UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

PEDRO KLIPPEL SATHLER LIMA

IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO DE UM RECEPTOR FM COM TECNOLOGIA RDS EM PLATAFORMA EMBARCADA

VITÓRIA – ES
12/2016

PEDRO KLIPPEL SATHLER LIMA

IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO DE UM RECEPTOR FM COM TECNOLOGIA RDS EM PLATAFORMA EMBARCADA

Parte manuscrita do Projeto de Graduação do aluno **Pedro Klippel Sathler Lima**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de engenheiro eletricista.

Orientador: Prof. Dr. Evandro Ottoni Teatini Salles

Co-orientador: Prof. Dr. Jair Adriano Lima Silva

VITÓRIA – ES
12/2016

PEDRO KLIPPEL SATHLER LIMA

IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO DE UM RECEPTOR FM COM TECNOLOGIA RDS EM PLATAFORMA EMBARCADA

Parte manuscrita do Projeto de Graduação do aluno **Pedro Klippel Sathler Lima**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de engenheiro eletricista.

Aprovado em 15 de dezembro de 2016.

COMISSÃO EXAMINADORA:

_____

Prof. Dr. Evandro Ottoni Teatini Salles
Orientador

_____

Prof. Dr. Jair Adriano Lima Silva
Co-Orientador

_____

Prof. Dr. Patrick Marques Ciarelli
Universidade Federal do Espírito Santo
Examinador

_____

Prof. Dr. Jorge Leonid Aching Samatelo
Universidade Federal do Espírito Santo
Examinador

VITÓRIA – ES
12/2016

DEDICATÓRIA

A Deus e à minha família.

AGRADECIMENTOS

Em primeiro lugar, agradeço muito a Deus porque sempre me deu muito mais do que jamais mereci.

Agradeço também aos meus pais, Claudia Klippel Sathler Lima e Wesley Sathler Lima e ao meu irmão Tiago Klippel Sathler Lima, que sempre cuidaram de mim e nunca deixaram que eu desistisse dos meus sonhos. Me ajudaram a cada dia, dando forças pra continuar e com seu companheirismo sempre me mantiveram firme na direção segura.

Agradeço também ao meu orientador Evandro Ottoni Teatini Salles e ao meu co-orientador Jair Adriano Lima Silva, pelos valiosos conselhos e conhecimento dispensados durante a execução deste projeto, além dos esforços para obtenção de recursos que eu não poderia ter conseguido por mim mesmo.

Agradeço também ao NERDS (Núcleo de Ensino em Redes Definidas por Software) por disponibilizar equipamentos e material humano, além do espaço físico, indispensáveis para a conclusão deste projeto.

Por fim, agradeço a todos os professores e colegas que, de alguma forma, fizeram parte não só deste projeto, mas de toda a minha jornada desde o início da graduação em Engenharia Elétrica, pelas experiências e conhecimentos proporcionados a mim.

RESUMO

Realiza-se neste projeto o desenvolvimento e programação de um receptor FM (*Frequency Modulation*) através da tecnologia RDS (*Rádio definido por software*) em uma plataforma embarcada. Avaliações de desempenho e aplicabilidade são realizadas, tendo em vista os recursos de entrada e saída, bem como computacionais, limitados de um *Raspberry Pi Model B*. Em um primeiro momento, objetiva-se compilar um programa capaz de ler os sinais obtidos na entrada do Raspberry Pi e converter, corretamente, para um sinal de áudio na saída do mesmo, através do software GNU Radio acoplado ao USRP2 (*Universal Software Radio Peripheral*), responsável pelo pré-processamento. Os testes realizados obtiveram êxito no sistema original porém com baixas taxas de amostragem. Em um segundo momento, aprimorou-se o desempenho do Raspberry Pi através de uma mudança no kernel do sistema operacional, implementando-se de um sistema de tempo real. Esta implementação visou o aumento da taxa de amostragem do sistema, bem como adição de tarefas simultâneas, pós implementação, com adição de uma tela *touchscreen* para interface. Em seguida, realiza-se um novo teste do receptor no Raspberry Pi com o sistema de tempo real, para avaliação de alterações no desempenho em comparação com o sistema de fábrica, entretanto, instabilidades no sistema inviabilizaram testes e aplicações posteriores.

ABSTRACT

This project was designed to develop and implement an FM (Frequency Modulation) Receiver via SDR (Software Defined Radio) Technology in an embedded platform. As embedded systems have limited processing capabilities, as well as IO limitations, performance analysis were run in order to effectively prove the applicability of the used hardware, Raspberry Pi 2 B, attending to the demands of an SDR. At first, the objective is to compile a program capable of reading signals sent to the Raspberry Pi 2 B input and correctly convert them to a quality audio signal at the output utilizing GNU Radio and the USRP2 (Universal Software Radio Peripheral), responsible for the pre-processing. Tests run in this condition were successful but suffered from low sample rates due to hardware limitations. As a second experiment, a real time Kernel was implemented in an attempt to study its behavior and analyze its performance. This implementation was executed, also, in an attempt to increase sample rate and make the system able to handle more simultaneous tasks, specially added input hardware, like a touchscreen interface. After that, new tests were run with the implemented FM receiver installed in the real time Kernel to evaluate performance differences, but system instabilities made it impossible to run new tests and applications with the selected software and hardware.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analog-to-Digital Converter*
AM	*Amplitude Modulation*
DAC	*Digital-to-Analog Converter*
DDC	*Digital Down Converter*
DSP	*Digital Signal Processing*
DUC	*Digital Up Converter*
FM	*Frequency Modulation*
FPGA	*Field Programmable Gate Array*
IRQ	*Interrupt Request*
LabTel	Laboratório de Telecomunicações
LRTBF	*Linux Real Time Benchmarking Framework*
ms	Milisegundo
NCO	*Numerically Controlled Oscillator*
NERDS	Núcleo de Ensino em Redes Definidas por Software
PDS	Processamento Digital de Sinais
RDS	Rádio definido por software
RTC	*Real Time Clock*
SDR	*Software Defined Radio*
SNR	*Signal-to-Noise Ratio*
Sps	*Samples per Second*
TCB	*Task Control Block*
UFES	Universidade Federal do Espírito Santo
UHD	*Universal Hardware Driver*
UTP	*Unshielded Twisted Pair*
USRP2	*Universal Software Radio Peripheral 2*
WBFM	*Wideband Frequency Modulation*

SUMÁRIO

1 INTRODUÇÃO

1.1 Modulação de Ondas de Rádio Frequência

Desde o final do século XIX, ondas de rádio vem sendo utilizadas para encurtar as distâncias de comunicação, seja com propósitos militares ou civis. Mesmo com o advento de novas tecnologias, diversos princípios mantiveram-se desde o início de sua utilização. Um desses princípios é a necessidade de realizar a modulação do sinal de mensagem antes de realizar sua transmissão. Isso dá-se pela natureza dos sinais de mensagem que, em sua grande maioria, não são apropriados para transmissão direta.

A técnica da modulação consiste em modificar o sinal que se deseja transmitir utilizando-se de um padrão conhecido, para que o receptor seja capaz de recuperar o sinal original usando as operações inversas. Existe uma grande diversidade de formas de modulação, mas as mais utilizadas para os radiofusão são AM (*Amplitude Modulation*) e FM (*Frequency Modulation*) que consistem de modulação em amplitude e frequência, respectivamente. Cada uma dessas técnicas possui vantagens e desvantagens e a escolha apropriada é feita baseada no que se deseja realizar com o sinal, como a distância de comunicação ou a densidade espectral de ruído externo no local de comunicação.

Dependendo da técnica de modulação escolhida, o hardware de transmissão e recepção deve ser adequado para cada caso específico. No entanto, com o tempo, e avanço de capacidade de processamento, percebeu-se a capacidade de realizar a modulação e demodulação digitalmente, via código, tornando possível um hardware adaptável a mais de uma técnica de modulação. A essa técnica, deu-se o nome de RDS (Rádio definido por Software) ou SDR (*Software Defined Radio*).

1.2 A Tecnologia Rádio definido por Software

A tecnologia RDS foi criada para possibilitar um hardware capaz de realizar uma variedade de tarefas sem necessidade de mudanças físicas. Para tornar isso possível, funções antes realizadas pela camada física são atribuídas a um software, que é reprogramável, muitas vezes em tempo real. Sendo assim, há a necessidade de uma maior flexibilidade do hardware, para que seja capaz de realizar novas tarefas requeridas por alterações no software (ITO; SCHENA, 2006).

Os padrões de comunicação sem fio estão sempre em constante mudança, visando maior eficiência, seja com mudanças de frequência ou protocolos de operação ou tipo de modulação. Com isso, é usual que equipamentos rapidamente tornem-se obsoletos e caiam em desuso, requerendo substituição, o que acarreta custos para as operadoras, modificando seu hardware para se adaptar às novas modalidades de operação. Nesses casos, especialmente, nota-se um grande potencial, inclusive econômico, da aplicação da tecnologia RDS, visto que após o investimento inicial em hardware, novas mudanças requerem apenas atualização de software (OLIVEIRA, 2014).

Idealmente, a arquitetura de um RDS seria composta apenas por antenas, conversores analógico/digital e uma unidade de processamento. Apesar dessa arquitetura não ser tão simples na situação real, a tecnologia RDS proporciona um sistema de baixo custo quando se deseja um sistema multi-modo e multi-banda, graças ao hardware versátil e simplificado.

A programação do software de um RDS pode ser feita de diversas maneiras e em diversas linguagens, especialmente em sistemas embarcados. No caso específico abordado neste projeto, o processador possui seu próprio sistema operacional e é capaz de executar um software que possibilita uma programação mais simplificada, por linguagem de diagrama de blocos. Tal programa é conhecido como GNU Radio e possui integração com o hardware utilizado neste projeto.

1.3 A Ferramenta GNU Radio

O GNU Radio é uma ferramenta livre e versátil capaz de se encarregar da programação de diversos aplicativos relacionados à comunicação sem fio. Trata-se de um ambiente de programação em linguagem gráfica ou textual capaz de simular, prototipar e executar programas de diversos níveis de complexidade. Por ser um software livre e aberto possui também uma vasta comunidade de desenvolvedores, sempre criando novas funcionalidades e blocos específicos (GNU RADIO, 2013).

As aplicações do GNU Radio são primariamente escritas em Python ou em blocos gráficos representativos de código Python, utilizando C++ no caminho crítico de processamento de sinal. Com isso, o programador é capaz de implementar sistemas de rádio de alto desempenho com tempo real através de uma plataforma simples e rápida (GNU RADIO, 2013).

1.4 Sistemas Operacionais de Tempo Real

Por muito tempo, microprocessadores e microcontroladores eram dependentes de programação. O usuário deveria escrever o código completo do que seriam as funções do mesmo e gravar em sua memória antes que pudesse ser utilizado. Com os avanços na tecnologia de microcontroladores, tem-se tornado comum encontrar no mercado placas de desenvolvimento que vem equipadas com um sistema operacional. Tal sistema pode ser visto como uma abstração do hardware, fazendo um papel intermediário entre o software e os componentes físicos do computador ou placa (PUHLMANN, 2014).

As funções do sistema operacional são diversas, dentre elas o gerenciamento dos recursos providos pelo hardware à sua disposição e fazer a interface humano-máquina, seja ela gráfica ou textual. Porém, esse gerenciamento nem sempre é adequado a qualquer caso. Exemplificando, se um paciente está na UTI e sofre uma variação brusca no batimento cardíaco, é necessario que o sistema que o monitora soe um alarme avisando os médicos o quanto antes, um atraso ou falha pode significar a perda de uma vida.

Nesses casos, onde os estímulos, tanto externos quanto internos, são críticos e devem ser tratados dentro de uma janela de tempo bem definida, é crucial que o sistema utilizado seja um sistema de tempo real. Esse sistema nada mais é do que um caso especial dos sistemas operacionais comuns, que funciona utilizando restrições de tempo em suas respostas ao ambiente e gerenciando suas tarefas da forma mais eficiente possível (PRADO, 2010).

No caso do SDR, um fluxo constante de dados é captado nas antenas e deve ser tratado de forma a recuperar a mensagem na saída. Se esses dados forem tratados em partes ou em ordem incorreta, o resultado gerado será diferente e incondizente com a mensagem desejada. Sendo assim, é interessante a aplicação desse tipo de sistema de tempo real, visando dar maior prioridade de processamento ao que nos interessa e deixando de lado tarefas menos importantes.

1.5 Objetivos deste projeto

Os objetivos deste projeto envolvem a colocação em prática do conceito RDS, utilizando uma plataforma embarcada como unidade de processamento. Para isso foram utilizados o USRP2 (*Universal Software Radio Peripheral*) como antena e pré-processador de sinal, e o *Raspberry*

Pi como unidade embarcada de processamento. Em uma primeira etapa, o software proposto é utilizado em conjunto com um teste simples, para comprovação de compatibilidade e capabilidade, na seção 5.1. Em seguida, é feita a implementação de um receptor FM simplificado, já contando com entrada de dados do mundo real através do USRP2, na seção 5.2. Posteriormente, comprovada a eficácia do hardware escolhido, foi realizada uma análise de desempenho em um *Raspberry Pi* com o sistema operacional modificado, de forma a trabalhar como um sistema de tempo real, a fim de averiguar as possíveis vantagem da aplicação do mesmo no *hardware* proposto, na seção 5.3. Por fim, na seção 5.4, é feita uma aplicação de conceito colocando-se o receptor FM implementado na primeira parte do projeto para trabalhar no sistema adaptado com *Kernel* de tempo real.

