Título

Subtítulo

Relatório Final de
pesquisa de Iniciação Científica

Aluno: Leonardo José Held
Orientador: Raimes Moraes

Departamento de Engenharia Elétrica e Eletrônica
Curso de Graduação em Engenharia Eletrônica
Universidade Federal de Santa Catarina
Brasil

Conteúdo

Capítulo 1

Resumo

Entre os diversos métodos de diagnóstico de enfermidades pulmonares, a asculta de sons respiratórios sobre o toráx tem como vantagens o fato de ser não invasivo e ter baixo custo. No entanto, devido à fatores técnicos como qualidade do estetoscópio, acurácia auditiva do examinador e atenuação do som pelo toráx do paciente, além da subjetividade pelo fator humano envolvido no exame, há esforços para tornar este método mais robusto, proporcionando um diagnóstico mais confiável. Este trabalho tem como objetivo aperfeiçoar estetoscópio eletrônico desenvolvido neste laboratório que digitaliza e transmite sons respiratórios, bem como, registra o deslocamento do tórax para identificação da fase respiratória. O sistema consiste-se de dois módulos, transmissor e receptor, que respectivamente amostram e recebem o som em um microcomputador pessoal para armazenar os sons captados de pacientes. O protocolo de transmissão utilizado é o *Bluetooth 5.0*. Neste trabalho, são reportados o estudo, teste e o aprimoramento do sistema.

Palavras-chave: Instrumentação médica. Sistema respiratório. Sons respiratórios. Estetoscópio Eletrônico. Processamento Digital de Sinais.

Capítulo 2

Introdução

O prematuro diagnóstico de enfermidades respiratórias pode representar uma redução significativa de óbitos. Dentre os métodos de diagnóstico, tem-se a asculta pulmonar, rotineiramente realizada por médicos treinados. Seus pontos fortes são o baixo custo e o baixo tempo de resposta, não demandando o deslocamento de pacientes acamados para salas de exames(CARVALHO V. O.; SOUZA, 2007). No entanto, o diagnóstico é fortemente dependente da experiência e capacidade auditiva do examinador. Neste contexto, surgiram estetoscópios eletrônicos de baixo custo que podem captar e registrar sinais que podem ser digitalmente processados com o objetivo de prover indíces quantitativos para orientar o diagnóstico, de tal forma a tornar o diagnóstico menos subjetivo e permitir a troca de informações entre especialistas. Esse trabalho propõe o aprimoramento com integração de funcionalidade adicionais a um sistema de aquisição de sons respiratórios desenvolvido no LCS - UFSC. O módulo (MATTOS, 2018), - que antes contava com receptor e transmissor *Bluetooth 5.0*, e um periférico de aquisição de sinais analógicos - foi aprimorado com a integração de sensor para registro da fase respiratória.

2.1 Objetivo

Este trabalho objetivou o aprimoramento de sistema de captação de sons respiratórios anteriormente desenvolvido, adequando-o para a aquisição de sons respiratórios em ambiente hospitalar.

Como desenvolvimento adicional, acrescentou-se sensor para registrar deslocamento da parede torácica para identificar a fase respiratória.

Capítulo 3

MATERIAL E MÉTODOS

3.1 Introdução

A Figura 1 apresenta o diagrama em blocos do sistema desenvolvido em trabalho anterior (MATTOS, 2018), no qual se incluiu um interface para acelerômetro e giroscópio.

O sistema possui módulo para a captação de sons respiratórios (módulo 1), firmware dos módulos de amostragem, transmissão e recepção de dados (módulos 3 e 4), bem como de aplicativo para sistema operacional Windows (módulo 5). Este aplicativo objetiva a apresentação, armazenamento e reprodução das formas de onda amostradas. O aplicativo conta com programa escrito em Python (desenvolvido neste trabalho) que obtém os dados recebidos pelo USB - via serial - do módulo de recepção dos dados, denominado **Módulo 4**, que são enviados pelo módulo de amostragem, ou **Módulo 3**.Cada um destes módulos é detalhado nas subseções seguintes.

O módulo 3 (Figura 1) controla a digitalização dos sinais do ADC via interface SPI, empacota as amostras e os transmite por radiofrequência (Bluetooth) para o módulo de recepção (módulo 4). Estas etapas são realizadas por meio do kit de desenvolvimento BLE da Texas Instruments (CC2640R2F `LAUNCHPAD-XL Kit`). Este kit com proto-

