clone https://github.com/pybombs/pybombs.git
  cd pybombs
  ./app_store.py
  ```

  Dessa forma, após a conclusão da instalação, será aberta uma interface gráfica onde é possível escolher o que se deseja instalar. Com o PyBOMBS instalado na sua máquina, sua execução é possível a partir do comando *./app_store.py* digitando a partir do diretório onde o mesmo está instalado.

Mais informações sobre esses processos de instalação, bem como outros modos, podem ser consultadas em [Radio 2015b].

1.4.2. GNU Radio Companion

GNU Radio Companion (GRC) é uma ferramenta gráfica livre para criar protótipos de sistemas de comunicação rápidos e funcionais a partir de diagramas de fluxos de sinais elétricos, permitindo gerar o respectivo código fonte desses fluxos. Programas no GNU Radio não precisam necessariamente serem feitos no GRC, entretanto ele permite uma prototipagem rápida e facilita o entendimento dos programas sendo desenvolvidos. Ele possui facilidades como sistemas para o desenvolvimento rápido de interfaces gráficas e análise estática do diagrama de módulos para encontrar erros simples de configuração. Nesta subseção vamos começar a explorar o GRC.

Para executar o ambiente GRC, após instalado o GNU Radio, abra um terminal e execute o comando `$ gnuradio-companion`. Inicialmente, temos que entender a posição dos elementos na interface. Há quatro partes: biblioteca (*library*), barra de tarefas (*toolbar*), terminal, e área de trabalho (*workspace*). Essas partes estão sinalizadas na Figura 1.9, respectivamente, com os números 1, 2, 3 e 4. A *Biblioteca* contém os diferentes blocos instalados no GRC. Nela encontramos blocos que estão pré-instalados no GNU Radio e blocos que estão instalados no sistema. No começo pode parecer difícil encontrar os blocos, mas há algumas praticidades: por exemplo, se quisermos gerar

uma forma de onda, em qual categoria devemos procurar? Vemos que há uma categoria *Waveform Generators*. E se quisermos exibir os nossos dados? Devemos inserir um *sink*, no entanto, procurando através da lista não encontramos uma categoria *sink*. A solução é usar o recurso de pesquisa clicando no ícone da lupa ou através do atalho *Ctrl + F* e, em seguida, começar a digitar uma palavra-chave para identificar o bloco. Vemos uma caixa branca aparecer no topo da Biblioteca com um cursor. Se digitamos *sink*, podemos encontrar todos os blocos que contêm as palavras *sink* e as categorias que vai encontrar em cada bloco.

O terminal é útil na compilação dos programas, pois apresenta os erros gerados. Além disso, podemos acompanhar a carga do programa na USRP e durante a execução podemos ver as saídas textuais do mesmo.

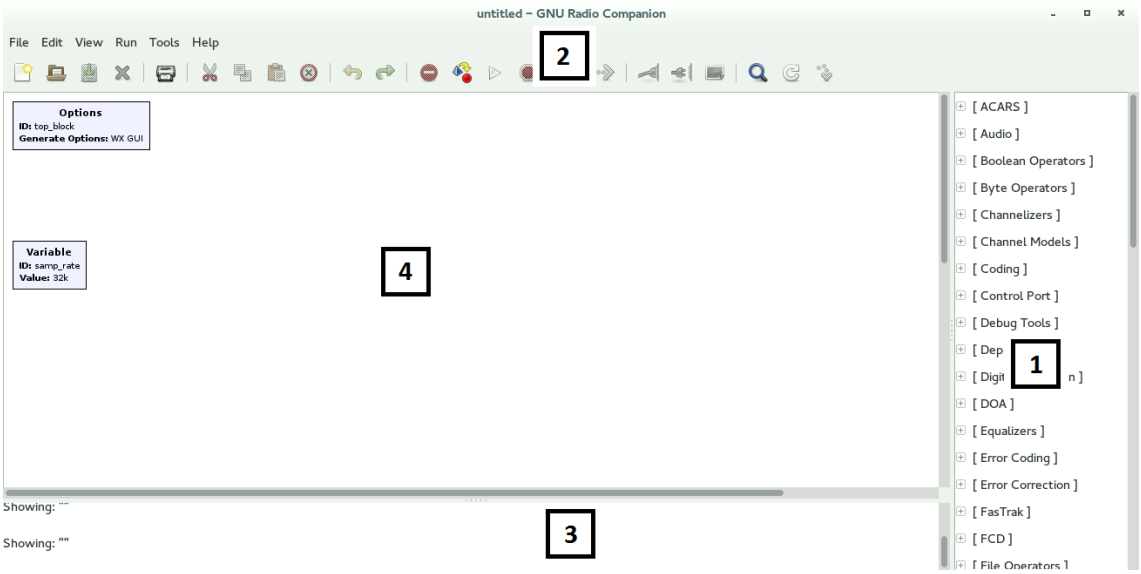

Figura 1.9. *Interface* do ambiente GNU Radio Companion

Uma aplicação é feita no GRC arrastando blocos da biblioteca para a área de trabalho, e realizando as conexões e configurações necessárias. Podemos modificar as propriedades de um bloco através de um duplo clique sobre o mesmo. As conexões entre blocos que permitem enlaces, como por exemplo os blocos *A* e *B*, são realizadas da seguinte forma: *clicamos* com o *mouse* na porta *out* do bloco *A* e em seguida na porta *in* do bloco *B*. Para remover uma conexão entre dois blocos, basta selecioná-lo e pressionar *Delete*.

Novos blocos podem ser adicionados por meio de importações de módulos, conforme faremos nas aplicações adiante, ou podem ser criados. Na subseção 1.4.6 apresentamos detalhadamente como criar um novo bloco.

Os tipos de dados no GRC podem ser visualizados no menu *Help* > *Types*. Realizando essa sequência, serão exibidos os tipos de dados possíveis com as respectivas cores associadas. Estas mesmas cores serão utilizadas nos fluxogramas e blocos à medida que se definem os tipos de dados. Para alterarmos o tipo de dados em um determinado bloco,

devemos ir nas propriedades do mesmo, normalmente em um campo identificado com *Type* ou *Output Type*, e selecionar o tipo de dado. Devemos ficar atento, contudo, com as compatibilidades de dados entre os blocos conectados, ou seja, o tipo de dados de saída de um bloco *A* deve ser do mesmo tipo de dados da entrada do bloco *B*, ao qual *A* se conecta. Caso haja incompatibilidade de tipo entre blocos, a conexão entre os mesmos será automaticamente sinalizada com uma seta vermelha, facilitando ao usuário perceber que naquela conexão está ocorrendo algum conflito entre os tipos de dados envolvidos. Para resolver, deve-se alterar o tipo em um dos blocos adequadamente. As posições dos blocos, informações de parâmetros e variáveis são salvas em um arquivo XML (*eXtensible Markup Language*) com extensão *.grc*, contendo basicamente o nome do bloco, a função a ser chamada para instanciar o bloco, parâmetros do bloco e seus tipos, e quantidade e tipos das portas de entrada e de saída.

O GNU Radio possui diversos tipos de blocos, por exemplo operações lógicas e matemáticas, filtros, moderadores e demoduladores, blocos de entrada e saída (chamados de *sources* e *sinks*), dentre outros. Entre eles, os blocos de *sink* e *source* são importantes. Todas as aplicações terão entradas e saídas, que são definidas pelos *sinks* e *sources*, respectivamente. Alguns exemplos de sinks e sources são arquivos binários, arquivos de som (por exemplo, WAV), ou placas USRP. Por exemplo, se queremos criar um receptor FM que grava a onda em um arquivo WAV, precisamos de um *source* que é uma placa USRP, e um *sink* que é um bloco que grava um arquivo WAV no disco, tal como o *Wav File sink*. Outros exemplos são arquivos de captura do Wireshark, fontes de seniores com frequência constante, ou *Null Sinks* e *Null sources*. Estes são interessantes em aplicações em que não queremos salvar dados mas somente transmitir, por exemplo.

Podemos ter mais de um *sink*, como iremos mostrar no exemplo de demodulação de sinais FM. Além disso, podemos ter uma aplicação com dois fluxos separados de blocos, um para a transmissão e outro para a recepção. Um exemplo simples está apresentado na aplicação de *walkie talkie* representada na Figura 1.10:

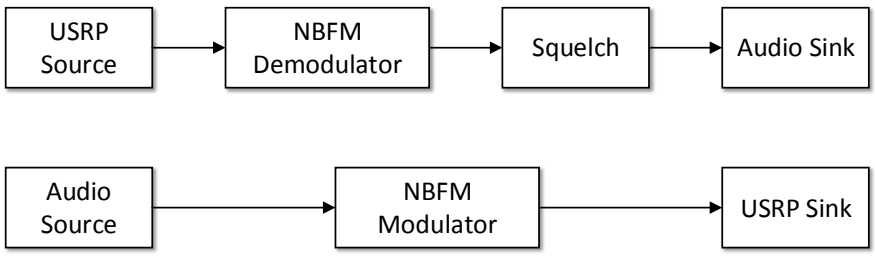

Figura 1.10. Exemplo de dois fluxos separados de blocos

Esta figura consiste de dois gráficos de fluxo separados, ambos rodando em paralelo. Um deles está relacionado com o caminho de Tx, e o outro com o caminho de Rx. Salienta-se que neste tipo de aplicação, é necessário um código extra (além dos gráficos de fluxo exibidos) para *silenciar* um caminho, enquanto o outro está ativo. Ambos os gráficos de fluxo exigem ainda, pelo menos, um *source* e um *sink*, cada. Uma aplicação nativa ao GNU Radio que realiza algo semelhante pode ser encontrada no ca-

minho *gr-uhd/examples/python/usrp_nbfm_ptt.py*. É importante frisar que fluxogramas que são completamente simulação (*i.e.*, sem blocos de áudio ou USRP) vão consumir 100% da CPU quando executados, e os elementos da GUI ficarão sem resposta. Estes gráficos de fluxos devem incluir um bloco *Misc* > *Throttle* para aumentar a taxa de dados de streaming. Somente precisamos de um único bloco acelerador (*Throttle*) em todo o fluxograma, mesmo se tivermos múltiplas fontes e sorvedouros(*sources/sinks*). Podemos pensar o bloco *Throttle* como um limite de velocidade; por outro lado, utilizando um *hardware* (USRP) há a imposição física de uma restrição à transferência, portanto, não é necessário nenhum bloco *Throttle*.