2 MODULAÇÃO DE ONDAS DE RÁDIO FREQUÊNCIA

O processo de transmissão de sinais de rádio inicia-se em um estúdio, através da gravação de sinais de áudio. Sinais de áudio são ondas mecânicas das mais variadas frequências. Dentre as frequências de ondas mecânicas, o ouvido humano reconhece aquelas que encontram-se na faixa de 20 – 20000 Hz. Porém, não se pode transmitir sinais de áudio via ondas de rádio nessas frequências, pois incorreriam em diversas dificuldades, como a interferência com diversos outros sinais existentes no canal de transmissão.

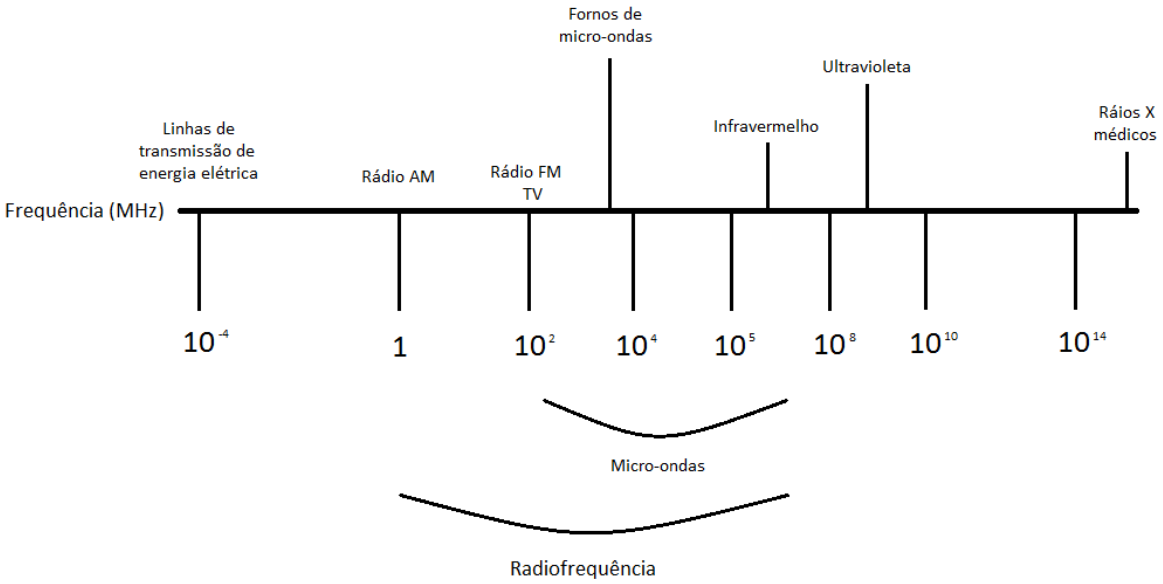
Todos os sinais de áudio possuem a mesma faixa de frequência, logo, o receptor não saberia separar o sinal desejado no meio de um emaranhado de sinais. Aliado a isso, quando diversos sinais diferentes passam pelo mesmo canal, há interferências construtivas e negativas de sinal que podem acarretar em drásticas reduções de alcance. Para resolver esses, e outros, problemas, foi criada a técnica da modulação.

A técnica de modulação consiste na variação de uma onda padrão, denominada portadora, acompanhando o sinal de informação. Os parâmetros a serem alterados podem ser a amplitude, frequência e o ângulo de fase da portadora (MUNIZ, 2010).

Em outras palavras, a modulação utiliza-se de uma onda conhecida, numa frequência desejável, para carregar as informações contidas no sinal de mensagem. Desta forma, voltando às ondas de áudios citadas anteriormente, pode-se transmitir os sinais, originalmente produzidos numa faixa fixa de frequência, em uma frequência escolhida no momento da modulação.

A Figura 1 apresenta o espectro de radiação eletromagnética e como é feita a sua divisão. Através dela, vemos que as rádios FM, tem sua frequência girando em torno de 100 MHz, por exemplo, com diferentes estações possuindo diferentes frequências. Isso é possível graças à modulação, onde, escolhendo-se a frequência da portadora, pode-se escolher também a frequência do sinal modulado e posicioná-lo na faixa do espectro conveniente.

Figura 1 - Espectro de Radiação Eletromagnética



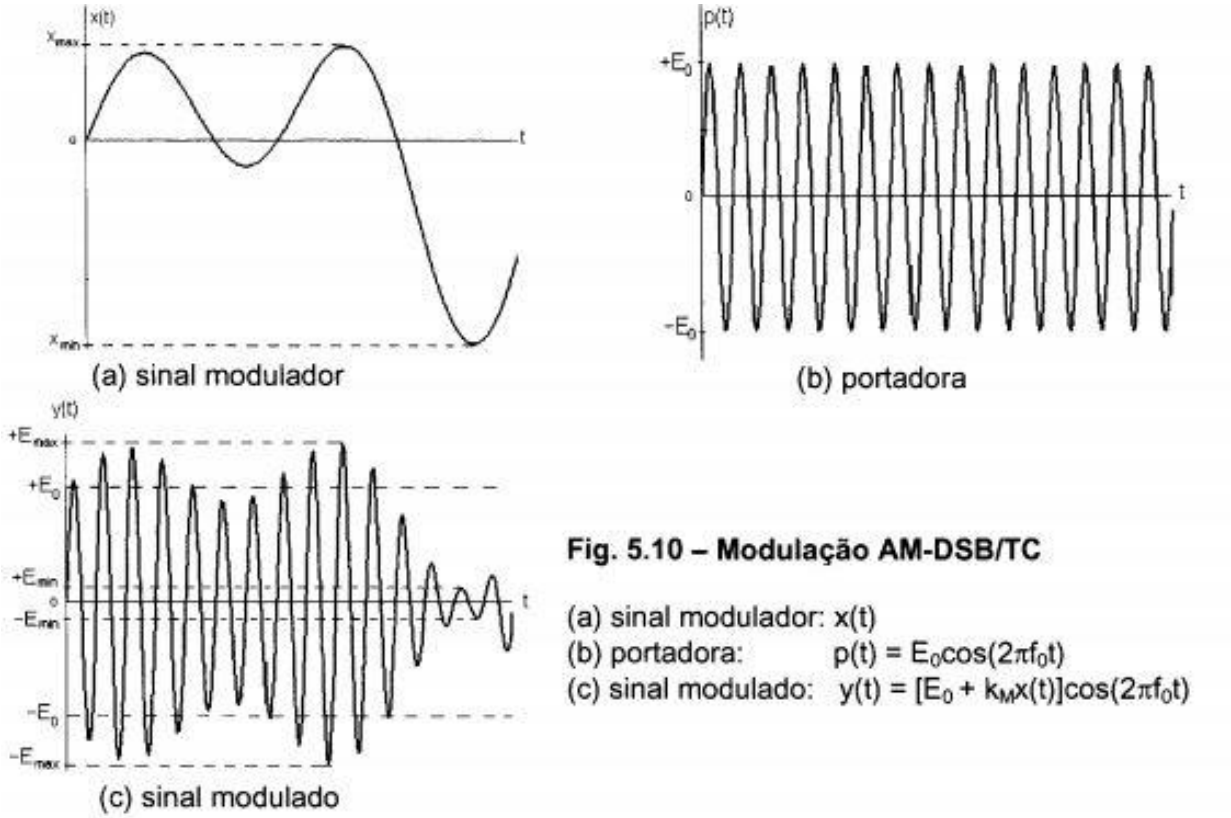
Fonte: Produção do próprio autor.

Como a modulação altera alguns parâmetros dos sinais, é necessário que o receptor conheça a modulação usada, para que ele possa, então, realizar o processo inverso, retirando a informação útil carregada por tal sinal.

Existem diversos modos de modulação. No caso das ondas de rádio, os mais utilizados são AM (*Amplitude Modulation*) e FM (*Frequency Modulation*), que se utilizam, respectivamente, de modulação de amplitude e de frequência.

A Figura 2 mostra, como exemplo, uma técnica de modulação de amplitude, denominada AM-DSB/TC (*Amplitude Modulated - Dual Side-band Transmitted Carrier*). O termo refere-se ao fato de que ela utiliza-se de uma dupla faixa lateral. Essa é uma das técnicas mais antigas, cujo resultado final é uma onda que possui a mesma frequência da portadora, mas com a forma modificada de acordo com o sinal de mensagem. Escolhe-se uma portadora na frequência desejada para a portadora e a mesma é utilizada para carregar os dados do sinal modulador. Como resultado, é obtido o sinal modulado mostrado (MUNIZ, 2010).

Figura 2 - Modulação de Amplitude



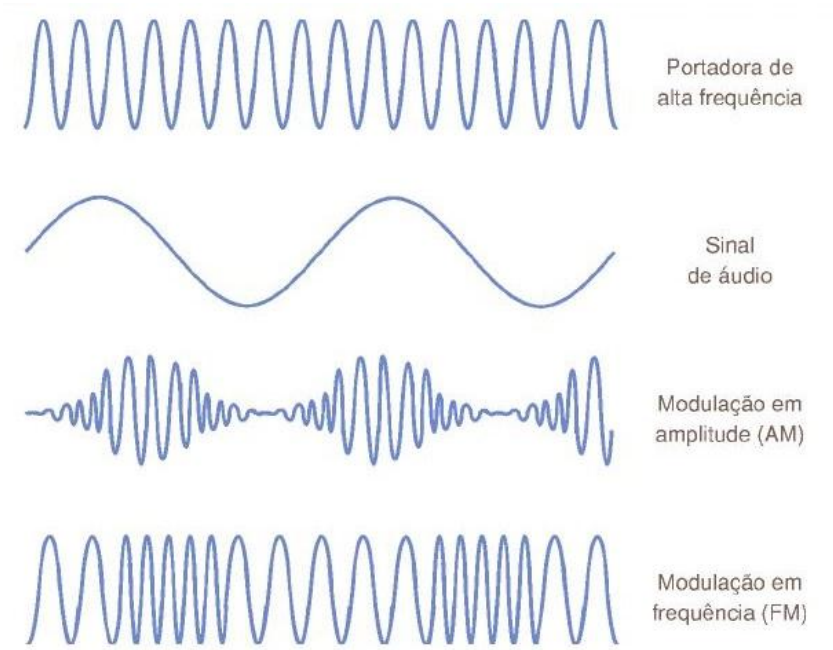
Fonte: MUNIZ, 2002.

É possível ver que, traçando uma curva que liga as extremidades superiores ou inferiores da onda demodulada, formando o que chama-se “envelope”, é obtido o sinal modulador original. Essa é a forma de demodulação de um sinal AM-DSB/TC, utilizando-se de um circuito de detecção de envelope.

No caso dos sinais FM, a amplitude do sinal resultante mantém-se constante e os dados são transmitidos através de uma desvios na frequência da portadora. A Figura 3 mostra um comparativo dos dois modos de modulação para um mesmo sinal senoidal de áudio e a mesma portadora.

A modulação FM apresenta diversas vantagens com relação à AM, para redes metropolitanas de menor alcance, como uma maior qualidade de áudio e maior imunidade a ruído, devido à sua amplitude constante. Essas e outras vantagens tornaram o modo FM mais frequentemente utilizado pelas estações de rádiodifusão de pequenas distâncias, na atualidade (CAMPOS, 2002). Neste projeto trabalhamos apenas com a modulação FM.

Figura 3 - Comparativo de Modulações de Amplitude e de Frequência



Fonte: TÉCNICAS, 2013.

Com os diversos modos de modulação utilizados atualmente, faz-se necessário que receptores adaptem-se a essas técnicas e sejam capazes de interpretar os sinais e decodificar as mensagens enviadas. Para dispositivos analógicos, com demodulação via hardware, é necessário um hardware específico para cada modo implementado, ocupando espaço e aumentando custos.

Para lidar com essa nova demanda, foi criada a tecnologia de rádio definido por software. Ela é capaz de interpretar os sinais, de diferentes modos, recebidos em sua entrada com um único hardware, genérico, com possibilidade de adição de funções, via programação.

3 A TECNOLOGIA RDS

Um receptor de sinais de rádio possui a capacidade de decodificar os sinais em sua entrada e interpretá-los. Em geral, em rádios controlados por software, utiliza-se um hardware que é capaz de realizar algumas etapas dessa decodificação, simplificando o trabalho do software que deve tratá-lo. Os rádios definidos por software diferem, porém, de rádios controlados por software.