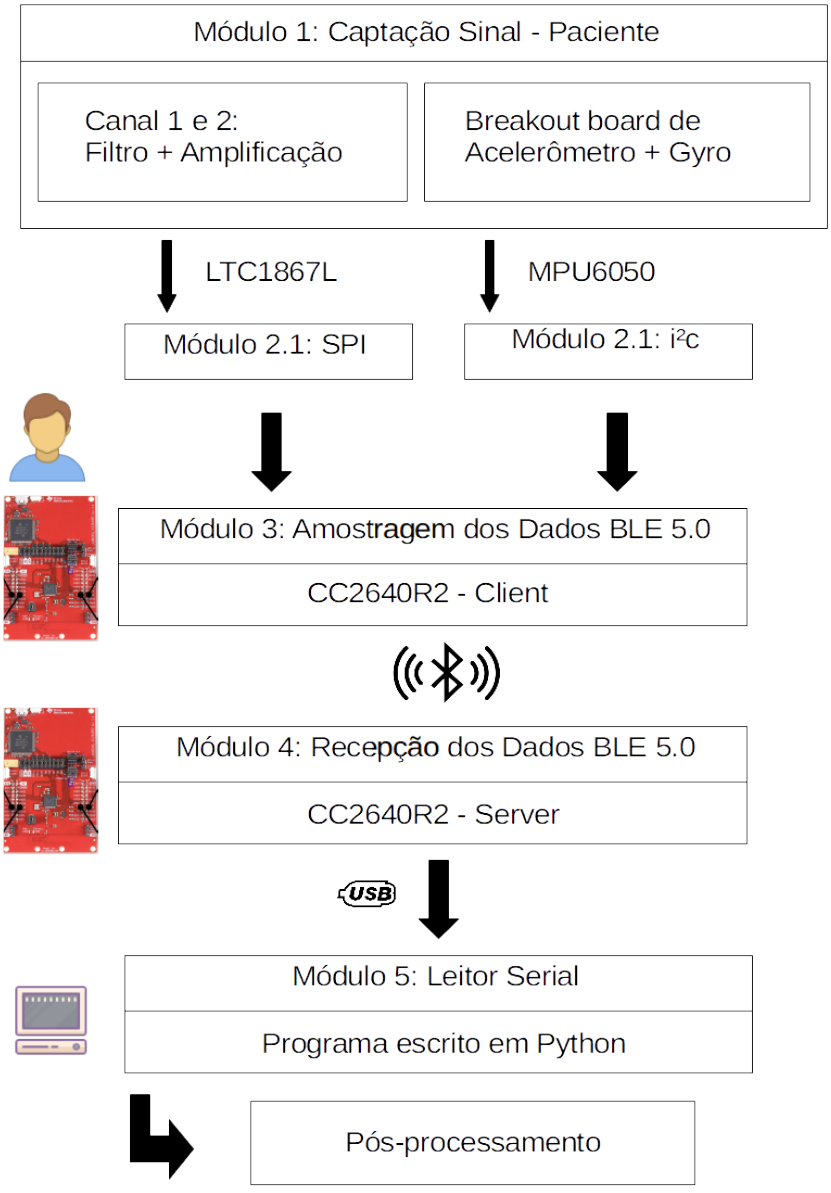

Figura 3.1: Fonte: O autor.

colo Bluetooth 5 proporciona uma considerável redução no tempo de desenvolvimento da solução BLE, oferecendo taxa de transmissão e consumo de energia que atendem às necessidades do projeto.

A transmissão por tecnologia *Bluetooth* em sua versão 5.0 foi selecionada devido a larga banda dados que devem ser transmitidos, visto que os sons respiratórios gravados são de alta fidelidade, e transmitidos sem compressão.

O dispositivo `LAUNCHPAD-XL Kit` baseia-se em processador ARM® Cortex®-M3 que gerencia a camada de aplicação e a pilha de protocolos BLE. Além disso, possui um sistema autônomo de rádio (utilizando um processador ARM Cortex®-M0) que gerencia todo o controle e processamento das camadas inferiores associadas à camada física e partes da camada de enlace. Para administrar todo este conjunto de tarefas, o módulo `LAUNCHPAD-XL Kit` utiliza um sistema operacional em tempo real proprietário da Texas Instruments (TIRTOS). Para melhor compreensão do sistema existente, este trabalho foi iniciado com o estudo do protocolo Bluetooth e o TIRTOS.

3.2 Características de módulos microcontrolados com capacidades *BLE 5.0*

Ambos módulos de Amostragem e Recepção (módulos 3 e 4, respectivamente na figura 1, seção 3.1) dos dados são placas de desenvolvimento da *Texas Instruments*®, codenome `LAUNCHPAD-XL-XL` contendo o chip *CC2640R2* e que suportam a tecnologia *BLE 5.0*, *Bluetooth Low Energy 5.0*.

A figura a seguir mostra o esquemático funcional de blocos dos módulos 3 e 4, que como mencionado na introdução deste trabalho, são plataformas de desenvolvimento *CC2640R2*.

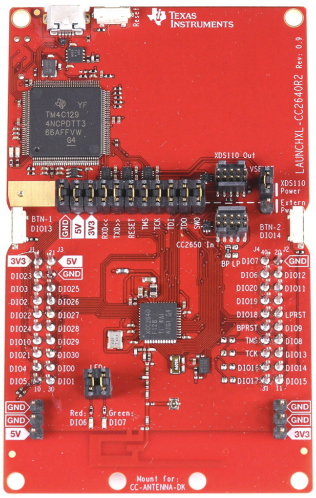

Figura 3.2: Fonte: Texas Instruments®

Como pode ser observado, ambos os módulos contam com dois processadores: ARM Cortex-M3 e ARM Cortex-M0. O primeiro atua como processador de propósito geral, supervisionando os pinos de entrada e saída (*I/O*) de cada placa. O Cortex-M0 controla o circuito de radiofrequência (*RF*) responsável pela implementação do protocolo *Bluetooth* da placa.

