INSTITUTO FEDERAL DE SANTA CATARINA

GUSTAVO CONSTANTE

Monitoramento remoto das colmeias de abelhas: desenvolvimento de PCB para o módulo colmeia.

São José - SC

Agosto/2022

MONITORAMENTO REMOTO DAS COLMEIAS DE ABELHAS: DESENVOLVIMENTO DE PCB PARA O MÓDULO COLMEIA.

Trabalho de conclusão de curso apresentado à Coordenadoria do Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Marcos Moecke

São José - SC

Agosto/2022

Gustavo Constante
Monitoramento remoto das colmeias de abelhas: desenvolvimento de PCB para o módulo colmeia./ Gustavo Constante. – São José - SC, Agosto/2022-
82 p. : il. digital.

Orientador: Marcos Moecke

Monografia (Graduação) – Instituto Federal de Santa Catarina – IFSC
Campus São José
Engenharia de Telecomunicações, Agosto/2022.

1. Sensoriamento. 2. Consumo energético. 3. Placa de circuito impresso. 4. Colmeias de abelhas I. Marcos Moecke. II. Instituto Federal de Santa Catarina. III. Campus São José. IV. Monitoramento remoto das colmeias de abelhas: desenvolvimento de PCB para o módulo colmeia.

GUSTAVO CONSTANTE

MONITORAMENTO REMOTO DAS COLMEIAS DE ABELHAS: DESENVOLVIMENTO DE PCB PARA O MÓDULO COLMEIA.

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 26 de agosto de 2022:

Prof. Marcos Moecke, Dr.
Orientador
Instituto Federal de Santa Catarina

Prof. Arliones Stevert Hoeller Jr, Dr.
Instituto Federal de Santa Catarina

Prof. Roberto de Matos, Dr.
Instituto Federal de Santa Catarina

Este trabalho é dedicado à minha família.
Pai, mãe, esposa e filhos que aguardaram
ansiosamente por esse momento.

AGRADECIMENTOS

Agradeço aos meus pais, Carlos e Neide, por me incentivar e ajudar incondicionalmente em todos os momentos.

A minha esposa Deise, por todo apoio ao longo desses anos.

Aos meus filhos queridos Luiz e Loise, por serem a motivação e me fazer entender o verdadeiro sentido da vida.

Ao meu orientador, professor Marcos Moecke, por toda sua dedicação e comprometimento em tornar esse trabalho possível.

E a todos os meus colegas, que em algum momento colaboraram para que esse momento chegasse.

"Peçam, e será dado;
busquem, e encontrarão;
batam, e a porta será aberta.
Pois todo o que pede recebe;
o que busca encontra;
e àquele que bate, a porta será aberta."
(Bíblia Sagrada, Mateus 7:7-8)

RESUMO

Ter acesso as informações de uma colmeia de abelhas de forma não invasiva, com precisão, baixo consumo de energia e garantir a mobilidade da colmeia é o principal desafio de um sistema de sensoriamento remoto de colmeias de abelhas. As tecnologias associadas a Internet das Coisas (do inglês *Internet of Things* (IoT)) podem apresentar uma solução para este problema, gerando uma base de dados para análise e tomada de ações para benefício da apicultura. Este trabalho tem por objetivo desenvolver um *hardware* capaz de coletar informações sensíveis para a criação de abelhas e enviar para uma base de dados online utilizando transmissão LoRa, com um baixo consumo de energia que possibilite diminuir componentes como bateria e painel fotovoltaico. Como consequência espera-se aumentar a mobilidade no momento do manejo das colmeias e reduzir o custo de implantação do sistema de monitoramento.

Palavras-chave: Sensoriamento. Consumo energético. Placa de circuito impresso. Colmeias de abelhas.

ABSTRACT

Having access to information from a bee hive in a non-invasive way, with precision, low energy consumption and ensuring the mobility of the hive is the main challenge of a bee hive remote sensing system. Technologies associated with the Internet of Things (IoT) can provide a solution to this problem, generating a database for analysis and taking actions for the benefit of beekeeping. This work aims to develop a *hardware* capable of collecting sensitive information for beekeeping and sending it to an online database using LoRa transmission, with a low energy consumption that makes it possible to reduce components such as battery and photovoltaic panel. As a consequence, it is expected to increase mobility at the time of hive management and reduce the cost of implementing the monitoring system.

Keywords: Sensing. Energy consumption. Printed circuit board. Beehive.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

ADC *Analog-to-Digital Converter* . . . 23

API *Application Programming Interface* . . . 45

CAD Projeto Assistido por Computador, do inglês (*Computer-Aided Design*) . . . 35

CAM *Computer Aided Manufacturing* . . . 38

CPU *Central Process Unit* . . . 21

CSS *Chirp Spread Spectrum* . . . 20

DRC *Design Rule Check* . . . 37

EMC compatibilidade eletromagnética . . . 31

EMI interferência eletromagnética . . . 31

EPAGRI Empresa de Pesquisa Agropecuária e Extensão Rural de Santa Catarina . . . 33

GPIO *General Purpose Input/Output* . . . 35

GPRS *General Packet Radio Services* . . . 17

IFSC Instituto Federal de Santa Catarina . . . 33

IoT *Internet of Things* . . . 7

ISM *Industrial Scientific and Medical* . . . 29

I2C *Inter-Integrated Circuit* . . . 9

MOSI *Master Output Slave Input* . . . 26

MISO *Master Input Slave Output* . . . 26

PCB Placa de Circuito Impresso, do inglês *Printed Circuit Board* . . . 30

RAM *Random Access Memory* . . . 21

RISC *Reduced Instruction Set Computer* . . . 21

RTC *Real-Time Clock* . . . 28

SCK *Serial Clock* . . . 25

SDA *Serial Data* . . . 25

SDR *Software Defined Radio* . . . 47

SPI *Serial Peripheral Interface* . . . 9

SS *Slave Select* . . . 26

TTN *The Things Network* . . . 19

USART *Universal Synchronous and Asynchronous Receiver and Transmitter* . . . 22

LISTA DE SÍMBOLOS

V_{DD}	Tensão de alimentação da fonte
I_D	Corrente de dreno de um transistor
GND	Referência de terra

SUMÁRIO

1 INTRODUÇÃO

O mel é considerado o produto apícola mais fácil de ser explorado, sendo também o mais conhecido e aquele com maiores possibilidades de comercialização. Além de ser um alimento, é também utilizado em indústrias farmacêuticas e cosméticas por suas conhecidas ações terapêuticas (FREITAS et al., 2014).

A evolução e o uso de ferramentas tecnológicas vem tornando-se essencial para que o produtor torne a atividade apícola cada vez mais rentável. Utilizar a tecnologia aliada a produção na apicultura não é uma prática nova, como pode-se observar em alguns trabalhos já realizados e produtos existentes no mercado.

Jesus (2017) criou um sistema de monitoração remoto com atuadores para ajuste da temperatura interna das colmeias, baseado no controlador ESP-12E, que possui um módulo *wireless* ESP8266[1] integrado, para realizar a leitura dos sensores e acionar os atuadores.

Em outro trabalho, Kridi (2014) realizou o sensoriamento da temperatura interna das colmeias de abelhas, para monitorar a temperatura na qual ocorre a enxameação, utilizando um microcontrolador ATMega328[2] e um módulo XBee - PRO 900HP[3] para enviar os dados para um servidor remoto.

Também existem sistemas que oferecem comunicação cabeada, como o proposto por Dutra (2016), o `Beehiveior` faz o sensoriamento de umidade e temperatura, utilizando um microcontrolador ATMega328 e envia os dados para um sistema de monitoramento remoto via uma conexão *ethernet*.

No mercado mundial, também existem produtos tais como o Arnia (2014) que monitora variáveis como umidade, temperatura, peso, acústica e chuva. Também possui uma *interface* no servidor, onde além de monitorar, pode-se criar alertas para envio de mensagens de texto ou e-mail cadastrado. Esse produto oferece como opções de transmissão dos dados o *General Packet Radio Services* (GPRS) ou modem via satélite para enviar as informações ao servidor.

Transmitir informações de umidade, temperatura e acústica da colmeia via rede *wireless*, é a proposta de Pollenity (2019). Possui uma plataforma fechada semelhante ao Arnia (2014) e *hardware* não informado pelo fabricante.

O Quadro 1 apresenta as soluções acadêmicas e produtos no mercado que consistem

[1] https://www.espressif.com/en/products/hardware/esp-wroom-02/overview
[2] https://www.microchip.com/wwwproducts/en/ATmega328
[3] https://www.digi.com/products/embedded-systems/rf-modules/sub-1-ghz-modules/xbee-pro-900hp

Quadro 1 – Comparação entre trabalhos e produtos existentes

Autores	**Variáveis Monitoradas**	**Conectividade**
Jesus (2017)	temperatura e umidade	WiFi
Kridi (2014)	temperatura	ZigBee
Dutra (2016)	temperatura e umidade	Ethernet
Arnia (2014)	temperatura, umidade, peso, acústica e quantidade de chuva	GPRS ou satélite
Pollenity (2019)	temperatura, umidade e acústica	WiFi

Fonte: Elaborada pelo autor.

em monitorar métricas bem conhecidas e que impactam diretamente no bem-estar da colmeia, como temperatura e umidade. A convergência para tecnologias sem-fio é na maioria dos casos uma necessidade pelo fato de que a localização das colmeias são normalmente em regiões de difícil acesso para instalação de sistemas fotovoltaicos *off-grid*, que são sistemas de alimentação isolados ou autônomos.

Em relação às soluções analisadas anteriormente, pode-se perceber que quase todas transmitem a informação usando alguma tecnologia sem fio, e também a maior parte é alimentada por energia da rede comercial ou baterias eletroquímicas. A bateria normalmente visa dar uma autonomia ao sistema quando há falta de energia, por outro lado esse modelo de alimentação dificulta o deslocamento das colmeias para novas localizações.

O objetivo deste trabalho, é desenvolver uma placa de circuito impresso para utilizar um microcontrolador com baixo consumo energético em *sleep-mode*, contendo conexões para as interfaces dos periféricos de sensoriamento e módulo de transmissão de dados utilizados no sistema. Viabilizando a utilização de componentes compactos para alimentação do módulo, como bateria e painel fotovoltaico, e garantindo também a mobilidade e boa área de cobertura com uma tecnologia de transmissão de baixa potência e longo alcance (*LoRa*).

Este trabalho está organizado em quatro capítulos. Neste primeiro foi apresentada uma introdução sobre o tema. No segundo é apresentado a fundamentação teórica e todas as tecnologias envolvidas que embasa este trabalho. No terceiro apresenta-se o desenvolvimento do projeto. Finalizando, no quarto capitulo comenta as conclusões obtidas e sugere-se trabalhos futuros.

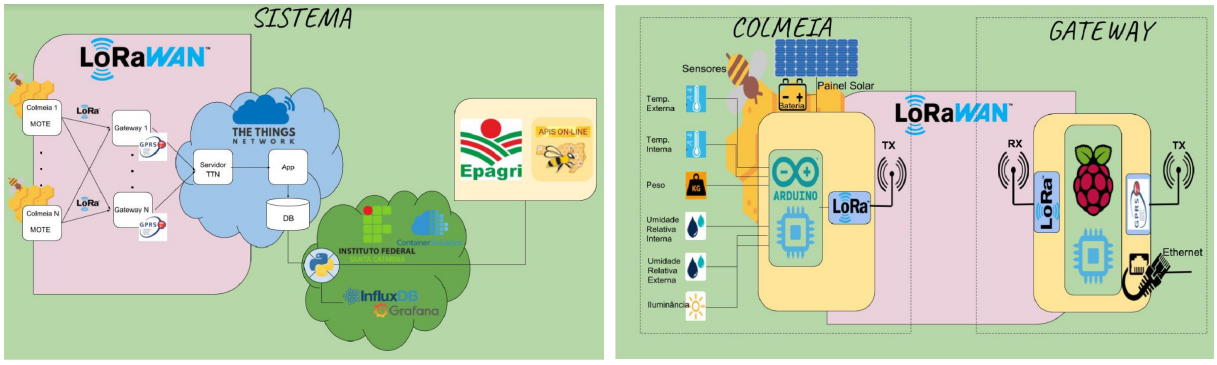
2 FUNDAMENTAÇÃO

2.1 Sistema RF-Abelhas

`RF-Abelhas` é o conjunto de tecnologia com base no conceito de `Internet das Coisas` (IoT) que foi proposto para atender a demanda de monitorar variáveis sensíveis para colmeias de abelhas (MOECKE et al., 2018).

Segundo MOECKE et al. (2018) o sistema pode ser dividido em módulo `Colmeia`, módulo *Gateway LoRa-IP*, rede *The Things Network* (TTN) e módulo *Cloud*. Na Figura 1a é apresentado o panorama de como o sistema está segmentado e como ocorre a interação entre os módulos e na Figura 1b apresenta-se a tecnologia *LoRaWan* escolhida para efetuar a transmissão dos dados coletados pelo módulo `colmeia` até o *gateway LoRa*.

Figura 1 – Sistema proposto para o `RF-Abelhas`



(a) (b)

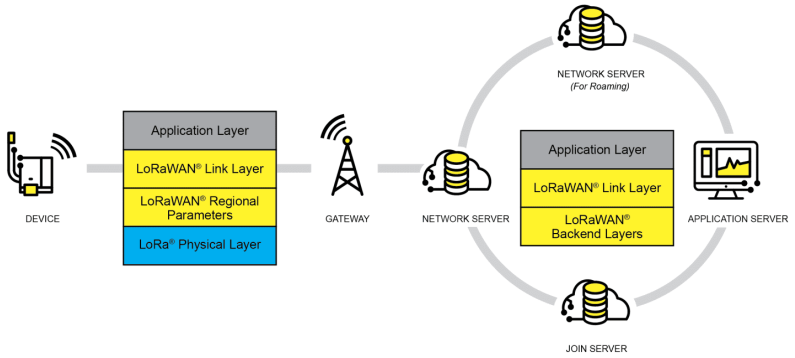
Fonte – (MOECKE et al., 2018)

O sistema prevê o monitoramento de variáveis que afetam diretamente as condições das abelhas como temperatura, umidade e peso da colmeia. Assim como variáveis que são referentes ao funcionamento do sistema como corrente, tensão e luminosidade.

2.2 LoRaWan

LoRaWan é uma especificação para protocolos de redes de baixo consumo que visam aplicações para IoT. A Figura 2 apresenta a rede *LoRaWan* e a implementação de sua arquitetura em estrela onde os *gateways* recebem mensagens dos *end-points* e as reencaminham para um servidor central onde são utilizadas por aplicações. A associação *LoRa Alliance* é responsável por atualizações e melhorias nas especificações do protocolo (ALLIANCE, 2022).

Figura 2 – Arquitetura LoRaWan



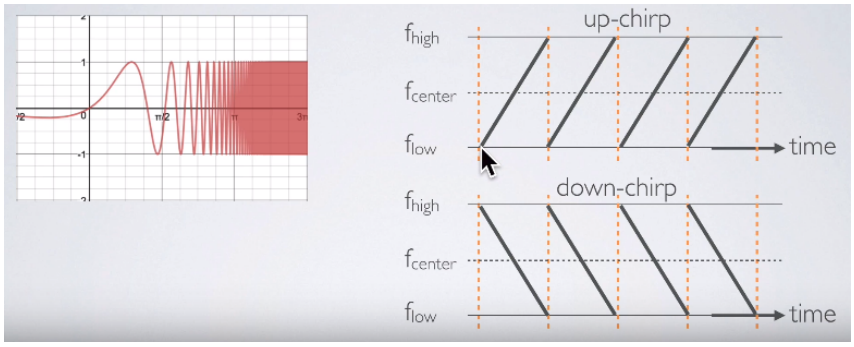
Fonte – (ALLIANCE, 2022)

2.2.1 Modulação LoRa

Um sistema *LoRa* é baseado em uma comunicação bidirecional entre dispositivos chamados *End-point* e *Gateway*, dentro deste sistema os dois dispositivos devem possuir a capacidade de enviar e receber mensagens.

Segundo Giserman (2019) a modulação *LoRa* é baseada na técnica *Chirp Spread Spectrum* (CSS) que carrega a informação na variação de sua frequência e ocorrem de forma cíclica, ou seja, quando se atinge o limite superior da frequência ela retorna para o limite inferior, formando os chamados saltos. Como podemos observar na Figura 3 estes saltos são decodificados e formam os *bits* que estruturam as mensagens LoRa.

Figura 3 – Modulação *CHIRP* no tempo e na frequência



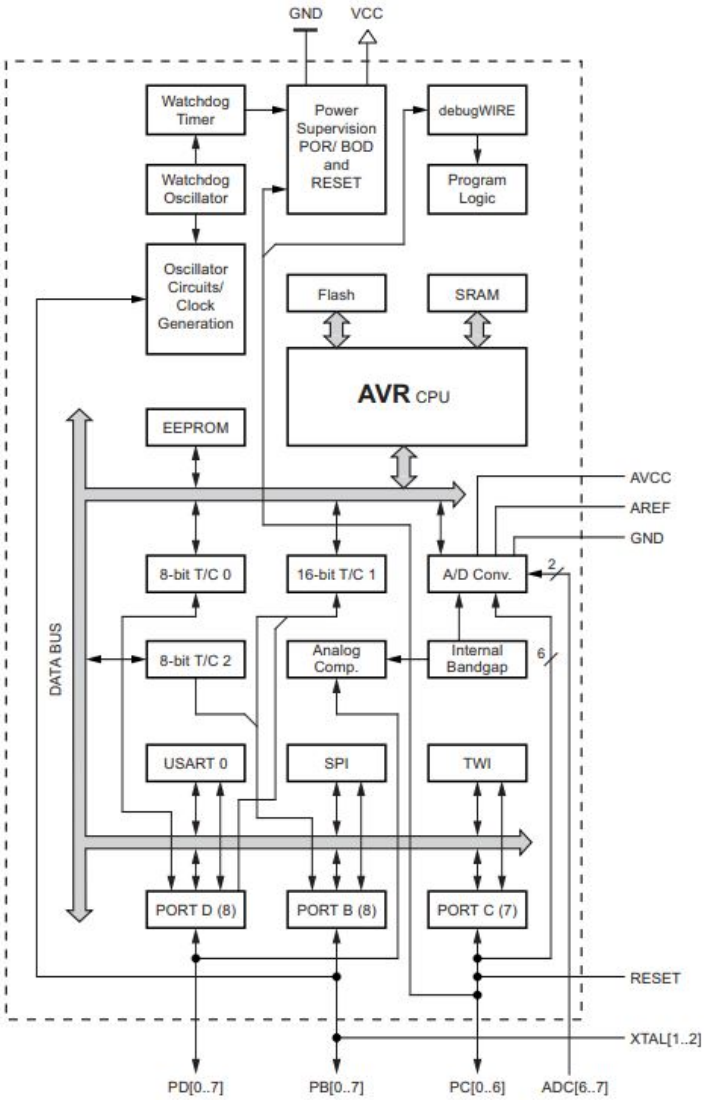
Fonte – (GISERMAN, 2019)

A taxa de dados da modulação está diretamente relacionada com o fator de espalhamento espectral (*SF*) utilizado e o tempo de duração no ar do *Chirp* é dado pelo *SF*, quanto maior este tempo menor a taxa dados, pois cada bit dura mais para ser transmitido, tornando a transmissão mais robusta. O fator de espalhamento pode ir de *SF7* até *SF12* e a cada passo do *SF* o tempo no ar da transmissão é dobrado.

2.3 Microcontroladores

Microcontroladores são circuitos integrados que possuem em seu encapsulamento, além da Unidade de Processamento Central (do inglês, *Central Process Unit* (CPU)), todos os periféricos necessários para realizar uma aplicação. Os periféricos inclusos em cada microcontrolador podem variar, porém, os mais comuns são memórias, barramentos, temporizadores, portas de comunicação, conversores de sinal analógico em digital entre outros (ANDRADE, 2010). Na Figura 4 é apresentado o diagrama de blocos do microcontrolador `ATMega328P` com os periféricos existentes e as associações existentes entre eles.

Figura 4 – Diagrama de blocos do microcontrolador ATMega328P



Fonte – (MICROCHIP, 2018)

O microcontrolador `ATMega328P` é baseado na arquitetura aprimorada *Reduced Instruction Set Computer* (RISC) e executa instruções de 8 bits. Possui *32kB* de memória *FLASH* e *2kB* de memória *Random Access Memory* (RAM). Conta com periféricos

de comunicação importantes como SPI, I2C, *Universal Synchronous and Asynchronous Receiver and Transmitter* (USART), além de 23 portas digitais configuráveis como entrada ou saída, sendo 2 com característica importante de interrupções externas, as quais, são utilizadas para que componentes externos possam ativar o microcontrolador quando está em estado de operação de baixo consumo.