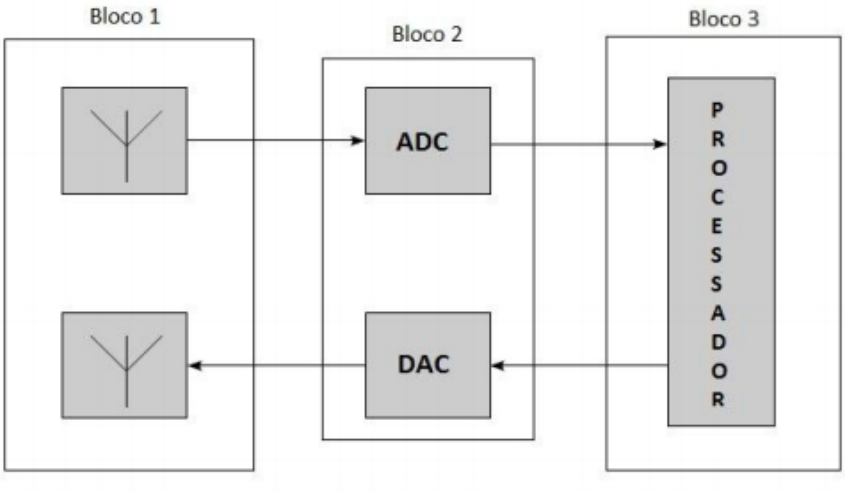
Os rádios controlados por software são capazes de realizar parte de seu processamento de sinais via código, porém, caso alguma alteração de parâmetros de operação seja necessária, o hardware também deverá ser adaptado em conjunto. Isso se dá devido ao fato de que os rádios controlados por software possuem hardware dedicado com propósito específico para o fim desejado em seu momento de projeto.

Os rádios definidos por software, porém, são projetados com o objetivo de serem mutáveis sem alterações físicas no equipamento. Para que isso seja possível, todo o hardware do equipamento deve ser projetado de forma geral, que atenda aos mais diversos propósitos possíveis. Para se obter um hardware geral, os RDS repassam a maior parte possível do tratamento de sinais para o domínio digital.

3.1 Hardware de um RDS

A arquitetura ideal de um RDS está mostrada na Figura 4. Tal arquitetura é baseada na conversão imediata dos sinais analógicos em digitais, através de um ADC (*Analog Digital Converter*) na entrada e um DAC (*Digital Analog Converter*) na saída. Com essa estrutura, o equipamento seria capaz de funcionar como um equipamento de uso geral, limitado apenas pela capacidade de processamento do processador e da precisão dos conversores Analógico/Digital.

Figura 4 - Arquitetura ideal de um rádio definido por software



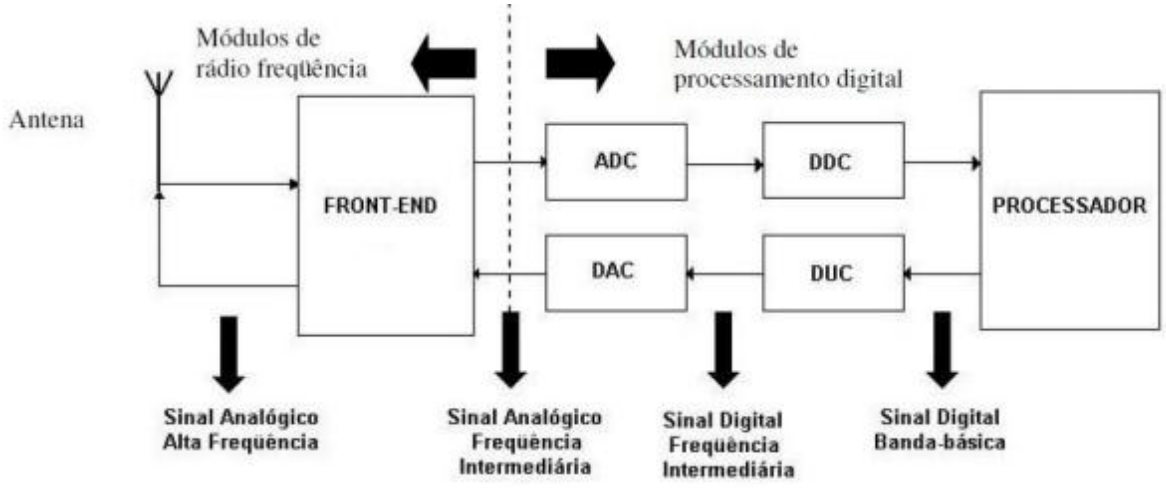
Fonte: JANSON, 2012.

Entretanto, as limitações citadas tornam essa arquitetura impraticável com elementos reais. Isso se dá devido às limitações tecnológicas dos componentes em questão. À medida que a demanda e requerimentos de precisão crescem, exige-se mais capacidade dos componentes. Com isso, atinge-se os limites tecnológicos de largura de banda e taxa de amostragem dos conversores. Esses requerimentos crescem conforme o aumento da frequência do sinal a ser recebido ou transmitido, tornando a aplicação inviável com os filtros realizáveis.

Aliado a essa limitação, caso fosse possível implementar tais conversores, seria enviado um grande volume de dados para o processador que, na maioria dos casos, deve processar em tempo real, ou seja, o tempo de processamento deve ser menor que o tempo de chegada de uma nova amostra. Sendo assim, a exigência de velocidade de processamento torna a prática de tal arquitetura inviável.

Para solucionar os problemas descritos, utiliza-se de um pré-processamento, realizado via hardware físico. Tal hardware, para se adequar à tecnologia RDS, deve ser o mais genérico possível, de modo que não limite sua mutabilidade por recodificação. É então proposta uma nova arquitetura, denominada arquitetura RDS real, ilustrada na Figura 5.

Figura 5 - Arquitetura real de um rádio definido por software



Fonte: JANSON, 2012.

A Figura 5 mostra uma arquitetura um pouco mais complexa que a ideal, mas que resolve os problemas descritos anteriormente. Na arquitetura ideal, o *front-end* seria composto apenas pelos ADC e DAC. Numa arquitetura real, o *front-end* torna-se um novo módulo com a função de realizar um pré-tratamento no sinal de entrada, ou tratamento do sinal de saída, de forma a resolver os problemas dos conversores. Esse módulo é responsável por uma amplificação do sinal, por controlar o ganho e deslocar o sinal para uma frequência intermediária (na recepção) ou para a frequência original do sinal (na transmissão), além de realizar o *anti-aliasing*. O deslocamento para uma frequência intermediária faz com que a parte subsequente do *hardware* seja capaz de tratar diversos sinais de entrada sem necessidade de alterações físicas e permite que os filtros necessários à aplicação tenham requisitos menores e realizáveis. O filtro *anti-aliasing* trata-se de um filtro passa baixas que remove componentes de frequência acima da taxa de amostragem, eliminando sobreposição no espectro amostrado (JANSON, 2012).

Com a adição desse módulo, resolvem-se os problemas físicos dos conversores analógico/digital. Para resolver os problemas do processador, porém, são necessários novos módulos, instalados entre a conversão analógico/digital e o processador. Esses módulos são os DDC (*Digital Down Converter*) e o DUC (*Digital Up Converter*).

Os dois módulos possuem função inversa, para serem usados na recepção (DDC) e transmissão (DUC). Como neste trabalho trataremos apenas da recepção, será mostrada apenas a arquitetura de um DDC.

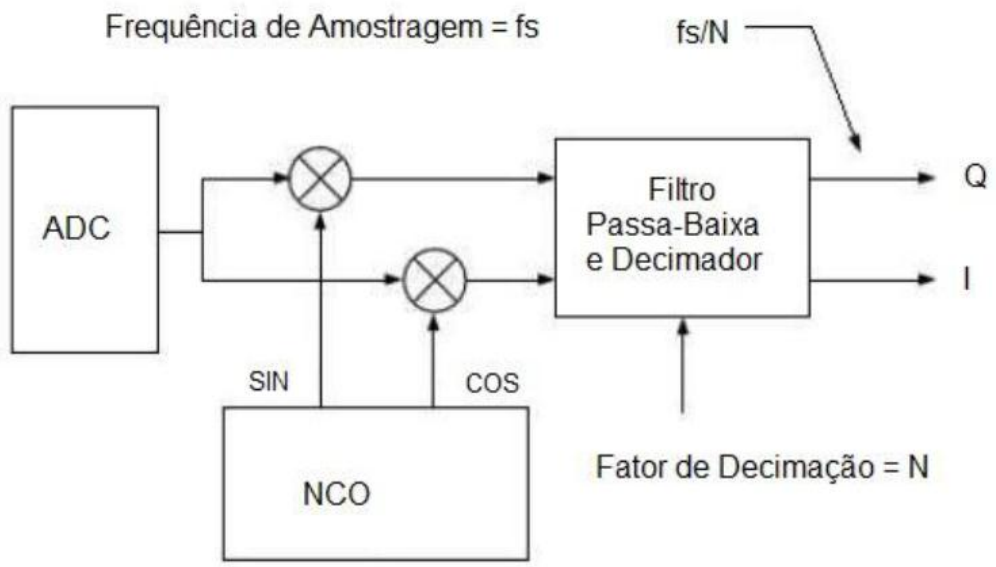
O DDC tem por função principal a diminuição do fluxo de dados enviados ao processador, reduzindo assim a sua carga de processamento. Para realizar tal operação, ele capta o sinal de saída do conversor, já digitalizado e em banda intermediária, e o transporta para a banda base.

No DDC, duas senóides são geradas, por um oscilador próprio, controlado pelo processador, em frequência intermediária em fase e em quadratura e utilizadas na decomposição do sinal de entrada. Essa decomposição gera componentes complexos em sua saída que passam por um filtro passa-baixas, seguido por um redutor de taxa de amostragem (decimador), uma vez que a largura de banda do sinal foi reduzida, que seleciona uma faixa de frequência de interesse. Com isso, há uma grande redução na banda e na quantidade de amostras que o processador deve tratar. A Figura 6 mostra um esquemático da arquitetura de um DDC (JANSON, 2012).

O bloco NCO trata-se de um Oscilador numericamente controlado (*Numerically Controlled Oscillator*), controlado pelo próprio processador do sistema, capaz de gerar sinais de frequência intermediária em fase e em quadratura, para a decomposição. O sinal digital da saída do conversor ADC é multiplicado por esses sinais de frequência intermediária, defasados de noventa graus, gerando dois novos sinais, em fase e em quadratura.

O sinal complexo de saída do DDC serve de entrada do processador que tem agora um fluxo de dados adequado para sua capacidade de processamento. O processador é responsável pelo tratamento dos dados recebidos. Essa etapa envolve a demodulação dos sinais recebidos, tratamento dos sinais, de áudio no caso, e preparação do sinal para utilização na saída a ser utilizada.

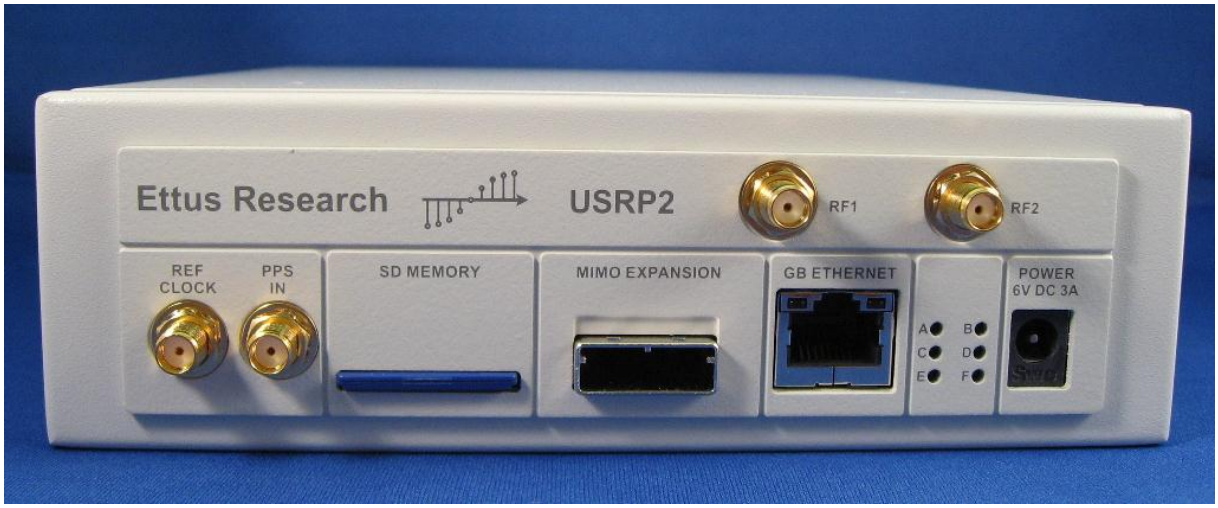
Figura 6 - Arquitetura de um DDC



Fonte: JANSON, 2012.

Neste projeto, utilizou-se o USRP2, disponibilizado pelo Laboratório de Telecomunicações (LabTel), da Universidade Federal do Espírito Santo (UFES). Trata-se de um dispositivo completo, produzido pela *Ettus Research LLC*, posteriormente adquirida pela *National Instruments*, capaz de realizar as funções requeridas do hardware de um RDS.

Figura 7 - Foto Ilustrativa do USRP2



Fonte: UCLA CORES, 2011.

O USRP2 é composto de uma placa mãe, placas filhas e antenas. As placas filhas são responsáveis pelo *front-end*, manipulando os sinais recebidos na antena para que estejam

adequados a serem tratados pela placa mãe. Para isso, elas selecionam uma certa faixa de frequência e deslocam os dados obtidos para uma banda intermediária, possibilitando que a placa mãe trabalhe com as mais variadas frequências, trocando apenas a placa filha. Existem diversos modelos de placas filhas e cada uma é adaptada para uma diferente faixa de frequências, conforme o Quadro 1 (ETTUS RESEARCH, 2014).