Existe um protocolo de comunicação entre os dois processadores (não acessível ao usuário da placa) que serve para transmissão de dados pelo Bluetooth. Fazemos uso, então, do *Cortex-M3* para acessar e obter dados dos sensores ou periféricos. Estes, por sua vez, são enviados via para para o módulo de recepção (segundo kit`LAUCHPAD-XL`) usando a API da *TI*.

Algumas limitações de hardware ainda são realidade para essa tecnologia, fato que, por extensão, limita a escolha de placas de desenvolvimento no mercado. A tecnologia *BLE 5.0*, embora o protocolo da presente versão tenha sido homologado em 2016, ainda não é totalmente adotada e nem

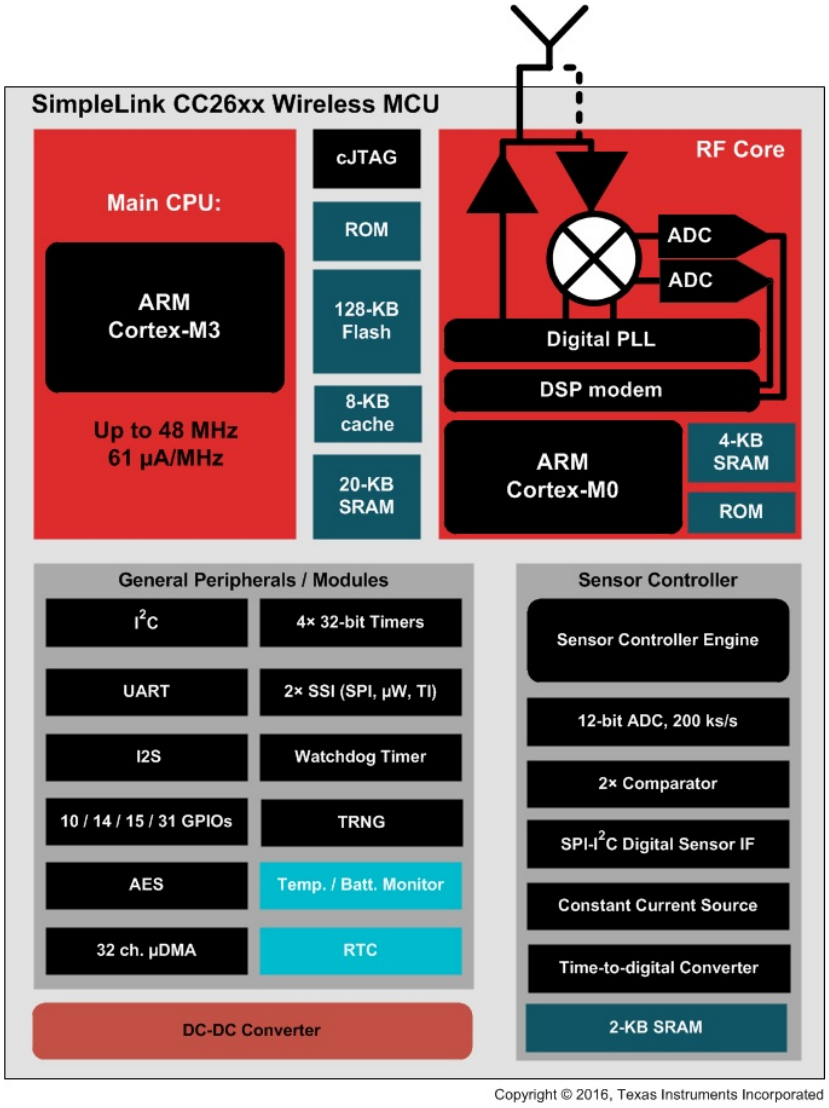

Figura 3.3: Fonte: Texas Instruments®

amplamente utilizada por vários dispositivos. Visto que o ecossistema do projeto original já estava baseado nesta plataforma, o atual projeto continuou sendo desenvolvido na mesma.

3.2.1 Bluetooth

O *Bluetooth 5.0* é uma tecnologia definida por um grupo de trabalho denominado *Bluetooth Special Interest Group*(SIG, 2019). Nesse, se define um padrão de comunicação sem fio entre dois - ou mais - dispositivos numa banda de frequência normalmente utilizada para fins industriais, científicos e médicos. Originalmente, o Bluetooth foi criado para estabelecer um tipo de comunicação simples, serial, que já existia na forma de conexão via cabo entre dispositivos.

De acordo com (AH, 2010), os principais pontos que fazem com que *Bluetooth* sejam uma boa escolha meios médicos são: interoperabilidade, baixo consumo, customização de formato de transmissão (pacote), compatibilidade eletromagnética (equipamentos médicos coexistem, não podendo inteferir negativamente uns com os outros), segurança na transmissão para proteger confidencialidade da informação e hub de sensores (diversos equipamentos médicos precisam ser atualizados em tempo real via internet para fins de telemedicina).