As variáveis no GRC mapeiam nomes simbólicos para valores, podendo definir uma constante global ou uma variável que pode ser utilizada em conjunto com a interface gráfica para controlar um sistema em execução. O bloco *Variable* é a forma mais básica de usar uma variável no GRC. O parâmetro ID do bloco é o *nome simbólico*. Este deve ser alfa-numérico (permitidos *underscores*) e começar com uma letra. Para usar a variável, basta digitar o nome simbólico de uma variável em um parâmetro de qualquer outro bloco.

Podemos ainda verificar se há erros em algum trecho de todo o fluxo no ambiente através do menu *View* > *Flowgraph Errors*, ou através do botão circular vermelho, localizado na barra de tarefas, que exibe o texto *View flow graph errors* ao se posicionar o ponteiro do mouse acima. Uma vez acionada essa funcionalidade será exibida uma mensagem descrevendo brevemente a quantidade de erros e o tipo de erro.

Além de blocos relativos ao processamento de sinais, temos também blocos que permitem a construção de gráficos e interfaces. Eles são úteis para construirmos uma interface gráfica, onde podemos modificar parâmetros dos blocos ou visualizar os sinais recebidos e transmitidos em tempo real. Temos objetos que representam os elementos mais comuns de interface gráficas, como rótulos de texto, botões, abas *etc*. O GNU Radio usa a biblioteca PyQt para a construção das interfaces gráficas.

O GRC manuseia arquivos em Python, assim, os fluxos gráficos serão compilados para essa linguagem e depois, se necessário, será transmitida uma imagem ao USRP. A compilação pode ser realizada através do menu *Run* > *Generate*, ou pressionando a tecla F5, ou ainda através do botão localizado na barra de tarefas, ao lado do botão *View flow graph errors*, representado por um ícone de uma pequena pirâmide, uma seta e um círculo, exibindo o texto *Generate the flow graph* ao posicionar o ponteiro sobre o mesmo. Se o fluxograma não estiver salvo, ao tentar compilar será solicitado que salve o fluxo em um arquivo com extensão *.grc*. Em seguida, podemos executar esse arquivo Python recém compilado acessando o menu *Run* > *Execute*, ou pressionando a tecla F6, ou ainda por meio do botão *Execute the flow graph*, identificado por um ícone triangular localizado ao lado do botão *Generate the flow graph*. Se o fluxo a ser executado envolver uso de *hardware* externo (*e.g.* USRP), uma imagem será transmitida ao equipamento; caso contrário, a saída dependerá do fluxo compilado, podendo ser apenas uma saída gráfica, áudio ou outras opções de saída.

O exemplo a seguir possibilitará uma melhor compreensão sobre o funcionamento do GRC. Mais informações sobre o GRC também podem ser encontradas em [Radio 2015a]. Existem diversos vídeos no Youtube e tutoriais, tanto no site do GNU Radio quanto em outros sites, que explicam em mais detalhes as funcionalidades do GRC.

Neste tutorial iremos discutir uma aplicação simples, que é um receptor FM, e depois iremos focar em aplicações mais relevantes para a pesquisa em redes de computadores.

1.4.3. Demodulação de Sinais FM com GNU Radio

Neste exemplo usamos o equipamento USRP B210 para construir um receptor de FM simples. O objetivo é ensinar alguns conceitos básicos de DSP e RF, incluindo: filtragem, demodulação e conversão da taxa de amostragem; mostrar como criar aplicações gráficas com GNU Radio Companion (GRC); e ilustrar a simplicidade das ferramentas de *software* que podem ser utilizadas com a família de produtos USRP.

O primeiro passo para a construção de uma aplicação do GNU Radio é identificar se o *hardware* possui o suporte adequado, ou seja, se temos uma placa filha com suporte à largura de banda e frequência empregadas, e se temos antenas com tamanho e ganho apropriados. Neste tutorial escolhemos a placa B210 porque ela suporta frequências de 70MHz a 6GHz, bem como a largura de banda das rádios FM. Já a largura de banda pode ser crítica caso queiramos implementar padrões de alta taxa de transmissão. Por exemplo, não é possível implementar um receptor IEEE 802.11ac na B210, porque ela suporta canais de até 56MHz de largura, e o 802.11ac suporta canais com até 80MHz de largura.

Nesta aplicação FM, o fluxograma pode ser melhor compreendido através da Figura esquemática 1.11.

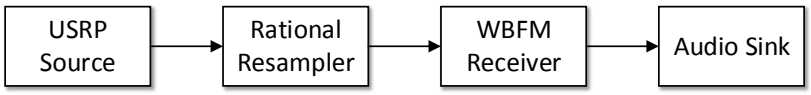

Figura 1.11. Esquema do fluxograma de demodulação FM

Precisaremos dos seguintes blocos, todos acessíveis pela biblioteca do GRC:

- USRP Source
- Low Pass Filter
- WBFM Receive
- Multiply Constant
- Rational Resampler
- Audio Sink

E dos seguintes blocos auxiliares:

- WX GUI Slider

- WX GUI Text Box
- WX GUI FFT Sink

No ambiente do GRC, inserimos o bloco *UHD:USRP Source* e alteramos o seu parâmetro *Samp Rate* para o valor *samp_rate*, assim ele ficará associado à variável *samp_rate* já definida em outro bloco de variável (criado por padrão no GRC). O bloco *UHD:USRP Source* permite o acesso às amostras do USRP. Em seguida associamos o parâmetro *Ch0: Center Freq* à variável *freq* para podermos controlar a frequência através da interface gráfica. Por último, definimos o ganho com o valor *15* alterando o parâmetro *Ch0: Gain*. O parâmetro *Antenna* vai depender do *hardware* que está sendo utilizado, neste caso utilizamos *TX/RX*.

No bloco da variável *samp_rate* (criado pelo GRC automaticamente) especificamos o valor de *5MHz* (*5e6*). Em seguida, adicionamos um novo bloco *WX GUI Text Box* para especificarmos a variável da frequência. No parâmetro *ID* digitamos *freq* e no campo *Default value* digitamos *105.7e6*, representando a frequência da estação FM que queremos sintonizar, neste caso, *105.7MHz*. Incluimos o componente de interface gráfica *WX GUI FFT Sink* para visualizarmos o espectro da frequência, conectando-o à saída do bloco *UHD: USRP Source*. Mudamos o parâmetro *Baseband Freq* para receber o valor da variável *freq*. Por fim definimos o parâmetro *Notebook* para *notebook_0,0*. Dessa forma, especificamos em qual aba da interface gráfica, que será definida mais adiante, será exibido o espectro.

Certamente há muitos ruídos no sinal que será recebido, logo devemos inserir um filtro para eliminar as freqüências além dos limites do intervalo especificado. Você pode tentar ajustar os parâmetros nesse ponto para tentar melhorar a qualidade do áudio e eliminar outras interferências. Assim, adicionamos um filtro "passa-baixa" (bloco *Low Pass Filter*). Definimos o parâmetro *Cutoff Freq* para e *100e3*, o *Transition Width* para *10e3* e o parâmetro *Decimation* para *20*. Conectamos a saída do bloco *UHD: USRP Source* na entrada do filtro *Low Pass*.

Neste ponto, a configuração de nosso penúltimo bloco principal é bastante simples. Precisamos adicionar o componente que decodifica o fluxo de entrada para o áudio real que foi codificado na corrente e os bits adequados para o envio à sua placa de som. Dessa forma, adicionamos o componente *WBFM Receive* e definimos o parâmetro *Quadrature Rate* para *250e3* e *Audio Decimation* para *1*. Conectamos a saída do filtro *Low Pass* à entrada do *WBFM Receive*. Como estamos recebendo, a partir do bloco *WBFM Receive*, a uma frequência de *250KHz*, precisamos convertê-la para a nossa taxa de amostragem de áudio de saída. Para isso, inserimos o componente *Rational Resampler* e especificamos ao parâmetro *Interpolation* o valor *250* e *Decimation* para *96*. Conectamos a saída do *WBFM Receive* à entrada do *Rational Resampler*. Em seguida, inserimos o componente *Audio Sink* que permite a escuta das amostras. Conectamos a saída do *Rational Resampler* à entrada do *Audio Sink*. Mudamos o parâmetro *Sample Rate* para *96000*.

Em seguida adicionamos o componente *Wav File Sink* para gravarmos as amostras em um arquivo de áudio. Definimos o parâmetro *File* para *fm_record* e *Sample rate* para *96000*. Conectamos a saída do *Rational Resampler* à entrada do *Wav File Sink*. Inserimos mais um componente *WX GUI FFT Sink* para visualizarmos o espectro do

áudio demodulado. Definimos o parâmetro *Sample Rate* para *250e3* e *Notebook* para *notebook_0,1*. Conectamos o *WBFM Receive* na entrada do *WX GUI FFT Sink*.

Por fim, adicionamos o componente *WX Gui Notebook* para criarmos uma interface gráfica com abas e definimos o parâmetro *labels* para *['RF Spectrum', 'Demod Spectrum']*.

Essas etapas devem acarretar em um fluxograma semelhante ao exibido na Figura 1.12. Após a compilação e execução desse fluxograma em um USRP, será possível sintonizar a frequência especificada com saída de áudio em tempo real e o mesmo salvo em arquivo na pasta de execução. A qualidade sonora dependerá de condições diversas, como do tipo da antena e da potência do sinal FM no ambiente.

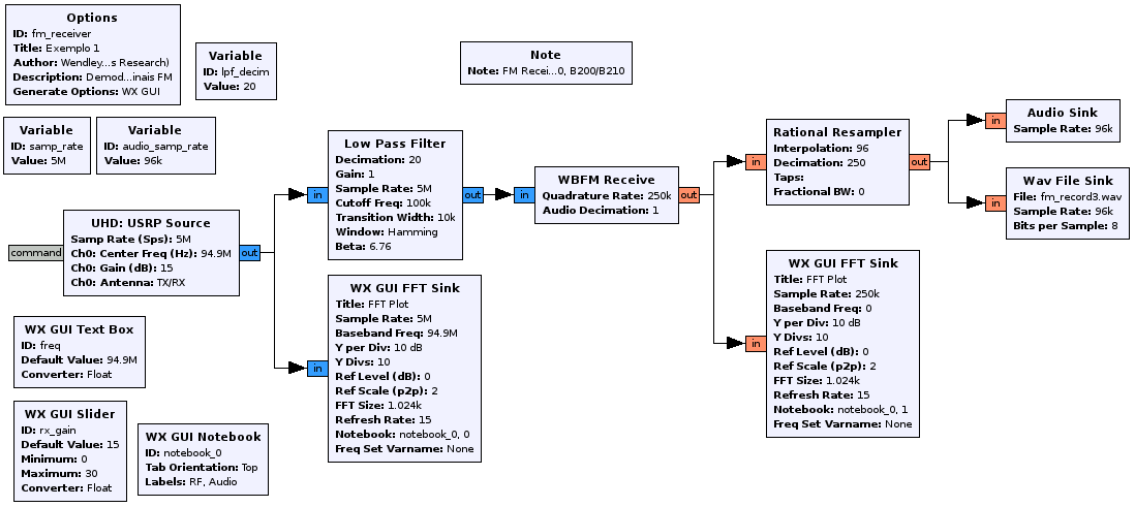

Figura 1.12. Fluxograma de Demodulação FM.