2.3.1 Consumo de potência

De acordo com Pedroni (2010), o consumo de potência pode ser separado em estático e dinâmico. A potência consumida no circuito em repouso, ou seja, quando não há transições de *clock* chamamos de estática, enquanto a potência dinâmica é consumida ao se trocar dados pelo circuito a cada transição positiva ou negativa dos sinais. Como os sinais são síncronos com o *clock*, então fica associado às mudanças do sinal de *clock*. Estas duas situações geram a potência total média (P_T):

$$P_T = P_{estática} + P_{dinâmica}, \tag{2.1}$$

na qual,

$$P_{estática} = V_{DD} * I_D, \tag{2.2}$$

e

$$P_{dinâmica} = \frac{1}{T}\int_0^T v(t)i(t) = \frac{1}{T}\int_0^T V_{DD}i(t)dt = \frac{V_{DD}}{T}\int_0^T i(t)dt = \frac{V_{DD}}{T}C_L V_{DD} \tag{2.3}$$

$$P_{dinâmica} = C_L V_{DD}^2 f. \tag{2.4}$$

A parcela de potência estática em sistemas digitais é normalmente desprezível, mas a medida que o tamanho dos transistores da tecnologia CMOS foi diminuindo, as correntes de fuga passaram a crescer e a potência estática passa a ser um valor não desprezível (PEDRONI, 2010). No caso de sistemas que permanecem a maior parte do tempo inativo, a potência estática passa a ter uma importância maior no consumo total. Por outro lado, a potência dinâmica é diretamente proporcional à tensão (V_{DD}) que se está alimentando o circuito, a frequência do *clock* (f) e a capacitância total do nó de saída (C_L) utilizada, conforme mostra a Equação 2.4. Por isso, projetos digitais tendem a utilizar tensões de alimentação cada vez mais baixas (PEDRONI, 2010).

Em sistemas digitais que garantem mobilidade e por isso são alimentados por baterias o consumo de potência é um fator muito importante que limita o tempo de operação sem que a bateria seja recarregada.

2.3.2 Modos de operação de baixo consumo

Os microcontroladores dispõem de mecanismos que visam minimizar o impacto da atividade das rotinas do microcontrolador na vida útil da bateria. Os chamados *sleep-modes*(em português, modos de suspensão), possuem características que atendem aplicações diversas desativando circuitos dos microcontroladores que não estiverem sendo utilizados e assim economizando energia.

Os principais modos de suspensão dos microcontroladores são: *Idle*, *Standby* e *Power-Down*. No caso específico do microcontrolador `ATMega328P`, o modo *Power-Down* mantém em operação apenas os circuitos periféricos de interrupção externa, *I2C*, *Watchdog* e *Brown-out Detector*. Sendo os dois primeiros para receber alguma sinalização externa para ativação do microcontrolador e os dois últimos circuitos de detecção e prevenção de falha elétrica. Adicionalmente é possível desativar periféricos diretamente estando no modo normal, reduzindo ainda mais o consumo de potência do sistema (MICROCHIP, 2018).

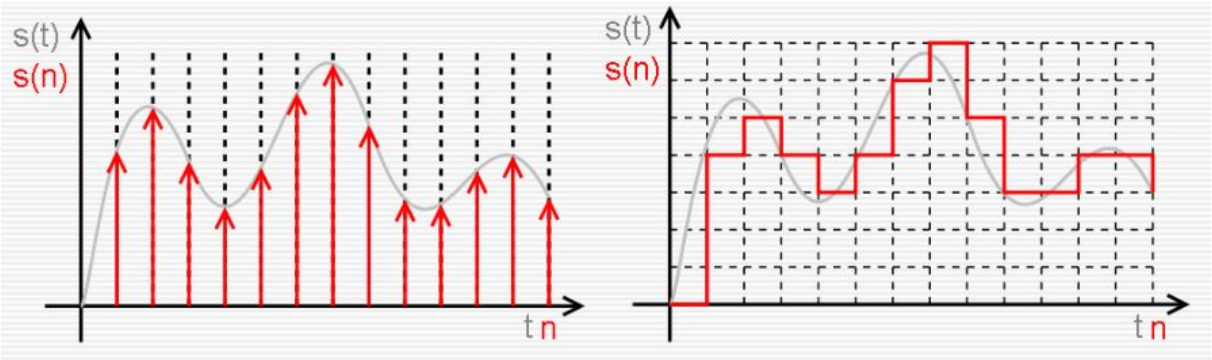
2.4 Interfaces

De acordo com Tocci (2003), conectar dispositivos de entrada e saída de um sistema que possuem características elétricas diferentes denomina-se interfaceamento. Sua função é condicionar os circuitos do acionador ao da carga e torná-lo compatível com os requisitos da mesma.

2.4.1 Portas digitais

Tocci (2003) diz que sinais analógicos são grandezas físicas que variam com valores contínuos ao longo do tempo e sinais digitais são discretos, assumindo quantidades de níveis lógicos finitos como ilustra a Figura 5. Desta forma, para ser utilizado no mundo digital, deve-se converter o sinal analógico com um *Analog-to-Digital Converter* (ADC) tornando-o discreto.

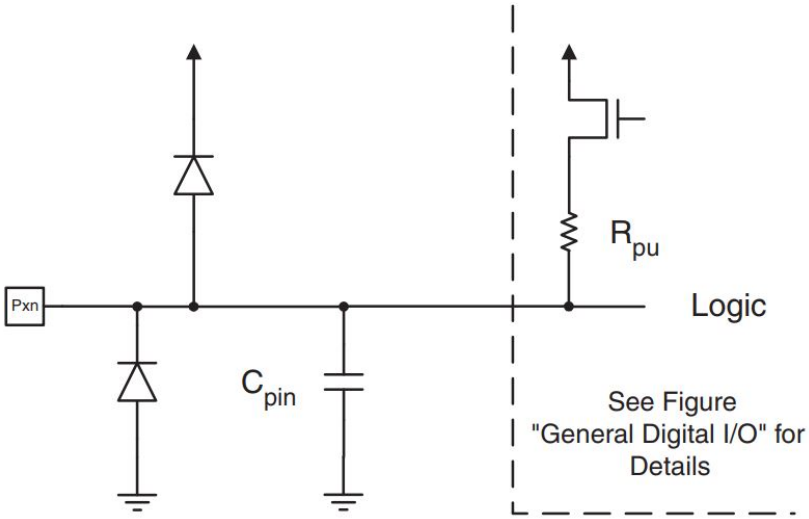
Figura 5 – Sinais analógicos sendo convertidos para digitais



Fonte – (HORAK, 2010)

Portas digitais podem ser configuradas como entrada ou saída, onde cada configuração possui um conjunto de circuitos associados para garantir seu correto funcionamento e proteção, como mostra a Figura 6.

Figura 6 – Circuitos associados a uma porta digital



Fonte – (MICROCHIP, 2018)

2.4.2 Comunicações seriais

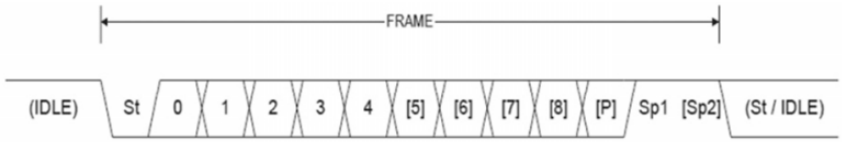
Segundo Forouzan (2007), com apenas um condutor é possível realizar uma comunicação serial diminuindo o custo em relação a transmissão por n condutores paralelos. A comunicação serial pode ser realizada de forma assíncrona, e também síncrona. No modo assíncrono, quadros de n *bits* de dados, são precedidos pelo *start bit* e sucedido por 1 a 2 *stop bits*, conforme ilustra a Figura 7. Nas transmissões síncronas os bits de dados são transmitidos de maneira contínua, não havendo *bits* extras e intervalos de repouso da linha de transmissão, o que possibilita uma maior taxa de transmissão.

2.4.2.1 Protocolo *USART*

De acordo com Lima (2012), o protocolo USART por possuir configuração flexível é amplamente utilizado em sistemas eletrônicos. Sua comunicação é *full-duplex* pois tem circuitos de transmissão e recepção independentes, suporta variar o tamanho do seu quadro entre cinco e nove *bits*, possui mecanismo de paridade por *hardware* e diferentes velocidades de transmissão.

No microcontrolador `ATMega328P` esta porta de comunicação também é utilizada para gravar o *firmware* do microcontrolador.

Figura 7 – Quadro do protocolo *USART*, comunicação assíncrona



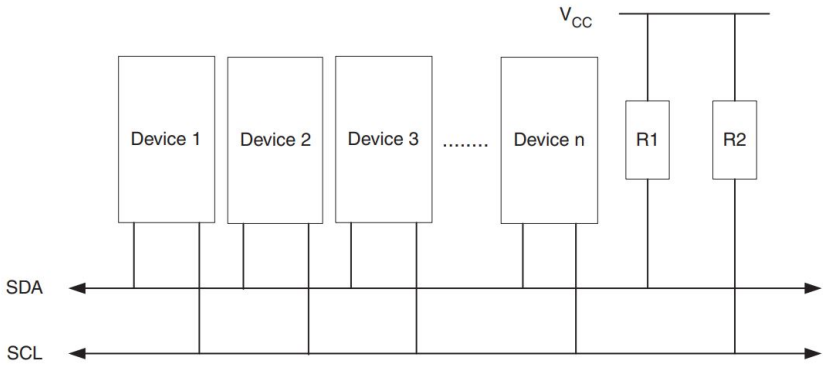
Fonte – (LIMA, 2012)

2.4.2.2 Protocolo *I2C*

O protocolo I2C, implementa comunicação serial síncrona com apenas dois fios, onde o *Serial Clock* (SCK) é responsável por gerar o relógio para sincronismo e o *Serial Data* (SDA) realiza a transmissão dos dados(NXP, 2014).

Por possuir apenas um fio de transferência de dados o I2C utiliza uma comunicação *half-duplex*, ou seja, envia e recebe dados entre os dispositivos em momentos diferentes. Utiliza o modelo de *mestre-escravo* para troca de mensagens, onde cada *escravo* é identificado por um endereço único. O campo de endereço possui 7 bits, permitindo ter um número máximo de 128 dispositivos conectados no barramento como podemos observar na Figura 8.

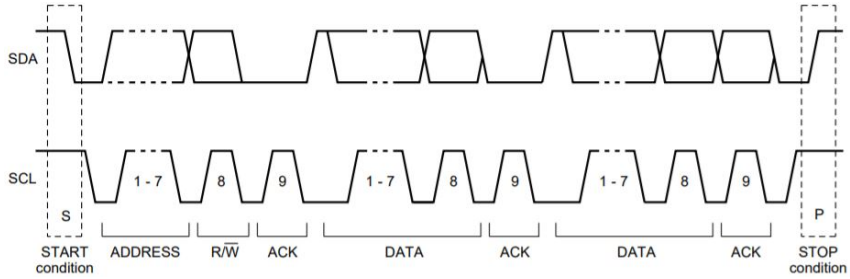
Figura 8 – Barramento I2C



Fonte – (MICROCHIP, 2018)

O início da transmissão do protocolo I2C é realizada por um dispositivo *mestre* com um sinal de *start bit*, seguido do envio do endereço de sete *bits* do dispositivo *escravo* e um *bit* sinalizando se será realizado uma leitura ou escrita no escravo. Após a sequência inicial do *mestre*, o *escravo* responde com um *bit* de reconhecimento (também chamado de *acknowledge* ou apenas *ack*), seguido dos *bits* de dados. Para finalizar a transmissão e liberar o barramento, o *mestre* sinaliza com um *stop bit*. Esta comunicação pode ser vista na Figura 9, que detalha a sequência completa de uma transmissão do protocolo I2C.

Figura 9 – Protocolo I2C

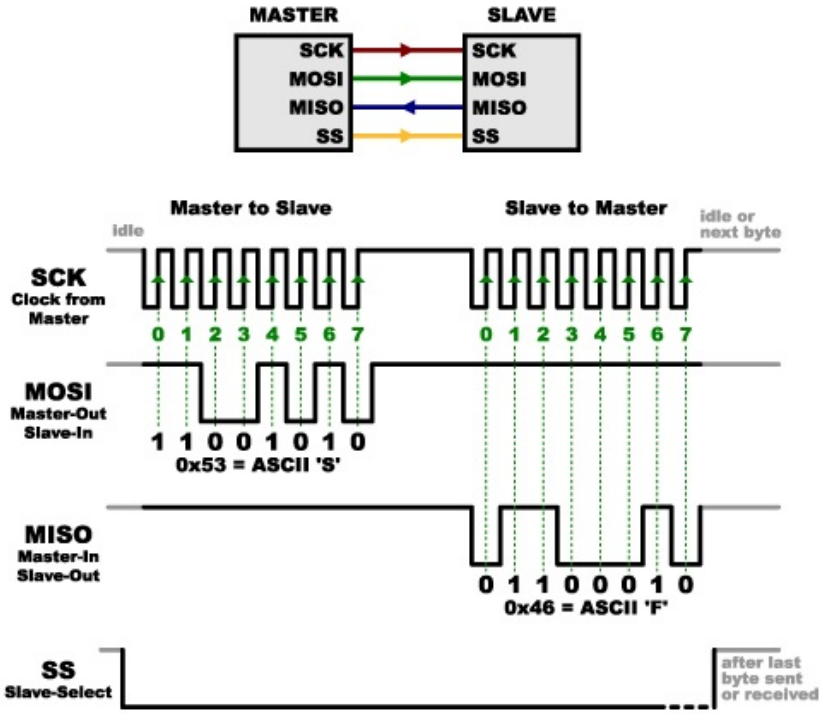


Fonte – (NXP, 2014)

2.4.2.3 Protocolo *SPI*

O *Serial Peripheral Interface* (SPI), é um protocolo serial, síncrono, *full-duplex* que realiza a comunicação utilizando quatro fios, sendo o *Master Output Slave Input* (MOSI) onde se envia os dados do *master* para o *slave*, o *Master Input Slave Output* (MISO) do *slave* para o *master*, para prover a comunicação entre múltiplos *slaves* é utilizado o *Slave Select* (SS) como sinal de controle e a troca de dados é sincronizada pelo SCK (MOTOROLA, 2000).

Figura 10 – Barramento e transmissão SPI



Fonte – (BODYRETPA, 2016)

Na Figura 10 é apresentada a conexão do barramento entre os dispositivos *master* e *slave*, e também como ocorre a sinalização e transferência de dados neste barramento. Para que ocorra um envio de dados, o dispositivo *slave* é acionado pelo *master* por meio

do sinal SS, logo após o conjunto de *bits* é encaminhado pela porta MOSI em sincronismo com o sinal SCK, da mesma forma que a resposta do *slave* para o *master* é enviada através da porta MISO.

2.5 Periféricos

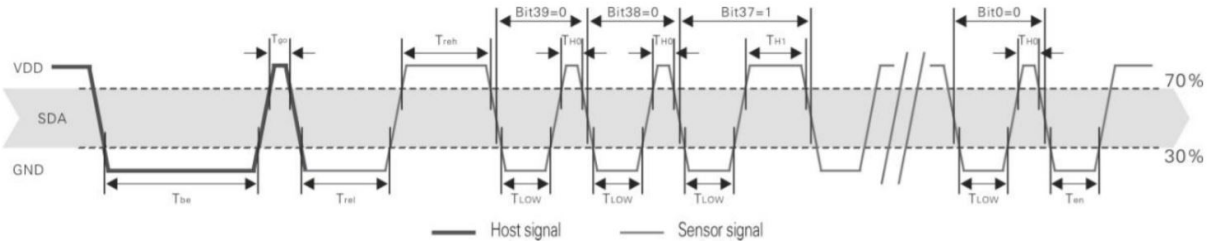
Segundo Bolton (2002), transdutor é um item que possibilita transformar a informação de uma grandeza física para outra forma que possa ser medida facilmente. Para um sistema microcontrolado, deve ser gerado sinais elétricos compatíveis com os níveis de entrada utilizados. Esta seção mostrará os transdutores que serão utilizados neste projeto.

2.5.1 Sensor de temperatura e umidade

O **DHT22** é um módulo que utiliza o sensor **AM2302** que possui resolução de 16 *bits* para medir temperatura e umidade, consegue realizar medições de 0 a 100% de umidade relativa e de -40 a 80°C de temperatura, com precisão de ±2% a 25°C e ±0.5°C respectivamente. Para comunicação com o microcontrolador utiliza uma porta digital onde possui um protocolo serial assíncrono similar ao USART com 40 bits de dados, sendo estes separados em 16 bits de umidade, 16 bits de temperatura e 8 bits de paridade (GUANGZHOU, 2018).

O funcionamento da comunicação entre o microcontrolador e o sensor ocorre de forma bidirecional. Onde o microcontrolador mantém a porta de dados em nível baixo por aproximadamente *1 ms* e em nível alto por *30 us*. Caracterizando um *start-bit*, logo após o sensor envia um pulso de *80 us* confirmando a requisição e envia os 16 bits de umidade, 16 bits de temperatura e por fim os 8 bits de paridade. Os bits '0' são formados por *50 us* em nível baixo e *26 us* em alto, já os bits '1' são formados por *50 us* em baixo e *70 us* em alto conforme mostra a Figura 11.

Figura 11 – Transmissão de bits no barramento SDA do sensor DHT22



Fonte – (GUANGZHOU, 2018)

2.5.2 Sensor de iluminância

A iluminância é a quantidade de fluxo luminoso que incide em uma determinada área. O **BH1750** é um sensor que utiliza um *fotodiodo* com resposta ao espectro muito próximo ao do olho humano.

Possui dois modos de operação para coleta de dados, o *One Time* onde o sensor após realizar a medida volta para *power down* automaticamente e o modo de medida contínua, onde o sensor encaminha as medidas para o microcontrolador de maneira repetida até que o usuário envie um comando que termine a ação. Cada modo de operação possui três resoluções que definem a precisão da leitura. A *Low-Resolution* possui *4lx* de resolução e *16ms* de tempo de resposta. A *Hi-Resolution2* possui *0,5lx* de resolução e leva *120ms* para completar um ciclo de leitura, este modo é indicado para casos onde há necessidade de diferenciar os níveis de escuridão do local monitorado. A recomendada para uso pelo fabricante é a *Hi-Resolution*, que possui *1lx* de resolução e leva os mesmos *120ms* para aquisição da medida pelo fotodiodo. Com os 16 *bits* de resolução que o sensor é capaz de gerar níveis entre 1 e 65535 *lx* e precisão tipica de 20%.

O sensor utiliza barramento *I2C* para comunicação com o microcontrolador e pode optar entre dois endereços (0x23 ou 0x5C) (ADAFRUIT, 2022), de acordo com o nível lógico do o pino *Address* do *CI* (ROHM, 2010).

2.5.3 Sensor de peso

O `HX711` é um módulo de conversão analógico para digital com resolução de 24 *bits*. Possui um multiplexador com dois canais de entrada que podem ser configurados com ganhos de escala de 32, 64 e 128. Pode operar no modo de baixo consumo, onde a corrente é aproximadamente *1uA* e em modo normal de operação pode consumir até *1,5mA* (AVIA, 2018).

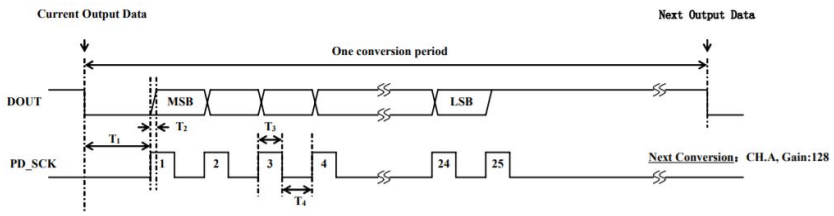
Utiliza entradas diferenciais onde são inseridos quatro células de cargas em um circuito conhecido como *ponte de wheatstone* que alteram seu valor ao aplicar-se uma pressão.

Utiliza comunicação serial síncrona a dois fios para enviar os dados ao microcontrolador. Para sincronizar os dados utiliza o sinal *PD_SCK* com período de *2us* , sendo *1us* em nível alto e *1us* em nível baixo. O sinal *DOUT* encaminha os 24 *bits* de dados que estão em complemento de dois ao microcontrolador conforme a Figura 12.

2.5.4 Relógio de tempo real

O **DS3231** é um *Real-Time Clock* (RTC) de alta precisão com cristal oscilador integrado muito utilizado em sistemas no qual é necessário interromper a energia principal.

Figura 12 – Protocolo serial do HX711



Fonte – (AVIA, 2018)

Possui entrada para bateria *backup*, o que garante a informação de cronometragem. Mantém dados de segundo, minuto, hora, dia, mês e ano. Contém dois alarmes programáveis, podendo ser utilizados para controlar interrupções de sistemas microcontrolados. A comunicação de dados, de leitura e escrita é feita pela porta de comunicação *I2C* (MAXIM, 2015).

Possui um consumo de corrente em modo ativo de *300uA*. Conta com um pino de saída *SQW* que pode ser utilizado para acionar interrupções em microcontroladores.

2.5.5 Sensor de corrente, tensão e potência

O **INA219** é um sensor digital de corrente contínua, tensão e potência que utiliza o método *high-side* de medição com um resistor *shunt* em série entre a fonte e a carga a ser analisada que pode variar de 0 a 26V de tensão. Possui resolução de até 12 *bits* em seu *ADC* e configuração de escala para 16 ou 32V. Quando opera em modo ativo tem um consumo de *700uA* e em modo *power-down* o consumo fica em aproximadamente *6uA*.

A comunicação com o microcontrolador é feita por meio do barramento *I2C* onde pode ser configurado até 16 endereços diferentes, utilizando os pinos *Address* (TEXAS, 2015).

2.5.6 Transceiver LoRa

O módulo de rádio frequência **RFM95PW** é equipado com o *transceiver* **SX1276** da *Semtech* e implementa as técnicas de modulação *FSK*, *GFSK*, *MSK*, *GMSK*, *OOK* e *LoRa*. Possui ajuste de potência de transmissão de até 30dBm para as frequências de 433/470MHz e até 27dBm para as frequências de 169/868/915MHz, frequências estas que estão na banda não-licenciada *Industrial Scientific and Medical* (ISM).

O rádio possui uma sensibilidade de recepção de -136dBm que com o seu ganho de transmissão de 30dBm forma um *link budget* de 166dB, que é a perda máxima permitida em percurso já considerando todas as atenuações inerentes a transmissão.