A necessidade de se utilizar uma tecnologia de ponta nesse projeto surge da já mencionada grande quantidade de dados envolvidos na transferência de sons respiratórios. Na especificação do *BLE* ou *Bluetooth Low Energy*, um padrão de Bluetooth utilizado pelo baixo consumo, com alcance de até 350 metros, o *throughput* é limitada em 2Mpbs(PANWAR; MISRA, 2017), porém, dado um *overhead* dependente do hardware onde o protocolo está sendo executado, a velocidade efetiva de transmissão é geralmente

menor do que a especificada.

Também é um protocolo aberto para implementação, realizado pelas principais empresas de tecnologia mundiais, fomentando os pontos de interoperabilidade de equipamentos e segurança.

3.2.2 Sistemas Operacionais em Tempo Real

Devido à necessidade de aquisição de diferentes sinais em intervalos de tempo regulares (de tal forma a atender ao teorema da amostragem), emprega-se Sistema Operacional em Tempo Real (a partir daqui denominado RTOS: Real Time Operating System) que é executado em ambos os módulos (transmissão e recepção).

De acordo com (BARRY, s.d.) um Sistema Operacional em Tempo Real como um sistema que precisa saber o tempo de término quão mais exato de uma tarefa. Ou seja, um sistema em tempo real se carateriza pelo caráter determínistico. É ideal para aplicações onde ações externas precisam ser induzidas em intervalos de tempo regulares e extremamente corretos, como aquisição de som - no caso de estudo deste trabalho - ou o disparo de um air-bag.

O princípio de funcionamento básico de um RTOS é fazer com que várias rotinas (também chamadas de funções) possam ser executadas em *quasi*-paralelismo. Ou seja, eficientemente alocando tempo de execução de código em um único processador para concluir várias tarefas diferentes, criando uma ilusão de paralelismo entre as mesmas. Além, a parte de "Tempo Real"se refere aos padrões rígidos que o sistema impõe para execução de tarefas (TANENBAUM; BOS, 2014).

A principal estrutura de dados de um Sistema Operacional é o de *Task*, *Tarefa* ou *Thread*. É uma estrutura contendo:

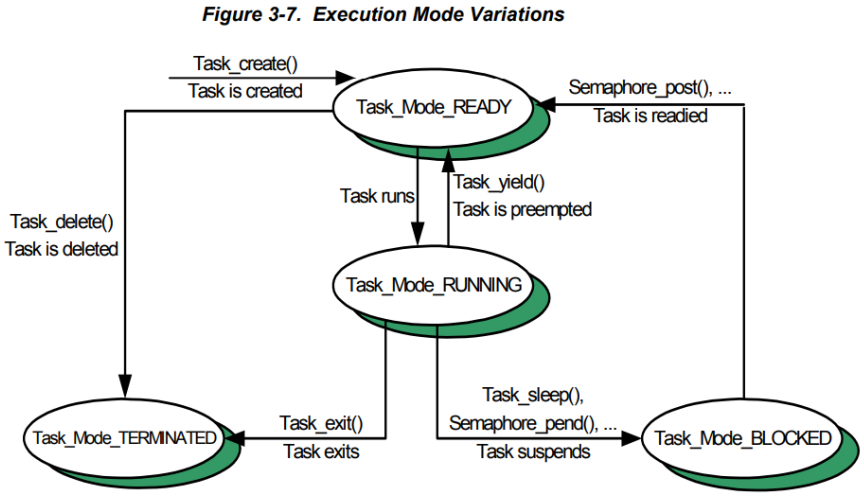

Figura 3.4: Fonte: Texas Instruments

1. A rotina (ou função) que deverá ser executada.
2. Metadados sobre que acompanham a rotina. Estes são o modo atual de execução da rotina, sua prioridade e tamanho de memória disponível para o uso da rotina.

Os metadados sobre a execução de uma tarefa, chamados de estado de execução mudam durante a execução da tarefa e podem ser `READY, RUNNING, BLOCKED` ou `TERMINATED`. A passagem de um estado para outro é sempre igual, e compõe a "vida de execução"de uma tarefa (INSTRUMENTS, s.d.).

Na figura abaixo é possível observar a vida de execução de uma tarefa.

Existem três blocos conceituais dentro de um Sistema Operacional: O *Kernel*, *User Space* e os *Middleware*. Essa divisão se objetiva por segregar a memória, afim de manter segurança na execução de aplicações (TANENBAUM; BOS, 2014).

O Kernel é a parte que tem controle total sobre o sistema, possuindo acesso irrestrito à memória do sistema,

sendo responsável pelo controle das interações entre software e hardware, alocando recursos e executando tarefas.

O *User Space* carateriza a parte da memória onde programas de usuário podem ser escritos, e podemos acessar funcionalidades do hardware via uma textitAPI - Application Programming Interface. Essa API configura um exemplo de *Middleware*. Outros tipos de middleware pode ser códigos relacionados com conectividade (Bluetooth, TCP/IP), sistemas de arquivo ou gerenciamento de energia dos módulos embarcados.