1.4.4. Modulação QPSK

PSK (*Phase shift keying*) é uma forma de modulação em que o sinal é codificado utilizando a fase da onda portadora. Uma das formas de fazer isso é deslocando a fase em 180º quando houver uma transição de bit 1 para 0 ou de bit 0 para 1, esta forma é chamada BPSK (*Binary Phase shift keying*), nesse sistema, quando não há transição, a portadora continua a transmitir com a mesma fase.

QPSK (*Quadrature phase shift keying*) é uma forma de PSK onde é utilizada uma quantidade maior de deslocamentos da fase da onda portadora para codificação do sinal. Neste caso, são utilizados 4 deslocamentos, assim, é possível transmitir 2 bits por símbolo. A fase da onda portadora terá 4 valores distintos e cada valor representará um dibit, por exemplo, (45º = 00), (135º = 01), (225º = 11) e (315º = 10) [Tenenbaum and Wetherall 2011].

No GNU Radio, para apresentar o exemplo de modulação QPSK, utilizamos os seguintes blocos:

- Options

- Random Source
- PSK Mod
- Noise Source
- Add
- Multiply Const
- Add
- UHD: USRP sink
- QT GUI Constellation sink

Inicialmente configuramos o bloco *Options*. Colocamos o valor do parâmetro *Generate Options* para *QT GUI*, em seguida precisaremos utilizar 3 blocos do tipo *Variable*, um para definir o valor do *sample rate*, outro para definir o valor da frequência e um outro para definir o valor do ganho. Assim, os parâmetros do primeiro bloco *Variable* terão os valores *ID*: *sample_rate*, *value*: *320e3*; os parâmetros do primeiro bloco terão os valores *ID*:freq, *value*: *900e6*; e para o último bloco, os valores serão *ID*:*gain*, *value*:*30*.

Feitas essas configurações iniciais, continuamos configurando o bloco *Random Source*. No parâmetro *Output Type* escolhemos o valor *byte*, para os parâmetros *Minimum* e *Maximum* definimos os valores *0* e *255* respectivamente, no parâmetro *Num Samples* escolhemos o valor *10e6* e para a opção *Repeat* setamos o valor *yes*. Este bloco será responsável por gerar o sinal que será modulado e transmitido. A fonte de sinal poderia ser outra, poderíamos escolher o bloco *Signal Source* e gerar sinais com com ondas específicas alterando o parâmetro *waveform*. Outra possibilidade seria a geração de sinais a partir de um arquivo utilizando os blocos *File Source* associado ao bloco *Packet Encoder*.

O dado de saída do bloco *Random Source* é do tipo *Complex* e deve ser ligada à entrada do bloco *PSK Mod* que é do mesmo tipo. O bloco *PSK Mod* é central neste exemplo, pois é ele quem modula o sinal que será transmitido. Configuramos seus parâmetros com os seguintes valores: *Number of constellations*: *4*, *Gray Code*: *yes*, *Differencial Encoding*: *yes*, *Samples/Symbols*: *2*, *Excess BW*: *350e-3*. O parâmetro *Number of constellations* deve ser um valor que seja potência de 2.

O bloco *Noise Source* não é essencial neste exemplo, ele existe para que se possa perceber diferenças no sinal a fim de identificá-lo, já que nenhuma mensagem específica será transmitida. Neste bloco, setamos o valor do parâmetro *Noise Type* como Gaussian e *Amplitude* como *0.125*.

Em seguida devemos adicionar este ruído ao sinal original, fazemos isso utilizando o bloco *Add*. Neste bloco configuramos os parâmetros *Num Inputs* e *Vec Length* como *2* e *1* respectivamente. O parâmetro *IO Type* setamos como *Complex*. Após essas configurações, ligamos a saída do bloco *PSK Mod* a uma entrada do bloco *Add* e ligamos a saída do bloco *Noise Source* à outra entrada do bloco *Add*.

O bloco *Multiply Const* apenas multiplica o sinal por *0,5*, isso é feito para evitar que o valor 1.0 chegue ao *UHD Sink*, o que saturaria o conversor analógico-digital. Qualquer valor acima de 1.0 causaria *clipping*. Assim, configuramos os parâmetros do bloco com só seguintes valores: *IO Type*: Complex, *Constant*: *0.5*, *Vec Length*: *1*. Em seguida, ligamos a saída do boclo *Add* à entrada do bloco *Multiply Const*.

O bloco seguinte a ser configurado é o *UHD: Usrp Sink*. Utilizamos as variáveis definidas com os blocos do tipo *Variable* para configurar este bloco. Os valores dos parâmetros devem ser: *Samp Rate (Sps)*: *samp_rate*, *Ch0: Center Freq (Hz)*: *freq*, *Ch0: Gain (dB)*: *gain*, *Ch0: Antenna*: *TX/RX*, *Ch0: Bandwidth (Hz)*: *samp_rate*. Ligamos então a saída do bloco *Multiply Const* na entrada do bloco *UHD: Usrp Sink*.

Por fim, ligamos também a saída do bloco *Multiply Const* na entrada do bloco *QT GUI Constellation sink*. Este bloco deve ter os parâmetros configurados da seguinte forma: *Number of points*: *1024*, *Y min* e *Xmin*: *-2*, *Y Max* e *X Max*: *2*, *Update Period*: *0.10*. Este bloco é responsável por exibir graficamente o sinal de saída do bloco *Multiply Const*, já modulado e pronto para ser entregue ao *UHD: Usrp Sink* [GNU Radio 2015].

O gráfico final deve ter o aspecto mostrado na Figura 1.4.4. Quando conectado ao

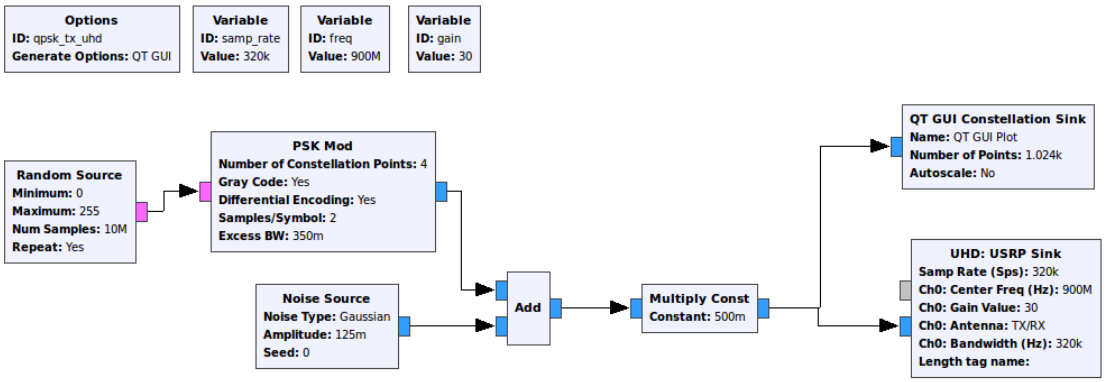

Figura 1.13. Flowgraph do modulador e transmissor QPSK

Equipamento B100 e executado, o resultado gráfico é exibido nas Figuras 1.14 e 1.15:

1.4.5. Comunicação com o protocolo IEEE 802.15.4

O padrão IEEE 802.15.4 é mantido pelo *Institute of Electrical and Electronics Engineers* (IEEE) e ficou responsável pela especificação das duas camadas mais baixas da tecnologia ZigBee, enquanto que a ZigBee Alliance trabalhava nas camadas superiores. Dessa forma, o padrão é a base para as especificações ISA100.11a, WirelessHART, e MiWi, além da ZigBee, cada uma das quais estende ainda mais o padrão através do desenvolvimento das camadas de roteamento e transporte, que não são definidas no padrão IEEE 802.15.4.

O ZigBee é projetado para oferecer conectividade com baixo custo de energia aos equipamentos que precisam extrair uma longa vida útil de uma bateria limitada, geralmente por vários meses ou anos, mas sem necessitar de taxas de dados tão elevadas quanto as proporcionadas pelo Bluetooth. Além disso, ZigBee pode ser implementada em redes de malha maiores do que é possível com a tecnologia Bluetooth [Ergen 2004].

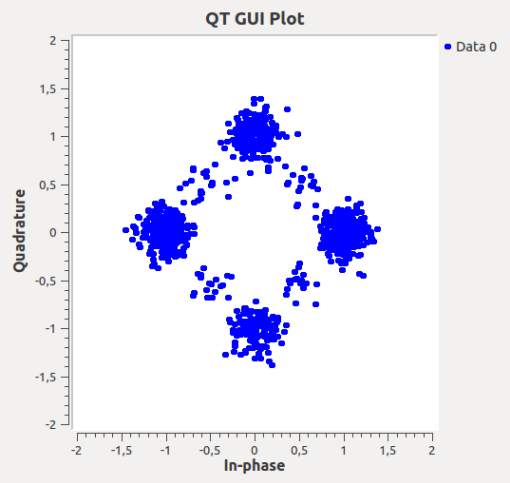

Figura 1.14. Sinal com ruído

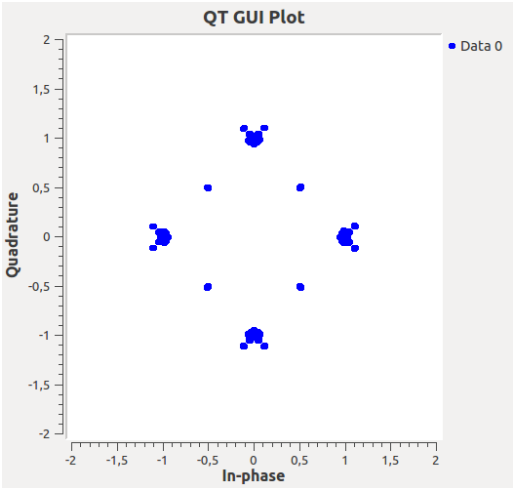

Figura 1.15. Sinal sem ruído

A pilha do protocolo pode ser melhor compreendida visualizando a Tabela 1.1 a seguir:

Usuário	Aplicação
ZigBee Alliance	Suporte à aplicação Rede (NWK) / Segurança (SSP)
IEEE 802.15.4	MAC PHY

Tabela 1.1. Representação esquemática da pilha de protocolo

A camada física (PHY) do ZigBee segue o protocolo 802.15.4 e é responsável por permitir a transmissão das PDUs (*Protocol Data Units*), unidades de dados, através de ondas de rádio. A PHY utiliza a modulação DSSS (*Direct Sequence Spread Spectrum*) que incorpora em cada bit de dado um padrão de redundância e os espalha pela largura de banda utilizada. Essa redundância permite não só que o dado seja identificado como pertencente a um determinado nó, como é claro, facilita a detecção de erros. Ao espalhar os dados em todas as frequências da banda, o sinal resultante se assemelha cada vez mais a um ruído, tornando-se mais robusto a interferências. Após o processamento da DSSS, o sinal é modulado em uma portadora para transmissão. As faixas de frequência utilizadas são as frequências livres de 2.4 GHz (global), 915 MHz (Américas) e 868 MHz (Europa). Cada uma das faixas implica em uma taxa de transmissão, número de canais e espectros diferentes. Outras funções da camada física são indicar qualidade de conexão, detectar potência dos canais, e reportar canais livres. A Tabela 1.2 apresenta características como modulação, taxa de bits e símbolos, de acordo com a banda de frequência em uso do padrão IEEE 802.15.4.