O consumo de corrente típico do módulo no momento de transmissão é de *300mA* em 169MHz e *650mA* nas demais frequências, já o consumo de corrente na recepção é de

13mA em todas as frequências. A corrente em *modo sleep* é de *5uA*.

Utiliza barramento *SPI* para comunicação com o microcontrolador e duas portas digitais para sinalização de conclusão de envio e recebimento de pacotes, sendo que uma das portas obrigatoriamente deve funcionar como interrupção no microcontrolador (HOPE, 2006).

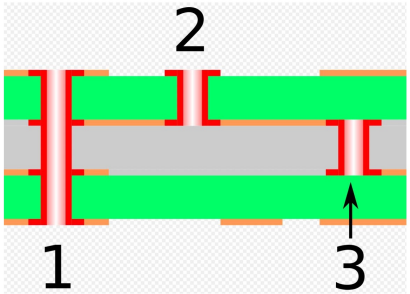
2.6 Placas de circuito impresso

Segundo Mehl (2005) a Placa de Circuito Impresso, do inglês *Printed Circuit Board* (PCB) foi desenvolvida em 1930 por Paul Eisler, inicialmente produzida com *fenolite* como material isolante, foi substituído por fibra de vidro que possui maior resistência a água. Outros materiais como *teflon* e *poliéster* são utilizados em circuitos com frequências mais elevadas por apresentar menor polarização dielétrica.

2.6.1 Camadas, vias, trilhas e ilhas

De acordo com Mehl (2005) uma placa de circuito impresso pode conter uma ou mais camadas formadas por material condutor, no qual o cobre é utilizado por ter uma excelente condução elétrica. A interconexão entre camadas é realizada por orifícios metalizados chamados vias conforme pode-se observar na Figura 13, quando uma via interliga todas as camadas é chamada de *through hole*, ao interconectar uma camada externa com uma interna denomina-se *blind via* e *burried via* ao realizar a interconexão entre duas camadas internas.

Figura 13 – Tipos de vias em circuitos impressos: Via(1), *blind* via(2) e *burried* via(3)



Fonte – (ADLER, 2008)

Conforme em Melo (2001) as trilhas são responsáveis por realizar a conexão elétrica entre os componentes eletrônicos. A área de fixação dos componentes na placa denomina-se ilhas e podem ser do tipo *through hole* no qual há a perfuração da placa ou *surface mounted* cuja solda é aplicada na mesma camada onde o componente encontra-se.

A unidade de medida padrão utilizada em projetos de placas de circuito impresso é a polegada (**inch**), que equivale a 2,54 cm. *Softwares* de desenvolvimentos adotam a unidade **mil**, que significa **um milésimo de polegada**, no qual o valor de referência é 0,1 *inch* ou 100 *mils*. Componentes comercializados mundialmente tem suas distâncias entre pinos dadas em *mils* (LIMA, 2010).

De acordo com Lima (2010) a distância entre trilhas depende da faixa de tensão que será utilizada por cada circuito, no Quadro 2 são mostradas as distâncias mínimas entre trilhas para assegurar um isolamento adequado de acordo com a tensão aplicada.

Quadro 2 – Distância entre trilhas de acordo com a diferença de potencial aplicada

Tensão (V)	**Distância entre trilhas**	
	(mm)	**(mils)**
0-30	0,1	8
31-50	0,6	25
51-100	1,5	60
101-170	3,2	150
171-250	6,4	300
251-500	12,5	500

2.6.2 Roteamento

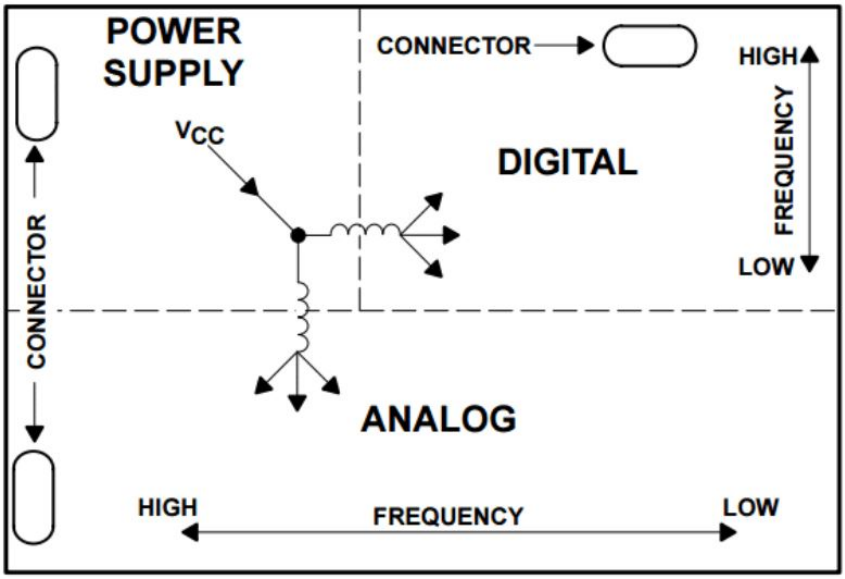
A criação dos roteamentos em uma placa de circuito impresso deve seguir regras para minimizar possíveis problemas com efeitos físicos indesejados como compatibilidade eletromagnética (EMC), interferência eletromagnética (EMI), *loop* de terra, acoplamento de ruído entre outros.

Segundo Lima (2013) utilizar técnicas como a criação de uma malha de terra em PCB multicamadas evita que um sinal ao retornar para o terra do circuito gere efeitos conhecidos como *loops* de terra. Esta técnica também evita que o caminho percorrido pelo sinal de retorno possa vir a formar uma antena, irradiando energia e interferindo em outros circuitos da placa. Com o agrupamento de funcionalidades como mostrado na Figura 14, além de isolar as interconexões com filtros a utilização de malhas de terra independentes nos subsistemas atenua efeitos de EMC.

De acordo com Lima (2010) em circuitos digitais que operam com frequências elevadas, como sinais de *clock*, a distância entre as trilhas de ligação dos componentes devem ser curtas e próximas ao malha de terra. No geral o traçado das trilhas deve ser o mais retilíneo possível, no entanto ao mudar de direção é recomendado utilizar ângulos chanfrados de 45º e quando viável empregar polígonos como mostra a Figura 15, o que diminui a reflexão de sinal e atenua efeitos de irradiação e evita EMI.

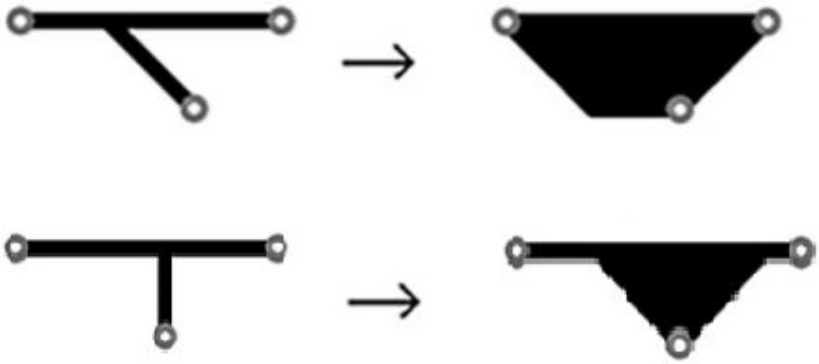
Ruídos de alta frequência são indesejados em componentes como circuitos integrados, pois provocam mal funcionamento na operação de um dispositivo. Desta forma, deve-se

Figura 14 – Agrupamento dos componentes por funcionalidade



Fonte – (SACCO, 2015)

Figura 15 – Emprego de poligonos no roteamento de trilhas



Fonte – (LIMA, 2010)

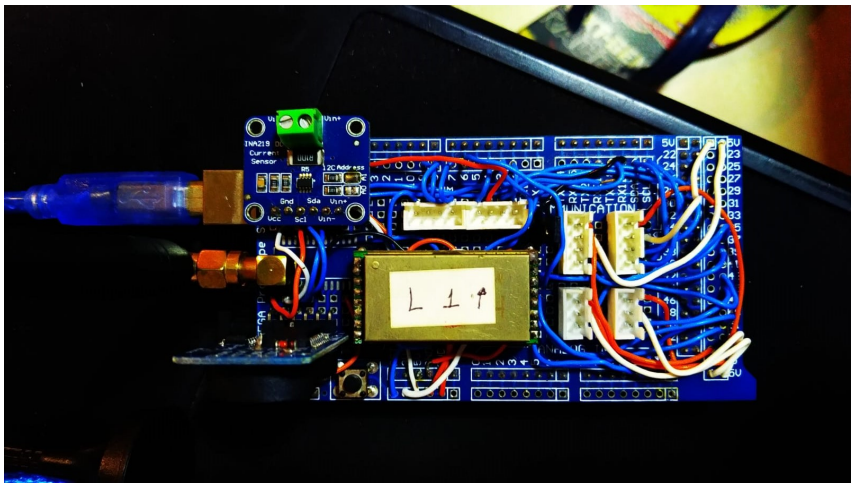
incluir capacitores de desacoplamento entre a alimentação e o terra dos *CI's*, realizando um filtro contra *ripples* indesejados. Capacitores cerâmicos de 100nF são tipicamente utilizados em cada ponto de entrada de alimentação de circuitos integrados (MICROCHIP, 2018).

3 DESENVOLVIMENTO

A proposta deste trabalho é desenvolver uma placa de circuito impresso para o módulo `Colmeia` do projeto `RF-Abelhas`, o qual nasceu de uma parceria entre o Instituto Federal de Santa Catarina (IFSC) e a Empresa de Pesquisa Agropecuária e Extensão Rural de Santa Catarina (EPAGRI) em 2018. Na versão original o protótipo do módulo foi desenvolvido em uma placa Arduíno MEGA equipado com um microcontrolador `ATMega2560`, devido a facilidade de uso de diferentes placas compatíveis, e a ampla quantidade de periféricos e bibliotecas disponíveis para a plataforma Arduíno.

O desenvolvimento da nova versão do hardware do módulo `Colmeia` iniciou com o estudo dos componentes do protótipo em Arduíno, que continha: (i) um sensor de tensão e corrente `INA219`, (ii) um sensor de iluminância `BH1750`, (iii) dois sensores de temperatura e umidade `DHT22`, (iv) um relógio de tempo real `DS3231`, (v) um sensor de peso `HX711` e (vi) um módulo de transmissão `LoRa RFM95PW`. No protótipo todos os componentes eram conectados por meio de fios a uma placa adaptadora conforme mostra a Figura 16.

Figura 16 – Protótipo em Arduíno

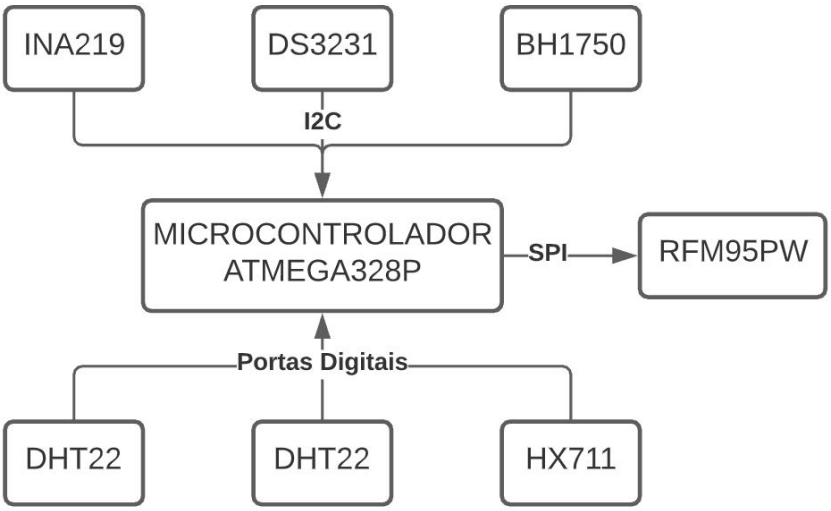


Fonte: Elaborada pelo autor.

3.1 Características funcionais do módulo a ser projetado

Com base no estudo realizado no protótipo, foram identificadas a disponibilidade das interfaces que permitem a interoperabilidade entre os componentes externos e o novo microcontrolador do módulo `colmeia`. A Figura 17 mostra o diagrama de blocos dos componentes e suas interfaces de comunicação com o microcontrolador. A partir da especificação das interfaces e das portas de comunicação necessárias, buscou-se selecionar um microcontrolador de menor custo que atendesse a todos os requisitos do sistema.

Figura 17 – Diagrama de blocos do módulo `Colmeia`



Fonte: Elaborada pelo autor.

Um dos objetivos desse projeto é reduzir o consumo de energia do módulo, sendo necessário escolher um microcontrolador que tenha modos de funcionamento de baixo consumo, tais como *sleep mode*, *power-down* ou *stand-by*. A existência desses modos permite reduzir o consumo de energia do microcontrolador durante o tempo em que não realiza nenhuma operação. Em um sistema de monitoramento é comum se utilizarem ciclos de leitura periódicos que podem ser de minuto em minuto até mesmo a ciclos onde a leitura é feita de hora em hora ou uma vez por dia, dependendo do tipo de dado que se deseja obter.

Quadro 3 – Requisitos de interfaces, modos de operação e memória do módulo `Colmeia`

Tipos de requisitos	**Necessário**	**ATMEGA328P**
Portas digitais exclusivas	6	10
Interfaces I2C	1	1
Interfaces SPI	1	1
Interrupções externas	2	2
Modos de baixo consumo	1	5
Memória flash [kB]	29	32
Memória RAM [kB]	2	2

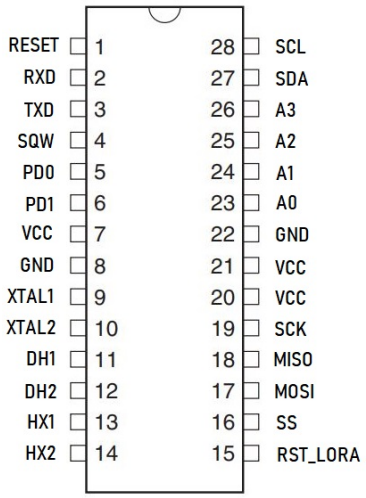
Fonte: Elaborada pelo autor.

O Quadro 3 lista os requisitos necessários e justifica a escolha do microcontrolador `ATMEGA328P` para o desenvolvimento da nova PCB do módulo `Colmeia`. E a escolha foi baseada no atendimento a todos os requisitos mínimos, baixo custo e disponibilidade para aquisição do componente no mercado brasileiro.

A Figura 18 apresenta o mapeamento dos pinos utilizados do microcontrolador `ATMEGA328P`. Pode-se observar que os pinos das *interfaces* SPI, I2C e `serial`, além de

pinos como *clock*, *reset* e interrupções não podem ser alterados, de forma que apenas 10 pinos digitais podem ser utilizados como *General Purpose Input/Output* (GPIO) para integração dos periféricos.

Figura 18 – Mapeamento dos pinos do ATMEGA328P



Fonte: Elaborada pelo autor.

3.2 Desenvolvendo o *hardware*

Para o desenvolvimento da PCB foi utilizado um software de Projeto Assistido por Computador, do inglês (*Computer-Aided Design*) (CAD) para o desenho dos diagramas esquemáticos e projeto da placa de circuito impresso. A ferramenta escolhida foi o *Eagle* CAD da AUTODESK, por dispor de uma versão acadêmica gratuita, cujas limitações de uso não afetam o tipo de projeto desenvolvido em dupla face. Entre os recursos disponibilizados pela ferramenta pode-se citar:

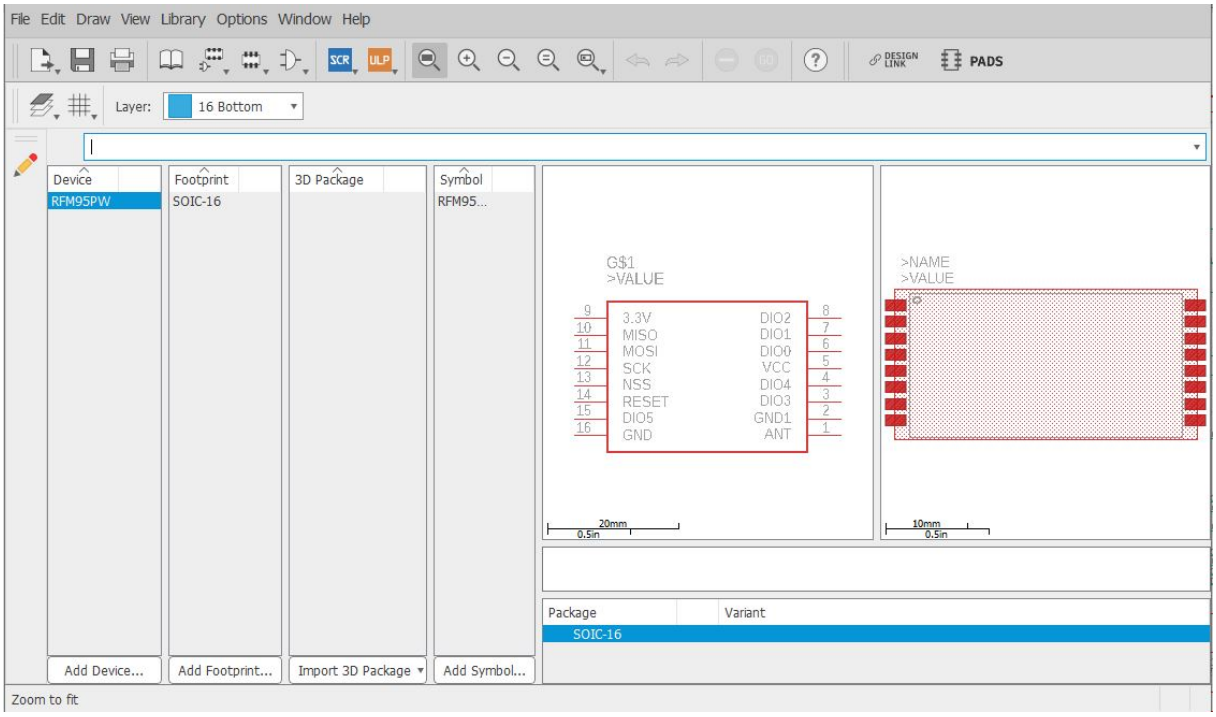
- possibilidade de criação de bibliotecas para componentes;
- importação de bibliotecas de fabricantes e terceiros;
- facilidade de intercambiar entre o editor de diagramas esquemáticos e design de PCB;
- configuração dos parâmetros básicos como trilhas, ilhas e pads;
- função de auto roteamento de trilhas;
- verificação de conexões de componentes;
- exportação dos arquivos *Gerber Files* para fabricação da placa.

3.2.1 Bibliotecas CAD

A biblioteca de um *software* CAD é constituída de um conjunto de informações relacionados ao componente, como modelos de encapsulamento, mapeamento de pinos, e os símbolos que são exibidos no esquemático e no PCB. O *software Eagle CAD* por padrão já contém bibliotecas de diversos componentes, algumas disponibilizadas por fabricantes e outras genéricas criadas pelos desenvolvedores da *AutoDesk*. Além disso, possibilita a criação de novos componentes, que podem ser personalizados pelo usuário do *software*. Outra opção é importar as bibliotecas disponibilizadas por fabricantes de componentes ou criadas por terceiros.

A Figura 19 mostra a biblioteca utilizada para o módulo LoRa `RFM95PW` no projeto da PCB, no caso deste módulo como não foi obtida nenhuma opção disponível para ser importada a forma encontrada foi criar uma biblioteca própria. Pode-se observar que cada componente criado contém o símbolo que é utilizado em cada editor (esquemático e PCB), assim como definições dos nomes que são associados aos pinos, posições de serigrafia e disposição da estrutura física do componente (PADs nos casos de componentes *SMD* e ilhas nos casos de componentes *through-hole*).

Figura 19 – Editor de componentes do EAGLE CAD



Fonte: Elaborada pelo autor.

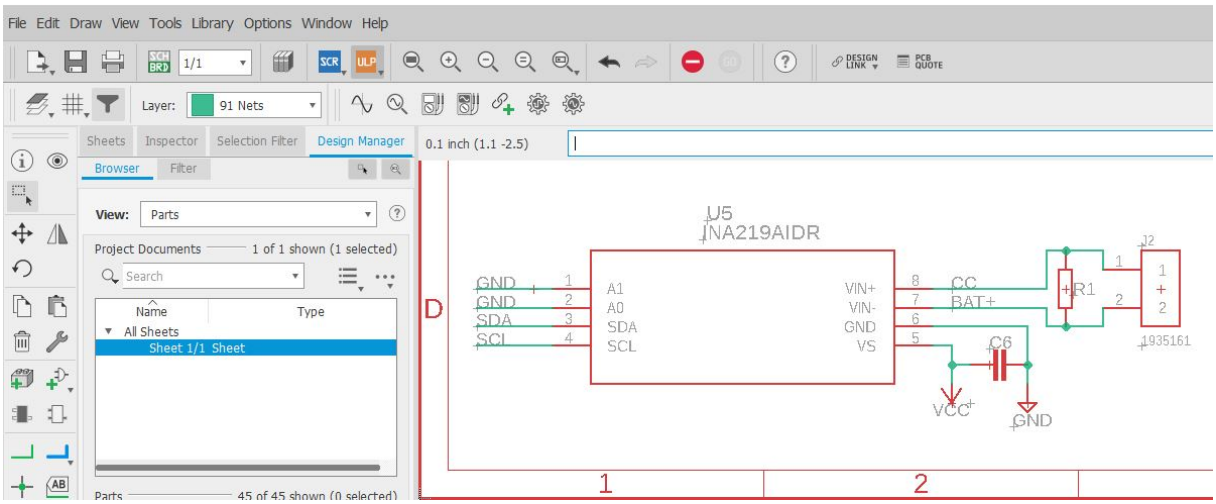
3.2.2 Edição do diagrama esquemático

O diagrama esquemático é a representação de um circuito eletrônico de forma a simplificar sua visualização e entendimento. Um editor de esquemáticos é composto por

um conjunto de ferramentas que permitem manipular os componentes de um diagrama. Este modelo de visualização por vezes é mantido também como parte das documentações de um projeto. O *software Eagle CAD* salva um diagrama esquemático como um arquivo de extensão *".sch"*.