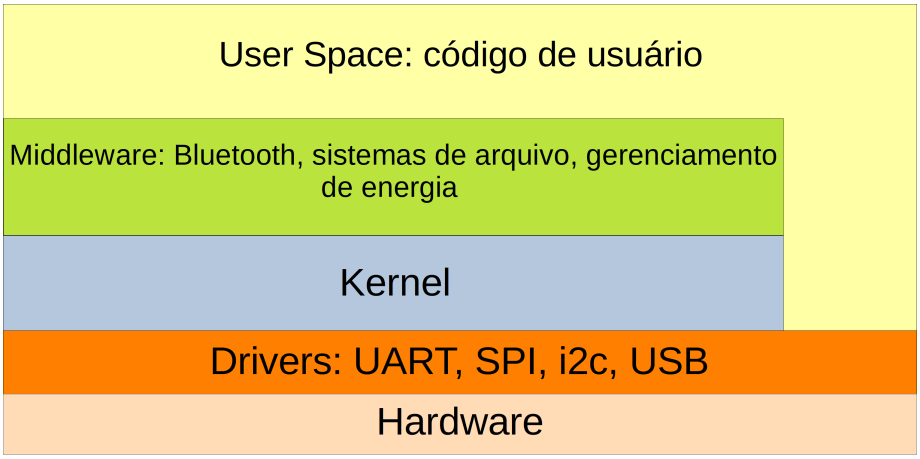

Figura 3.5: Fonte: O autor

O programa que realiza a funcionalidade de *quasi*-paralelismo num RTOS é o escalonador, que reside no Kernel. A função principal do escalonador é criar uma fila das tarefas disponíveis escritas pelo programador e garantir que estas possam ser executadas dentro de um tempo limite chamado de *time-slice* (na ordem de microssegundos). O escalonador pode ser observado, essencialmente, como um algoritmo de organização que minimiza o tempo de execução levando em consideração os metadados de cada tarefa(TANENBAUM; BOS, 2014).

Para garantir que cada tarefa possa ser executada dentro

do *time-slice*, o escalonador pode dinamicamente fazer a *preempção* de quaisquer tarefa por outra. Ou seja, se uma tarefa está sendo executada (modo de execução `RUNNING`), o escalonador pode parar a execução da mesma se

- Seu *time-slice* tiver acabado ou
- Exista outra tarefa de prioridade mais alta que esteja no estado `READY`.

Tais sistemas são de suma importância para diversas aplicações na indústria e na ciência, sendo exaustivamente utilizado em diversos setores, como robótica, equipamentos biomédicos e quaisquer sistemas sensíveis a requerimentos de tempo. O compreendimento desses mecanismos são essenciais para a programação do ambos firmware dos módulos de recepção e amostragem.

Ademais, a Texas Instruments disponibiliza juntamente com os `LAUCHPAD-XL` um Sistema Operacional em Tempo Real denominado `TIRTOS` juntamente com um kit de desenvolvimento (SDK), este último contendo um Middleware que possibilita a utilização do protocolo Bluetooth na programação das placas. Por estes, fazemos uso das facilidades do hardware de Bluetooth que existem nessa plataforma.

O TIRTOS é um software capaz de gerenciar funções e tratadores de interrupção escritos em linguagem de programação, levando em consideração questões de tempo de execução e prioridade(INSTRUMENTS, s.d.). A empresa, no entanto, não disponibiliza o código do mesmo, mas sim uma *API - Application Programming Interface*, que permite seu uso. Essa *API* contém *drivers*, ou camadas de abstração entre o código e o uso do hardware. Por exemplo, um driver de timer para controlar ciclos de tempo, ou um driver de periférico como *SPI*.

Entre essas APIs, existem drivers de periféricos (como SPI, timers e pinos de propósito geral), mecanismos de criação de tarefas e sistemas de gerenciamento de energia das placas.

A empresa também disponibiliza uma *IDE - Integrated Development Environment*, ou Ambiente Integrado de Desenvolvimento chamado `Code Composer`. Esse ambiente contém um editor de texto, compilador, depurador e várias ferramentas de análise em tempo real para diversas plataformas.

Por este software também disponibilidade exemplos de aplicação que podem ser extendidos e modificados livremente, sob licença aberta. Um desses é o *Simple Port Profile*, que é um firmware para conectar duas placas de desenvolvimento *LAUNCHPAD-XL* de tal forma a transferir dados numa configuração bidirecional entre as duas.

Esse exemplo foi compilado, estudado e analisado, visando compreendimento base do Sistema Operacional. O projeto também usa esse exemplo de aplicação como projeto-base.

3.3 Desenvolvimento de firmware para testes de sinais simulados de aúdio

A partir disso, foram iniciados estudos sobre como o sistema de aquisição e transmissão criado por (MATTOS, 2018). Durante esse passo, notamos as limitações promovidas pela construção e configuração padrão do hardware da plataforma *CC2640R2*.