Quadro 1 - Placas Filhas disponíveis no Mercado

Modelo da Placa Filha	Função	Faixa de Frequência
BasicRX	Receptora	1 – 250 MHz
BasicTX	Transmissora	1 – 250 MHz
LFRX	Receptora	DC – 30 MHz
LFTX	Transmissora	DC – 30 MHz
TVRX2	Receptora	50 – 860 MHz
DBSRX2	Receptora	800 – 2300 MHz
WBX-40	**Transceptora**	**50 – 2200 MHz**
WBX-120	Transceptora	50 – 2200 MHz
SBX-40	Transceptora	400 – 4400 MHz
SBX-120	Transceptora	400 – 4400 MHz
CBX-40	Transceptora	1200 – 6000 MHz
CBX-120	Transceptora	1200 – 6000 MHz
UBX-40	Transceptora	10 – 6000 MHz
UBX-160	Transceptora	10 – 6000 MHz
TwinRX	Receptora	10 – 6000 MHz

Fonte: ETTUS RESEARCH, 2016.

As placas diferenciam-se não só pela faixa de frequência, como também pela função transmissora, receptora ou transceptora, bem como ganhos e largura de banda. Cada placa mãe pode trabalhar com até duas placas filhas, desde que sejam uma receptora e uma transmissora, ou com apenas uma placa transceptora. Sendo assim, é importante que a(s) placa(s) escolhida(s) seja(m) ideal(is) para o que se quer realizar com o equipamento. No caso deste projeto, foi utilizada a placa WBX-40, disponível em laboratório.

As placas filhas se conectam à placa mãe, que é responsável pelas conversões analógico-digital e digital-analógico e por realizar as funções do DDC e DUC. Como as placas filhas se encarregam de deslocar o sinal para uma banda intermediária, independente de qual era a frequência original do sinal, o hardware da placa mãe pode ser único, genérico e especificado para tal frequência intermediária. A placa mãe é composta de um conversor A/D de 14 bits, com taxa de amostragem de 100MS/s, um conversor D/A de 16 bits, com taxa de amostragem de 400MS/s e uma FPGA (*Field Programmable Gate Array*) Xilinx Spartan XC3S2000 com conversores DDC e DUC com taxas de decimação e interpolação programáveis (ETTUS RESEARCH, 2014).

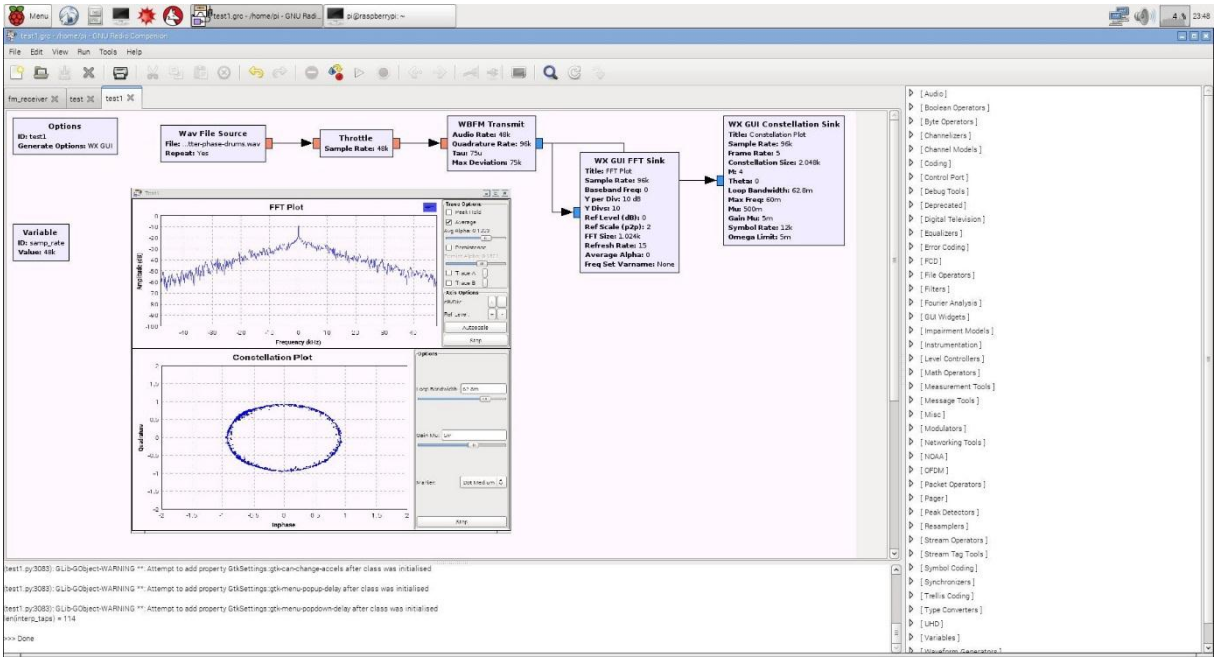
O USRP2 conecta-se com o mundo exterior através de uma saída *Gigabit Ethernet*, presente na placa mãe. A conexão se dá através de um cabo UTP (*Unshielded Twisted Pair*), com conectores RJ45 e permite um fluxo de até 1 Gb/s, quando em comunicação bidirecional. A unidade de processamento no caso deste projeto, foi o *Raspberry Pi 2 B*, um potente micro processador equipado com o *Debian*, uma distribuição simplificada do *Linux*.

3.2 O Software de um RDS

Esta parte do RDS é responsável pela maior parte do processamento de sinal. O sinal entregue ao processador recebe apenas um pré-tratamento para redução de fluxo de dados, para diminuir os requisitos de processamento. Com isso, todo o tratamento do sinal, desde filtragem, interpolação, demodulação à geração de sinais de saída de áudio ou texto, entre outros, é função exclusiva da unidade de processamento. Existem diversos meios de realizar este tratamento, desde linha de código à utilização de softwares com linguagem gráfica, através de blocos. Para o caso deste projeto, utilizou-se o GNU Radio.

Essa escolha foi feita devido à sua simplicidade de utilização, integração com o hardware escolhido para o projeto e possibilidade de simples implementação de qualquer função que eventualmente não fosse nativa do programa. O GNU Radio é uma ferramenta livre e gratuita que funciona em distribuições do *linux*, como a presente no *Raspberry Pi 2 B*, que possui uma interface simplificada de programação.

Figura 8 - Interface Gráfica do GNU Radio



Fonte: Produção do próprio autor.

Na Figura 8 pode-se ver a interface do software. Trata-se de um ambiente como um quadro branco para o qual blocos são arrastados. Tais blocos são representações gráficas de abstrações de código em *Python* ou C++. A barra localizada à direita da área de trabalho do programa é utilizada para pesquisar os blocos desejados. O GNU Radio vem equipado com uma grande quantidade de blocos em diversas áreas de *Digital Signal Processing* (DSP), ou Processamento Digital de Sinais (PDS), aptos a realizarem as mais diversas operações.

Quando a programação é realizada, o GNU Radio gera um arquivo *Python* que ele mesmo executa e gera as saídas requeridas. O software também fornece funções que permitem analisar se as saídas estão sendo geradas de acordo com o esperado, produzindo gráficos em tempo real do que está passando pelo ponto de inserção do bloco de análise.

Também na Figura 8, observa-se um exemplo simples de modulação FM associado a dois blocos de análise, realizando a transformada rápida de fourier e o diagrama de constelação da saída, modulada. Através desses gráficos, pode-se ver que a modulação foi feita corretamente e que o sinal está sendo gerado na faixa de frequência programada.

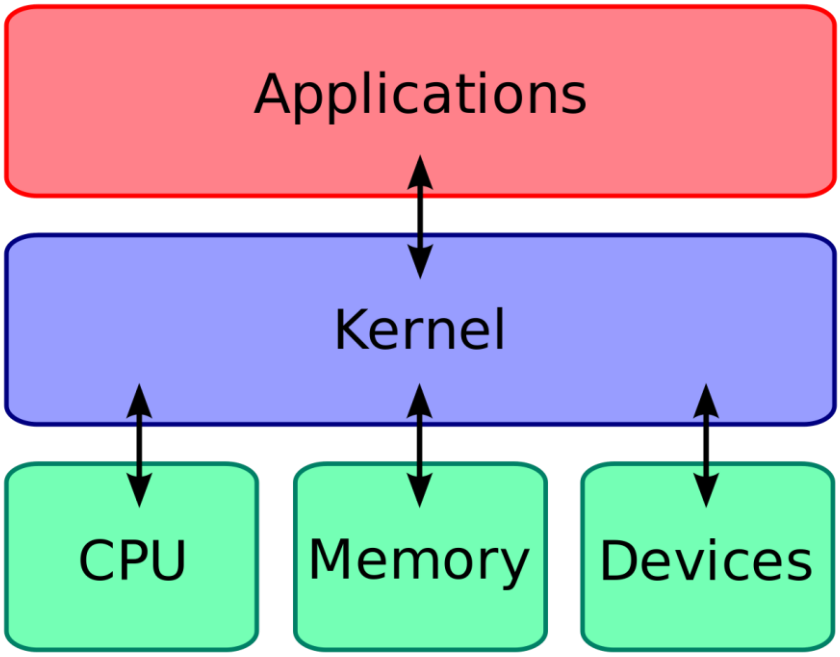
4 SISTEMA OPERACIONAL COM KERNEL DE TEMPO REAL

Sistemas operacionais, em geral, são softwares capazes de realizar a abstração do hardware, para o usuário. Dentre suas diversas funções estão o gerenciamento dos recursos do sistema, definição de prioridades para o processador e a interface gráfica ou textual, utilizada pelo usuário para interagir com o hardware.

4.1 O *Kernel*

Cada sistema operacional possui um *Kernel*, que é o núcleo do mesmo. O *Kernel* possui controle total sobre todo o hardware e software envolvido. É ele quem realiza as principais interações entre aplicações, processador, memória e dispositivos. A Figura 9 demonstra graficamente a função do *Kernel* dentro do sistema operacional, se comunicando tanto com as aplicações de interface humana, quanto com as camadas de *hardware* internas e realizando a abstração, de modo que o usuário seja capaz de se comunicar com as funções primitivas através de linguagens mais simplificas, usualmente gráficas.

Figura 9 - Blocos componentes de um Sistema Computacional



Fonte: JOSH, 2015

O comportamento do *Kernel* é definido no momento da compilação do sistema operacional mas nem sempre se adequa ao que se quer realizar com o mesmo. Isso se dá porque o mesmo possui

prioridades de execução que nem sempre estão em harmonia com as necessidades do usuário. Com isso, tarefas que exigem um tempo de resposta mais estável podem sofrer com falta de prioridade de processamento, enquanto tarefas menos importantes ocupam todo o tempo do processador. Para solucionar esses e outros problemas, foram criados os sistemas de tempo real. No caso das distribuições do linux, existe a possibilidade de implementação de tal *Kernel* através de um simples *patch*, que é um arquivo contendo código a ser inserido em um programa para aprimorá-lo ou corrigir erros.

4.2 O *Kernel* de Tempo Real

Tarefas computacionais são iniciadas através de eventos, ou estímulos, que podem ser internos ou externos. Dado o estímulo, é realizado o processamento adequado e é gerada uma saída. Quando o sistema de controle das tarefas está funcionando em tempo real, ele impõe restrições de tempo para a realização de cada etapa. Sendo assim, sempre que necessita-se realizar tarefas onde o tempo de execução é crítico, há uma grande vantagem na utilização de sistemas de tempo real. Vale frisar que um sistema de tempo real não é necessariamente mais rápido, mas seu comportamento determinístico traz consigo diversas vantagens (PRADO, 2010).

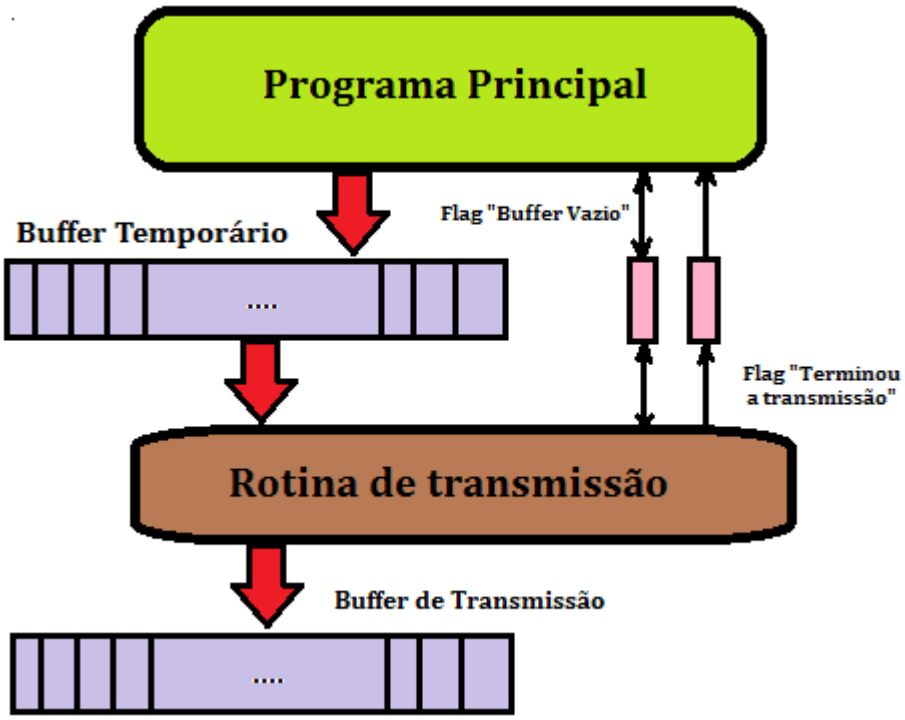
Algumas das principais particularidades de um sistema de tempo real são: rotinas de processamento, em geral, curtas e especializadas, para executar a sua tarefa o mais rápido possível, forte paralelismo na execução de atividades e a definição de prioridades para as tarefas. Devido a esses motivos, as atividades especializadas em paralelo devem ter alguma forma de comunicação para que sejam informadas de tarefas concluídas. Essa comunicação é feita através de *flags* (semáforos) e/ou *buffers* (filas) (PUHLMANN, 2014).