A Figura 20 mostra parte do esquemático referente ao circuito do sensor de corrente e tensão utilizado para o projeto da `colmeia`. Neste editor utiliza-se a simbologia de blocos para apresentar os componentes do diagrama e para interligá-los são usados sinais chamados *"NET"*. Estes sinais recebem um nome que além de identificá-los, serve para mapear pontos que são comuns dentro do esquemático. No Apêndice A está o esquemático completo desenvolvido para o projeto.

Figura 20 – Editor de Esquemático EAGLE CAD



Fonte: Elaborada pelo autor.

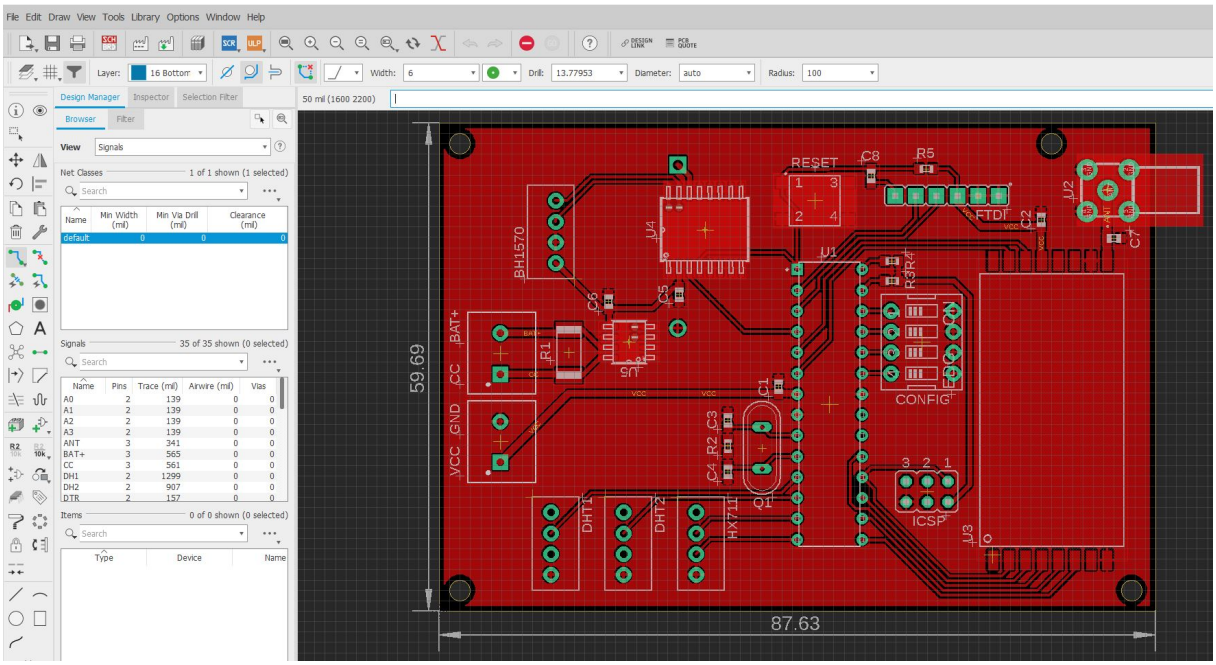
3.2.3 Editor de projeto da PCB

O editor de projeto é a interface de criação da placa de circuito impresso, onde os símbolos criados no editor esquemático são convertidos em *footprints*, como são chamados os símbolos que representam e possuem as dimensões exatas dos componentes reais. O dimensionamento da placa, a disposição e a criação das trilhas de conexões dos componentes são realizadas neste editor. Da mesma forma que o editor de esquemático, o editor de projetos salva um arquivo com extensão *".brd"* para que haja a possibilidade de serem feitas alterações na PCB.

A Figura 21 mostra as ferramentas disponíveis para serem utilizadas na criação das PCB. É apresentado também na figura a camada superior desenvolvida para a placa da `colmeia` (AUTODESK, 2020).

O editor possui a ferramenta *Design Rule Check* (DRC) como mostra a Figura 22, que permite inserir valores padrões para os parâmetros que serão utilizados no desenvolvi-

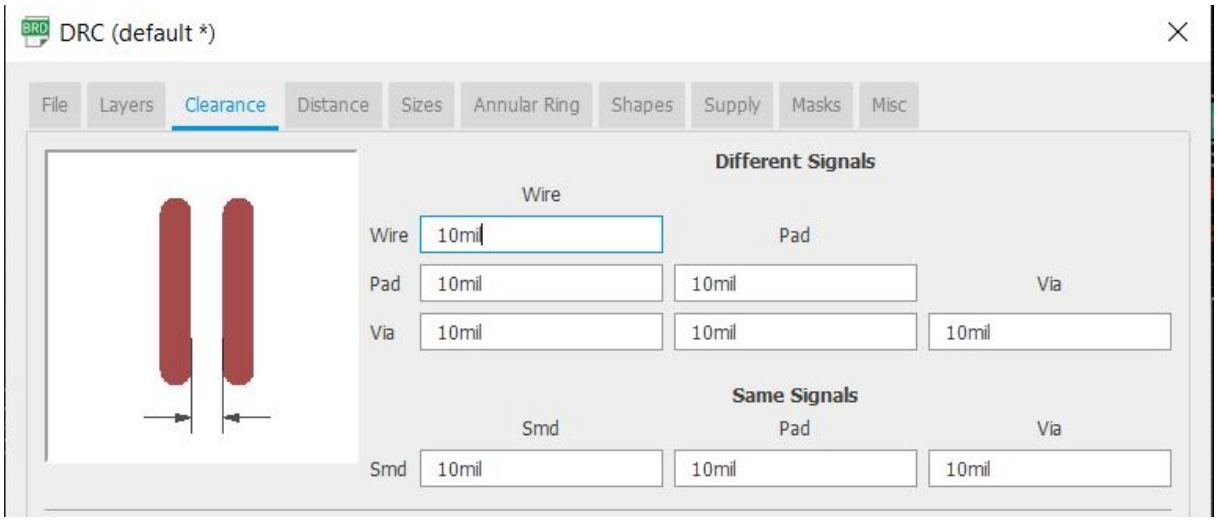
Figura 21 – Editor de projeto de PCB



Fonte: Elaborada pelo autor.

mento da PCB, desta forma ao optar-se pelo roteamento automático da placa o *software* usará como base estes valores em seu algorítmo. Além do auto roteamento o DRC é utilizado para gerar alerta ao usuário em casos de erros que possam comprometer a PCB (AUTODESK, 2016).

Figura 22 – Ferramenta *Design Rule Check*



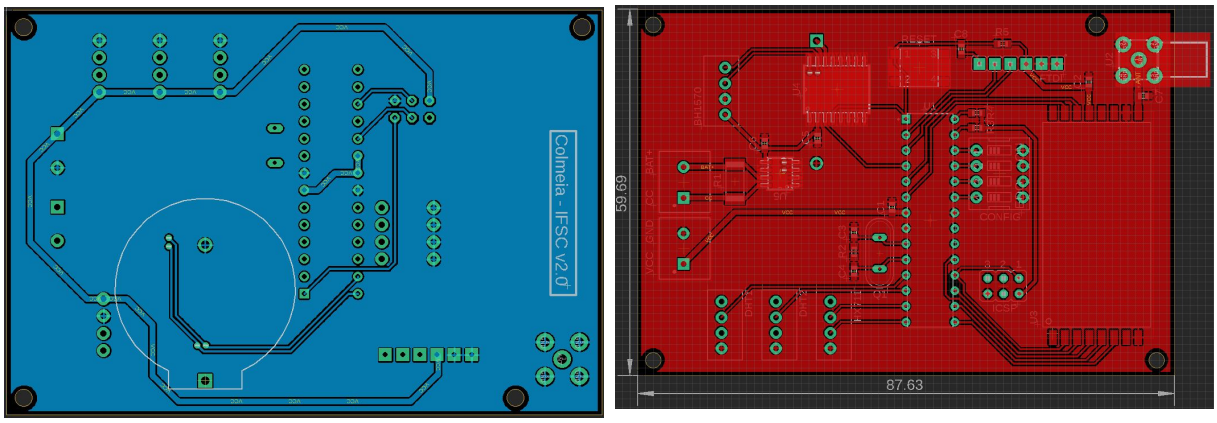
Fonte: Elaborada pelo autor.

3.2.4 Produção da PCB

Após o desenvolvimento do projeto no CAD, obtém-se as camadas superior e inferior da PCB, como é apresentado na Figura 23. A ferramenta *Computer Aided Manufacturing*

(CAM) do *software CAD*, realiza o mapeamento dos dados de localização das trilhas, posição da furação das vias, máscara de solda, serigrafia, entre outras informações da placa desenvolvida e exporta elas para um arquivo conhecido como *Gerber Files*, os quais são utilizados para a manufatura da placa.

Figura 23 – Placas em ambiente de desenvolvimento



(a) Camada inferior (b) Camada superior

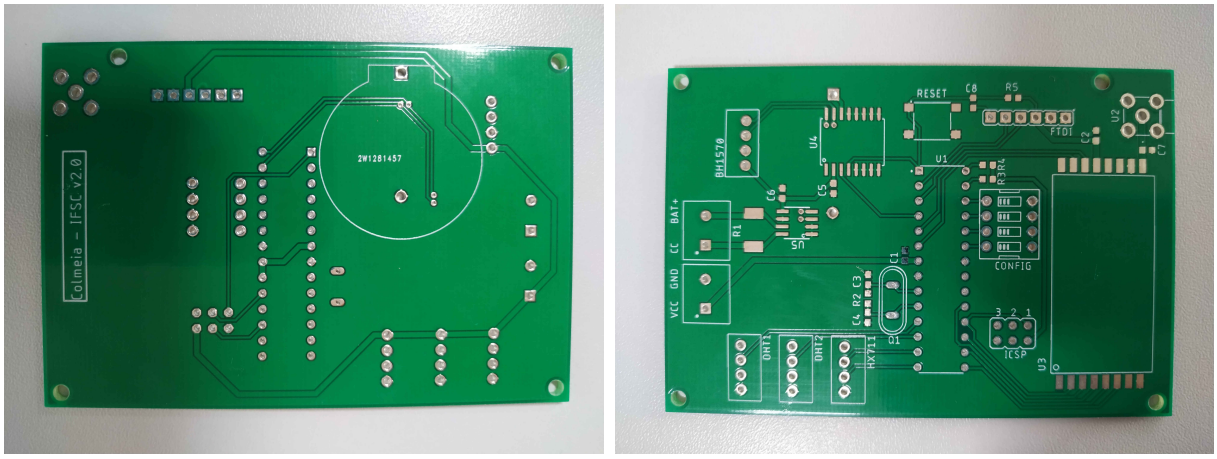
Fonte: Elaborada pelo autor.

As máquinas de manufatura utilizadas na confecção de Placa de Circuito Impresso, do inglês *Printed Circuit Board* (PCB) possuem limitações características de cada equipamento e ao se desenvolver um projeto em *software CAD* você deve configurar os parâmetros inerentes a estas limitações, como por exemplo distância mínima entre ilhas, espessura mínima de trilhas, bitola mínima de vias. Sendo respeitados estes parâmetros diminui-se a chance de problemas ocasionados no processo de confecção das placas.

Para este projeto foi realizado uma consulta em diversas empresas de manufaturas de PCB no Brasil, mas por tratar-se de um protótipo de poucas peças, o custo de produção por unidade se tornou alto. O menor custo obtido nos orçamentos (feito em 25/06/2020) foi de R$38,50 por unidade e com produção mínima de 10 peças. Assim, decidiu-se pela manufatura da PCB fora do país (China) por meio do site <https://www.allpcb.com/small_quantity_pcb_manufacturing.html>, conseguindo-se um custo de U$1,99 para a produção de 6 unidades e com o custo de U$10,67 referente a importação e frete. No fim do processo o valor obtido com o câmbio do dólar a R$5,88 foi de R$74,56, proporcionando um custo de R$12,42 por unidade produzida.

Como podemos observar na Figura 24 as placas entregues pela manufatura possuem excelente qualidade, o material isolante é a fibra de vidro `FR-4 Shengyi TG130`, o cobre possui espessura de *1 oz*, as vias foram metalizadas e as placas receberam uma camada de verniz em seu processo de produção, cuja finalidade é evitar a oxidação do cobre.

Figura 24 – Placa de circuito impresso executadas pela manufatura



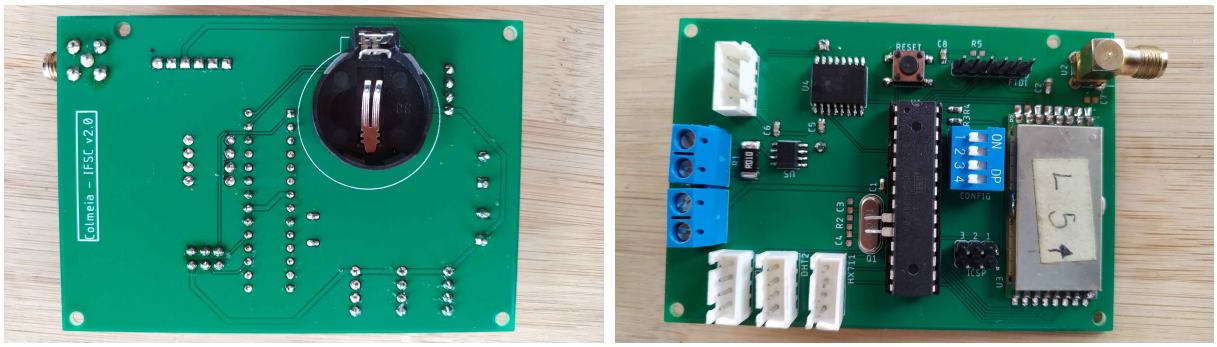
(a) Face inferior (b) Face superior

Fonte: Elaborada pelo autor.

3.2.5 Montagem da `Colmeia`

Após confirmar o correto processo de fabricação da placa medindo a continuidade das trilhas com o auxílio de um multímetro, foi realizada a montagem e soldagem dos componentes e conectores na PCB, como mostrado na Figura 25 para posteriormente serem realizados os testes de funcionamento e de consumo de corrente com auxílio do osciloscópio e ponteira de corrente.

Figura 25 – Placa de circuito impresso com componentes



(a) Face inferior (b) Face superior

Fonte: Elaborada pelo autor.

3.3 Desenvolvendo o *firmware*

Para que uma unidade de processamento realize uma tarefa é necessário que exista uma sequência de instruções indicando quais posições da área de memória serão utilizadas. *Firmware* são *software* embarcados que ficam armazenados em memórias conhecidas como "memórias de programa"e são responsáveis pela inicialização e configuração das funções que o *hardware* irá executar.

Neste projeto foi utilizado o microcontrolador `ATMEGA328P` que possui 2 *kB* de memória *Random Access Memory* (RAM), para armazenar as variáveis criadas e atualizadas dinamicamente ao longo da execução do programa, e 32 *kB* de memória *FLASH*, para armazenamento do código do *firmware*.

No decorrer do desenvolvimento do projeto, a biblioteca[1] do `RFM95PW` sofreu uma atualização, ocasionando o travamento da execução do programa devido ao aumento do uso de memória RAM do microcontrolador. Para dar continuidade no projeto sem perdas do que já havia sido produzido, foi adotada a estratégia de desmembrar o *firmware* em duas partes, a primeira realizando os testes de funcionamento dos sensores e utilizando a biblioteca `LMIC` antiga do `RFM95PW` e a segunda testando o funcionamento da transmissão dos dados para a TTN com o módulo LoRa e a nova biblioteca `LMIC`. Desta forma, foi possível validar todos os periféricos da `colmeia`, deixando a otimização do *firmware* para trabalhos futuros.

O *firmware* foi desenvolvido utilizando a IDE do Arduino, que faz uso da linguagem C++ e possui compilador para o microcontrolador `ATMEGA328P`. Também foram incluídas no projeto bibliotecas de terceiros para os módulos periféricos.

3.3.1 Bibliotecas dos sensores e módulo LoRa

Código 3.1 – Bibliotecas utilizadas no *firmware*

```
1  #include <lmic.h>
2  #include <hal/hal.h>
3  #include <DS3232RTC.h>
4  #include <HX711.h>
5  #include <Adafruit_INA219.h>
6  #include <BH1750.h>
7  #include <DHT.h>
8  #include <avr/sleep.h>
9  #include <Wire.h>
10 #include <SPI.h>
```

No Código 3.1 podemos observar as bibliotecas que foram inclusas no *firmware* desenvolvido. A biblioteca `<lmic.h>` utilizada no *firmware* de envio de dados pelo módulo *LoRa* é referente à <https://github.com/mcci-catena/arduino-lorawan>, sendo amplamente utilizada em dispositivos conectados a *The Things Network* (TTN) versão 3 que entrou em atividade em janeiro de 2021. A `<lmic.h>` inclusa no *firmware* de testes dos sensores é referente à <https://github.com/matthijskooijman/arduino-lmic> que teve seu projeto de desenvolvimento descontinuado, perdendo a interoperabilidade com a TTN quando os serviços da versão 2 foram desativados em setembro de 2021. A biblio-

[1] subprogramas auxiliares responsáveis por executar determinadas partes do código

teca `<hal/hal.c>` é a estrutura global que torna compatível o mapeamento de pinos do microcontrolador com a biblioteca *LMIC*.

As bibliotecas `<HX711.h>`, `<Adafruit_INA219.h>`, `<BH1750.h>`, `<DHT.h>` são utilizadas nas leituras dos sensores. A `<DS3232RTC.h>` efetua as configurações de período de envio das interrupções do *Real-Time Clock* (RTC) ao microcontrolador. As bibliotecas `<SPI.h>` e `<Wire.h>` são responsáveis pelas comunicações *Serial Peripheral Interface* (SPI) e *Inter-Integrated Circuit* (I2C) respectivamente.

3.3.2 Instanciação dos objetos

As bibliotecas utilizadas no *firmware* implementam o método de programação orientado a objeto, desta forma foram instanciados objetos para cada sensor, onde os valores coletados são guardados em forma de atributos. O Código 3.2 apresenta as instâncias dos objetos que foram criadas para cada sensor e também o objeto `sendjob` que é utilizado no escalonador da biblioteca `LMIC` do módulo LoRa.

Código 3.2 – Instancias dos objetos

```
1 HX711 sensor_peso;
2 Adafruit_INA219 ina219(0x40);
3 BH1750 lightMeter;
4 DHT dht1(DHT1_PIN, DHTTYPE);
5 DHT dht2(DHT2_PIN, DHTTYPE);
6 static osjob_t sendjob;
```

3.3.3 Configurando o RTC

A biblioteca `<DS3232RTC.h>` implementa funções necessárias para efetuar a configuração do RTC que é usado para gerar as interrupções que alteraram o modo de operação do microcontrolador de *sleep* para ativo. A função `setAlarm()` apresentada no Código 3.3 é usada para configurar o `Alarme 1` do RTC com o parâmetro `ALM1_MATCH_SECONDS` onde uma vez a cada minuto a saída `SQW` é alterada para nível lógico baixo, gerando um sinal de interrupção no microcontrolador. A função `alarmInterrupt()` habilita o alarme configurado, permitindo que as interrupções sejam sinalizadas e a função `alarm()` retorna a saída `SQW` para nível lógico alto.

Código 3.3 – Chamadas de funções do RTC

```
1 RTC.setAlarm(ALM1_MATCH_SECONDS, 1, 0, 0, 0);
2 RTC.alarm(ALARM_1);
3 RTC.alarmInterrupt(ALARM_1, true);
```

3.3.4 Efetuando a leitura dos sensores

O Código 3.4 apresenta a implementação das funções que são utilizadas para coletar os atributos de cada sensor. Por padrão os sensores encontram-se em modo de baixo consumo e em alguns casos como podemos observar nas linhas 13 e 19 é necessário que seu estado seja alterado para modo ativo antes de realizar as medições e ao término das leituras deve-se retorná-lo ao modo de baixo consumo como mostram as linhas 16 e 21.