O principal esforço se deu a trabalhar dentro de limitações extremas de memória volátil (memória não persistente). Com apenas 8`kb` de memória do tipo volátil, sendo que o Sistema Operacional em uso pode ocupar cerca de 25%

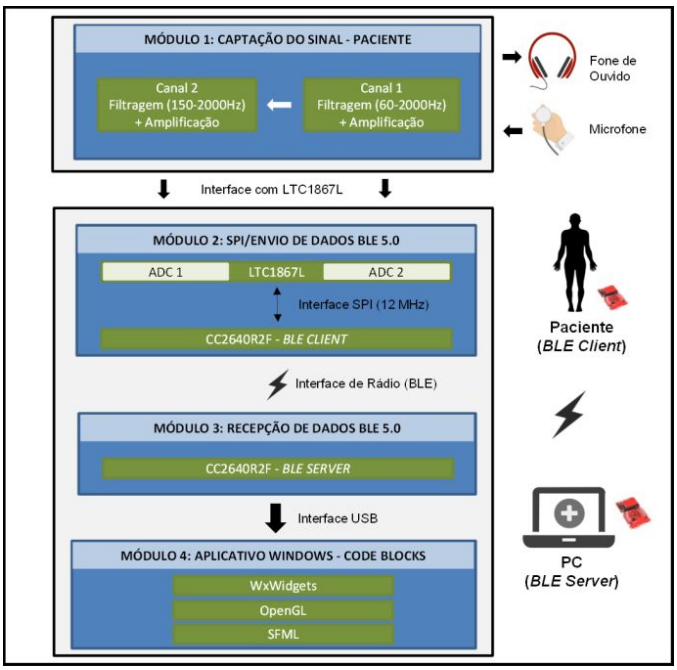

Figura 3.6: Fonte: (MATTOS, 2018)

desse valor, fazendo com que inserção de rotinas novas necessitem de cuidado para evitar possíveis *overflows*, ou estouro de memória por sua insuficiência.

O sistema conta com um *ADC* de 16 *bits* de dois canais modelo *LTC1867LCGN* da *Linear Devices®*. Este se conecta a uma front-end desenvolvida por [trabalho que desenvolveu a frontend]

O projeto pode ser utilizado para se obter sinais de com pouca interferência e distúrbio para pacientes. Os sinais podem, *a posteriori* ser processados e analisados por técnicas específicas de processamento de sinal. A interface também pode ser vista como um estetoscópio eletrônico de baixo custo e performance comparável aos dispositivos existentes no mercado, que são de código fechado e valor proibitivo para uma grande parcela de possíveis aplicações.

A comunicação $ADC \rightarrow CC2640$ é realizada pelo padrão *SPI*, tal como citado na seção 2.1 com as seguintes confi-

gurações:

Parâmetro	**Valor**
Modo de comunicação CC2640R2	Mestre
Modo de comunicação LTC1867L	Escravo
Taxa do clock	12MHz
Tamanho da Amostra	16 bits
CPOL (Clock Polarity)	0
CPHA (Clock Phase)	0
Modo	Callback

Tabela 3.1: Parâmetros de configuração *SPI*

Foram corrigidos avisos gerados por conversões digitais implícitas que existiam nos valores amostrados do *ADC* quando levadas ao buffer de transmissão utilizado pelo projeto.

Para disparar a aquisição desse periférico é utilizado um relógio em modo periódico que faz a requisição dos sinais a cada $100\mu s$.

O sistema foi então testado com sinais alternados tanto gerados pelo próprio microcontrolador quanto externos e obteve resposta satisfatória.

3.4 Desenvolvimento de Aplicativo para Notebook

O projeto original também conta com um aplicativo para Notebooks e Computadores Windows, que captura os pacotes transmitidos via Bluetooth pelo *Client* do sistema por emulação de porta serial via USB. O primeiro problema foi a falta de documentação sobre o projeto, que não provinha de instruções de compilação. As bibliotecas utilizadas no projeto também eram de versões antigas, com diversos problemas de interoperabilidade.

Ultrapassando o problema de compilação do projeto, percebemos que o mesmo utilizada a *API* do próprio sistema Windows NT. O código utilizado para acesso ao pe-

riférico da porta serial emulada sobre USB era datado, não se comportando bem em novas versões do sistema Windows NT. Decidimos então criar um novo programa que capture as informações de quaisquer porta serial, com a necessidade de fácil manutenção e pouca modificação entre sistemas operacionais host. Dessa forma, foi escrito um programa na linguagem de programação `Python 3.8` que obtém os dados da porta serial e os salva para pós-processamento, que mostrou bom desempenho. Se necessário, métodos de pós-processamento podem ser programados de maneira não obtusa no programa, gerando os dados de maneira rápida.

Outra vantagem desse novo programa é que o mesmo foi escrito com princípios de programação *concorrente*. No programa anterior, caso o usuário estivesse fazendo aquisição de dados, o programa necessariamente teria terminar a aquisição para começar outra rotina ("travando"a tela do programa), não fazendo total uso da capacidade processamento disponível nos notebooks e microcomputadores modernos.

Essa capacidade fez uso da biblioteca `asyncio`(ASYNCIO..., s.d.), de Python, que implementa código concorrente baseado em `async/await`.

Esse programa também define um tratador de códigos de operação e uma interface estilo *Shell* entre as partes integrantes do sistema. Via códigos predefinidos, se pode ativar operações em quaisquer uma das placas. Esse método será utilizado na implementação da inicialização da aquisição de sinais via interrupção externa (próxima seção).

3.4.1 Código para atendimento de interrupção externa

Considerando que acessar o equipamento notebook que pode estar a alguns metros de distância do conjunto paciente, examinador e equipamento Bluetooth, surgiu a neces-

sidade de um botão que dispare o programa de computador supracitado.