Para melhor ilustrar o funcionamento do sistema, a Figura 10 traz um exemplo de aplicação do sistema de tempo real. Trata-se de um sistema de transmissão de mensagens via USB. Em um sistema convencional, de maneira simplificada, o sistema monta a mensagem e chama a rotina de transmissão. Apenas após a mensagem ser transmitida o programa retorna às suas atividades normais.

No caso do sistema de tempo real, o programa principal checa a disponibilidade do *buffer* e, caso esteja vago, envia dados para o mesmo. Ao finalizar o salvamento de dados, muda a sinalização para informar às outras rotinas que o *buffer* está em uso. Após essa modificação na

sinalização, a rotina de transmissão já sabe que há uma nova mensagem a ser transferida e começa a ser executada, como uma interrupção.

Figura 10 - Diagrama Simplificado da Transmissão de Caracteres em Tempo Real



Fonte: PUHLMANN, 2014.

Os caracteres, então, vão sendo transferidos um a um pela rotina de transmissão. Como a mesma é executada como interrupção, as demais atividades do sistema que independem desse hardware específico podem executar normalmente. Ao finalizar o envio da mensagem, a rotina modifica a sinalização de "*buffer* vazio", para indicar que ele já está disponível para novos dados e a sinalização de "Terminou a transmissão", para que o programa principal possa enviar novas mensagens, caso haja alguma.

4.3 Escalonamento de Tarefas

O escalonamento de tarefas é o algoritmo que permite que o *Kernel* determine quando alguma tarefa mais importante precisa ser realizada. Para que o *Kernel* seja capaz de realizar o escalonamento, é necessário que ele assuma o controle do processador. Isto ocorre através de um destes eventos:

- Quando ocorre uma troca de mensagens entre tarefas, o *Kernel* assume o controle do sistema para preparar a mensagem para envio;
- Se uma tarefa requisita um *delay*, ou tempo de espera, o controle é dado ao *Kernel* até que o tempo se finalize, possiblitando o escalonamento neste intervalo;
- Acontece um evento, que pode ser um estímulo externo, esperado por uma tarefa de maior prioridade, forçando que o *Kernel* interrompa a execução da tarefa atual;
- Quando uma interrupção do *timer* do sistema acontece e é tratada pelo *Kernel*. Neste caso o *Kernel* coloca a tarefa atual em espera e realiza o escalonamento para uma tarefa mais prioritária. Sistemas com essa funcionalidade são também conhecidos como sistemas preemptivos.

Em todos os casos citados o escalonamento é realizado quando o *Kernel* recebe de volta o controle do sistema. Para que o escalonamento seja realizado, são mantidas tabelas de controle. Cada tarefa tem uma prioridade definida que pode ser alterada de acordo com as necessidades do usuário. O *Kernel* também mantém uma tabela com estados atuais de todas as tarefas em execução, denominada *Task Control Block* (TCB) (PRADO, 2010).

Os dados armazenados no TCB também são chamados de contexto, pois envolvem o estado atual da tarefa e o conteúdo de alguns registradores salvos no momento de suspensão da tarefa. Isso permite ao sistema não só saber em que ponto da tarefa estava, como também retomar de onde havia parado sem perda de dados armazenados em registradores temporários. Sempre que o escalonamento é realizado, o TCB é atualizado. Este processo de mudança de tarefa também é conhecido como mudança de contexto (PRADO, 2010).

Devido ao escalonamento de tarefas, o sistema de tempo real garante que tarefas com alta prioridade serão tratadas com baixo tempo de resposta. Além disso, dada a tabela de prioridades e o salvamento de estados, o sistema, como um todo, torna-se bastante previsível em seu funcionamento. Essa é uma grande vantagem quando trata-se de tarefas onde o tempo de resposta é crítico para seu funcionamento e pode acarretar em falhas ou respostas indesejadas.

5 METODOLOGIA E ETAPAS DE DESENVOLVIMENTO

O desenvolvimento deste trabalho é composto de três etapas principais. Em um primeiro momento, objetivava-se desenvolver um programa capaz de colocar em prática a teoria do Rádio definido por Software em uma plataforma embarcada, avaliando limitações de integração entre o *hardware* e o *software*, com inclusão da dados externos num segundo experimento.

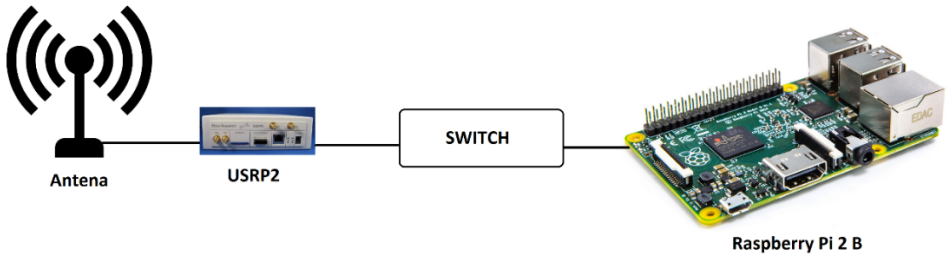
Em seguida, o sistema operacional do *Raspberry Pi 2 B* foi adaptado, atreavés da implementação de um *Kernel* de tempo real e testes de desempenho foram realizados, de modo a avaliar o comportamento de tal sistema e os tempos de resposta para processos com alta prioridade. Finalmente, num último momento, um experimento, visando uma possível aplicação de tal *hardware* realiza-se com a implementação do receptor FM em um sistema de tempo real embarcado.

Visto que a capacidade de processamento do sistema embarcado é reduzida em relação à de um computador, era necessário provar que o sistema era capaz de desenvolver a tarefa. Para tal, foi utilizado um *Raspberry Pi 2 B*, conectado a um kit USRP2. Devido à limitação da placa de rede do *Raspberry Pi*, que não possui conexão *Gigabit Ethernet*, foi necessária a adição de um *Switch* intermediário, que era responsável pela troca de dados entre as duas partes do *hardware*. Vale ressaltar que, apesar da placa de rede limitada, esta não seria um gargalo do sistema, visto que era mais do que capaz de suportar toda a troca de dados necessária.

O diagrama representativo da montagem feita encontra-se na Figura 11. A montagem foi realizada dentro do Núcleo de Estudos em Redes Definidas por Software (NERDS), na UFES.

A frequência de operação para os testes girava em torno dos 100 MHz, por visar estações metropolitanas de rádio FM. Sendo assim, a placa filha WBX-40 foi utilizada, disponível em laboratório. As características do hardware do Raspberry Pi 2 B são apresentadas no Quadro 2.

Figura 11 - Diagrama da montagem



Fonte: Produção do próprio autor.

Quadro 2 - Especificações do *Raspberry Pi 2 Model B*

Dispositivo	Processador	Frequência do Proc.	Memória RAM
Raspberry Pi 2 B	Quad-core ARM Cortex-A7	900 MHz	4 GB

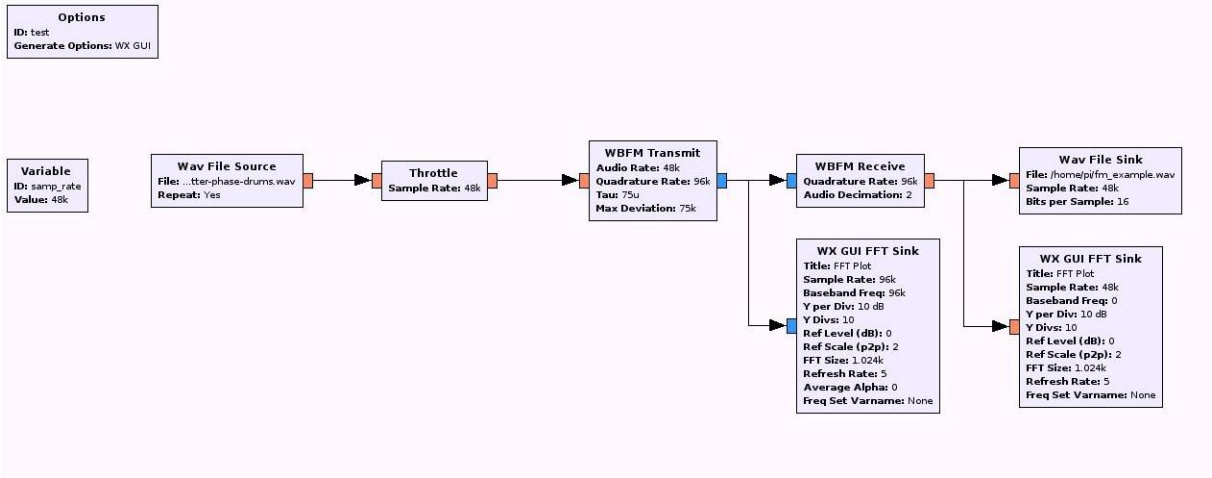
Fonte: Produção do próprio autor.

5.1 Teste Inicial de Hardware e Software

Antes de aplicar a montagem completa do sistema, realizou-se um pequeno teste envolvendo o GNU Radio em sua versão disponibilizada para dispositivos embarcados. Esse teste tinha como objetivo averiguar as condições e viabilidade de realização da montagem completa. Dentre as características desejadas na realização do teste, estavam: a qualidade da saída do som, para comparação com resultados posteriores, disponibilidade de todos os blocos necessários à programação final e capacidade de geração de ferramentas gráficas de análise de resultados.

Para realizar tal teste, foi feita uma simples montagem, representada na Figura 12. O programa consiste de um arquivo de áudio em constante replay, enviando um fluxo constante de dados para um modulador FM. Logo em seguida, um demodulador é colocado, desconsiderando perdas no canal para efeito de teste. O sinal de saída, como áudio demodulado, é então levado a um bloco que tem por função gravar todos os dados recebidos em formato de áudio.

Figura 12 - Programa simplificado para teste



Fonte: Produção do próprio autor.

O bloco *Wav File Source* é o bloco que utiliza um arquivo de áudio presente no sistema de arquivos e o transmite como um sinal analógico, amostrado, para o próximo bloco, seguindo as setas de fluxo. A modulação e domodulação são executadas, logo em seguida, pelos blocos *WBFM Transmit* e *WBFM Receive*, respectivamente. *WBFM* é a sigla para *Wideband Frequency Modulation*, ou modulação de frequência de banda larga e é a modulação mais comumente utilizada em estações de rádio. O bloco transmissor permite a configuração da frequência da portadora que, nesse caso, foi de 98kHz, duas vezes a taxa de amostragem, de forma a não perder dados na transmissão. Adicionalmente, é possível ajustar *Tau* (τ), que representa a constante de tempo do filtro passa-baixa que limita a banda a ser transmitida e o desvio máximo. No receptor, é definida a frequência de quadratura a ser recebida e realizado o processo de *Audio Decimation*, que ajusta a taxa de amostragem dos dados recebidos para a taxa desejada.

O bloco *throttle* é responsável por determinar um limite máximo de transferência de dados através de si, evitando que o programa se perca num *loop* infinito de envio de dados. Ele se faz necessário, nesse caso, visto que tanto entrada quanto saída são independentes do mundo real e, portanto, são limitadas apenas pela capacidade de processamento do sistema.

Ainda nesse teste, são includos dois blocos *WX GUI FFT Sink*, que executam a transformada rápida de *Fourier* no sinal passando pelo ponto de conexão. Através de tal operação, podemos ver o espectro na frequência do sinal. Os blocos foram posicionados na saída do transmissor e na saída do receptor, para verificar a correta execução da modulação e demodulação, aliado à

avaliação qualitativa provida pelo arquivo de áudio salvo pelo programa através do bloco *Wav File Sink*. Os gráficos obtidos através de tais blocos encontram-se na Figura 13.

Através dos gráficos, pode-se ver que o sinal modulado tem um pico justamente na frequência de quadratura programada e decai ao se afastar desse ponto, como esperado. Já o sinal demodulado, possui uma forma bem distribuída desde a frequência tendendo a zero até os 18kHz, aproximadamente, que é boa parte do espectro captado pelo ouvido humano, e caracteriza um sinal de áudio traduzido corretamente.

Durante a fase de testes, a interface gráfica das transformadas rápidas de Fourier foram desabilitadas, de forma que não diminuíssem a capacidade de processamento do sistema. Os resultados obtidos através das gravações foram, então, analisados auditivamente, em comparação aos arquivos originais e as diferenças foram imperceptíveis ao ouvido humano. O programa teste teve, então, resultado satisfatório, quanto à qualidade de transmissão, evitando perda de informação relevante e mantendo a fidelidade ao arquivo transmitido, dentro dos padrões auditivos dos seres humanos..