Código 3.4 – Chamadas de funções dos sensores

```
1  //sensor de temperatura e umidade interna
2  float h_in = dht1.readHumidity();
3  float t_in = dht1.readTemperature();
4
5  //sensor de temperatura e umidade externa
6  float h_ex = dht2.readHumidity();
7  float t_ex = dht2.readTemperature();
8
9  //sensor de luminosisadade
10 float luz = lightMeter.readLightLevel();
11
12 //sensor de corrente e tensão
13 ina219.powerSave(false); //Alterando para modo ativo
14 float Im = ina219.getCurrent_mA();
15 float Vm = ina219.getBusVoltage_V();
16 ina219.powerSave(true); //Entrando em modo sleep
17
18 //sensor de peso
19 sensor_peso.power_up(); //Alterando para modo ativo
20 float peso_f =  sensor_peso.get_units(10) * (1);
21 sensor_peso.power_down(); //Entrando em modo sleep
```

3.3.5 Controle do modo de baixo consumo e interrupção do microcontrolador

A biblioteca `<avr/sleep.h>` implementa funções para controlar os modos de baixo consumo do microcontrolador. No Código 3.5 pode-se observar a função *set_sleep_mode()* onde é definido o modo de baixo consumo que será utilizado, *sleep_enable()* é uma permissão para que o modo *sleep* seja utilizado, *sleep_cpu()* ativa o modo de baixo consumo e a partir deste momento o microcontrolador só retorna ao modo ativo ao receber uma interrupção. A função *attachInterrupt()* prepara o microcontrolador para que uma interrupção possa ser efetuada, para que ela funcione de forma correta deve-se passar como parâmetros o pino (`digitalPinToInterrupt(SQW)`) do microcontrolador que recebera a interrupção externa, a função `ISR` (`acordar`) que será executada logo após o acionamento da interrupção e o estado (`FALLING`) que a porta de interrupção deve estar para que

a interrupção ocorra. Assim que a função *ISR* de interrupção é acionada ela executa *sleep_disable()* para que o microcontrolador volte para o modo ativo.

Código 3.5 – Chamadas de funções para os modos de baixo consumo

```
//criando a interrupção no microcontrolador
attachInterrupt(digitalPinToInterrupt(SQW), acordar, FALLING);

void dormir() {
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);// Define o modo sleep como power down
    sleep_enable(); // Habilita o modo sleep
    sleep_cpu();  // Ativa o modo sleep
}

void acordar() {
    sleep_disable(); // Desabilita o sleep
}
```

3.3.6 Utilizando a biblioteca `lmic`

A biblioteca `lmic` é responsável por encapsular e encaminhar os dados coletados dos sensores para a TTN de forma segura e confiável. Para realizar a comunicação com o servidor da TTN os dispositivos recebem um *DEVEUI* e um *APPKEY* que são únicos dentro da rede para efetuar a autenticação como podemos observar no Código 3.6.

Código 3.6 – Funções de autenticação de dispositivos LoRa na TTN

```
static const u1_t PROGMEM APPEUI[8]={ 0x00, 0x00, 0x00, 0x00,
                                      0x00, 0x00, 0x00, 0x00};
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

static const u1_t PROGMEM DEVEUI[8]={ 0x01, 0x00, 0x00, 0x00,
                                      0x00, 0x00, 0x00, 0x00};
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

static const u1_t PROGMEM APPKEY[16] = { 0x2B, 0x7E, 0x15, 0x16,
                                         0x28, 0xAE, 0xD2, 0xA6,
                                         0xAB, 0xF7, 0x15, 0x88,
                                         0x09, 0xCF, 0x4F, 0x3C};
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
```

O Código 3.7 apresenta o mapeamento de pino responsável pela comunicação entre o microcontrolador e o módulo *LoRa*. O módulo ainda utiliza o barramento de comunicação SPI, além dos pinos que podem ser mapeados considerando a disponibilidade das portas do microcontrolador discriminados no *firmware*.

Código 3.7 – Mapeamento de pinos lmic

```
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {3, 4, LMIC_UNUSED_PIN},};
```

A biblioteca `lmic` conta com uma *Application Programming Interface* (API) que disponibiliza um conjunto de funções e estruturas globais `LMIC` que são utilizadas para alterar parâmetros de configurações específicas. O Código 3.8 mostra as alterações das estruturas `LMIC` que foram efetuadas para enquadrar o projeto dentro das especificações que a rede da TTN exige para uso do LoRa no Brasil.

Código 3.8 – Funções da biblioteca lmic

```
LMIC_reset();
LMIC_setAdrMode(false);
LMIC_setLinkCheckMode(0);
LMIC_setDrTxpow(DR_SF9, 2);
LMIC.dn2Dr = DR_SF9;
LMIC_selectSubBand(1);

os_init();
do_send(&sendjob);
os_runloop();
```

A biblioteca `lmic` é baseada em eventos e conta com um escalonador interno executado pela função *os_runloop()* que consome os objetos enfileirados pela função *do_send()*, porém é necessário que toda a aplicação seja iniciada pela função *os_init()*.

O Apêndice B mostra os códigos propostos para os testes de cada conjunto de periféricos, assim como todas as bibliotecas importadas no projeto.

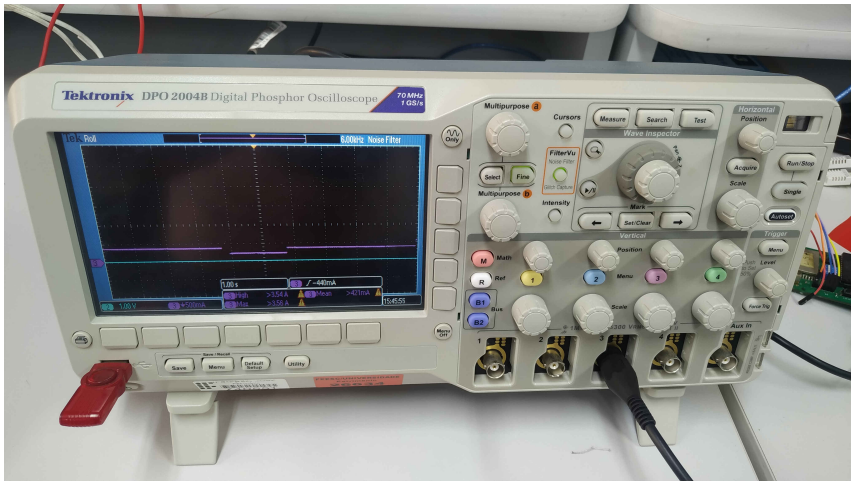
3.4 Instrumentação

A instrumentação é parte fundamental para conseguir realizar as medições e registros do comportamento dos equipamentos que se quer analisar. Equipamentos adequados geram alta confiabilidade e aumentam a credibilidade nos resultados obtidos. Para a análise deste projeto, foram utilizados os instrumentos como osciloscópio, ponteiras, dongle SDR e o *software* GQRX.

3.4.1 Osciloscópio

O osciloscópio `DPO2004B` é um instrumento da *Tektronix* que auxilia nas aquisições de sinais de corrente e tensão, contém uma gama de funcionalidades para diversas finalidades de depuração e possui recursos como:

Figura 26 – Osciloscópio DPO2004B



Fonte – (TEKTRONIX, 2020)

- *Wave Inspector®* para facilitar o controle de navegação dos sinais adquiridos;
- *FilterVu™* um filtro passa-baixa variável para remoção de sinais ruidosos enquanto captura transições rápidas;
- *TekVPI®* uma interface que suporta sondas ativas, de corrente e diferenciais;
- 1GS/s de taxa de amostragem;
- 70MHz de largura de banda;
- 4 canais analógicos e 16 digitais;
- Interface *USB* para captura de imagens de sinais.

3.4.2 Ponteira A622

A ponteira de prova `A622` da `Tektronix` utiliza sensor de corrente de efeito *Hall* para entregar uma saída de tensão ao osciloscópio e possui características como:

- Correntes AC ou DC com frequência de até *100 KHz*;
- *50 mA* a *100 A* de pico;
- Saída amplificada de *10 mV* ou *100 mV*.

Figura 27 – Ponteira de corrente A622

Fonte – (TEKTRONIX, 2017)

3.4.3 Uso do `GQRX` e *dongle SDR*

O dongle *R820T SDR & DVB-T* da `NooElec` é um dispositivo modificado para uso de *Software Defined Radio* (SDR), que por meio de uma interface *USB* entrega ao software a responsabilidade de implementação das configuração como filtros, demoduladores, amplificadores, AGC, misturadores, etc. Estes dispositivos possuem recursos como:

Figura 28 – Dongle NooElec SDR

Fonte – (NOOELEC, 2013)

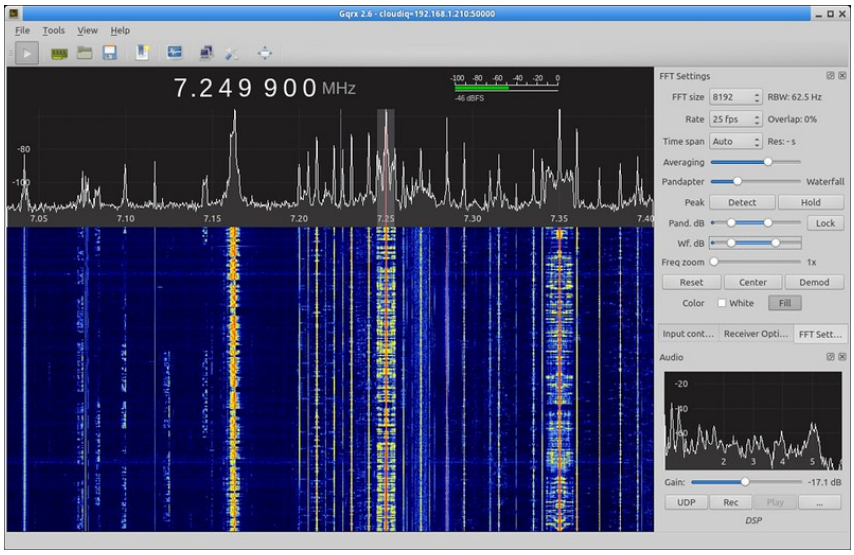
- Sintonizador *R820T*;
- Interface *USB* `Realtek` *RTL2832U*;

- Range de frequência de 24MHz a 1766MHz;
- Largura de banda *3,2MHz*;
- *8 bits* de resolução;
- Taxa de símbolo de *2,4MS/s*.

O `GQRX` é um receptor de rádio frequência definido por *software* de código aberto disponível para sistemas operacionais *Linux* e *MacOS*. Tem suporte a muitos *hardwares SDR* como pode-se observar na lista disponível em <https://gqrx.dk/supported-hardware>. Como pode-se observar na Figura 29 o `GQRX` conta com recursos como:

- Ajuste de frequência e ganho;
- Demoduladores (mono e estéreo);
- Filtro de banda passante ajustável;
- Ajuste de taxa de amostragem;
- Gráficos de *FFT* e *Waterfall*;
- Analisador de espectro.

Figura 29 – *Software* GQRX



Fonte – (CSETE, 2013)

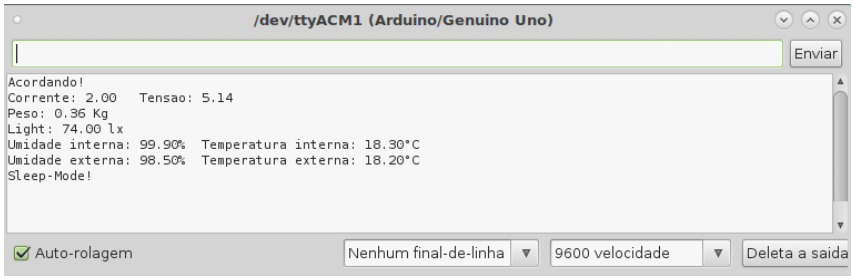
3.5 Teste do funcionamento dos sensores e transmissão LoRa

Inicialmente será descrito como foram feitos os testes que visam mostrar que o *hardware* desenvolvido está funcional e interagindo de forma correta com seus periféricos.

3.5.1 Leitura dos sensores e transmissão LoRa utilizando o terminal

O *software* `terminal` foi o primeiro método de teste utilizado para executar os *firmwares* implementados de forma isolada dos demais componentes do sistemas. Ele utiliza uma *interface* serial com possibilidade de ajuste da taxa de transmissão para receber os dados e apresentá-los na tela do computador. Na Figura 30 pode-se observar os valores obtidos em um ciclo de leitura dos sensores.

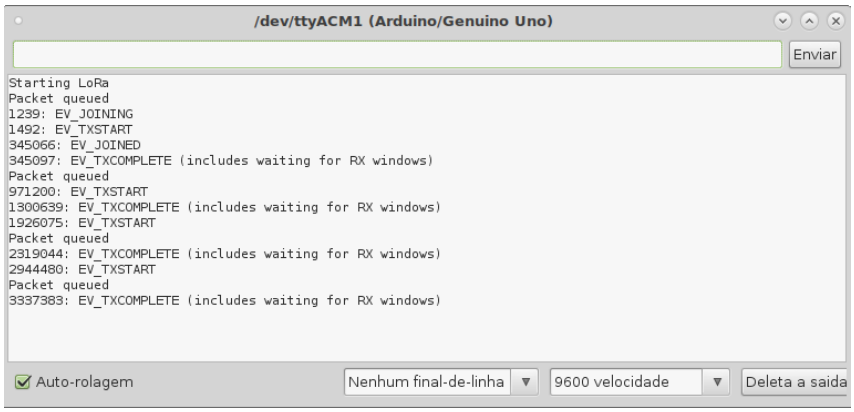
Figura 30 – Leitura dos sensores pelo console



Fonte: Elaborada pelo autor.

A Figura 31 apresenta com o auxílio do `terminal` o procedimento de transmissão dos pacotes LoRa para a TTN, onde em um primeiro momento o dispositivo troca informações de parâmetros para ingresso na rede utilizando a mensagem *EV_JOINING*. Após o primeiro pacote enviado e a autenticação na rede efetuada, o dispositivo começa a encaminhar seus pacotes de dados para o servidor.

Figura 31 – Leitura da transmissão LoRa pelo console

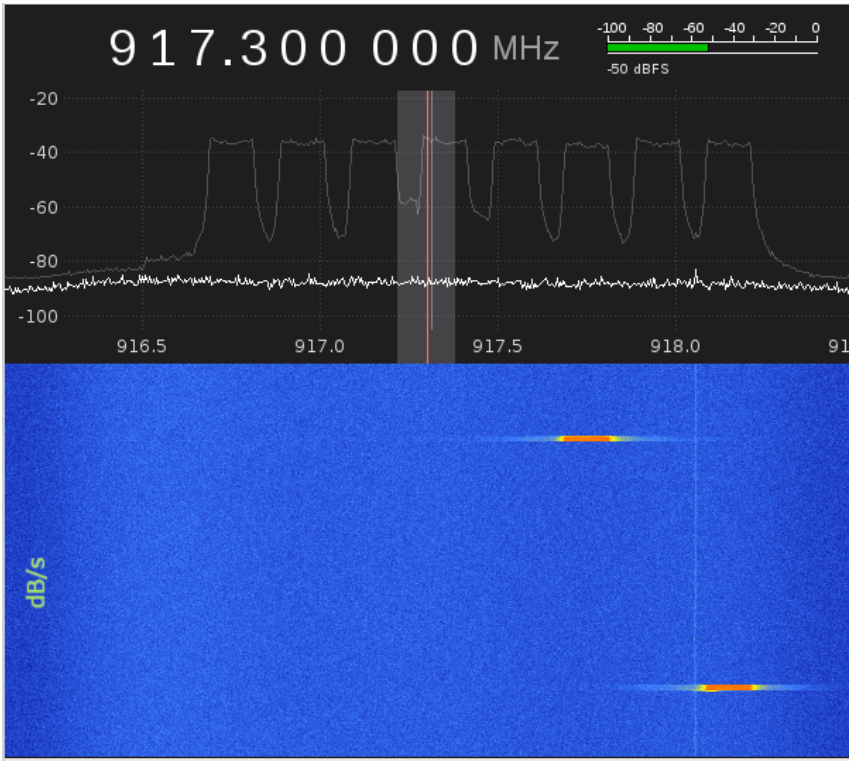


Fonte: Elaborada pelo autor.

3.5.2 Leitura da transmissão LoRa com `GQRX`

Para confirmar o envio dos dados coletados foi utilizado o *dongle SDR* para verificar o sinal de radiofrequência (também chamado de `RF`) emitido da `colmeia` para o *gateway*. A Figura 32 mostra o sinal capturado pelo *dongle SDR* utilizando o *software GQRX*. Pode-se observar uma elevada intensidade de sinal de aproximadamente *-40dBm* recebida pelo `dongle` (devida a pequena distância entre a `colmeia` e o *dongle*) e também o conjunto de frequências que são utilizadas pelo módulo LoRa. O *software* também possui um gráfico chamado *waterfall* (cascata) onde pode ser observado o sinal enviado variando na frequência devido ao protocolo LoRa que utiliza a modulação CSS, assim como a relação entre a cor que indica a intensidade do sinal (vermelho indica uma intensidade maior e azul menor) e o tempo de duração do sinal no ar dado em segundos. O gráfico ilustra também a técnica de salto de frequência (*frequency hopping*), no qual percebe-se que a cada transmissão realizada ocorre uma mudança na frequência do *chirp*.

Figura 32 – Captação do sinal de `RF` utilizando o *software* `GQRX`



Fonte: Elaborada pelo autor.

3.5.3 Recepção dos dados pela TTN

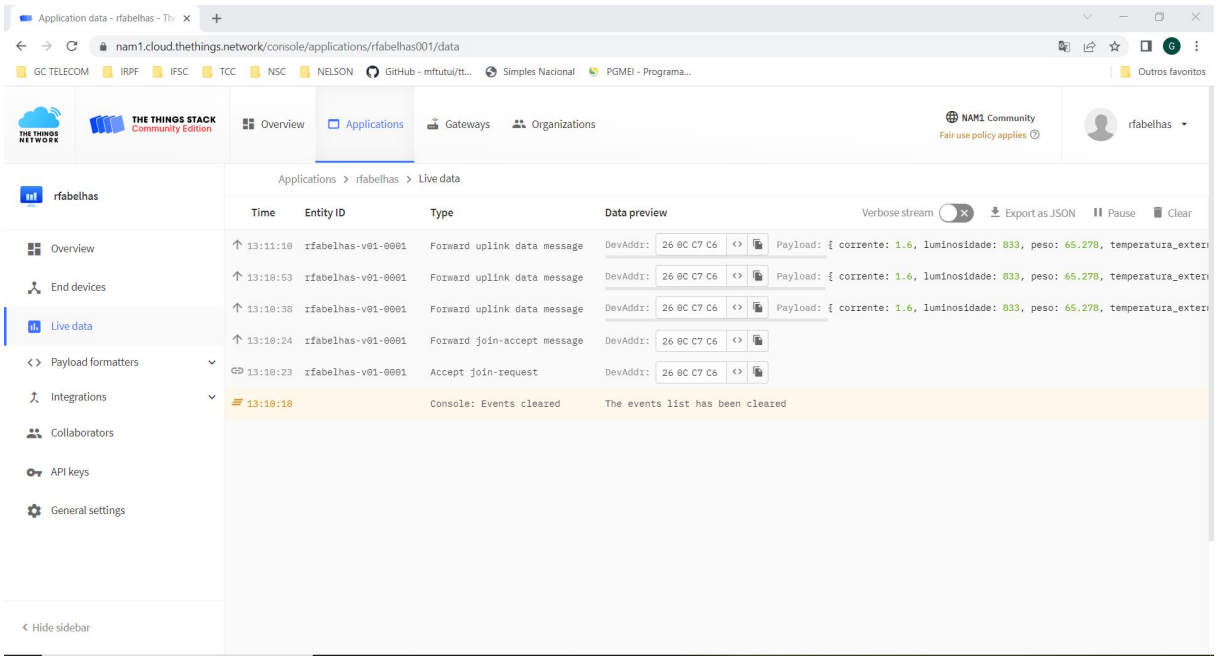
Para verificar a chegada dos dados na TTN, que é uma rede formada por um conjunto de *gateways* disponibilizados e compartilhados por seus próprios usuários, é utilizado uma interface WEB disponibilizada pela própria TTN que fornece um conjunto de recursos para o usuário cadastrar novos *gateways* e dispositivos ou utilizar os *gateways*

compartilhados por um outro usuário e uma gama de configurações que devem ser efetuadas para o correto funcionamento dos dispositivos IoT.

Os dados de sensores coletados pela rede TTN podem ser enviados para aplicações proprietárias e que estão alocadas em servidores fora da sua rede. Nestas aplicações, ocorre o processamento dos dados adquiridos e disponibilização dos resultados para os usuários (ver Figura 1).

Para a validação do módulo LoRa usado no projeto foi transmitido um conjunto de dados e verificado a sua recepção na TTN, ao mesmo tempo que foi utilizado o GQRX para confirmar a transmissão do sinal de RF. A Figura 33 mostra a recepção dos dados na tela de *Applications* da interface *web* da TTN. Conforme indicado na figura, foram transmitidas e recebidas 3 mensagens com os dados de corrente, luminosidade, peso e temperatura, validando o módulo LoRa.

Figura 33 – Dados sendo coletados pela `TTN`



Fonte: Elaborada pelo autor.

3.6 Medição do consumo de corrente e durações de tempo

Conforme definido na introdução deste trabalho, um dos principais objetivos foi desenvolver uma nova PCB que mantivesse todas as funcionalidades do protótipo original mas reduzisse significativamente o consumo de corrente (energia). Nesta seção serão medidos e analisados os consumos dos diversos periféricos utilizados para sensoriamento e transmissão dos dados, monitorados pelo sistema.