A camada MAC do padrão 802.15.4 é responsável pelo processo de encapsulamento dos dados oriundos das camadas superiores, preparando-os para serem transmitidos. O método de acesso ao meio caracteriza a rede em dois modos de operação: modo *Beaconing* e *Non-Beaconing*. No primeiro modo os roteadores transmitem periodica-

PHY (MHz)	Banda de frequência (MHz)	Parâmetros de espalhamento		Parâmetros dos dados		
		Taxa de espalhamento (kchip/s)	Modulação	Taxa de bit (kb/s)	Taxa de símbolo	Símbolos
868/915	868-868.6	300	BPSK	20	20	Binário
	902-928	600	BPSK	40	40	Binário
2450	2400-2483.5	2000	O-QPSK	250	62.5	16 símbolos

Tabela 1.2. Bandas de frequência e taxas de dados

mente sinalizadores (*beacon frames*) para confirmar sua presença na rede. Após o *beacon*, há os tempos de acesso CAP (*Contension Access Period*), onde todos os dispositivos competem entre si utilizando CSMA-CA (*Carrier Sense Multiple Access with Collision Avoidance*) e o CFP (*Contension Free Period*) que garante reservas de tempo para cada dispositivo. Utilizando-se de boa sincronia, os nós da rede (exceto o coordenador) podem permanecer inativos entre os *beacon frames* e economizar energia. Nesse modo é utilizada a estrutura de *superframe*. O modo *Non-Beaconing* se caracteriza por manter a maioria dos nós ativos constantemente, ocasionando um maior consumo energético [Ergen 2004].

A aplicação que será analisada a seguir é do tipo *out-of-tree*, ou seja, é uma componente GNU Radio que não foi desenvolvida nem é mantida dentro dos arquivos *fontes* do GNU Radio. A aplicação é baseada nas implementações de [Schmid 2006] e de [Bloessl et al. 2013a], tendo como principais modificações a segmentação do arquivo *transceiver.grc* que possuia função tanto de recepção como de transmissão, em dois arquivos denominados *transceiver_onlyRx.grc* e *transceiver_onlyTx.grc*, cada um com rearranjos de blocos a fim de possibilitar o isolamento das funcionalidades.

Nesta aplicação, mostramos um transceptor IEEE 802.15.4 para GNU Radio v3.7 que apresenta como principais características:

- Camada física empregando O-QPSK e encapsulada em um bloco hierárquico;
- Bloco *RIME Stack*, que implementa a pilha de comunicação Rime (Rime é uma pilha de comunicação projetada para redes de sensores sem fio e faz parte do sistema operacional Contiki) [Leitner 2013];
- Exemplo de aplicativo que transmite e recebe mensagens no padrão IEEE 802.15.4;
- Exemplo de aplicativo que visualiza valores do nós sensores.

Algumas propriedades interessantes da implementação são que os pacotes podem ser canalizados para Wireshark, que é um *software* de captura de pacotes [Foundation 2015]. A modulação física completa é feita com simples blocos do GRC, e a implementação é interoperável com sensores TelosB motes e com Contiki; e utiliza um bloco para marcar rajadas de pacotes com *Tx* e *Rx*. Estas marcas são compreendidas pelos blocos UHD e permitem a comutação rápida entre transmissão e recepção, necessária quando Tx e Rx estão sendo executados em um único equipamento. A aplicação discutida está configurada para ser executada com o mínimo de dois equipamentos, mas é possível fazer rearranjos nos blocos do fluxograma para que Tx eRx executem em apenas um USRP, conforme apresentado em [Bloessl et al. 2013a].

Uma visão geral sobre a arquitetura do transceptor pode ser vista na Figura 1.16. A estrutura do transceptor SDR é modular, em camadas, como exposto no GRC. A camada física é encapsulada em um bloco hierárquico e os pacotes entre o MAC e a camada física são capturados pelo conector Wireshark. Devido à estrutura modular deste fluxo, a forma como a mensagem trafega a partir do aplicativo de envio à USRP podem ser facilmente rastreados: o bloco *Socket PDU* transforma a mensagem enviada pela aplicação (neste caso, enviada pelo bloco *Message Strobe*) em um formato de PDU, em seguida, ela é encapsulada pelo bloco *Rime Stack* na camada de rede e pelo bloco *IEEE802.15.4 MAC* na subcamada MAC. Depois disso, a mensagem é transformada em um sinal físico, modulado e enviado pelo bloco *IEEE802.15.4 PHY*.

Várias conexões podem ser abertas pelo bloco *Rime Stack*, fornecendo vários números identificadores de canais para cada tipo de conexão desejada na janela de opções do bloco. O GRC cria automaticamente um par de portas entrada/saída para cada conexão aberta. Há dois pares de conexões de *broadcast* (rotuladas como *bcinX* e *bcoutX*), de *unicast* (rotuladas como *ucinX* e *ucoutX*) e um par de conexões *unicast* confiáveis (marcada com *rucin* e *rucout*). Além disso, pode-se configurar o endereço Rime do transceptor. Se o bloco Rime é conectado à camada MAC pelas suas portas *to/fromMAC*, as mensagens podem ser enviadas por ligação a um bloco de emissão de PDU [Leitner 2013]. Na Figura 1.16, o bloco *Socket PDU* gera mensagens de dados no formato PDU que serão enviadas para o socket UDP na porta 52001, podendo este valor ser alterado pelo usuário nas propriedades do bloco.

O bloco MAC está atualmente limitado à funcionalidade mais básica que permite a conectividade: encapsula os pacotes das camadas mais altas com um cabeçalho IEEE 802.15.4 válido e acrescenta a soma de verificação CRC. No processo de recepção, ele faz o inverso, removendo o cabeçalho MAC e verificando se o CRC está correto. Em particular, a camada MAC ainda não realiza detecção de portadora, enviando uma mensagem imediatamente, nem implementa o sistema de *back-off* ou a separação entre períodos CAP e CFP.

O processo de instalação desse módulo de comunicação com o protocolo IEEE 802-15-4 (*gr-ieee802-15-4*) se dá através da aplicação Git. Para que seja possível realizar a instalação e execução correta desse módulo, é necessário instalar, inicialmente, um módulo complementar chamado *gr-foo*, desenvolvido por [Bloessl et al. 2013a]. Para isso, devemos executar a seguinte sequência de comandos:

```
git clone https://github.com/bastibl/gr-foo.git
cd gr-foo
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Por fim, para baixar e instalar o módulo *gr-ieee802-15-4* devemos executar os comandos listados a seguir:

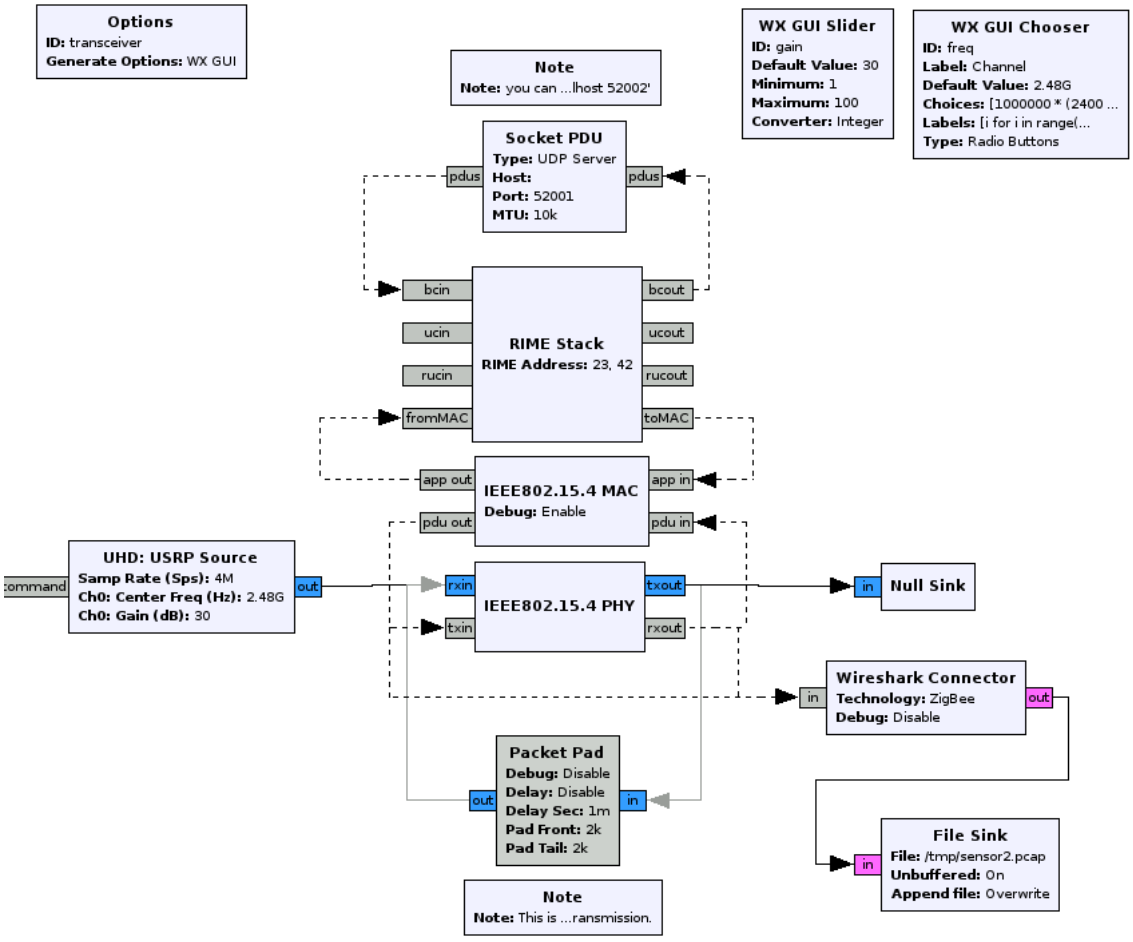

Figura 1.16. Arquitetura do transceptor

```
git clone git://github.com/wendley/gr-ieee802-15-4.git
cd gr-ieee802-15-4
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Concluídas essas etapas, o GRC de sua máquina já deve exibir o módulo *gr-ieee802-15-4* na sua biblioteca. Precisamos instalar o bloco hierárquico; para isso, abrimos o arquivo *examples/ieee802_15_4_PHY.grc* no GRC e o compilamos (tecla F5). Isso instala o bloco hierárquico no diretório onde o GRC possa encontrá-lo, normalmente em /.grc_gnuradio. Pode ser necessário reiniciar o GRC ou todo o sistema do GNU Radio para que o bloco seja corretamente detectado. Podemos conferir se a instalação foi corretamente concluída procurando pelo arquivo *examples/transceiver.grc* e verificando no GRC se todos os blocos estão conectados com sucesso.