Figura 13 - Análise Gráfica dos Resultados da Modulação (13a) e Demodulação (13b) FM

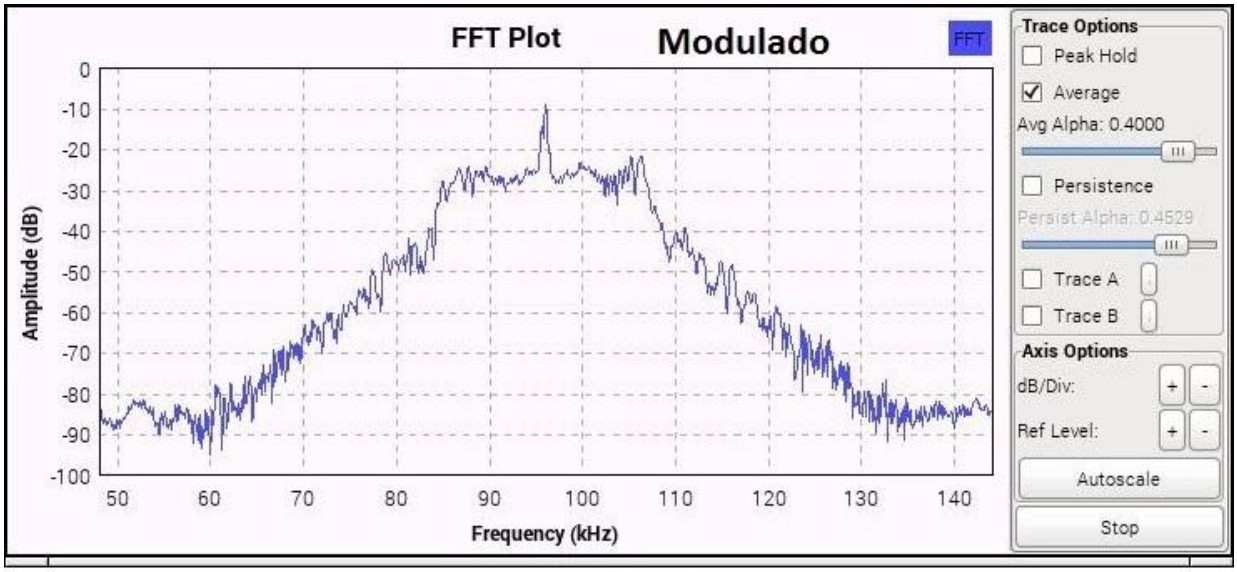


Figura 13a)

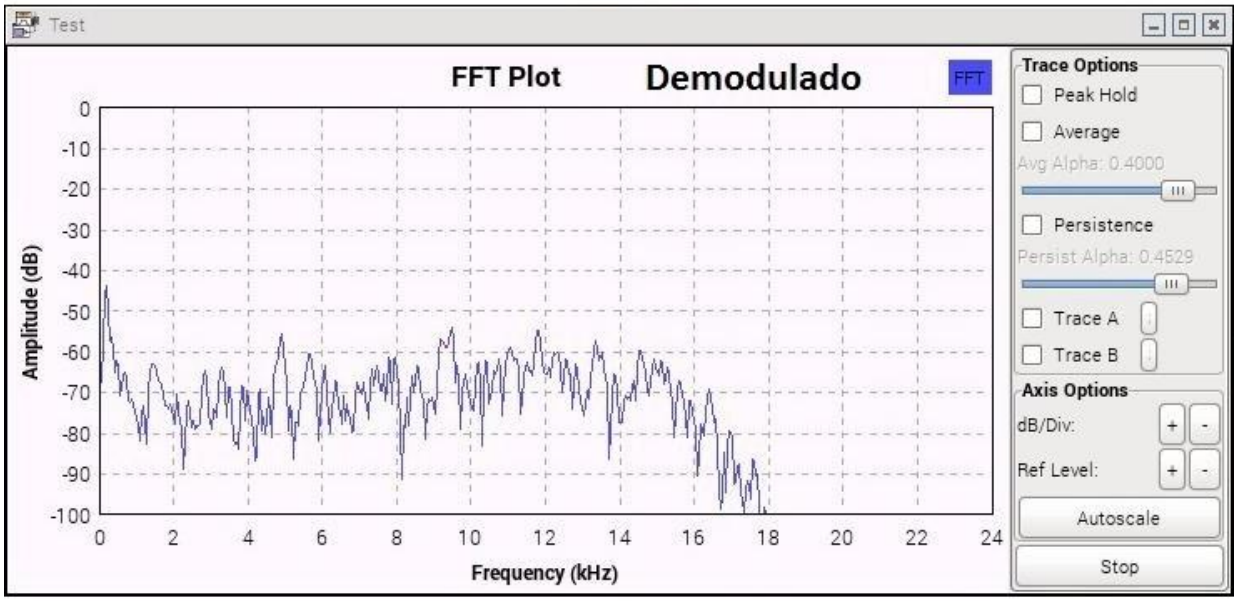


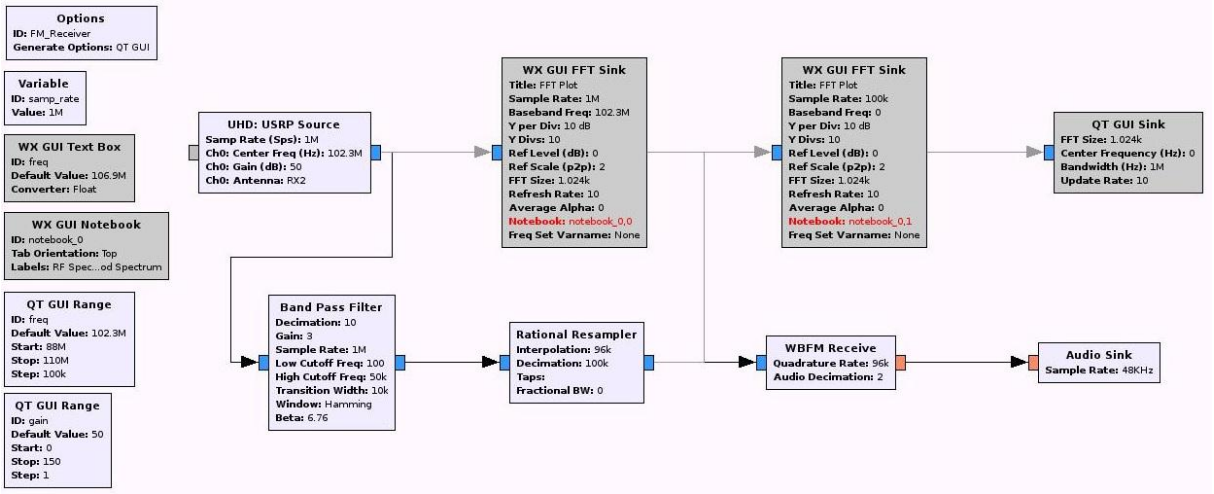
Figura 13b)

Fonte: Produção do próprio autor.

5.2 Implementação do Receptor FM

A partir do resultado positivo no teste inicial, inicia-se a próxima etapa, de implementação de um receptor FM de estações de rádio. Para isto, foi desenvolvido o programa exposto na Figura 14.

Figura 14 - Diagrama de blocos de um Receptor FM



Fonte: Produção do próprio autor.

Nesta nova montagem, por haver entrada de dados reais, o bloco *Throttle* pôde ser eliminado, considerando que as próprias limitações físicas não provocariam travamentos no software. O bloco *UHD: USRP Source* é responsável pela entrada de dados no sistema. Este bloco conecta-se ao USRP2 através de um endereço indicado no mesmo. Caso o campo endereço seja deixado em branco, ele procura por algum USRP na rede interna e realiza a conexão. Nele também é definida a taxa de amostragem, frequência central, ganho e qual das antenas da placa filha será utilizada.

Dependendo da placa filha do USRP sendo utilizada, é possível definir qual será a largura de banda enviada ao sistema. Porém, no caso da placa utilizada, WBX-40, a banda é fixa e definida em 40MHz. O que se extrai disso é que cada amostra recebida do USRP possui em sua composição uma quantidade grande de informações irrelevantes, quando se trata de estações de rádio que possuem banda de aproximadamente 200 – 400 kHz. Considerando isso, e que os dados na saída do USRP2 estão em banda base, é utilizado um filtro passa banda para remover ruído e reduzir drasticamente a largura de banda e, consequentemente, o volume de dados que precisam ser processados. No filtro também é realizada uma decimação, reduzindo a taxa de amostragem em dez vezes, no caso demonstrado na Figura 14.

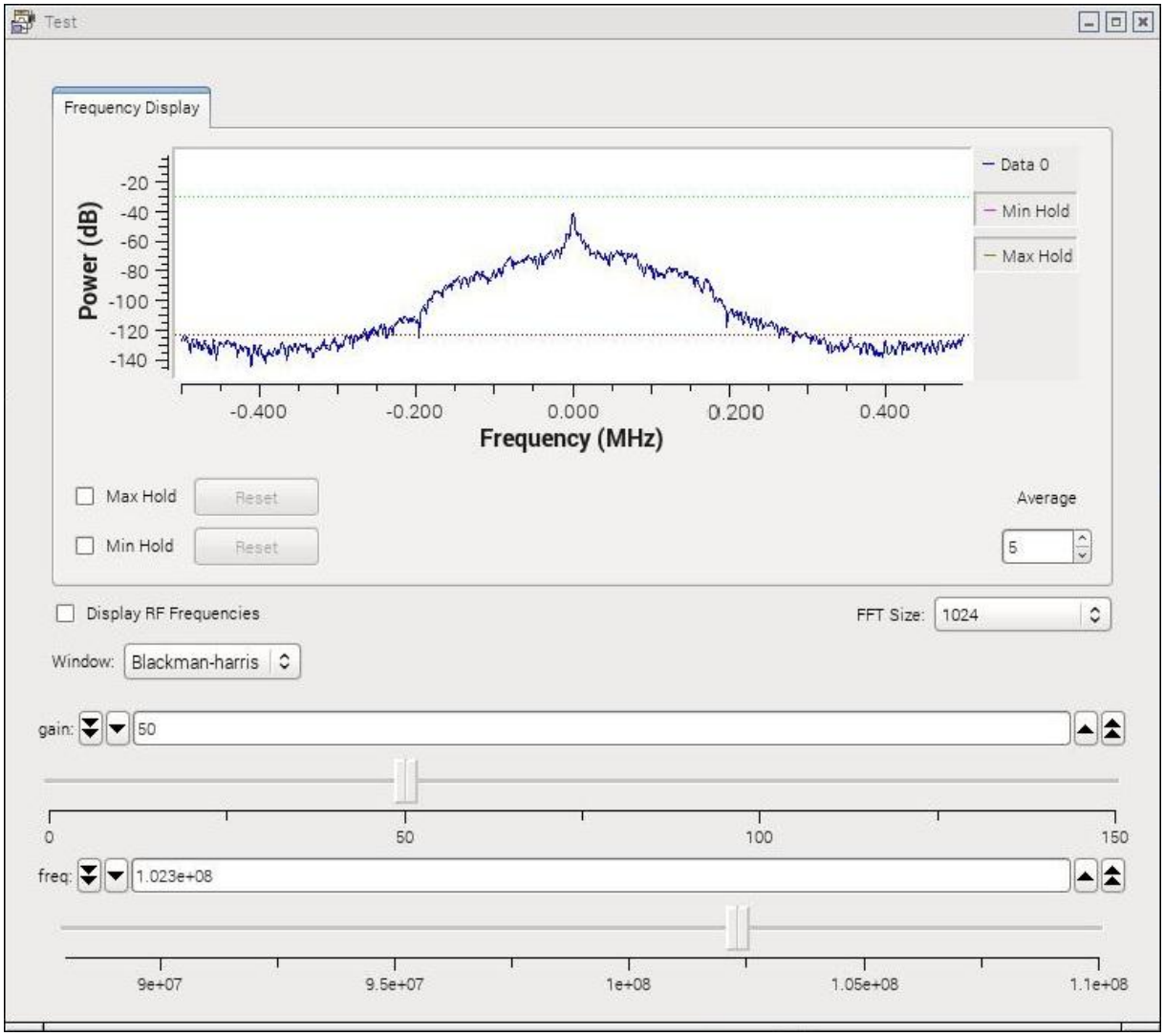
O bloco *Rational Resampler* se faz necessário por tratar-se de uma aplicação de áudio. É importante, para uma boa qualidade de áudio na saída, que o sinal que chega ao receptor de áudio esteja em uma taxa de amostragem múltipla da taxa que se deseja utilizar para a

reprodução ou gravação de áudio. Para tal, foi utilizado este bloco que é capaz de realizar operações que decimam e interpolam os dados recebidos. Assim, é possível alterar a taxa de amostragem para os mais variados valores finais desejados. No caso demonstrado na Figura 14, foi feita uma redução de 100 kHz para 96 kHz, que é um múltiplo da frequência final desejada.

Nesta aplicação, foram realizados testes com dois tipos diferentes de interfaces gráficas, geradas pelo GNU Radio. As interfaces, que envolvem seu próprio grupo de blocos específicos, são denominadas, nesse caso, WX e QT. Todos os ensaios primeiramente executados foram feitos baseados na interface WX, que é a *default* do programa. Entretanto, foram percebidas diversas quedas de capacidade de processamento ao utilizar-se blocos de visualização gráfica, especialmente quando aliados ao bloco *notebook*, que gera uma visualização com abas. A queda de desempenho provocava, inclusive, a não responsividade dos comando de configuração apresentados na barra lateral aos gráficos, durante a execução e a falha ao finalizar o programa, resultando numa finalização forçada que impedia o salvamento de arquivos de áudio corretamente, por falta de cabeçalho e terminação de acordo com os padrões do arquivo.