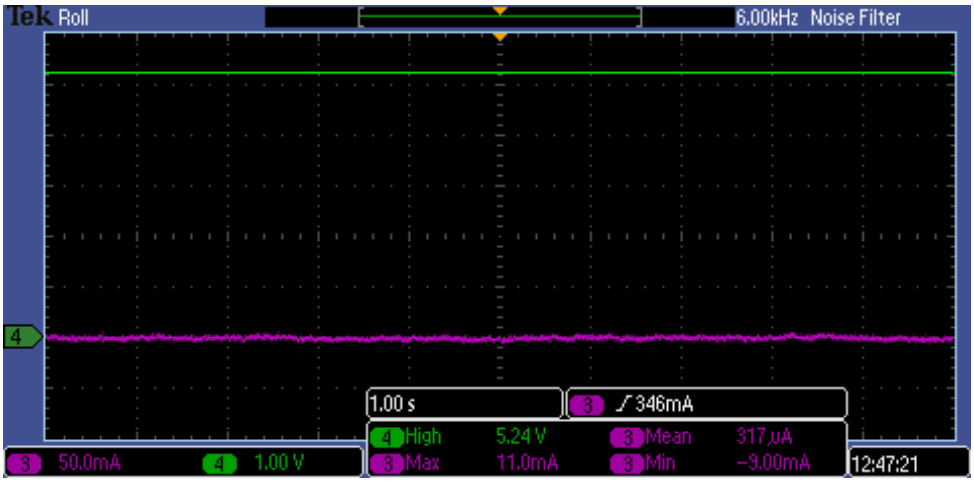
Para a medição do consumo de corrente dos diversos sensores, foi adotado um

procedimento de monitorar a corrente na entrada de alimentação do módulo utilizando um osciloscópio com ponteira de corrente. Em virtude de serem correntes muito baixas, foram necessários levar em conta que os valores medidos apesar de representarem de forma proporcional as correntes, não representam valores exatos, devido aos ruídos de aquisição e também a escala da ponteira usada.

3.6.1 Calibração da ponteira de corrente

Para iniciar a coleta de dados com o auxilio do osciloscópio é preciso efetuar a calibração da ponteira A622 que possui uma bateria de 9V que alimenta seu circuito interno e a cada medida realizada a calibração do zero deve ser verificada. Para que a calibração seja feita de forma correta, a ponteira deve estar ligada e sem corrente fluindo pelos condutores que possam interferir neste procedimento. A Figura 34 mostra uma calibração realizada na ponteira antes da coleta de uma medida de corrente, com valor médio de ruído de 317uA.

Figura 34 – Calibração do zero da ponteira A622



Fonte: Elaborada pelo autor.

A ponteira A622 é especificada para medições de corrente entre 50mA à 100A e como boa parte da nossa região de operação está em microamperes foi utilizado o filtro de ruído ativo do osciloscópio configurado em 6KHz para que se pudesse observar de maneira mais clara a forma de onda do sinal . Além do filtro, foi utilizada a amplificação da corrente da ponteira do osciloscópio, com a passagem de múltiplas voltas do condutor de alimentação da placa da `colmeia` pela mandíbula da ponteira. Com isso conseguimos fazer com que a corrente medida fique mais distante da zona de ruído. Um fator de amplificação de 10 foi obtido com 5 espiras dos condutores positivos e negativos da alimentação. Assim, os sinais de medidas de corrente no osciloscópio durante os testes na `colmeia` sempre terão um fator de amplificação de 10x maior que a corrente real do circuito.

3.6.2 Modos de operação

Um ciclo de operação completo da `colmeia` é composto pelos momentos em modo ativo e *sleep*, onde o modo ativo se divide em leitura dos sensores e transmissão dos dados e o modo *sleep* é o tempo em que a `colmeia` ficará operando em baixo consumo. Na prática, a duração do modo ativo é fixo e bem determinado, pois o ciclo de leitura dos sensores e a transmissão dos dados pelo módulo LoRa não se altera. A duração do modo *sleep* é definido levando em consideração a carga da bateria, sendo relevante apenas o valor do consumo de corrente e não o tempo de duração para a realização dos testes. Os momentos citados possuem consumo de corrente específicos e determinar estes valores corretamente é essencial para saber quantos ciclos de operação da `colmeia` são efetuados com determinada capacidade de carga de bateria.

A Figura 35 mostra um ciclo da `colmeia` saindo de modo *sleep* para ativo e retornando para *sleep* e as situações que ocasionam essa troca de contexto, (1) inicialmente o módulo está em modo *sleep*, no qual ocorre o menor consumo de potência, (2) ao receber uma interrupção do RTC, (3) passa para o modo ativo do microcontrolador, ao entrar neste modo é o momento em que posteriormente será realizado as leituras dos sensores e o envio dos dados pelo módulo LoRa. Para finalizar o modo ativo (4) o microcontrolador envia uma *flag* ao RTC para limpar a interrupção que foi gerada e (5) entra em modo *sleep* novamente.

Figura 35 – Momentos das transições entre modo *sleep* e ativo

Fonte: Elaborada pelo autor.
Nota: Corrente medida com módulo de amplificação 10 vezes. Zoom de 100x da tela.

A Tabela 1 mostra os valores de consumo de corrente máxima dos componentes em seus respectivos datasheets, tanto em modo ativo como em modo *sleep*. Assim, com estas informações levantadas pôde ser feito uma comparação com os valores obtidos nas medições utilizando o osciloscópio.

Tabela 1 – Consumo de corrente informado pelo datasheet

Periférico	Ativo (mA)	Sleep (uA)
Microcontrolador	11	7.5
Relógio de tempo real	0.3	300
Sensor de iluminância	0.2	10
Sensor de temperatura	0.3	30
Sensor de corrente	0.7	15
Sensor de peso	1.5	0.5
Módulo LoRa Tx	700	12
Módulo LoRa Rx	15	12
Total	729 mA	375 uA

Fonte: Elaborada pelo autor.

3.6.3 Modo *sleep*

O modo *sleep* tem por característica possuir o menor consumo de corrente médio da `colmeia`, isto acontece pelo fato de que todos os componentes terem um modo de baixo consumo, o qual deve ser ativado para se ter uma leitura correta.

A Figura 36 mostra o valor médio do consumo de corrente da `colmeia` no modo *sleep*. O valor medido de corrente é de *5,42mA*, mas considerando o fator de amplificação utilizado (10x), o valor real medido é de *542uA*, o qual é aproximadamente a soma do consumo de corrente de todos os componentes em modo *sleep* (*375uA*) mostrado na Tabela 1. Nesta análise temos o filtro *FilterVu* do osciloscópio ativo, e o valor medido está abaixo do mínimo (50mA) especificado para a ponteira de corrente utilizada, por isso, não será possível ter uma melhor precisão neste valor. Não custa lembrar que a medição com o osciloscópio visa mostrar as variações de corrente de acordo com os componentes que forem ativados, e este valor exato será insignificante nestes casos.

Para diminuir o consumo de corrente da `colmeia` a Tabela 1 mostra que é necessário manter o microcontrolador e os demais periféricos em modo de baixo consumo, para isso foi proposto utilizar um circuito com o *Real-Time Clock* (RTC) e gerar uma interrupção externa no microcontrolador que o 'acorde' do modo *sleep*. Apesar do RTC consumir *300uA*, isto representa apenas *2,6%* do consumo do microcontrolador em modo ativo.

3.6.4 Modo ativo

No modo ativo da `colmeia` há dois momentos a serem analisados, a coleta de dados dos sensores e a transmissão dos dados pelo módulo LoRa. Os valores mostrados na Tabela 2 de consumo corrente dos sensores de iluminância, tensão e corrente da bateria e peso e também do módulo LoRa se referem ao adicional em relação ao valor do microcontrolador. Essa medição foi realizada utilizando a diferença de corrente entre os cursores horizontais do

Figura 36 – Média de consumo de potência em modo sleep

Fonte: Elaborada pelo autor.
Nota: A corrente foi medida com módulo de amplificação 10 vezes.

osciloscópio. Assim, a Tabela 2 apresenta o consumo medido e o tempo de funcionamento em modo ativo dos sensores que podem ser observados pelo consumo de corrente. É possível verificar que aproximadamente *52%* do consumo de energia da `colmeia` é relacionada a transmissão dos dados pelo módulo LoRa, e aproximadamente *41%* do consumo de energia é gasta pelo microcontrolador no momento em que está no modo ativo, sendo o consumo dos sensores aproximadamente *7%* do modo ativo. Na Figura 37[2] são apresentados os sinais medidos com o osciloscópio, nos momentos de leitura dos sensores e da transmissão pelo módulo LoRa, e foram utilizados para a elaboração da Tabela 2.

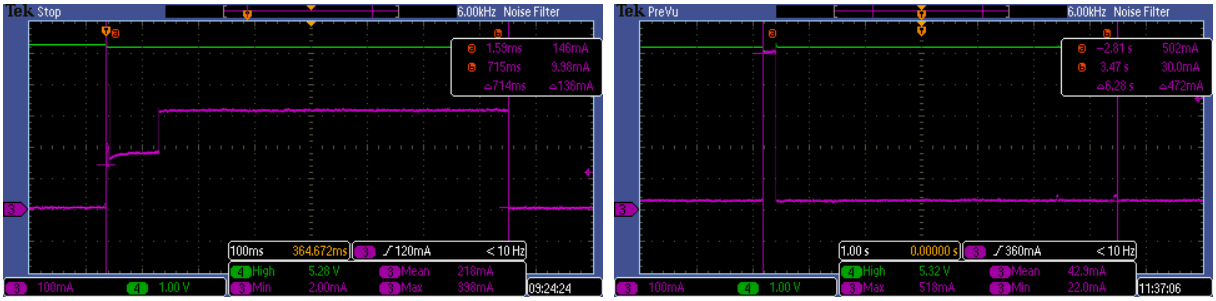
Tabela 2 – Consumo de corrente medido da `colmeia` e tempo de execução

Periférico	Ativo (mA)	Tempo (ms)	Carga (mA.s)	% Energia
Microcontrolador	12,9	6694	86352,6	41,38%
Sensor de iluminância	7,4	0,56	4,144	0,00%
Sensor de corrente	8,8	1,76	15,488	0,01%
Sensor de peso M1	4,8	88,4	424,32	0,20%
Sensor de peso M2	18	620	11160	5,35%
Sensor de temp/umidade	0	0	0	0,00%
Módulo LoRa Tx	480	228	109440	52,45%
Módulo LoRa Rx1	16	24,8	396,8	0,19%
Módulo LoRa Rx2	16	54,4	870,4	0,42%
Média	**31,17**	**6694**	**208663,8**	**100,00%**

Fonte: Elaborada pelo autor.

[2] E demais medidas realizadas e apresentadas no Apêndice C

Figura 37 – Tempo de execução de leitura dos sensores e transmissão do módulo LoRa



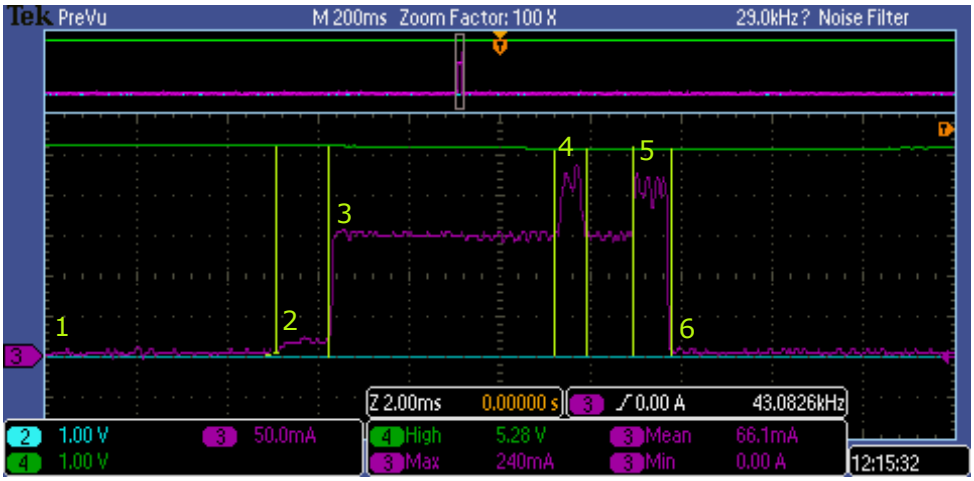
(a) Leitura dos sensores (b) Transmissão módulo LoRa

Fonte: Elaborada pelo autor.
Nota: A corrente foi medida com módulo de amplificação 10 vezes.

Leitura dos sensores

Para haver uma coleta de dados o microcontrolador executa uma sequência de interações com o sensor, podemos observar na Figura 38 as interações que ocorrem na leitura do sensor de iluminância. Nesta sequência nota-se que a `colmeia` (1) encontra-se no modo *sleep* e (2) ao receber uma interrupção gerada pelo RTC (3) passa para o modo ativo, em seguida podemos ver o microcontrolador (4) realizando uma aquisição de dados por meio do barramento de comunicação I2C, (5) a *flag* para limpar a interrupção do RTC é enviada pelo microcontrolador que (6) retorna ao modo *sleep*.

Figura 38 – Momento da leitura do Sensor de iluminância



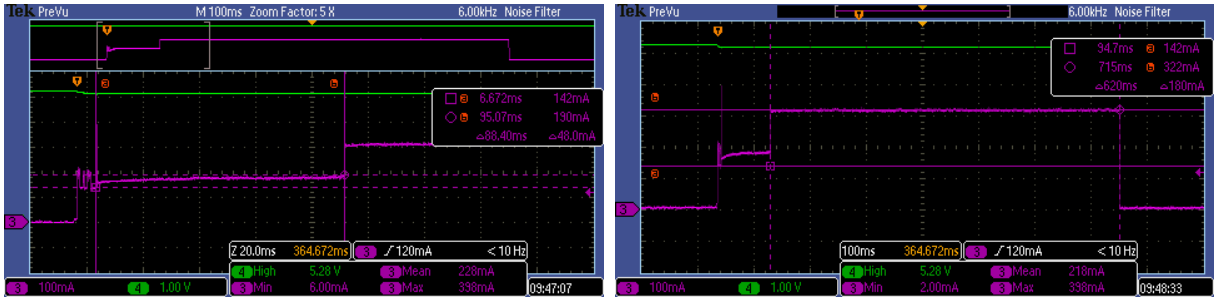
Fonte: Elaborada pelo autor.
Nota: A corrente foi medida com módulo de amplificação 10 vezes. Para visualizar o instante mostrado foi utilizado um zoom de 100x da tela do osciloscópio.

Os sensores possuem interações diferentes com o microcontrolador, e essas interações resultam em consumo de corrente diferentes, como mencionado no exemplo da Figura 38 o sensor de iluminância utiliza a interface *I2C* para realizar o envio de dados requisitados pelo microcontrolador e os sinais de consumo que conseguimos observar no osciloscópio

são basicamente provenientes desta comunicação, devido ao baixo consumo de corrente que o transdutor do sensor possui ao realizar uma aquisição física de fato.

O sensor de peso possui um consumo expressivo de corrente comparado aos demais por utilizar quatro células de cargas resistivas ligadas entre si em formato de ponte de *Whearstone*, sendo criados dois divisores de tensão em paralelo. A Figura 39 apresenta dois momentos da leitura do sensor de peso, o primeiro é a ativação do sensor onde ele sai do modo de baixo consumo, pode-se observar que neste momento o sensor apresenta um consumo de aproximadamente *4,8mA*. O segundo momento é quando as células de carga são ativadas para que de fato se realize as medidas de peso, o consumo de corrente neste momento passa para *18mA*. Os demais sensores da `colmeia`, por não possuírem um consumo de corrente que seja possível medir, com a ponteira de corrente disponível, foram agregados ao valor de consumo de corrente do microcontrolador em modo ativo. Os sinais da leitura do restante dos sensores estão no Apêndice D.

Figura 39 – Leitura do sensor de peso



(a) sensor saindo do modo baixo consumo (b) células de carga são ligadas

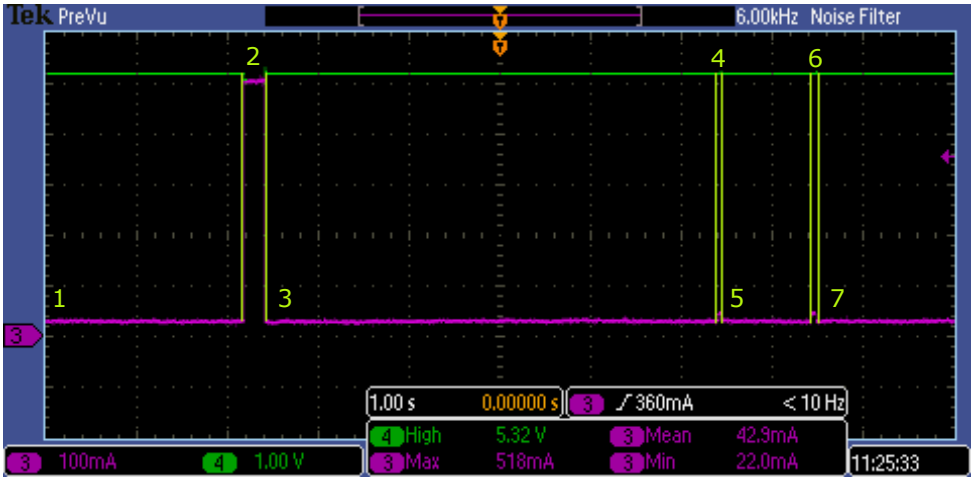
Fonte: Elaborada pelo autor.
Nota: A corrente foi medida com módulo de amplificação 10 vezes.

Transmissão pelo módulo LoRa

Outro momento importante de análise do consumo de corrente da `colmeia` está relacionado com a transmissão dos dados pelo módulo LoRa. O módulo utiliza a implementação LoRa classe A que possui uma janela de transmissão seguida de duas janelas de recepção com intervalos determinados.

Por padrão a transmissão é iniciada (1) em modo ativo do microcontrolador e é responsável por (2) um pico no consumo de corrente, onde acontece a transmissão dos dados pelo módulo, (3) um atraso de cinco segundos antecede a (4) primeira janela de recepção, (5) seguido de um novo atraso de um segundo (6) para a segunda janela de recepção, (7) liberando o microcontrolador para novas interações caso necessário. A Figura 40 mostra que o momento de transmissão é a ocorrência do maior pico de consumo de corrente da `colmeia`.

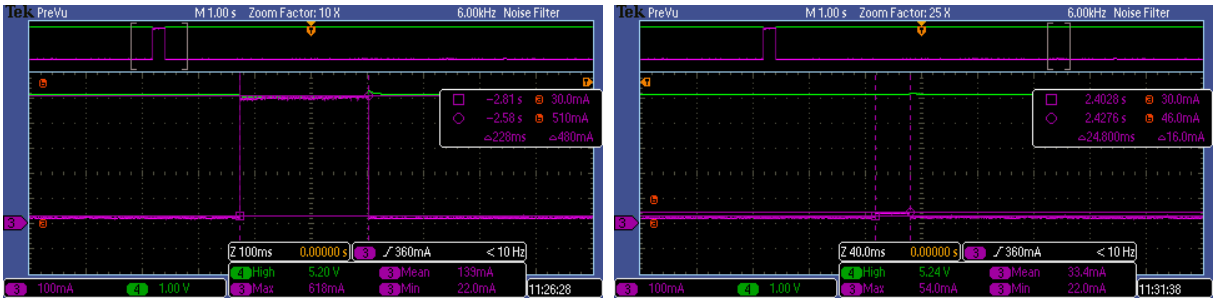
Figura 40 – Momento da transmissão dos dados pelo módulo LoRa



Fonte: Elaborada pelo autor.
Nota: A corrente foi medida sem módulo de amplificação 10 vezes.

A Tabela 2 apresenta o consumo medido de *480mA* no momento da janela de transmissão do módulo LoRa e *16mA* nas janelas de recepção, estes consumos foram obtidos lendo os sinais mostrados na Figura 41. Os gráficos contendo os sinais das medidas realizadas com o osciloscópio no momento de transmissão do módulo LoRa estão apresentados no Apêndice D.

Figura 41 – Transmissão do módulo LoRa



(a) consumo de corrente na janela de transmissão (b) consumo de corrente na janela de recepção

Fonte: Elaborada pelo autor.
Nota: A corrente foi medida sem módulo de amplificação 10 vezes.

3.6.5 Autonomia do sistema

Um ciclo de operação completo do sistema consiste na soma dos momentos em modo ativo e modo *sleep* da `colmeia`. A Tabela 3 mostra que o consumo máximo por ciclo é de `31,172mAh` e acontece quando a `colmeia` está a todo momento no modo ativo. O consumo mínimo por ciclo ocorre quando apenas uma de operação de leitura e transmissão é feita por dia, que corresponde a `0,377mAh`.

A autonomia da `colmeia` esta relacionada diretamente ao seu consumo de corrente por ciclo de operação e a capacidade de carga da bateria que será utilizada. Considerando uma bateria de `1000mAh`, no consumo da `colmeia` medido, e também no número de ciclos de operação diário variando de mais de 1440 até 1, a estimativa de autonomia da `colmeia` é de `11` dias, ou seja, `262` horas em modo ativo e `110,41` dias respectivamente, conforme mostra a Tabela 3. Esse resultado mostra claramente que o *hardware* desenvolvido atingiu o seu principal objetivo que era uma maior autonomia que o protótipo inicial construído com um Arduíno, o qual tinha uma autonomia de 105 horas utilizando uma bateria de 2,6Ah (JESUS, 2017).

Tabela 3 – Consumo de corrente da `Colmeia` e tempo de autonomia de bateria. Considerando tempo ativo de 6,7 seg, corrente no modo sleep de 375uA, e bateria com capacidade de 1000mAh, e sem perdas na carga da bateria.

Ciclo de operação (min)	**Ciclos por dia**	**Consumo tempo ativo (mAh)**	**Consumo no modo *sleep* (mAh)**	**Consumo ciclo (mAh)**	**Autonomia em horas @ 1Ah**	**Autonomia em dias @ 1Ah**
0,112	12907	31,172	0,000	31,172	32	1,34
1	1440	3,478	0,333	3,811	262	10,93
4	360	0,869	0,365	1,234	810	33,77
15	96	0,232	0,372	0,604	1.655	68,98
60	24	0,058	0,374	0,432	2.313	96,39
120	12	0,029	0,375	0,404	2.478	103,23
720	2	0,005	0,375	0,380	2.633	109,71
1440	1	0,002	0,375	0,377	2.650	110,41

Fonte: Elaborada pelo autor.