Em seguida, podemos iniciar as comunicações entre os USRPs através da execução dos arquivos *transceiver_onlyRx* e *transceiver_onlyTx*. Vale salientar que cada um destes deverão ser executados em microcomputadores distintos. Dessa forma, se quisermos a configuração de um equipamento transmitindo e três na recepção, teremos que dispor de quatro máquinas, cada uma com o GNU Radio instalado e um USRP associado. Os fluxogramas dos arquivos *transceiver_onlyRx* e *transceiver_onlyTx* diferem entre si basicamente pela forma de conexão dos blocos: enquanto o Rx dispõe de um bloco *Message Strobe* e não possui um *USRP Sink*, o Tx não possui um bloco *Message Strobe* e possui um *USRP Sink*. Essa diferença fica mais evidente comparando-se a Figura 1.16, que representa um transceptor Rx, e a Figura 1.17, que representa um transceptor Tx.

A aplicação *transceiver_onlyTx* pode ser iniciada através da compilação (F5) e execução (F6) do arquivo *transceiver_onlyTx*. Podemos editar a mensagem a ser enviada (no nosso exemplo, o texto *Msg Tx SBRC 2015*) nas propriedades do bloco *Periodic Message Strobe*. Além do conteúdo da mensagem, podemos alterar o período (em milissegundo) e a quantidade de mensagens que serão transmitidas a cada execução. Observa-se da Figura 1.17 que os blocos *Wireshark Connector* e *File Sink* estão desativados (identificáveis através do preenchimento em cor cinza), visto que, embora possível, não é necessário salvar as mensagens em um arquivo. A desativação ou ativação de blocos é possível clicando-se com o botão direito do *mouse* sobre o bloco e escolhendo, respectivamente, *Disable* ou *Enable*. Uma funcionalidade que pode ser adicionada no futuro é obter a mensagem através de um arquivo.

Para a recepção, devemos iniciar a aplicação *transceiver_onlyRx* compilando (F5) e executando (F6) o arquivo *transceiver_onlyRx*. O USRP então coletará as mensagens que estão sendo transmitidas na frequência especificada e o bloco *RIME Stack* fará a decodificação, salvando-as em um arquivo *.pcap* através do bloco *Wireshark Connector*. Dessa forma, pode-se verificar no arquivo recém gerado *.pcap* se a mensagem enviada pelo transmissor foi recebida corretamente pelo USRP e devidamente demodulada.

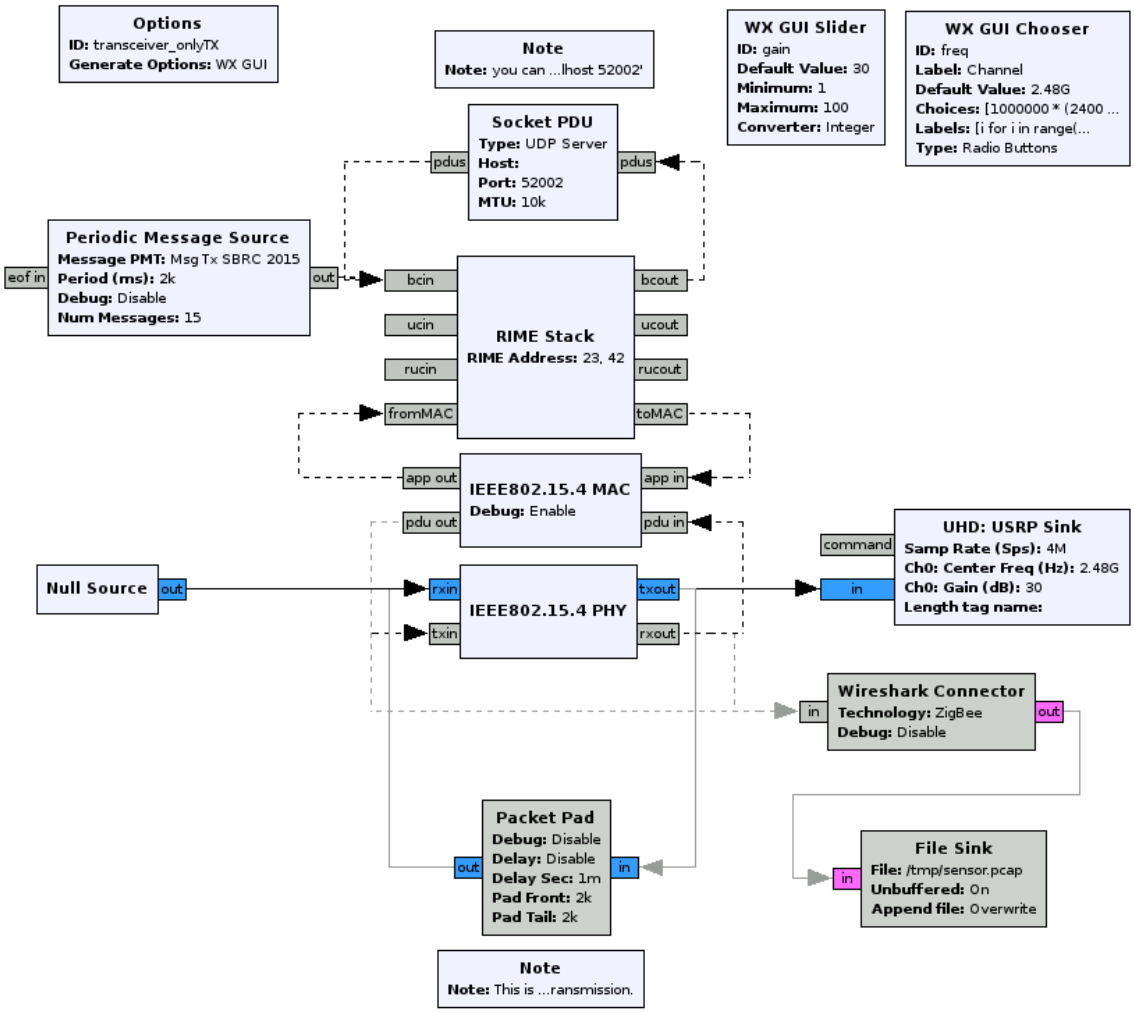

Figura 1.17. Arquitetura do transceptor Tx

Configuramos o transmissor para enviar a mensagem de texto *Msg Tx SBRC 2015* a cada 2 segundos, no canal 15, selecionado na janela gráfica logo após o início da transmissão. O receptor foi configurado para receber no mesmo canal e salvar as capturas no arquivo *msgBee.pcap* . Com o Wireshark, abrimos esse arquivo e podemos visualizar as mensagens transmitidas, incluindo seu conteúdo de texto. A Figura 1.17 exibe a tela de captura do Wireshark, onde se observa na porção superior da imagem todos os pacotes recebidos e na base da imagem temos a mensagem textual que foi enviada. A Figura 1.19 exibe um resumo, realizado pelo próprio Wireshark, avaliando quais as proporções de protocolos presentes no arquivo. Neste caso, todos os 42 pacotes recebidos correspondiam ao protocolo IEEE 802.15.4.