Ao utilizarmos a interface QT, no entanto, o sistema apresentou uma melhora significativa no desempenho por se tratar de uma interface mais leve e simplificada. Os resultados gráficos do sinal obtido, já na nova interface, são apresentados na Figura 15. Tal interface é também dotada de barras de rolagem capazes de mudar a frequência a ser ouvida da entrada, alterando a estação de rádio, durante a execução do programa.

Figura 15 - Representação Gráfica Gerada na Execução do Receptor FM com a Potência do Sinal representada em dBm, onde m indica Potência medida em relação a 1 mW



Fonte: Produção do próprio autor.

Duas variáveis foram adicionadas à interface de saída, para controle do programa. *Gain* armazena o ganho do sinal recebido do USRP2 , enquanto *freq* controla a frequência de sintonização. Ambas variáveis são controladas por barras de rolagem cujos valores máximo e mínimo são definidos no momento de programação de seus respectivos blocos. Através da Figura 15 percebe-se que o sinal recebido apresenta uma boa relação SNR (*Signal to Noise Ratio*) de cerca de 90dB, com um pico elevado na frequência central, chegando a -30dB, em seu máximo.

Os dados recebidos do bloco *UHD: USRP* Source já receberam um pré-tratamento e se encontram em banda base, centrada na frequência desejada, demonstrada pela frequência

central em zero, na Figura 15. Com isso, a recepção de dados digitais, apesar da alta frequência, torna-se realizável visto que, pelo teorema de Nyquist, a taxa de amostragem deve ser no mínimo duas vezes a maior frequência presente no sinal a ser recebido. Caso os dados não fossem deslocados para a banda base, a recepção de um sinal a 100MHz seria impossibilitado pela incapacidade do sistema de processar um volume de dados da ordem de 200MSps (*Samples per Second*).

Como a banda recebida era de aproximadamente 400 kHz e as frequências relevantes, mais altas, da ordem de 200 kHz, segundo o teorema de Nyquist, seria necessária uma taxa de amostragem de, no mínimo, 400 kSps. Foram, então, realizados testes com diversas taxas de amostragem de entrada, desde 500 kSps a 2,5 MSps. As diferenças ao modificar tal taxa eram imperceptíveis visualmente, através dos gráficos, mas se distinguiam nitidamente no áudio gerado. Vale ressaltar que, de modo a evitar interferências e queda de capacidade de processamento, as ferramentas de análise gráfica eram desabilitadas sempre que se desejada gravar arquivos de áudio.

A 500 kSps o programa executa com perfeição, sem sobrecargas do sistema e com baixa utilização do processador. Entretanto, o áudio gerado na saída deixa muito a desejar, qualitativamente, utilizando o ouvido humano como referência. A baixa presença das frequências mais graves tornou o sinal pobre, auditivamente, e infiel ao áudio transmitido original. Apesar da taxa de amostragem ser maior do que a recomendada, isso se justifica pois quanto menor a taxa de amostragem, em sistemas digitais, maiores são as perdas de informação do sistema.

Com os incrementos na taxa de amostragem de dados na entrada, percebe-se um resultado sonoro cada vez mais nítido, porém os gargalos de processamento vão se mostrando visíveis. Durante os testes em 2,5MSps, o dispositivo era capaz de reproduzir áudio por um período máximo de 3 segundos antes que a pilha (banco de dados que armazena dados recebidos) na entrada entrasse em *Overflow*, caracterizado por uma entrada de dados mais rápida que a saída deles, enchendo todos os espaços de armazenamento e impossibilitanto o recebimento de novos dados. Ainda que, durante tais 3 segundos, a qualidade sonora era desejável, o sistema, apesar das otimizações de interface e código aplicadas, não se mostrou capaz de executar o programa, utilizando-se de tal taxa de amostragem, em regime permanente.

Por fim, ao executar-se o programa a uma taxa intermediária, como 1 MSps, como demonstrada na Figura 14, há um balanço entre a sobrecarga do sistema e os resultados obtidos. Há que se levar em consideração, também, que, por impossibilitade de movimentação do kit USRP2 para uma área externa, a qualidade do sinal recebido ficou comprometida, até mesmo devido à localização da área de testes e espessura das paredes de alvenaria que formam o prédio.

5.3 Análise de Desempenho do Patch de Tempo Real

Para realizar os testes de desempenho, inicialmente foi necessária a implementação do *Kernel* de tempo real no sistema do *Raspberry Pi*. Essa instalação é feita através de um *patch*, disponibilizado gratuitamente pelos desenvolvedores do sistema. A aplicação do *patch* se dá através de uma simples concatenação com os arquivos de base para compilação do *Kernel*.

A aplicação do *patch* habilita uma série de novas possíveis configurações do *Kernel* antes de sua compilação. Dentre elas a habilitação de um RTC (*Real-time Clock*) e escolha do modo de tempo real a ser utilizado. Para o caso deste projeto, foi escolhido o modo *Full Preemptive RT*, que é o mais determinístico e rígido com tempo de execução, entre os disponíveis. Após os passos de configuração, é executada a compilação que, no caso do *Raspberry Pi*, levou algo em torno de 16 horas, incluindo a breve etapa de instalação subsequente.

Com o software preparado, pude-se então começar a realizar testes. Primeiramente foi realizado um teste generalista de tempo de resposta. Para tal, foi utilizado o *cyclictest*, uma ferramenta distribuída gratuitamente, altamente configurável e com diversos modos diferentes de operação. Para o este caso, o comando utilizado definia o teste com um milhão de amostras, separadas entre si por 400 micro segundos. Foi dada ao teste a prioridade mais alta possível em ambos os sistemas operacionais e o teste foi executado com apenas um dos núcleos do processador, com um único *thread*. O comando utilizado, que executa o teste de acordo com as condições descritas, foi:

Sudo cyclictest –l1000000 –m –n –a0 –t1 –p99 –i400 –h400 -q

O comando citado possui alguns parâmetros adicionais de forma a tornar a execução mais justa e passível de comparativos, visto que objetivava-se a comparação de dois sistemas distintos. O significado dos parâmetros pode ser encontrado no Quadro 3.

Quadro 3 - Parâmetros utilizados no comando do *Cyclictest*

Parâmetro	Função
L1000000	Executa programa com 1 milhão de *loops*, ou amostras
M	Bloqueia alocação dinâmica durante a execução
N	Habilita utilização do *clock_nanosleep*, para possibilitar intervalos mais curtos entre amostras
A0	Seleciona o processador numerado “0”, para a execução
T1	Executa utilizando apenas um *thread*
P99	Configura a prioridade para a máxima possível
I400	Seleciona o intervalo entre amostras para 400 microsegundos
H400	Imprime no console os primeiros 400 valores de tempo de resposta
Q	Ativa o modo *quiet*, com impressão dos resultados apenas após execução

Fonte: Produção do Próprio Autor.

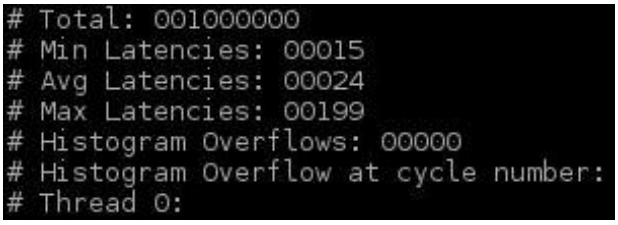
Os parâmetros M, A0, T1 e P99, em especial, são responsáveis por tornar tal comparação mais justa. Sua adição se faz necessária para que sejam evitados eventos aleatórios que poderiam ser provocados pelo próprio sistema durante a execução, provocando disparidades nos resultados. Com adição desses comandos, selecionando sempre o mesmo processador, com o mesmo número de *Threads* e prioridade, aliado ao bloqueio da alocação dinâmica, o sistema torna-se mais imune a desvios do comportamento esperado durante a fase de testes, aumentando assim a confiabilidade dos resultados.

O teste foi executado no mesmo hardware, com *Kernel* diferente em cada caso. A Figura 16 apresenta os resultados obtidos no sistema *Vanilla*, como foi distribuído pelos criadores do *Raspberry Pi*, sem modificações, enquanto a Figura 17 mostra os resultados obtidos no mesmo hardware, mas já com um *Kernel* de tempo real aplicado.

Os tempos dados nos resultados estão contados em microsegundos. Quando trata-se de testes de latência, o dado mais importante a ser considerado é a latência máxima. Este fator determina até onde pode-se arriscar levar a frequência de operação do sistema com a garantia de que o

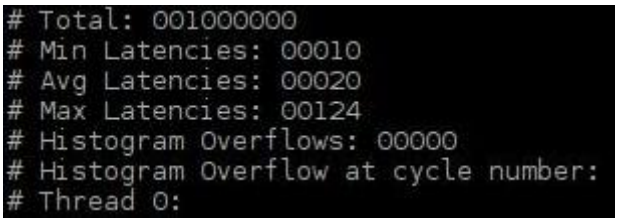
mesmo será capaz de atender ao que é dele requerido. Há reduções em todas as latências, quando aplicamos o *patch* de tempo real, porém tal diferença é muito mais visível quando analisa-se para a latência máxima. Isso se torna uma grande vantagem desse sistema na aplicação em tarefas que possuem tempo crítico.

Figura 16 - Resultados do Cyclictest em Kernel Vanilla



Fonte: Produção do próprio autor.

Figura 17 - Resultados do Cyclictest em Kernel de Tempo Real



Fonte: Produção do próprio autor.

Como o teste que foi realizado através *Cyclictest* é apenas um teste de latência simplificado, não há muita sobrecarga do sistema, provocando baixos tempos de latência e ausência de *Overflows*, independente do sistema utilizado, como também pode ser visto nas Figuras 16 e 17.

De forma a realizar um teste mais completo, e obter resultados um pouco mais concretos e específicos, foi utilizada mais uma ferramenta de análise, capaz de testar diversos componentes do sistema operacional de forma isolada. Para tal, foi utilizado o LRTBF (*Linux Real Time Benchmarking Framework*).

Para a realização de tais testes, são necessários 2 computadores além do que se deseja testar. Estes possuem funções distintas, são quais sejam: *target*, *host*, e *logger*. O *Raspberry Pi*, neste caso, foi utilizado como *target*, ou alvo e trata-se do computador, ou sistema, que se deseja testar. O computador *host* é um sistema de controle e também é usado para coletar dados a

respeito do *target*. Já o *logger* é uma máquina especial encarregada de provocar interrupções na máquina *target* e gravar os tempos de resposta da máquina alvo.

Os testes foram realizados em 5 fases distintas, de modo a isolar características específicas e analisar a forma em que elas afetam o desempenho do sistema como um todo. Primeiramente foi feito um teste simples, sem nenhuma interferência externa ou sobrecarga interna. Em seguida foram feitos testes com carga. No primeiro deles, a máquina *logger* provoca interrupções por comunicação serial a cada 1ms (milisegundo). No segundo caso, o *host* entra em um *loop* de envio de *pings* endereçados ao *target*, provocando um *ping flood*, traduzido literalmente para uma inundação de pedidos de resposta a serem tratados concomitantemente com o *script* de *benchmarking*.

No terceiro teste, as duas sobrecargas anteriores são postas em ação simultaneamente e no quarto teste com carga, é provocada uma sobrecarga no HD do *target* ao mesmo tempo que o *logger* provoca interrupções de *hardware*. A sobrecarga no HD é criada através de um comando que entra em um *loop* infinito de criação de arquivos, no *target*.

Num primeiro momento, os 5 testes foram executados em sua totalidade três vezes e o tempo médio de duração da execução foi gravado. Os resultados se encontram no Quadro 4. Nota-se que os sistemas possuem tempos parecidos em três das cinco etapas de teste enquanto nas outras duas, o sistema *Vanilla* é mais rápido no tempo total de execução. Os resultados são justificáveis pela preocupação maior do sistema de tempo real em checar a todo instante se está executando a tarefa prioritária. Seu foco deixa de ser a redução do tempo máximo para ser a redução das latências máximas de atividades julgadas mais importantes para o usuário. As maiores disparidades presentes nos testes com *Ping Flood* possivelmente são ocasionadas por uma maior prioridade do *Ping* em relação às demais sobrecargas.

Quadro 4 - Tempo total de Execução do teste LRTBF

Kernel	Sem Carga	Teste IRQ	*Ping Flood*	IRQ & *Ping*	IRQ & HD
Vanilla	183s	182s	213s	212s	279s
Preempt RT	183s	185s	234s	230s	279s

Fonte: Produção do próprio autor.

Em seguida, mais um teste foi realizado medindo os tempos de resposta a interrupções nas condições do teste anterior, excluindo interrupções como carga e com adição de dois *scripts* diferentes, denominados *LMbench e doHell*. *LMbench* é um *script* de *benchmarking* que, neste caso, foi usado como sobrecarga paralela às interrupções.

O *script doHell* tem por função sobrecarregar o sistema em paralelo à atividade principal que se deseja executar, simulando uma situação em que o sistema se encontra sobrecarregado mas ainda tem por função responder a alguma atividade de alta prioridade. O Quadro 5 contém os resultados obtidos através dos testes.