Nota: Para o valor do consumo em modo *sleep* foi utilizado o valor do datasheet, pois o medido é abaixo da especificação da corrente mínima da ponteira.

3.6.6 Comparação com o sistema prototipado em Arduíno

O desenvolvimento da placa da `colmeia` foi proposto para reduzir o consumo obtido e apresentado no projeto 'Controle de Potência em um sistema de monitoramento de abelhas *off-grid*' por Moecke (2019), pag 84, Tabela 6. A Tabela 4 apresenta os cenários `CC3 @5V - sleep` e `CC3 @5V - ativo` que serão utilizados na comparação por serem equivalentes em ambos os protótipos. A primeira linha traz o consumo de corrente do protótipo em Arduíno quando no modo *sleep*,`7,08mA`, o qual comparado com o novo consumo de `542uA`, do *hardware* proposto, traz uma redução de 13 vezes no consumo da `colmeia`. Em relação ao consumo de corrente no modo ativo, a redução é menor, cerca de 1,15 vezes, passando de `14,87mA` para `12,9mA`. Apesar de parecer uma redução pequena quando considerado de forma isolada, esse resultado leva a um aumento da autonomia da colmeia para o mesmo sistema de alimentação, ou uma redução significativa do custo, tamanho e peso final da colmeia.

Tabela 4 – Comparação entre o protótipo em Arduíno e o protótipo colmeia

	protótipo arduino (mAh)	**protótipo colmeia (mAh)**
CC3 @5V - sleep	7,08	0,542
CC3 @5V - ativo	14,87	12,9

Fonte – (MOECKE, 2019)

Foi alcançado uma redução expressiva também no peso do conjunto do sistema que possuía uma bateria 7Ah @12V (2,3Kg), controlador de carga (120gr), caixa ambiental (760gr) e o painel solar 12V (1,2Kg) e pesava aproximadamente `4,5Kg`. Com o novo protótipo poderia ser usado um sistema de alimentação de menor volume e pesando menos de `600g` com uma bateria de 2,6Ah @3,6V (150gr), controlador de carga de (80gr), caixa ambiental de (200gr) e painel solar 5V de (160gr) que é aproximadamente *13%* do peso do sistema anterior.

4 CONCLUSÕES

Este trabalho teve como proposta desenvolver um hardware específico do módulo `colmeia` do projeto "RF-Abelhas", para substituir o protótipo utilizado em Arduino proposto no trabalho "Controle de Potência em um sistema de monitoramento de abelhas *off-grid*", reduzindo o consumo de corrente nos modos de operação ativo e *sleep* para viabilizar a utilização de baterias e painel solar de pequenas dimensões e pesos.

Para alcançar o objetivo foi desenvolvido um *hardware* com o auxílio do *software Eagle*, uma ferramenta Projeto Assistido por Computador, do inglês (*Computer-Aided Design*) (CAD) onde se projetou o diagrama esquemático e a Placa de Circuito Impresso, do inglês *Printed Circuit Board* (PCB) que foi produzida em uma manufatura fora do país (CHINA), com um custo de aproximadamente *30%* do valor encontrado para produção no Brasil. Foi implementado um *firmware* que pudesse ser utilizado para testar as funcionalidades de leitura dos sensores e transmissão dos dados utilizando o módulo LoRa para a rede *The Things Network* (TTN), que é muito utilizada para implantações de produtos baseados no conceito de *Internet of Things* (IoT).

Utilizando um osciloscópio, foi possível realizar as medições de consumo de corrente do *hardware* proposto neste projeto, além de que esses dados puderam ser utilizados para fim de comparação do consumo de corrente com o protótipo desenvolvido com o Arduino. Com esta comparação, pôde-se observar que foi alcançado uma redução 13 vezes no consumo de corrente em modo *sleep* e 1,15 vezes no consumo em modo ativo. Esta redução impacta diretamente no aumento da autonomia da bateria, podendo chegar a 110 dias sem recarga, caso se faça uma transmissão diária.

Além do aumento da autonomia da bateria, a redução do consumo de corrente traz a possibilidade de diminuir o tamanho dos componentes como baterias, painel solar, carregador e caixa ambiental, passando de aproximadamente `4,5Kg` para `590gr` o que reduz o custo destes produtos e traz um ganho na mobilidade e no manejo das colmeias pelos apicultores.

Deve-se considerar que apesar de as medidas de corrente terem sido feitas com boa precisão de tempo, para valores inferiores a *5mA* elas estão abaixo da especificação da ponteira de corrente utilizada, e por isso valores baixos de corrente medidos podem carregar erros superiores a *20%*.

4.1 Trabalhos futuros

Durante o desenvolvimento do projeto da Placa de Circuito Impresso, do inglês *Printed Circuit Board* (PCB), foram encontradas adversidades que podem ser alteradas e são sugeridas para trabalhos futuros:

- Alteração do microcontrolador com maior memória *RAM*, que foi um obstáculo encontrado após a atualização da biblioteca `LMIC`. Como sugestão, o `ATmega4808` possui *48kB* de memória de programa e *6kB* de memória RAM. Tem suporte às bibliotecas do projeto com poucas alterações no *firmware*;
- A criação de um *firmware* específico para o *hardware* tornando o código mais otimizado, Estes atrasos são as funções delay() utilizadas no código para cada sensor de acordo com o fabricante. No protótipo a leitura é feita de forma sequencial, a proposta é ligar os diversos sensores em paralelo e após o tempo previsto de atraso efetuar as leituras, otimizando o tempo total. sem atrasos de espera entre outras melhorias;
- Estudar a possibilidade de colocar o microcontrolador em modo *sleep* durante a janela de espera de *5s* e *1s* do protocolo LoRa.
- Melhoria nos conectores de acoplamento dos sensores;
- Realizar testes de campo para verificar o comportamento do *hardware* em condições reais.
- Melhorar o consumo de corrente do sensor de peso que consome *5,35%* da energia da `colmeia`;
- Estudar a possibilidade de desligar a alimentação dos sensores através de um micro chaves, após a realização das medições;
- Desenvolver o sistema de alimentação solar para assegurar uma boa autonomia e tamanho reduzido;
- Acomodar o hardware em uma caixa ambiental;

REFERÊNCIAS

ADAFRUIT. *The List | I2C addresses! | Adafruit Learning System*. 2022. Disponível em: <https://learn.adafruit.com/i2c-addresses/the-list>. Citado na página 28.

ADLER, M. *Different types of vias: (1) Through hole. (2) Blind via. (3) Buried via. The gray and green layers are nonconducting, while the thin orange layers and vias are conductive.* 2008. Disponível em: <https://en.wikipedia.org/wiki/Via_(electronics)#/media/File:Via_Types.svg>. Citado na página 30.

ALLIANCE, L. *What is LoRaWAN® Specification - LoRa Alliance®*. 2022. Disponível em: <https://lora-alliance.org/about-lorawan/>. Citado 2 vezes nas páginas 19 e 20.

ANDRADE, A. S. de Oliveira e Fernando Souza de. *Sistemas Embarcados. Hardware e Firmware na Prática*. [S.l.]: Érica, 2010. ISBN 8536501057. Citado na página 21.

ARNIA. *Arnia remote hive monitoring system*. 2014. Disponível em: <https://www.arnia.co.uk>. Citado 2 vezes nas páginas 17 e 18.

AUTODESK, I. *Design Rule Check: PCB Layout Basics 3*. 2016. Disponível em: <https://www.autodesk.com/products/eagle/blog/design-rule-check-pcb-layout-basics-3/>. Citado na página 38.

AUTODESK, I. *EAGLE CAD*. 2020. Disponível em: <https://www.autodesk.com/products/eagle/overview>. Citado na página 37.

AVIA, S. *24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales*. 2018. Disponível em: <https://img.filipeflop.com/files/download/Datasheet_HX711.pdf>. Citado 2 vezes nas páginas 28 e 29.

BODYRETPA. *Serial Programing SPI Enable*. 2016. Disponível em: <https://bodyretpa.weebly.com/serial-program-ing-spi-enable.html>. Citado na página 26.

BOLTON, W. *Instrumentação & Controle*. [S.l.]: Hemus, 2002. ISBN 852890119X. Citado na página 27.

CSETE, A. *Gqrx SDR*. 2013. Disponível em: <https://gqrx.dk/>. Citado na página 48.

DUTRA, T. F. S. *Beehiveior sistema de monitoramento de colmeias de produção apícola*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, jul 2016. Citado 2 vezes nas páginas 17 e 18.

FOROUZAN, B. A. *Comunicação de Dados e Redes de Computadores (Em Portuguese do Brasil)*. [S.l.]: Mc Graw Hill, 2007. ISBN 8586804886. Citado na página 24.

FREITAS, D. G. F. et al. *Nível tecnológico e rentabilidade de produção de mel de abelha (Apis mellifera) no Ceará*. 2014. Citado na página 17.

GISERMAN, B. L. G. e. A. C. J. L. F. *LoRa - Long Range*. 2019. Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/lora/index.html>. Citado na página 20.

GUANGZHOU, A. E. C. L. *Temperature and humidity module AM2302 Product Manual*. 2018. Disponível em: <http://aosong.com/products-22.html>. Citado na página 27.

HOPE, M. C. L. *Enhanced Power Long Range Transceiver Module*. 2006. Disponível em: <https://www.hoperf.com/data/upload/portal/20190301/RFM95PW.pdf>. Citado na página 30.

HORAK, K. *Sinal Discreto*. 2010. Disponível em: <http://midas.uamt.feec.vutbr.cz/ZVS/Exercise01/content_cz.php>. Citado na página 23.

JESUS, F. T. *Sistema de calefação para ninhos de abelhas-sem-ferrão com controle e leitura de temperatura interna por sistema remoto*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, jan 2017. Citado 3 vezes nas páginas 17, 18 e 59.

KRIDI, D. S. *Monitoramento de padrões térmicos em colmeias de abelhasS via redes de sensores sem fio*. Dissertação (Mestrado) — Universidade Federal do Ceará, aug 2014. Citado 2 vezes nas páginas 17 e 18.

LIMA, C. B. de. *Técnicas de Projetos Eletrônicos com os Microcontroladores AVR*. first. [S.l.]: clube de autores, 2010. Citado 2 vezes nas páginas 31 e 32.

LIMA, C. B. de. *AVR e Arduino Técnicas de Projeto*. second. [S.l.]: dos autores, 2012. Citado 2 vezes nas páginas 24 e 25.

LIMA, T. *Placas de Circuito Impresso Multicamadas*. 2013. Disponível em: <https://https://embarcados.com.br/placas-de-circuito-impresso/>. Citado na página 31.

MAXIM, I. *Extremely Accurate I2C-Integrated RTC/TCXO/Crystal*. 2015. Disponível em: <https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>. Citado na página 29.

MEHL, E. L. de M. *Projeto de Placas de Circuito Impresso com o Software Eagle*. 2005. Citado na página 30.

MELO, P. R. de S. *PLACAS DE CIRCUITO IMPRESSO: MERCADO ATUAL E PERSPECTIVAS*. 2001. Disponível em: <https://web.bndes.gov.br/bib/jspui/bitstream/1408/13440/2/BS014PlacasdeCircuitoImpresso_MercadoAtualePerspectivas_P_BD.pdf>. Citado na página 30.

MICROCHIP. *AVR® Microcontroller with Core Independent Peripherals and picoPower® Technology*. 2018. Disponível em: <http://ww1.microchip.com>. Citado 5 vezes nas páginas 21, 23, 24, 25 e 32.

MOECKE, F. E. *Controle de Potência em um Sistema de Monitoramento de Abelhas off-grid*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, jul 2019. Citado 2 vezes nas páginas 59 e 60.

MOECKE, M. et al. *PROJETO RF-ABELHAS: SENSORIAMENTO REMOTO DAS CONDIÇOES AMBIENTAIS DE COLMEIAS DE ABELHAS UTILIZANDO RÁDIO FREQUÊNCIA*. 2018. Disponível em: <https://wiki.sj.ifsc.edu.br/images/3/31/PosterSEPEI2018.pdf>. Citado na página 19.

MOTOROLA, I. *SPI Block Guide V03.06*. 2000. Disponível em: <https://opencores.org/usercontent/doc/1499360489f>. Citado na página 26.

NOOELEC, I. *Some Measurements on DVB-T Dongles with E4000 and R820T Tuners: Image Rejection, Internal Signals, Sensitivity, Overload, 1dB Compression, Intermodulation*. 2013. Disponível em: <https://www.nooelec.com/store/downloads/dl/file/id/35/product/17/e4000_r820t_tuner_comparison.pdf>. Citado na página 47.

NXP, S. *I2C-bus specification and user manual*. 2014. Disponível em: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>. Citado 2 vezes nas páginas 25 e 26.

PEDRONI, V. *Eletr^{a}nica Digital Moderna e Vhdl*. [S.l.]: Elsevier, 2010. ISBN 8535234659. Citado na página 22.

POLLENITY. *uHive smart beehive equipment*. 2019. Disponível em: <https://pollenity.com>. Citado 2 vezes nas páginas 17 e 18.

ROHM, C. L. *Digital 16bit Serial Output Type Color Sensor IC*. 2010. Disponível em: <http://www.elechouse.com/elechouse/images/product/Digital%20light%20Sensor/bh1750fvi-e.pdf>. Citado na página 28.

SACCO, F. *10 mandamentos da PCB*. 2015. Disponível em: <https://www.embarcados.com.br/10-mandamentos-da-pcb>. Citado na página 32.

TEKTRONIX, I. *Current Probes A621 & A622 Datasheet*. 2017. Disponível em: <https://download.tek.com/datasheet/A621-A622-Current-Probes-Datasheet-60W150813.pdf>. Citado na página 47.

TEKTRONIX, I. *MSO/DPO2000B Series Mixed Signal Oscilloscope*. 2020. Disponível em: <https://download.tek.com/datasheet/MSO-DPO2000-Datasheet-EN-US-3GW-28413-8.pdf>. Citado na página 46.

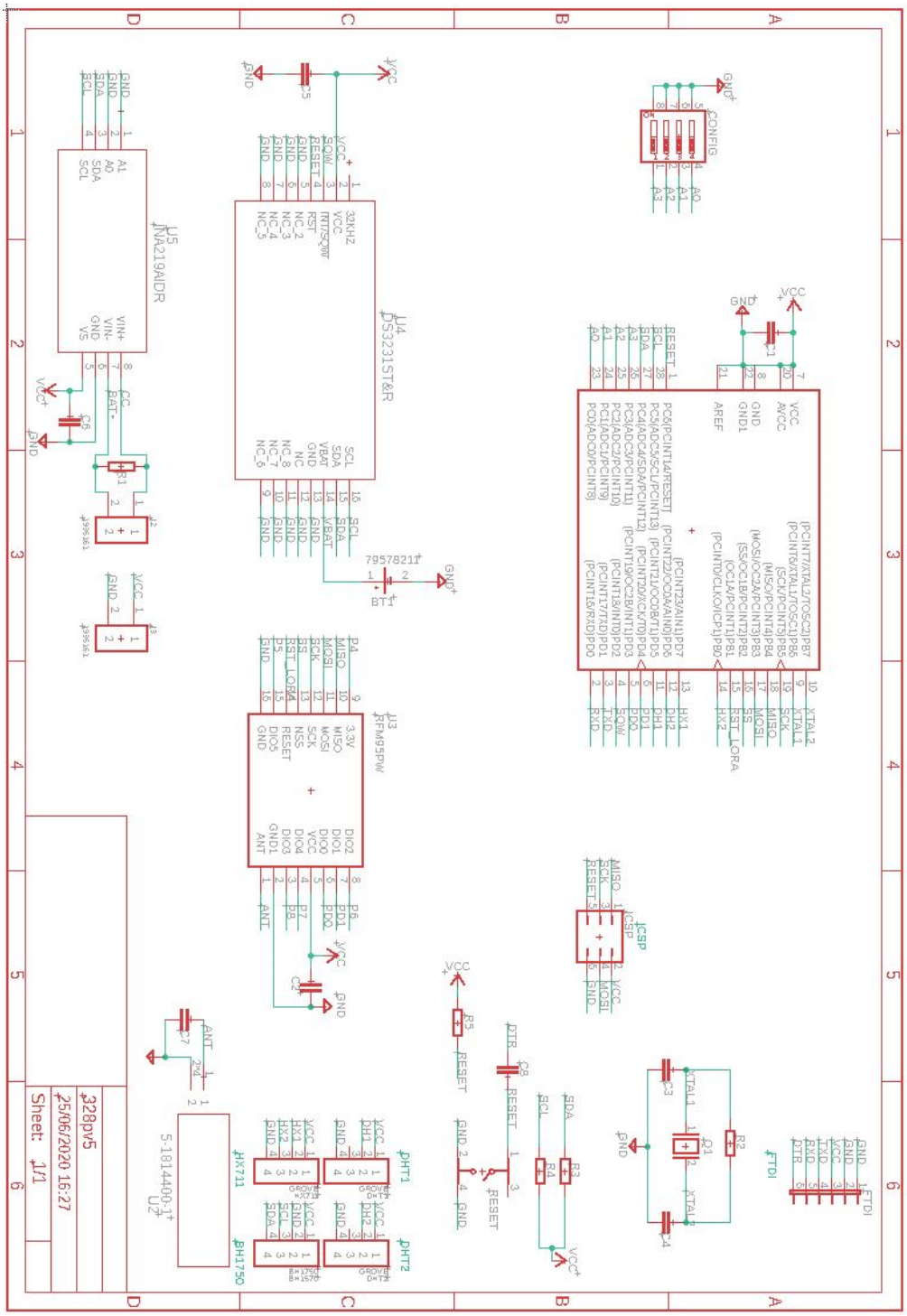
TEXAS, I. I. *INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I 2C Interface*. 2015. Disponível em: <http://www.ti.com/lit/ds/symlink/ina219.pdf>. Citado na página 29.

TOCCI, R. *Sistemas digitais : principios e aplicacoes*. Sao Paulo: Prentice Hall, 2003. ISBN 8587918206. Citado na página 23.

Apêndices

APÊNDICE A – ESQUEMÁTICO COMPLETO

Figura 42 – Esquemático completo



Fonte: Elaborada pelo autor.