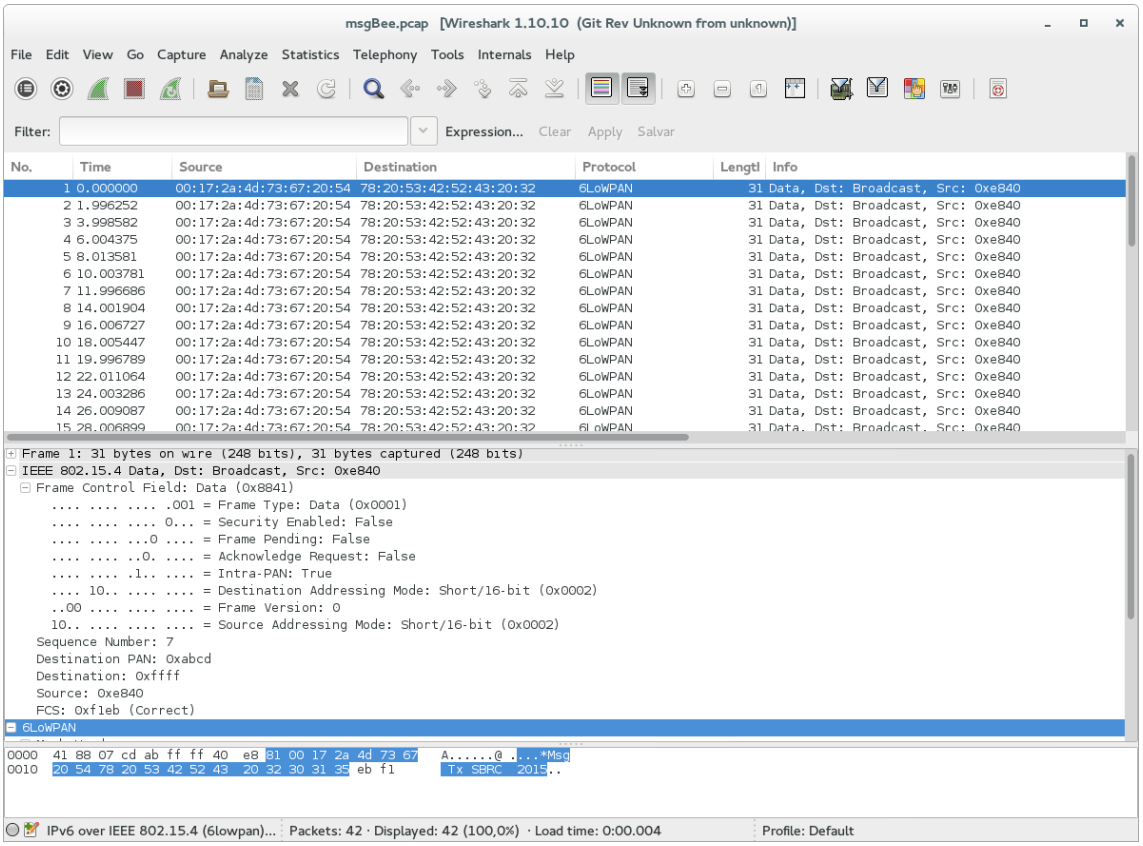

Figura 1.18. Tela do Wireshark exibindo o conteúdo do arquivo *msgBee.pcap*

1.4.6. Criando novos blocos

Como o principal objetivo do uso de SDR em pesquisa em redes de computadores é o desenvolvimento de novas técnicas e protocolos, nesta subseção apresentamos uma breve descrição de como construir um novo bloco para o GNU Radio. Iremos apresentar um exemplo de um bloco simples, que realiza a contagem de mensagens recebidas pela aplicação *transceptor Rx*. O bloco realiza incrementos sucessivos em um contador a cada vez que o mesmo é instanciado, e imprime no console o valor desse contador. Salienta-se que essa saída pode ser exibida também em uma interface gráfica, em livre escolha na biblioteca do GRC.

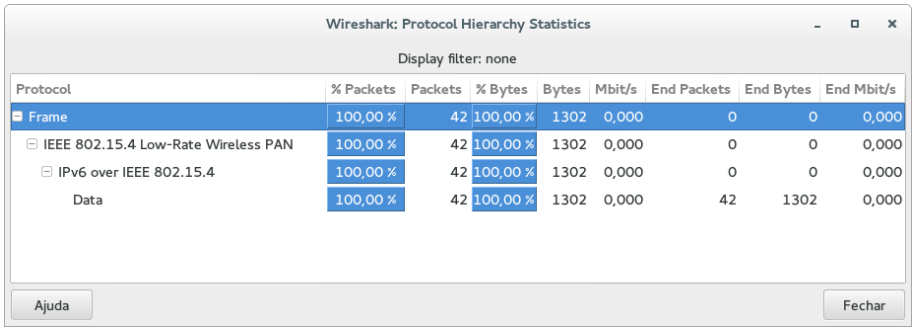

Figura 1.19. Resumo de protocolos do arquivo *msgBee.pcap*

Inicialmente, precisamos mencionar que grande parte do esforço é realizado pela ferramenta *gr_modtool*, que já compõe o sistema GNU Radio. Informações detalhadas sobre as opções dessa ferramenta podem ser obtidas com o comando `$ gr_modtool help`. Vamos criar um bloco chamado *Multiply* dentro da categoria *Tutorial*. Para isso, fazemos `$ gr_modtool newmod tutorial` . Como saída, teremos:

```
Creating out-of-tree module in ./gr-tutorial... Done.
Use 'gr_modtool add' to add a new block to this currently empty
module.
```

Foi criada uma pasta chamada *gr-tutorial*. Devemos entrar nessa nova pasta e executar o comando especificado na primeira linha da sequência abaixo, e entrar com as respostas para as solicitações abaixo descritas:

```
gr-tutorial$ gr_modtool add -t sync -l python
GNU Radio module name identified: tutorial
Language: Python
Enter name of block/code (without module name prefix): counter_py_ff
Block/code identifier: counter_py_ff
Enter valid argument list, including default arguments: counter
Add Python QA code? [Y/n] n
Adding file 'Python/counter_py_ff.py'...
Adding file 'grc/tutorial_counter_py_ff.xml'...
Editing grc/CMakeLists.txt...
```

Em seguida devemos abrir o arquivo *counter_py_ff.py* (o nome do arquivo é constituído por *nomedomodulo.extensão*, sendo que a extensão é *.py* porque selecionamos a linguagem do bloco como Python), que está na pasta *python* recém criada e alterá-lo de forma que tenhamos a configuração abaixo. As inserções ou alterações ocorreram nas linhas 8, 13, 14, 21, 22 e 23. Na linha 8 instanciamos uma variável global, que será o contador. Nas linhas 13 e 14 definimos o tipo de dados como *float*, e finalmente nas linhas 21 a 23 realizamos o incremento da variável *contador* e imprimimos a saída.

```
1  import numpy
2  from gnuradio import gr
3
4  class counter_py_ff(gr.sync_block):
5      """
6  ␣␣␣␣docstring␣for␣block␣counter_py_ff
7  ␣␣␣␣"""
8      contador = 0
9
10     def __init__(self, counter):
11         gr.sync_block.__init__(self,
12             name="counter_py_ff",
13             in_sig=[numpy.float32],
14             out_sig=[numpy.float32])
15
16
17     def work(self, input_items, output_items):
18         in0 = input_items[0]
19         out = output_items[0]
20         # <+signal processing here+>
21             self.contador = self.contador + 1
22             out[:] = out[:] = in0*self.contador
23         print self.contador
24         return len(output_items[0])
```

Por fim, devemos alterar o arquivo XML *tutorial_counter_py_ff.xml*, acessível na pasta recém criada *grc*, para inserir o novo bloco na biblioteca do GRC. Poucas alterações são necessárias no arquivo, mais especificamente nas linhas 14, 22, 23, 32 e 33. Estas alterações são necessárias para registrar as entradas e saídas do bloco no GRC, indicando os seus respectivos tipos e nomes. Ao final da edição, o arquivo deve conter o texto abaixo:

```
1  <?xml version="1.0"?>
2  <block>
3    <name>counter_py_ff</name>
4    <key>tutorial_counter_py_ff</key>
5    <category>tutorial</category>
6    <import>import tutorial</import>
7    <make>tutorial.counter_py_ff($counter)</make>
8    <!-- Make one 'param' for every Parameter you want on the GUI.
9         Sub-nodes:
10        * name
11        * key (makes the value accessible as $keyname...
12        * type -->
13
14   <!-- No parameters in this block counter-->
15
16   <!-- Make one 'sink' node per input. Sub-nodes:
17        * name (an identifier for the GUI)
```

```
18          * type
19          * vlen
20          * optional (set to 1 for optional inputs) -->
21    <sink>
22      <name>in</name>
23      <type>float</type>
24    </sink>
25
26    <!-- Make one 'source' node per output. Sub-nodes:
27          * name (an identifier for the GUI)
28          * type
29          * vlen
30          * optional (set to 1 for optional inputs) -->
31    <source>
32      <name>out</name>
33      <type>float</type>
34    </source>
35  </block>
```

Uma vez criado o arquivo XML de definições do bloco, devemos indicar ao GRC que existe um novo bloco. Para instalar o bloco no GRC, devemos sair do diretório *.../grc* e criar uma pasta denominada *build*. A partir dela, devemos executar os comandos a seguir:

```
cmake ../
make
sudo make install
sudo ldconfig
```

Abrindo o GRC, o novo bloco *counter_py_ff* deve aparecer no final da listagem da biblioteca, na categoria *tutorial*. Este bloco pode ser adicionado ao final do fluxo da aplicação *transceptor Rx* para que possa realizar uma contagem da quantidade de pacotes recebidos; contudo precisamos inserir o bloco conversor *IChar to Complex* e outro bloco conversor *Complex to Float* antes do bloco *Counter*. O bloco *Counter* pode ficar posicionado entre os blocos *Wireshark Connector* e *File Sink*. Informações complementares sobre construção de novos blocos e como adicioná-los à biblioteca podem ser encontradas em [Radio 2015c].

1.4.7. Dicas Práticas

Nesta seção iremos dar algumas dicas práticas de uso e instalação do GNU Radio, que identificamos com o uso da plataforma em diversas aplicações. Elas são:

- **Escolha da versão do GNU Radio e SO a ser empregada**. Muitos dos padrões de comunicação modernos possuem implementações em GNU Radio (por exemplo, IEEE 802.11, IEEE 802.15.4, RFID), entretanto elas não fazem parte dos módulos suportados pela *release* oficial do GNU Radio. Estes módulos são conhecidos no

Gnu Radio como *out-of-tree*, ou OOT. Nestes casos, muitas vezes os módulos fazem referências a funcionalidades e métodos de versões específicas do código. Além disso, os códigos podem requerer bibliotecas específicas do sistema em uma dada versão. Uma dica importante, nestes casos, é sempre perguntar ao autor do módulo qual foi a versão do GNU Radio empregada, bem como o SO.

- **Evite máquinas virtuais para aplicações de alta largura de banda**. O GNU Radio requer muito processamento quando toda a aplicação é executada no CPU. Assim, para usos de alta largura de banda, evite máquinas virtuais. Elas adicionam atrasos importantes no processamento, devido à virtualização da rede e USB.

1.4.8. Alternativas ao GNU *Radio*

Apresentaremos a seguir três principais alternativas ao uso do GNU Radio e, ao final, realizamos uma comparação entre as plataformas discutidas.

1.4.8.1. ASGARD

Desenvolvido pela *Aalborg University*, da Dinamarca, o ASGARD (*Application-oriented Software on General purpose processors for Advanced Radio Development*) [AsgardAAU 2015a] é um *framework* para desenvolvimento de aplicações para SDR e rádios cognitivos. Escrito em linguagem C++, é executado como uma aplicação no espaço de usuário do sistema operacional Linux, tem como principal objetivo fornecer o grau de flexibilidade necessário para a implementação de sistemas de comunicação multicamadas reconfiguráveis [Tavares et al. 2012]. Diferentemente do GNU Radio, o ASGARD não dispõe de um fluxo gráfico dos dados para edição ou visualização. A arquitetura do *software* é baseada em APIs (*Application Programmable Interfaces*), utilizando-se de componentes, módulos, aplicativos e objetos de comunicação. A seguir encontramos uma breve descrição de cada um desses elementos [AsgardAAU 2015b]:

- Componente é a função atômica básica do ASGARD, não havendo restrições de tipos de dados de entrada;
- Módulo é um processo que contém uma lista de componentes, executando-os de acordo com uma sequência definida;
- Aplicativos são responsáveis por especificar as relações entre os componentes, possuindo o mais alto nível de abstração, aceitando, inclusive, configuração por XML;
- Objetos de comunicação permitem a troca de dados entre diversos componentes e os módulos.