Quadro 5 - Tempo de Resposta a Interrupções sob diferentes Cargas (µs)

Kernel	Carga	Médio	Máximo	Mínimo	Desvio Padrão
Vanilla	Nenhuma	11,2	61,3	10,5	0,3
	Ping	**11,2**	**59,4**	10,5	**0,8**
	LMbench	**11,5**	82,8	10,5	**0,9**
	LMbench + *Ping*	**11,5**	63,5	10,5	**1,2**
	LMbench + HD	**11,9**	152,9	10,5	3,6
	doHell	**12,4**	573,2	10,5	7,9
Preempt RT	Nenhuma	**11,1**	**58,9**	10,5	**0,2**
	Ping	12,4	70,1	10,5	1,6
	LMbench	12,6	**60,5**	10,5	1,6
	LMbench + *Ping*	13,1	**60,4**	10,5	2,0
	LMbench + HD	12,6	**74,1**	10,5	**1,9**
	doHell	12,7	**59,8**	10,5	**1,4**

Fonte: Produção do próprio autor.

Após a realização dos testes, é possível comprovar que o *Kernel* de tempo real não aumenta a capacidade de processamento do sistema. Os tempos mínimos são constantes pois são resultado de interrupções ocorrendo em momentos em que o processador se encotrava ocioso e refletem o tempo mínimo de resposta do sistema a uma interrupção.

Os tempos médios, na maioria dos casos, aumentam devido a uma mais frequente mudança de contexto ocasionada pelo constante escalonamento de tarefas do *Kernel* de tempo real,

envolvendo leitura e escrita de variáveis e dados temporários para evitar perdas. Embora estas operações descritas reduzam o tempo total e médio de processamento, a grande vantagem da aplicação deste tipo de sistema vem com a redução da latência máxima.

Devido ao núcleo de tempo real, o sistema torna-se mais determinístico, constante e previsível. Isso quer dizer que o mesmo tem um comportamento altamente configurável e se comporta exatamente da forma como o usuário programar. O sistema torna-se injusto, na divisão de tarefas, priorizando as tarefas configuradas com alta prioridade. Isso pode ser comprovado pelas reduções em tempo máximo de resposta, especialmente nos casos onde o sistema encontra-se sobrecarregado.

Essa redução nos tempos máximos é provocada pelo sistema de prioridades. Mesmo que o sistema esteja sobrecarregado com atividades paralelas, quando uma interrupção de alta prioridade acontece, o sistema a executa dentro de um tempo limite, definido pelo próprio *Kernel*. Com isso, é obtida uma maior constância em relação a tempo máximo de resposta.

Quando do momento de projeto de *hardware*, para uma tarefa específica, deve-se sempre avaliar o pior caso possível (*Worst Case Scenario*). Com isso, o tempo que realmente tem valor, para um projeto, é o tempo máximo de execução. Em cima desse valor é definida a máxima frequência de operação sem erros. Com isso, podemos concluir que, para atividades que possuem tempo crítico de resposta, os sistemas de tempo real são os mais indicados.

5.4 Implementação do Receptor FM em plataforma com Kernel de Tempo Real

Na etapa final deste projeto, aplica-se o programa do receptor FM no sistema com *Kernel* de tempo real, como uma aplicação prática do que foi estudado e comprovado. Através deste experimento é possível avaliar alterações de desempenho do programa e verificar se tal aplicação tornava possível um incremento da taxa de amostragem na entrada sem que houvesse *overflow*.

Após a preparação do sistema e instalação dos requerimentos para que o programa fosse executado, o sistema começou a apresentar falhas com travamentos completos, resolvidos através da remoção do cabo de alimentação e reinicialização do sistema.

Tais falhas podem ter sido ocasionadas por alguns motivos, segundo pesquisas realizadas. O primeiro motivo considerado foi um conflito de prioridades do sistema. Possivelmente alguma tarefa no *background* estava assumindo o controle do sistema e entrando em um *loop* infinito, impossibilitando que o *Kernel* retomasse o controle e desse continuidade às suas tarefas. Outra possibilidade seria a incompatilibidade do *Kernel* de tempo real com *software* pré-existente do *Kernel* anterior.

Muitas manobras foram realizadas para tentar evitar que isto acontecesse novamente. Uma das medidas foi a completa recompilação de diversos *softwares* imprescindíveis ao funcionamento do GNU Radio, visando solucionamento de conflitos com versões pré-existentes. Além disto, foram feitas mudanças na interface do GNU Radio, na tentativa de diminuir a carga sobre o sistema. Aliado a isso, a tabela de prioridades do sistema foi aberta e foram realizadas tentativas de averiguar casos de outras tarefas recebendo prioridades conflitantes com o que se queria realizar, tomando o controle do sistema e, possivelmente, entrando em *loops* infinitos, prevenindo que o sistema retomasse o controle sobre o processador. Entretanto, apesar das tentativas, os problemas persistiam, sem indicações de motivação reportadas pelo sistema.

Depois de muitas tentativas, uma pesquisa revelou que as distribuições de *patch* para aplicação em *Raspberry Pi* são feitas em caráter experimental e, portanto, são passíveis de instabilidades, algumas conhecidas. Tal instabilidade impossibilitou que o programa fosse executado para novas avaliações em tal distribuição de tempo real.

Contudo, apesar da impossibilidade, é cabível uma previsão dos resultados esperados caso a aplicação obtivesse êxito. Devido aos resultados comprovados nos experimentos da seção 5.3, observa-se que os tempos médios do sistema, de modo geral, aumentam, caracterizando uma leve redução na capacidade de processamento de dados. No entanto, há uma redução nas latências máximas, especialmente em casos de sobrecarga do sistema. Através das reduções de latência máxima, há também um aumento na taxa de amostragem máxima que pode ser feita com segurança, utilizando a latência máxima como período de amostragem.

Sendo assim, há duas possbilidades de resultados da implementação que dependem da forma como os *overflows* estavam ocorrendo. Se os mesmos foram ocasionados por sucessões de respostas de alta latência, até que a pilha enchesse, o sistema de tempo real seria capaz de

possibilitar um aumento na taxa de entrada. No entanto, se os *overflows* eram provocados por respostas de tempo médio do *Kernel Vanilla*, uma mudança de *Kernel* para tempo real provocaria redução da taxa de amostragem, visto que o mesmo possui tempos médios em geral mais lentos.

No entanto, caso a implementação desejada envolva tarefas realizadas concomitantemente, como, por exemplo, a adição de entrada de dados via interrupção por um teclado ou tela *touchscreen*, o sistema de tempo real seria superior. Isso dá-se pelo fato de que essas cargas de entrada de dados não possuem prioridade e podem ser tratadas de forma mais lenta sem que afete a experiência do usuário. Sendo assim, num sistema de tempo real, a prioridade de tais tarefas pode ser reduzida drasticamente de modo a certificar que não alterem o desempenho da tarefa principal, que é a recepção de dados do USRP2. Assim, uma situação que fatalmente provocaria uma redução da taxa de amostragem num sistema *Vanilla*, não afetaria grandemente um sistema de tempo real.

6 CONCLUSÃO E PROJETOS FUTUROS

Este trabalho teve como objetivos principais a implementação de um receptor FM com tecnologia RDS em uma plataforma embarcada, com recursos limitados, e a realização de análises de desempenho no sistema com aplicação de variações no *Kernel*, de modo a provar resultados teóricos.

Isto foi realizado através da utilização de um *Raspberry Pi 2 B* com e sem a aplicação de um *Kernel* de tempo real, em conjunto com um kit USRP2 e o software livre GNU Radio. Inicialmente foram feitos testes sem interferências do mundo externo e, em seguida, com a programação de um receptor FM com frequência e taxa de amostragem variáveis. Diversos testes de desempenho foram realizados em tais condições.

Em seguida, o *patch* de tempo real foi aplicado ao *Raspberry Pi* e novos testes foram realizados para comparar seu desempenho e funcionamento com o sistema distribuído pelo fabricante. Na etapa final, foi implementada a união das etapas anteriores, para que se pudesse perceber diferenças em seu funcionamento devido ao sistema de tempo real.

Este trabalho obteve êxito em seus resultados no que toca as duas primeiras etapas principais, uma vez que foi possível implementar o que se queria fazer. As amostras de áudio obtidas na saída do receptor eram condizentes com a entrada e possuiam qualidade satisfatória, levando em condição as limitações, tanto de *hardware,* quanto geográficas (paredes grossas, localização isolada, provocando qualidade mais baixa de sinal de entrada).

Também foi provada a capacidade do RDS de ser extremamente mutável e versátil, podendo ser aplicado nas mais diversas configurações e, até mesmo, em uma plataforma embarcada, com recursos de *hardware* extremamente limitados, em comparação com um computador. Além disso, através do mesmo é possível receber uma infinidade de sinais diferentes, com modulações diferentes no mesmo *hardware* apenas com uma reconfiguração de *software*, feito de forma simplificada através do GNU Radio.

Os testes realizados no sistema de tempo real foram capazes de provar o que a teoria afirmava sobre o sistema e sobre suas vantagens com relação a aplicações que possuem tempo crítico,

com drásticas reduções de latência e tempo de resposta a interrupções quando o sistema apresenta sobrecargas.

Subsequente a esta etapa, encontra-se o maior desafio deste projeto, que foi a implementação do receptor FM no sistema com *Kernel* de tempo real. O sistema operacional se mostrava inconsistente e apresentava falhas e travamentos constantes, impossibilitando que os testes fossem realizados com êxito. Diversas técnicas e medidas foram aplicadas para tentar contornar os problemas apresentados, mas sem sucesso.

Como projetos futuros, é possível sugerir uma ampliação do que foi feito aqui, através da implementação de múltiplas técnicas de demodulação em um mesmo programa, oferecendo uma chave seletora que escolhesse a técnica a ser aplicada. Além disso, por ser uma plataforma embarcada e portátil, é possível a implementação de uma tela *touchscreen* com uma interface gráfica simplificada, tornando a experiência do usuário mais agradável.

Por outro ângulo, é possível realizar a mesma implementação sem a utilização dos blocos do GNU Radio, mas através de linha de código. Tal implementação torna-se mais eficiente por usar código menos generalista, desde que programado visando eficiência. Desta forma, pode-se obter taxas de amostragem elevadas em comparação com as obtidas neste projeto.

Finalmente, pode ser proposta uma avaliação a fundo das inconsistências da aplicação conjunta do GNU Radio com o sistema de tempo real, de modo a solucionar os problemas apresentados durante a execução deste projeto e fazer uma avaliação real sobre as vantagens e desvantagens da aplicação do sistema de tempo real para este fim específico.

REFERÊNCIAS BIBLIOGRÁFICAS

CAMPOS, S. **O que é modulação e que modos são utilizados**. Disponível em: <http://www.sarmento.eng.br/Modulacao.htm>. Acesso em 1 jun. 2015.

ETTUS RESEARCH. **USRP N200/N210 Networked Series**. Disponível em: <https://www.ettus.com/content/files/07495_Ettus_N200-210_DS_Flyer_HR_1.pdf>. Acesso em: 15 jun. 2015.

GNU RADIO. **Core Concepts of GNU Radio.** Disponível em: <http://gnuradio.org/redmine/projects/gnuradio/wiki/TutorialsCoreConcepts>. Acesso em: 2 nov. 2016.

JANSON, J. **Radio Definido por Software: Estudo e Realização de Teste com uma Plataforma Livre**. Trabalho de Conclusão de Curso (Curso Superior em Sistemas de Telecomunicação) – Centro Federal de Educação Tecnológica de Santa Catarina. São José, 2012.

JOSH. **Demystifying the Linux Kernel.** Disponível em: <https://blog.digilentinc.com/demystifiying-the-linux-kernel/>. Acesso em: 15 nov. 2016.

MUNIZ, R. C. **Comunicações Analógicas e Digitais**. 1. ed. Vitória: LTC. 2010.

ITO, M. S.; SCHENA, R. **Gnu Radio e Independência do Hardware em Sistemas Embarcados: Considerações Sobre a Aplicabilidade de SCA como Alternativa em Busca de Maior Flexibilidade.** Disponível em: <http://www.revdigonline.com/uploads/docs/volume-6-revista-digital-online-issn-1679-4389.pdf>. Acesso em 2 nov. 2016.

OLIVEIRA, M. A. de. **Implementação da multiplexação OFDM conforme Padrão IEEE 802.11p via tecnologia de radio definido por software**. Trabalho de Conclusão de Curso (Curso Superior em Engenharia Elétrica) – Universidade Federal do Espírito Santo. Vitória, 2014.

PRADO, S. **Sistemas de Tempo Real.** Disponível em: <https://sergioprado.org/sistemas-de-tempo-real-part-1>. Acesso em 5 nov. 2016.

PUHLMANN, H. **Sistemas Operacionais de Tempo Real.** Disponível em: <http://www.embarcados.com.br/sistemas-operacionais-de-tempo-real-rtos>. Acesso em: 5 nov. 2016.

RASPBERRY PI FOUNDATION. **What is a raspberry pi?** Disponível em: <https://www.raspberrypi.org/help/what-is-a-raspberry-pi/>. Acesso em: 17 jun. 15.

TÉCNICAS de modulação AM/FM. Disponível em: <http://www.mecatronicaatual.com.br/educacao/1202-tcnicas-de-modulao-amfm?showall=&limitstart=>. Acesso em 1 jun. 2015.

UCLA CORES. **Cognitive Radio Testbeds.** Disponível em: <http://cores.ee.ucla.edu/index.php?title=Cognitive_Radio_Testbeds>. Acesso em: 8 dez. 16.