Para a implementação desse *feature*, foi utilizada uma interrupção externa do *client* (**Módulo 3**), disparada pelo botão esquerdo do mesmo (podendo ser reconfigurado para qualquer pino de propósito geral da placa). Como esse módulo fica perto do paciente, é possível realizar a operação sem movimentação até o notebook ou até o server (**Módulo 4**).

O fluxo de tratamento dessa interrupção para inicialização remota da gravação acontece da seguinte forma:

1. O botão é pressionado no **Módulo 3** disparando uma interrupção externa.

2. Essa interrupção externa faz com que seu tratador envie uma série de dados pré-definidos chamados de *código de operação* via bluetooth para o **Módulo 4**.

3. Esse código de operação é enviado pelo **Módulo 4** via USB ao notebook/microcomputador.

4. No código do aplicativo presente notebook/microcomputador existe um loop que faz *polling* dos dados recebidos do **Módulo 4**. Dentro desse loop, existe um código de tratamento que localiza o código recebido no fluxo de dados.

5. Esse código de tratamento levanta um evento que envia via **Módulo 4**, por bluetooth, um outro código de operação que é detectado pelo **Módulo 3**, inicializando a operação de transmissão e aquisição dos dados.

O código abaixo é uma cópia exata do console do aplicativo desenvolvido após o botão esquerdo ser pressionado:

```
$ python .\serial_reader.py
$ Iniciando shell do estetoscópio
$ >
$ Código de operação recebido
$ Inicializando aquisição e salvando dados
$ Aquisição completa
```

3.5 Aprimoramentos Realizados

Uma das propostas concebidas pelo orientador foi a integração de um acelerômetro e giroscópio, dispositivo que mede aceleração e posição absoluta, de forma a obter dados espaciais do tórax do paciente. A partir dessa proposta foi elaborado uma interface parar aquisição de dados de pequenos sensores que contém tanto acelerômetro quando giroscópio.

O periférico de giroscópio e acelerômetro escolhido foi o `MPU6050`. Seu baixo custo, alta precisão e plena disponibilidade oferecem um bom dispositivo. O `MPU6050` é conta com 6 *ADCs*, cada um de 16 *bits*. Cada um desses *ADCs* é responsável pela digitalização de um eixo do acelerômetro ou do giroscópio embarcados na *breakout board*. Também oferece um filtro passa-baixas digitalmente programável, descartando a necessidade de pós-processamento e baixo consumo (em torno de $3.6mA$ de acordo com a fabricante).

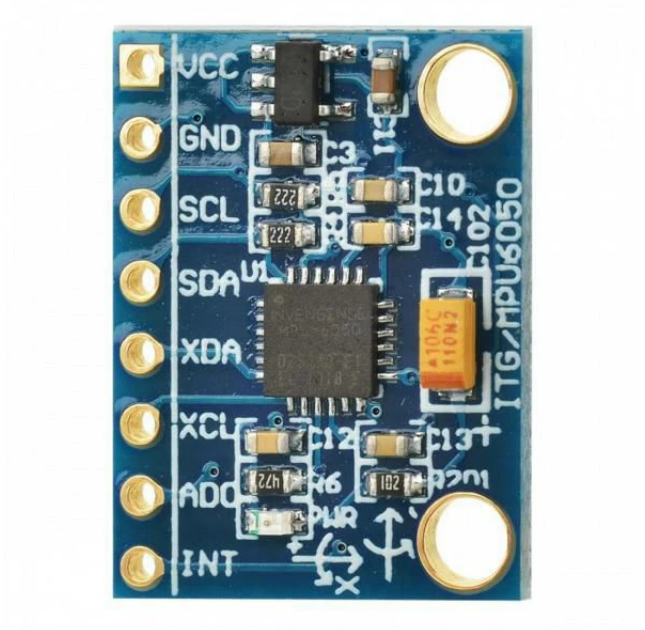

Figura 3.7: `MPU6050` *breakout board*. Fonte: www.easytronics.com.br

O `MPU6050` possui um sistema simples de endereçamento, que permite endereçar até dois chips usando apenas um periférico i^2c. Como a plataforma atual possui dois desses periférico, é possível o endereçamento de até 4 combos de sensores de posição espacial. Qualquer pino pode ser reprogramado para funcionar como esse protocolo.

Abaixo são listados os parâmetros de configuração i^2c.

Parâmetro	**Valor**
Modo de comunicação CC2640R2	Mestre
Modo de comunicação `MPU6050`	Escravo
Taxa do clock	400kHz
Tamanho da Amostra	16 bits

Tabela 3.2: Parâmetros de configuração i^2c

A metodologia de programação desse periférico se deu no desenvolvimento de um *driver* do mesmo, que abstraí rotinas de baixo nível do periférico via uma *API*. Como no programa escrito em Python para recebimento dos dados

pela serial, optamos por uma forte filosofia de portabilidade, escrevendo a biblioteca de uma maneira portável e agnóstica à plataforma, podendo ser utilizada em outros microcontroladores e até outros projetos.