1.4.8.2. IRIS

IRIS (*Implementing Radio in Software*) é uma arquitetura de *software* para rádios cognitivos desenvolvida para processadores de propósito geral, compatível com Windows e Linux, que permite a criação de redes altamente reconfiguráveis.

Assim como GNU Radio, esta plataforma é orientada a componentes. Processadores de sinais como filtros e moduladores são implementados como componentes genéricos passíveis de reconfiguração. Esses componentes são conectados entre si para produzir os canais de transmissão e recepção, que por sua vez, são conectados às camadas superiores da rede.

O IRIS possui um foco muito grande em flexibilidade, e sua arquitetura possui grande suporte para reconfiguração do rádio em tempo real. Além disso, possui suporte não só para camada física, mas para todas as camadas da rede, e provê ainda uma interface bem definida para que os controladores possam tomar decisões baseadas nas observações do ambiente e do próprio rádio. IRIS suporta plataformas avançadas de processamento que incluem FPGA e CellBE e, além disso, é baseada em C++, portável para diversos sistemas operacionais e arquiteturas de CPU. A representação de seus componentes é feita utilizando XML, que permite até mesmo a reconfiguração de rádios em funcionamento.

Em comparação com GNU Radio, IRIS possui um suporte maior em relação à capacidade de reconfiguração em tempo real e também quanto à pilha de protocolos da rede, já em relação à linguagem de desenvolvimento, suporta apenas C++, enquanto o GNU Radio permite também o uso de Python [Sutton et al. 2010].

1.4.8.3. OSSIE

OSSIE (*Open-Source Software Communication Architecture Implementation - Embedded*) [NSF 2015] é um projeto aberto de SDR desenvolvido e mantido pela *Virginia Tech*, EUA. Ele foi criado para ser utilizado para prototipagem rápida e em provas de conceito. É patrocinado pela Fundação Nacional de Ciência dos EUA (*U.S. National Science Foundation*) e implementa uma versão aberta do *software* de comunicação SCA (*Software Communication Architecture*) desenvolvido pelo departamento de defesa norte-americano (*U.S. Department of Defense*). De acordo com [Snyder et al. 2011], OSSIE consiste de quatro partes principais:

- Núcleo do *framework*;
- uma "biblioteca" (*workshop*) de formas de ondas;
- uma biblioteca de componentes pré-fabricados e ondas; e
- um conjunto de ferramentas para desenvolvimento rápido de componentes e aplicações SDR.

O sistema OSSIE é escrito em C++, utilizando omniORB CORBA ORB, e é projetado para sistemas operacionais Linux. Por sua vez, ele é distribuído sob a licença GNU GPL (General Public License) 2.0 ou posterior (para ferramentas e componentes de processamento de sinal), e GNU Lesser GPL 2.1 (para o núcleo do *framework*). Em contraposição ao GNU Radio, OSSIE permite a integração com diversos outros ambientes de desenvolvimento, como o Eclipse [Eclipse Foundation 2015] e o próprio GNU Radio.

1.4.8.4. Comparação

A Tabela 1.3 apresenta uma comparação das principais características das plataformas de desenvolvimento SDR, incluindo o GNU Radio.

	Linguagem	**Fluxo gráfico**	**Custo**	**Observações**
GNU Radio	C++ e Python	Sim	Grátis, com licença GPL	Código aberto; ampla comunidade de usuários
ASGARD	C++	Não	Grátis, com um termo de acordo	Desenvolvido pela Universidade de Aalborg
IRIS	C++	Sim	Grátis apenas para membros	Desenvolvido pela Universidade de Dublin
OSSIE	C++	Sim	Grátis, com licença GPL	Suporte do governo norte-americano

Tabela 1.3. Tabela comparativa com as principais plataformas SDR

1.5. Conclusões e desafios

Nesta seção apresentamos os desafios do SDR hoje e suas limitações, bem como a nossa visão para o futuro da área e as conclusões.

1.5.1. Desafios

A seguir apresentamos os principais desafios para a evolução do SDR e das suas plataformas no meio científico.

- Maior largura de banda (*bandwidth*) dos transceptores. Os padrões mais recentes de telecomunicação tem empregado canais cada vez mais largos. O 802.11ac, por exemplo, poderá empregar canais de até 160MHz. Isso implica no desenvolvimento de novas plataformas de SDR com filtros suportando canais tão largos. Além disso, a quantidade de informação a ser processada irá aumentar, requerendo novos mecanismos como barramentos com maior capacidade de transmissão de dados, sistemas operacionais com baixa latência e alto *throughput*.

- Linguagens de alto nível e depuração. Da mesma forma que os bancos de dados possuem linguagens específicas, facilitando a programação e depuração de consultas, os SDR deveriam empregar linguagens para a escrita de módulos de comunicação e protocolos de controle do meio que possuam alto nível de abstração. A máquina de estados de FLAVIA é um exemplo, pois utiliza instruções ligadas ao problema a ser tratado, que é a escrita de protocolos MAC. As linguagens de propósito específico, quando aliadas a ambientes de programação e depuração adequados, permitiriam o desenvolvimento e a verificação dos protocolos desenvolvidos de forma mais fácil.

- Suporte ao *software*. As bibliotecas de módulos e componentes são muitas vezes suportadas por grupos acadêmicos ou comunidades que não possuem financiamento

perene para as suas tarefas. O resultado é que os esforços de desenvolvimento são perdidos, pois se tornam incompatíveis com o lançamento de novas versões da plataforma de *hardware* ou do *software*, e existe pouca documentação ou correção de erros. A comunidade de SDR deve sensibilizar as agências de fomento para iniciativas de manutenção de *software* para a pesquisa, da mesma forma que hoje as plataformas experimentais possuem chamadas específicas para projetos. Um bom começo é o suporte da plataforma OSSIE pelo Departamento de Defesa dos EUA.

- Plataformas experimentais de menor custo, consumo de energia e tamanho. Apesar de SDR ter democratizado a pesquisa experimental em redes sem fio, o seu uso em experimentos de larga escala ainda é proibitivo devido ao seu custo, tamanho e consumo de energia. Apesar de pesquisas indicarem que é possível desenvolver SDRs capazes de serem empregados em telefones e outros dispositivos móveis que executam padrões de comunicação de alto *throughput* [Lin et al. 2006], atualmente não existem plataformas de SDR para pesquisa realmente portáteis. Isto impede que experimentos de longa duração e escala, em ambientes móveis e utilizando voluntários, possam ser realizados.

1.5.2. Visão do futuro

É inegável a importância de SDR, pois estas plataformas acelerarão o processo de prototipagem e o desenvolvimento de novas tecnologias de comunicação sem fio. O tempo necessário no processo de desenvolvimento de novos padrões e protocolos de comunicação sem fio pode diminuir quando utilizamos componentes de *software*.

SDR podem e devem ser utilizados para tornar rádios cognitivos uma realidade. Com isso, teremos uma melhor eficiência do espectro de radiofrequência. Além disso, o uso de SDR em produtos comerciais permitirá que a atualizações de novas tecnologias de comunicação sem fio sejam implementadas por *software* e irá acelerar a adoção de novos protocolos. Como vimos durante o texto, o conceito de SDR já é aplicado parcialmente na infra-estrutura celular.

Finalmente, o SDR pode ser uma ferramenta importante para a evolução rápida dos padrões de comunicação e a diminuição do problema de compatibilidade entre padrões novos e de legado. Quando os padrões 802.11g,n e ac foram especificados, eles precisaram ser compatíveis com todos os padrões anteriores, até mesmo com o primeiro padrão, o IEEE 802.11b. Isso acarreta em limitações no que pode ser melhorado no protocolo, gerando restrições no que pode ser modificado e diversas "gambiarras", que acabam complicando os protocolos e limitando a vazão da rede. Por exemplo, os novos padrões 802.11 tiveram que adotar a confirmação de mensagens por rajadas para poder manter a compatibilidade. Caso os rádios empregassem SDR, não haveria problemas de compatibilidade, pois poderíamos atualizar os rádios já existentes mais rapidamente, e haveria uma melhor utilização do espectro de radiofrequência.

1.5.3. Considerações finais

O potencial do paradigma de Rádios Definidos por *Software* está apenas começando a ser explorado, mas já há grande interesse entre pesquisadores e empresas da área, tanto que

algumas plataformas comerciais já foram desenvolvidas, e diversos produtos incorporam SDR mesmo que parcialmente.

Neste trabalho apresentamos uma visão dos aspectos teóricos e práticos no desenvolvimento de pesquisa em SDR. Na parte teórica, apresentamos os usos de SDR, algumas plataformas populares na pesquisa em redes e telecomunicações, resultados de pesquisa recente que só foram possíveis devido ao uso de SDR, e os fundamentos de transmissão digital. Na parte prática, focamos na plataforma GNU Radio. Mostramos como utilizar o GNU Radio, seus módulos e ferramentas.

Dado o seu potencial, consideramos que existe um campo amplo para o desenvolvimento de novos projetos de pesquisa em Rádios Definidos por *Software*, seja como ferramenta para o desenvolvimento de novos padrões e protocolos de comunicação, seja como alvo de estudos sobre novas soluções de implementação. Certamente, os trabalhos mais interessantes na área ainda virão.

Referências

[Ope] Open WRT. `http://openwrt.org/`. Acessado em Maio de 2013.

[Amiri et al. 2007] Amiri, K., Sun, Y., Murphy, P., Hunter, C., Cavallaro, J., and Sabharwal, A. (2007). WARP, a unified wireless network testbed for education and research. In *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*, pages 53–54.

[AsgardAAU 2015a] AsgardAAU (2015a). Asgard - platform for cognitive radio. `http://blog.asgard.lab.es.aau.dk`. Acessado em Março de 2015.

[AsgardAAU 2015b] AsgardAAU (2015b). Asgardaau. `http://blog.asgard.lab.es.aau.dk/sites/default/files/material/other/asgard_commercial.pdf`. Acessado em Março de 2015.