APÊNDICE B – FIRMWARES DESENVOLVIDO PARA TESTES DOS SENSORES E MÓDULO LORA

Código B.1 – Firmware sensores

```
#include <lmic.h>
#include <hal/hal.h>
#include <DS3232RTC.h>
#include <avr/sleep.h>
#include <HX711.h>
#include <Wire.h>
#include <Adafruit_INA219.h>
#include <BH1750.h>
#include <DHT.h>

#define SQW 2
#define DIO0 3          //pino digital 0 módulo lora
#define DIO1 4          //pino digital 1 módulo lora
#define DHT1_PIN 5      //pino associado ao sensor de umidade e temperatura 1
#define DHT2_PIN 6      //pino associado ao sensor de umidade e temperatura 2
#define SCK1 7          //SCK clock sensor de peso
#define DATA1 8         //Data sensor de peso
#define RST_LORA 9      //pino resete módulo lora
#define NSS 10          //chip select módulo lora
#define DHTTYPE DHT22   //modelo sensor temperatura
#define PRINTER         //se definido, imprime informações na serial

HX711 sensor_peso;             //balanca
Adafruit_INA219 ina219(0x40); //PVI i2c=0x40
BH1750 lightMeter;             //luminancia
DHT dht1(DHT1_PIN, DHTTYPE);   //umidade e temperatura 1
DHT dht2(DHT2_PIN, DHTTYPE);   //umidade e temperatura 2

const unsigned TX_INTERVAL = (5);
static osjob_t sendjob;

byte mydata[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00};
const lmic_pinmap lmic_pins = { .nss = NSS, .rxtx = LMIC_UNUSED_PIN, .rst =
    RST_LORA, .dio = {DIO0,                                DIO1, LMIC_UNUSED_PIN},
    };
```

```
34
35 static const u1_t PROGMEM APPEUI[8] = { 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0
       x00 , 0x00  };
36 static const u1_t PROGMEM DEVEUI[8] = { 0xD4, 0x3D, 0x25, 0xF0, 0x28, 0x87, 0x96, 0
       x00 };
37 static const u1_t PROGMEM APPKEY[16] = { 0xCC, 0x4F, 0x8C, 0xAF, 0x28, 0x10, 0x9F,
       0xED, 0xCB,  0x69,                                      0x34, 0x25, 0xAF, 0xA9,
        0xC7, 0x0D };
38
39 void os_getArtEui (u1_t* buf) {  memcpy_P(buf, APPEUI, 8);}
40 void os_getDevEui (u1_t* buf) {  memcpy_P(buf, DEVEUI, 8);}
41 void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
42
43 void onEvent (ev_t ev) {
44   Serial.print(os_getTime());
45   Serial.print(": ");
46   switch (ev) {
47     case EV_SCAN_TIMEOUT:
48       Serial.println(F("EV_SCAN_TIMEOUT"));
49       break;
50     case EV_BEACON_FOUND:
51       Serial.println(F("EV_BEACON_FOUND"));
52       break;
53     case EV_BEACON_MISSED:
54       Serial.println(F("EV_BEACON_MISSED"));
55       break;
56     case EV_BEACON_TRACKED:
57       Serial.println(F("EV_BEACON_TRACKED"));
58       break;
59     case EV_JOINING:
60       Serial.println(F("EV_JOINING"));
61       break;
62     case EV_JOINED:
63       Serial.println(F("EV_JOINED"));
64       break;
65     case EV_RFU1:
66       Serial.println(F("EV_RFU1"));
67       break;
68     case EV_JOIN_FAILED:
69       Serial.println(F("EV_JOIN_FAILED"));
70       break;
71     case EV_REJOIN_FAILED:
72       Serial.println(F("EV_REJOIN_FAILED"));
73       break;
74     case EV_TXCOMPLETE:
75       Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
76       if (LMIC.txrxFlags & TXRX_ACK)
```

```
77          Serial.println(F("Received ack"));
78        if (LMIC.dataLen) {
79          Serial.println(F("Received "));
80          Serial.println(LMIC.dataLen);
81          Serial.println(F(" bytes of payload"));
82        }
83        do_send(&sendjob);
84        break;
85      case EV_LOST_TSYNC:
86        Serial.println(F("EV_LOST_TSYNC"));
87        break;
88      case EV_RESET:
89        Serial.println(F("EV_RESET"));
90        break;
91      case EV_RXCOMPLETE:
92        Serial.println(F("EV_RXCOMPLETE"));//data received in ping slot
93        break;
94      case EV_LINK_DEAD:
95        Serial.println(F("EV_LINK_DEAD"));
96        break;
97      case EV_LINK_ALIVE:
98        Serial.println(F("EV_LINK_ALIVE"));
99        break;
100     default:
101       Serial.println(F("Unknown event"));
102       break;
103   }
104 }
105
106 void do_send(osjob_t* j) {
107   // Check if there is not a current TX/RX job running
108   if (LMIC.opmode & OP_TXRXPEND) {
109     Serial.println(F("OP_TXRXPEND, not sending"));
110     Serial.flush();
111   } else {
112     LMIC_setTxData2(1, mydata, sizeof(mydata) - 1, 0);
113     Serial.println(F("Packet queued"));
114     Serial.flush();
115   }
116   // Next TX is scheduled after TX_COMPLETE event.
117 }
118
119 void setup() {
120   Serial.begin(9600);
121   Serial.println(F("Starting"));
122   pinMode(SQW, INPUT_PULLUP);
123   ina219.begin();
```

```
124   ina219.setCalibration_16V_400mA();
125   sensor_peso.begin(DATA1, SCK1);
126   sensor_peso.set_scale(22038.f);
127   sensor_peso.tare();
128   lightMeter.begin();
129   RTC.setAlarm(ALM1_MATCH_SECONDS, 1, 0, 0, 0);
130   RTC.alarm(ALARM_1);
131   RTC.alarmInterrupt(ALARM_1, true);
132   RTC.squareWave(SQWAVE_NONE);
133   attachInterrupt(digitalPinToInterrupt(SQW), acordar, FALLING);
134   ler();
135   os_init();
136   LMIC_reset();
137 }
138
139 void loop() {
140   dormir();
141   ler();
142 }
143
144 void ler() {
145   ina();
146   delay(1);
147   hx();
148   delay(1);
149   bh();
150   delay(10);
151   dht_int();
152   delay(10);
153   dht_ext();
154   delay(10);
155
156 }
157 void dht_ext() {
158   float h_ex = dht2.readHumidity();
159   float t_ex = dht2.readTemperature();
160   if (isnan(h_ex) || isnan(t_ex)) {
161     Serial.println("Falha na leitura do sensor DHT2!");
162     Serial.flush();
163     return;
164   }
165 #ifdef PRINTER
166   Serial.print(F("Umidade externa: "));
167   Serial.print(h_ex);
168   Serial.print(F("%  Temperatura externa: "));
169   Serial.print(t_ex);
170   Serial.println(F("C"));
```

```
171 #endif
172 }
173
174 void dht_int() {
175   float h_in = dht1.readHumidity();
176   float t_in = dht1.readTemperature();
177   if (isnan(h_in) || isnan(t_in)) {
178     Serial.println("Falha na leitura do sensor DHT1!");
179     Serial.flush();
180     return;
181   }
182 #ifdef PRINTER
183   Serial.print(F("Umidade interna: "));
184   Serial.print(h_in);
185   Serial.print(F("%  Temperatura interna: "));
186   Serial.print(t_in);
187   Serial.println(F("C"));
188 #endif
189 }
190
191 void bh() {
192   float luz = lightMeter.readLightLevel();
193 #ifdef PRINTER
194   Serial.print("Light: ");
195   Serial.print(luz);
196   Serial.println(" lx");
197 #endif
198 }
199
200 void ina() {
201   ina219.powerSave(false);
202   delay(1);
203   float Im, Vm = 0;
204
205   Im = ina219.getCurrent_mA();
206   Vm = ina219.getBusVoltage_V();
207   ina219.powerSave(true);
208
209 #ifdef PRINTER
210   Serial.print("Corrente: ");
211   Serial.print(-Im);
212   Serial.print("   Tensao: ");
213   Serial.println(Vm);
214 #endif
215 }
216
217 void hx() {
```

```
218   sensor_peso.power_up();
219   delay(0.5);
220   float peso_f =  sensor_peso.get_units(10) * (1);
221   if (peso_f < 0) peso_f = 0;
222   sensor_peso.power_down();
223 #ifdef PRINTER
224   Serial.print("Peso: ");
225   Serial.print(peso_f);
226   Serial.println(" Kg");
227 #endif
228 }
229
230 void acordar() {
231   sleep_disable();
232 #ifdef PRINTER
233   Serial.println(F("Acordando!"));
234 #endif
235 }
236
237 void dormir() {
238   RTC.alarm(ALARM_1);
239   set_sleep_mode(SLEEP_MODE_PWR_DOWN);
240   sleep_enable();
241 #ifdef PRINTER
242   Serial.println(F("Sleep-Mode!"));
243 #endif
244   delay(50);
245   sleep_cpu();
246 }
```

Código B.2 – Firmware lora

```
1  #include <lmic.h>
2  #include <hal/hal.h>
3  #include <SPI.h>
4  #include <avr/sleep.h>
5
6  #define DIO0 3          //pino digital 0 módulo lora
7  #define DIO1 4          //pino digital 1 módulo lora
8  #define RST_LORA 9      //pino resete módulo lora
9  #define NSS 10          //chip select módulo lora
10 //pinos 11, 12 e 13 são para SPI(MOSI, MISO e SCK respectivamente) do módulo lora
11
12 byte mydata[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00};
13 static const u1_t PROGMEM APPEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0
       x00 };
```

```
14 static const u1_t PROGMEM DEVEUI[8] = { 0xD4, 0x3D, 0x25, 0xF0, 0x28, 0x87, 0x96, 0
       x00 };
15 static const u1_t PROGMEM APPKEY[16] = { 0xCC, 0x4F, 0x8C, 0xAF, 0x28, 0x10, 0x9F,
       0xED, 0xCB, 0x69,                                        0x34, 0x25, 0xAF, 0xA9,
        0xC7, 0x0D };
16 void os_getArtEui (u1_t* buf) {  memcpy_P(buf, APPEUI, 8);}
17 void os_getDevEui (u1_t* buf) {  memcpy_P(buf, DEVEUI, 8);}
18 void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
19 static osjob_t sendjob;
20 const unsigned TX_INTERVAL = 5;
21 const lmic_pinmap lmic_pins = {  .nss = NSS,  .rxtx = LMIC_UNUSED_PIN,  .rst =
       RST_LORA,  .dio = {DIO0,                                DIO1, LMIC_UNUSED_PIN}, };
22
23 void onEvent (ev_t ev) {
24   Serial.print(os_getTime());
25   Serial.print(": ");
26   switch (ev) {
27     case EV_SCAN_TIMEOUT:
28       break;
29     case EV_BEACON_FOUND:
30       break;
31     case EV_BEACON_MISSED:
32       break;
33     case EV_BEACON_TRACKED:
34       break;
35     case EV_JOINING:
36       Serial.println(F("EV_JOINING"));
37       break;
38     case EV_JOINED:
39       Serial.println(F("EV_JOINED"));
40       LMIC_setLinkCheckMode(0);
41       break;
42     case EV_JOIN_FAILED:
43       break;
44     case EV_REJOIN_FAILED:
45       break;
46     case EV_JOIN_TXCOMPLETE:
47       break;
48     case EV_TXCOMPLETE:
49       Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
50       if (LMIC.txrxFlags & TXRX_ACK)
51         Serial.println(F("Received ack"));
52       if (LMIC.dataLen) {
53         Serial.print(LMIC.dataLen);
54       }
55       cont++;
56       break;
```

```
57      case EV_LOST_TSYNC:
58        break;
59      case EV_RESET:
60        break;
61      case EV_RXCOMPLETE:
62        Serial.println(F("EV_RXCOMPLETE"));
63        break;
64      case EV_LINK_DEAD:
65        break;
66      case EV_LINK_ALIVE:
67        break;
68      case EV_TXSTART:
69        Serial.println(F("EV_TXSTART"));
70        break;
71      default:
72        break;
73    }
74  }
75
76  void do_send(osjob_t* j) {
77    // Check if there is not a current TX/RX job running
78    if (LMIC.opmode & OP_TXRXPEND) {
79      Serial.println(F("OP_TXRXPEND, not sending"));
80    } else {
81      mydata[0] = 0xCF5 & ~(0xCF5 & 0x0f00);
82      mydata[1] = ((0xCF5 >> 8) << 4) | 0xDCF >> 8;
83      mydata[2] = 0xDCF & ~(0xDCF & 0x0f00);
84      mydata[3] = 0xFF05 & ~(0xFF05 & 0x0f00);
85      mydata[4] = ((0xFF05 >> 8) << 4) | 0x1CFF >> 8;
86      mydata[5] = 0x1CFF & ~(0x1CFF & 0x0f00);
87      mydata[6] = 0x341 >> 8;
88      mydata[7] = 0x341 & ~((0x341 >> 8) << 8);
89      mydata[8] = 0x7F7F >> 8;
90      mydata[9] = 0x7F7F & ~((0x507F >> 8) << 8);
91      mydata[10] = ((0x3F0 & 0x0Ff00) >> 8);
92      mydata[11] = (0x3F0 & 0x00ff);
93      mydata[12] = ((0x10 & 0x0ff00) >> 8);
94      mydata[13] = ( 0x10 & 0x00FF);
95      LMIC_setTxData2(1, mydata, sizeof(mydata), 0);
96      Serial.println(F("Packet queued"));
97    }
98  }
99
100 void setup() {
101   Serial.begin(9600);
102   os_init();
103   LMIC_reset();
```

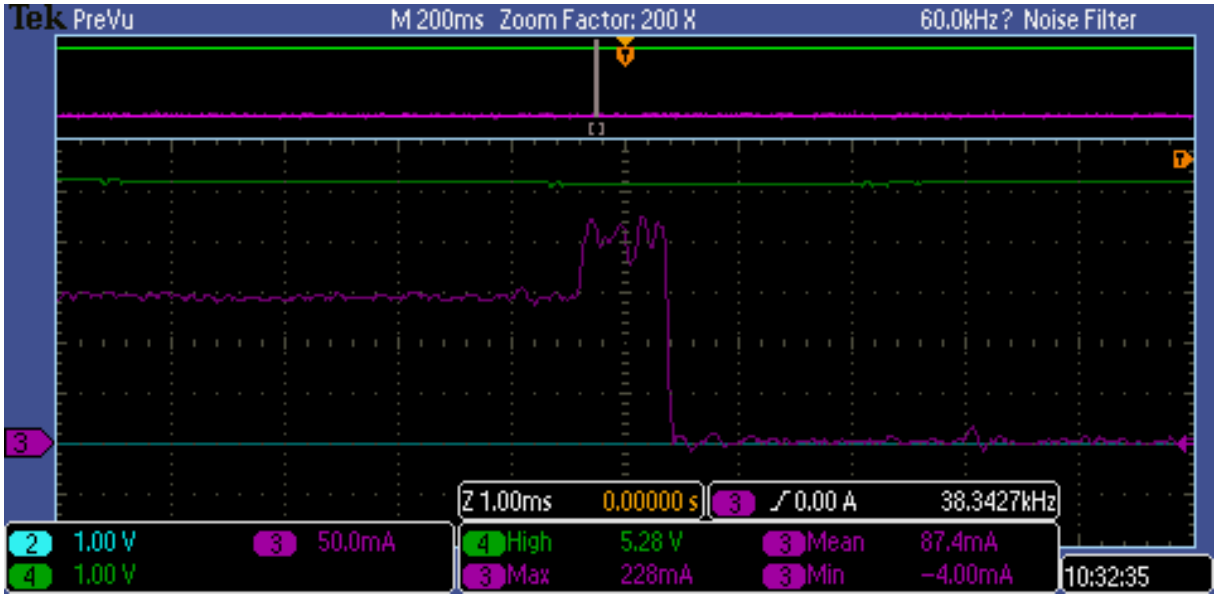
```
104   LMIC_setAdrMode(false);
105   LMIC_setLinkCheckMode(0);
106   LMIC_setDrTxpow(DR_SF9, 2);
107   LMIC.dn2Dr = DR_SF9;
108   LMIC_selectSubBand(1);
109   do_send(&sendjob);
110 }
111
112 void loop() {
113   os_runloop_once();
114   if (cont == 1) {
115     Serial.println(F("sleep"));
116     delay(80);
117     set_sleep_mode(SLEEP_MODE_PWR_DOWN);
118     sleep_enable();
119     sleep_cpu();
120   }
121 }
```

APÊNDICE C – MEDIDAS DE CORRENTES DOS MODOS ATIVO E SLEEP COM O OSCILOSCÓPIO

Figura 43 – Momento da transição entre modo sleep e ativo

Fonte: Elaborada pelo autor.Nota: A corrente foi medida com módulo de amplificação 10 vezes.

Figura 44 – Momento da transição entre modo ativo e sleep

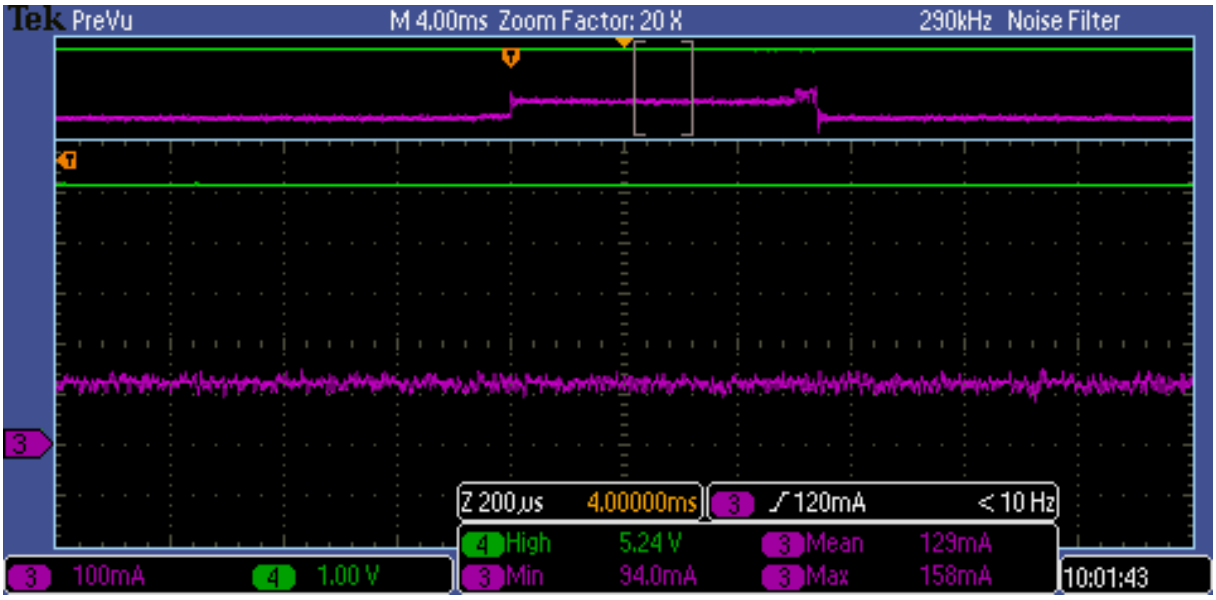


Fonte: Elaborada pelo autor.

Figura 45 – Média de consumo de potência em modo sleep

Fonte: Elaborada pelo autor.

Figura 46 – Média de consumo de potência em modo ativo



Fonte: Elaborada pelo autor.

APÊNDICE D – MEDIDAS DE CORRENTES DOS SENSORES COM O OSCILOSCÓPIO

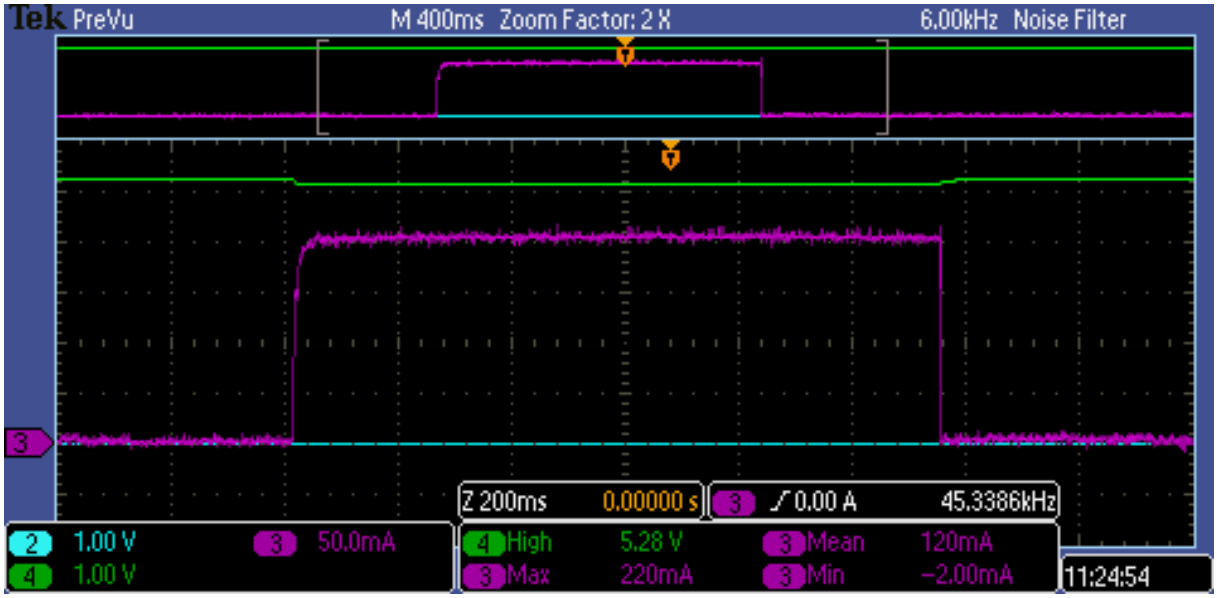
Figura 47 – Momento da leitura do sensor de tensão e corrente

Fonte: Elaborada pelo autor.

Figura 48 – Momento da leitura do sensor de temperatura e umidade

Fonte: Elaborada pelo autor.

Figura 49 – Momento da leitura do sensor de peso

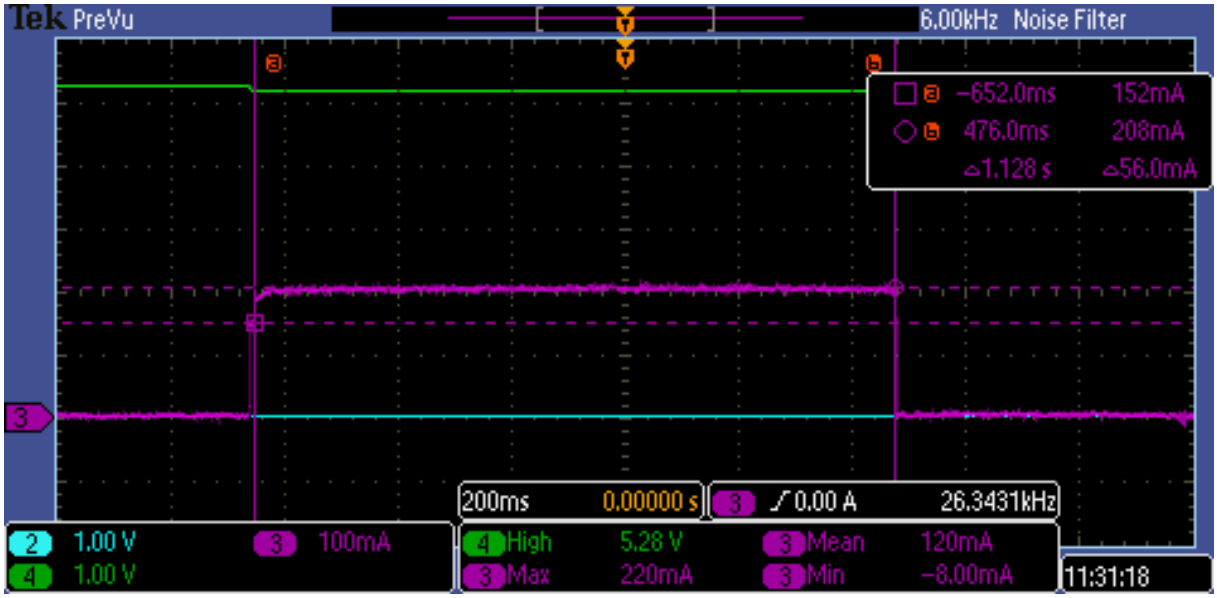


Fonte: Elaborada pelo autor.

Figura 50 – Pico de corrente na leitura do sensor de peso

Fonte: Elaborada pelo autor.

Figura 51 – Tempo e potência na leitura do sensor de peso



Fonte: Elaborada pelo autor.

Figura 52 – Momento da leitura do sensor de iluminância

Fonte: Elaborada pelo autor.