A taxa de aquisição desse sensor foi programada em função da taxa de aquisição do *ADC*. A cada disparo do relógio, existe uma simples lógica de contador que divide esse clock afim de realizar a aquisição do acelerômetro + giroscópio. Dessa forma, a taxa de aquisição do acelerômetro + giroscópio nunca pode ser maior do que a do *ADC*. De qualquer forma a taxa de 12MHz do *ADC* é maior que a taxa de 1KHz do acelerômetro e 8kHz do giroscópio.

Capítulo 4

RESULTADOS E DISCUSSÃO

As condições para realização dos testes não foi propícia devido aos eventos externos ocorridos no ano de 2020, impossibilitando tanto contato quanto acesso aos equipamentos e profissionais necessários para continuar os trabalhos normalmente, dadas as questões sanitárias. De qualquer forma, houveram testes regulares em todas as partes do projeto, o que garante o funcionamento em ambiente de desenvolvimento. À exceção da aquisição de sinais de pacientes, todos os objetivos do projeto foram atingidos.

Dado que a forma de conexão dos equipamentos e seu hardware continuaram constantes na realização deste trabalho, a distância de operação entre os equipamentos é, de forma aparente, 10 a 5 metros, a mesma realizada por (MATTOS, 2018).

O sistema sofre ainda, com falta de estabilidade em certos pontos da conexão. Um aumento nos tempos de atualização do Bluetooth já feitos no trabalho anterior mostraram uma melhora, apesar disso, quedas ainda existem. Se cogita que uma das possíveis soluções seja portar o projeto para utilizar uma versão mais nova do firmware distribuído pela fabricante, que conta com possível correção de bugs e

ofereça maior estabilidade.

A falta de estabilidade geral do sistema também é ocasionada pela pouca memória volátil existente no sistema. O tamanho de pacote transmitido pelo protocolo sofreu aumento para acomodar os dados do giroscópio e acelerômetro, causando maior uso na memória volátil.

Como o firmware escrito para os módulos foi realizado em nível alto e portável dentro da linha suportada pela fabricante, e o driver do acelerômetro e giroscópio também foi escrito com este escopo de portabilidade em mente, se sugere o uso de uma plataforma com maior memória volátil disponível, ou plataforma que possa fazer acesso de memória externa a unidade do microprocessador. A organização interna do código e arquitetura geral do firmware foi desenvolvida em torno da pouca memória volátil disponível. Visando a manutenção do código, uma programação mais orientada e seguindo de forma mais próxima a filosofia de Sistemas Operacionais também se beneficiaria com o aumento da quantidade de memória.

A principal motivação educacional nesse trabalho consistiu do estudo de ambientes de programação, plataformas profissionais de desenvolvimento, compiladores, e principalmente de Sistemas Operacionais em Tempo Real e programação paralela, que foi empregada em todos os níveis de desenvolvimento do sistema. Dentro de Sistemas Operacionais, a ênfase foi dada na depuração e resolução de problemas, essenciais no escopo prático de desenvolvimento.

O estudo também incluiu uma sólida introdução as camadas do protocolo bluetooth, com exemplos práticos de como essa tecnologia pode ser empregada no desenvolvimento de sistemas embarcados, incluindo conhecimento de pontos fortes e limitações da tecnologia.

Referências

AH, Omre. Bluetooth low energy: wireless connectivity for medical monitoring. **J Diabetes Sci Technol**, v. 4,2, n. 457-63, 2010.

ASYNCIO - Asynchronous I/O¶. [S.l.: s.n.]. Disponível em: ¡https://docs.python.org/3/library/asyncio.html¿.

BARRY, Richard. **Mastering the FreeRTOS™ Real Time Kernel**. [S.l.].

CARVALHO V. O.; SOUZA, G. E. C. O estetoscópio e os sons pulmonares: uma revisão da literatura. **Revista de Medicina**, v. 86, n. 4, p. 224–231, 2007.

HUANG, Ying hui et al. The respiratory sound features of COVID-19 patients fill gaps between clinical data and screening methods. **medRxiv**, Cold Spring Harbor Laboratory Press, 2020. DOI: 10.1101/2020.04.07.20051060. eprint: https://www.medrxiv.org/content/early/2020/04/10/2020.04.07.20051060.full.pdf. Disponível em: ¡https://www.medrxiv.org/content/early/2020/04/10/2020.04.07.20051060¿.

INSTRUMENTS, Texas. **TI-RTOS Kernel (SYS/BIOS) User's Guide**. [S.l.].

MATTOS, Willian Daniel de. **Dissertação (mestrado)**. [S.l.: s.n.], 2018.

PANWAR, G.; MISRA, S. Inside Bluetooth Low Energy (Gupta, N.) [Book Review]. **IEEE Wireless Communications**, v. 24, n. 4, p. 2–3, 2017.

SIG, Bluetooth. **Bluetooth 5 Core Specification**. [S.l.: s.n.], 2019.

STALLMAN, Richard M.; DEVELOPERCOMMUNITY, GCC. **Using The Gnu Compiler Collection: A Gnu Manual For Gcc Version 4.3.3**. Scotts Valley, CA: CreateSpace, 2009. ISBN 144141276X.

TANENBAUM, Andrew S.; BOS, Herbert. **Modern Operating Systems**. 4th. USA: Prentice Hall Press, 2014. ISBN 013359162X.