[Balint 2015] Balint (2015). Gnu radio tutorial series. `https://www.youtube.com/watch?v=N9SLAnGlGQs&list=PL618122BD66C8B3C4`. Acessado em Março de 2015.

[Bharadia et al. 2013] Bharadia, D., McMilin, E., and Katti, S. (2013). Full duplex radios. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, SIGCOMM '13, pages 375–386, New York, NY, USA. ACM.

[Bloessl et al. 2013a] Bloessl, B., Leitner, C., Dressler, F., and Sommer, C. (2013a). A GNU Radio-based IEEE 802.15.4 Testbed. In *12. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN 2013)*, pages 37–40, Cottbus, Germany.

[Bloessl et al. 2013b] Bloessl, B., Segata, M., Sommer, C., and Dressler, F. (2013b). An IEEE 802.11a/g/p OFDM receiver for GNU radio. In *Proceedings of the Second Workshop on Software Radio Implementation Forum*, Software Radio Implementation Forum (SRIF), pages 9–16.

[Cass 2013] Cass, S. (2013). A $40 software-defined radio. *IEEE Spectrum*, 50(7):22–23.

[Chen and Duan 2011] Chen, K. and Duan, R. (2011). C-RAN –the road towards green RAN. Technical report, China Mobile. http://labs.chinamobile.com/cran/wp-content/uploads/CRAN_white_paper_v2_5_EN.pdf.

[Cidon et al. 2012] Cidon, A., Nagaraj, K., Katti, S., and Viswanath, P. (2012). Flashback: decoupled lightweight wireless control. *SIGCOMM Comput. Commun. Rev.*, 42(4):223–234.

[Cimini 1985] Cimini, L. J. (1985). Analysis and simulation of a digital mobile channel using orthogonal frequency division multiplexing. *Communications, IEEE Transactions on*, 33(7):665–675.

[Clark 1983] Clark, A. (1983). Principles of digital data transmision. *London, Pentech Press, 1983, 312 p.*, 1.

[Dillinger et al. 2003] Dillinger, M., Madani, K., and Alonistioti, N. (2003). *Software Defined Radio: Architectures, Systems and Functions*. Wiley & Sons.

[Eclipse Foundation 2015] Eclipse Foundation (2015). Eclipse - The Eclipse Foundation open source community.

[Ergen 2004] Ergen, S. C. (2004). Zigbee/ieee 802.15. 4 summary. *UC Berkeley, September*, 10:17.

[Ettus Research a] Ettus Research, I. Ettus research. http://www.ettus.com.

[Ettus Research b] Ettus Research, I. Ettus research N210 product details. https://www.ettus.com/product/details/UN210-KIT.

[Force 2002] Force, S. P. T. (2002). Spectrum policy task force report et docket no. 02-155. Technical report, FCC.

[Forum 2011] Forum, W. I. (2011). Software defined radio - rate of adoption. http://www.wirelessinnovation.org/sdr_rate_of_adoption.

[Foundation 2015] Foundation, W. (2015). Wireshark - go deep. https://www.wireshark.org/. Acessado em Março de 2015.

[GNU Radio 2013] GNU Radio (2013). Welcome to GNU Radio! http://gnuradio.org/. Acessado em Maio de 2013.

[GNU Radio 2015] GNU Radio (2015). Tutorial: PSK Symbol Recovery.

[Gringoli and Nava] Gringoli, F. and Nava, L. OpenFWWF – open firmware for WiFi networks. http://www.ing.unibs.it/~openfwwf/.

[Gudipati and Katti 2011] Gudipati, A. and Katti, S. (2011). Strider: automatic rate adaptation and collision handling. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 158–169, New York, NY, USA. ACM.

[Haykin and Moher 2006] Haykin, S. S. and Moher, M. (2006). *Introduction to analog and digital communications*, volume 1. Wiley New York.

[Hong et al. 2012] Hong, S. S., Mehlman, J., and Katti, S. (2012). Picasso: flexible RF and spectrum slicing. *SIGCOMM Comput. Commun. Rev.*, 42(4):37–48.

[Iannucci et al. 2012] Iannucci, P. A., Perry, J., Balakrishnan, H., and Shah, D. (2012). No symbol left behind: a link-layer protocol for rateless codes. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 17–28, New York, NY, USA. ACM.

[III 2000] III, J. M. (2000). *Cognitive Radio – An Integrated Agent Architecture for Software Defined Radio*. PhD thesis, Royal Institute of Technology (KTH).

[Intel 2013] Intel, I. (2013). Packet processing on intel architecture. `http://www.intel.com/go/dpdk`.

[Katti et al. 2008] Katti, S., Rahul, H., Hu, W., Katabi, D., Médard, M., and Crowcroft, J. (2008). XORs in the air: practical wireless network coding. *IEEE/ACM Trans. Netw.*, 16(3):497–510.

[Kumar et al. 2013] Kumar, S., Cifuentes, D., Gollakota, S., and Katabi, D. (2013). Bringing cross-layer MIMO to today's wireless LANs. In *Proceedings of the ACM SIGCOMM 2013 conference*, SIGCOMM '13, pages 387–398, New York, NY, USA. ACM.

[Leech 2015] Leech, M. (2015). Build-Gnuradio. `http://www.sbrac.org/files/build-gnuradio`. Acessado em Março de 2015.

[Leitner 2013] Leitner, C. (2013). *Integration of the Rime Network Stack into GNU Radio*. Bachelor thesis, University of Innsbruck.

[Lin et al. 2008] Lin, K. C.-J., Kushman, N., and Katabi, D. (2008). ZipTx: Harnessing partial packets in 802.11 networks. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, MobiCom '08, pages 351–362, New York, NY, USA. ACM.

[Lin et al. 2006] Lin, Y., Lee, H., Woh, M., Harel, Y., Mahlke, S., Mudge, T., Chakrabarti, C., and Flautner, K. (2006). SODA: a low-power architecture for software radio. In *Computer Architecture, 2006. ISCA '06. 33rd International Symposium on*, pages 89–101.

[Murphy et al. 2006] Murphy, P., Sabharwal, A., and Aazhang, B. (2006). Design of warp: A wireless open-access research platform. In *European Signal Processing Conference*.

[Nagurney 2009] Nagurney, L. S. (2009). Software defined radio in the electrical and computer engineering curriculum. In *Proceedings of the 39th IEEE international conference on Frontiers in education conference*, FIE'09, pages 1489–1494.

[NSF 2015] NSF (2015). OSSIE - SCA-Based Open Source Software Defined Radio.

[Perry et al. 2012] Perry, J., Iannucci, P. A., Fleming, K. E., Balakrishnan, H., and Shah, D. (2012). Spinal codes. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 49–60, New York, NY, USA. ACM.

[Pinto and de Albuquerque 2002] Pinto, E. L. and de Albuquerque, C. P. (2002). A técnica de transmissão ofdm. *Revista Científica*, 1516:2338.

[Radio 2015a] Radio, G. (2015a). Guided tutorial grc. `http://gnuradio.org/redmine/projects/gnuradio/wiki/Guided_Tutorial_GRC`. Acessado em Fevereiro de 2015.

[Radio 2015b] Radio, G. (2015b). Instalando o gnu radio. `http://gnuradio.org/redmine/projects/gnuradio/wiki/InstallingGR_PTBR`. Acessado em Março de 2015.

[Radio 2015c] Radio, G. (2015c). Out-of-tree modules. `http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules`. Acessado em Março de 2015.

[Rappaport 2001] Rappaport, T. (2001). *Wireless Communications: Principles and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.

[Reis et al. 2012] Reis, A. L. G., Barros, A. F., Lenzi, K. G., Meloni, L. P., and Barbin, S. (2012). Introduction to the software-defined radio approach. *IEEE Latin America Transactions*, 10(1):1156–1161.

[Research 2015] Research, E. (2015). Ettus research product detail - b210. `http://www.ettus.com/product/details/UB210-KIT`. Acessado em Março de 2015.

[Schmid 2006] Schmid, T. (2006). Gnu radio 802.15. 4 en-and decoding. *unpublished document and source*.

[Shokrollahi 2006] Shokrollahi, A. (2006). Raptor codes. *IEEE/ACM Trans. Netw.*, 14(SI):2551–2567.

[Sklar 2001] Sklar, B. (2001). *Digital communications*, volume 2. Prentice Hall NJ.

[Snyder et al. 2011] Snyder, J., McNair, B., Edwards, S., and Dietrich, C. (2011). Ossie: An open source software defined radio platform for education and research. In *International conference on frontiers in education: computer science and computer engineering (FECS'11). World congress in computer science, Computer engineering and applied computing. Las Vegas, NV*.

[Sutton et al. 2010] Sutton, P., Lotze, J., Lahlou, H., Fahmy, S., Nolan, K., Ozgul, B., Rondeau, T., Noguera, J., and Doyle, L. (2010). Iris: an architecture for cognitive radio networking testbeds. *Communications Magazine, IEEE*, 48(9):114–122.

[Tan et al. 2009] Tan, K., Zhang, J., Fang, J., Liu, H., Ye, Y., Wang, S., Zhang, Y., Wu, H., Wang, W., and Voelker, G. M. (2009). Sora: High performance software radio using general purpose multi-core processors. In *USENIX International Symposium on Networked Systems: Design and Implementation (NSDI)*, pages 75–90.

[Tavares et al. 2012] Tavares, F. M. L., Tonelli, O., Berardinelli, G., Cattoni, A. F., and Mogensen, P. (2012). Asgard: the aalborg university cognitive radio software platform for dsa experimentation.

[Tenenbaum and Wetherall 2011] Tenenbaum, A. S. and Wetherall, D. (2011). *Redes de computadores*. Pearson, São Paulo, SP.

[Tinnirello et al. 2012] Tinnirello, I., Bianchi, G., Gallo, P., Garlisi, D., Giuliano, F., and Gringoli, F. (2012). Wireless MAC processors: Programming MAC protocols on commodity hardware. In *IEEE INFOCOM*, pages 1269 –1277.

[Vieira et al. 2013] Vieira, L. F. M., Gerla, M., and Misra, A. (2013). Fundamental limits on end-to-end throughput of network coding in multi-rate and multicast wireless networks. *Computer Networks*, 57(17):3267–3275.

[Wireless Innovation Forum] Wireless Innovation Forum. Wireless innovation forum. `http://www.wirelessinnovation.org`.

[Wyglinski and Pu 2013] Wyglinski, A. M. and Pu, D. (2013). *Digital Communication Systems Engineering with Software-Defined Radio*. Artech House.