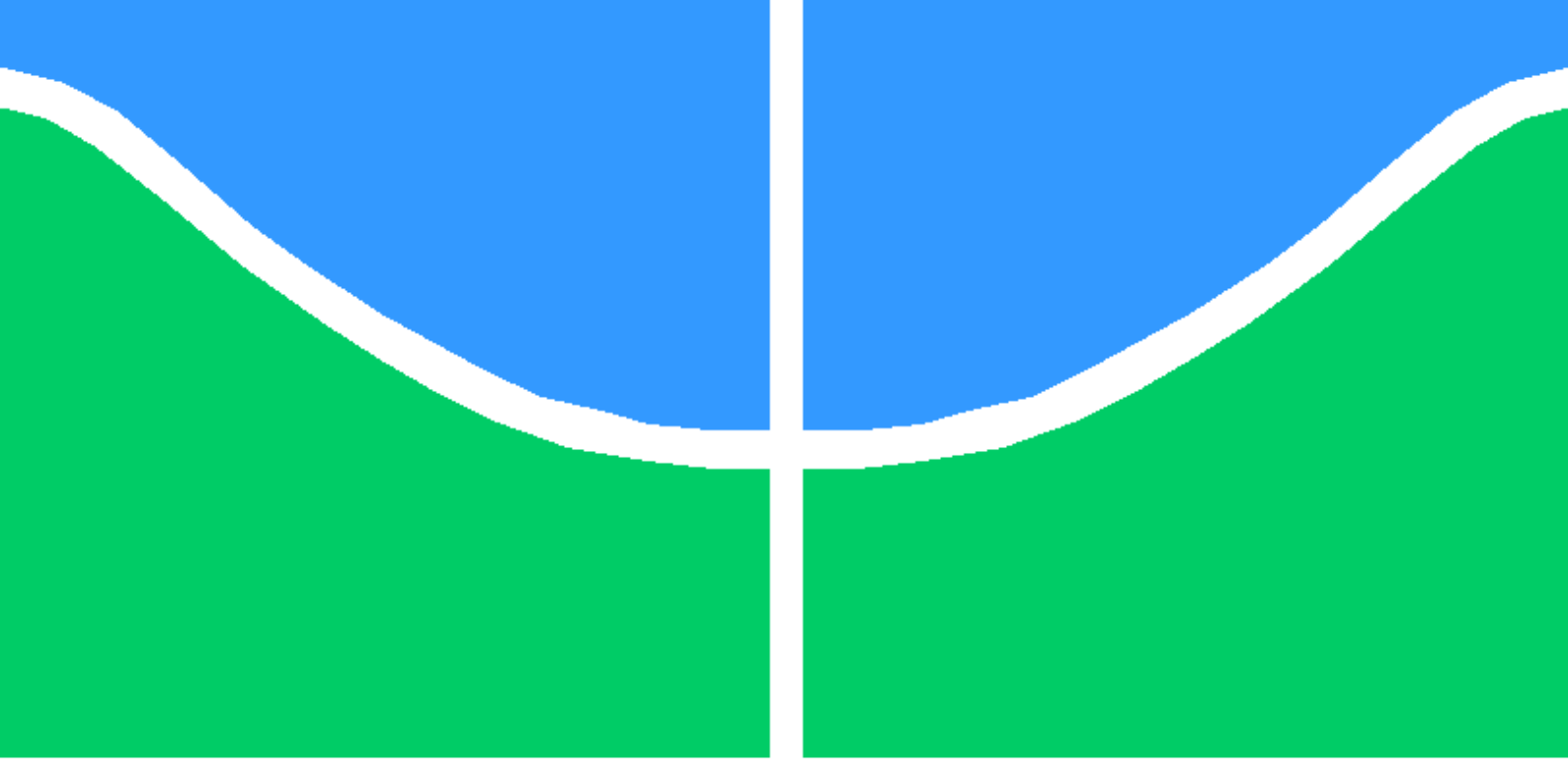

RECURSO DE COMUNICAÇÃO VIA BLUETOOTH PARA USO EM SISTEMAS DE AQUISIÇÃO DE SINAIS MÉDICOS

LUIZ FELIPE VILELA SANTOS DE ARAUJO

PROJETO FINAL DE GRADUAÇÃO EM ENGENHARIA DE REDES DE COMUNICAÇÃO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RECURSO DE COMUNICAÇÃO VIA BLUETOOTH PARA USO EM SISTEMAS DE AQUISIÇÃO DE SINAIS MÉDICOS

LUIZ FELIPE VILELA SANTOS DE ARAUJO

Orientador: PROF. DR. ADSON FERREIRA DA ROCHA, ENE/UNB

PROJETO FINAL DE GRADUAÇÃO EM ENGENHARIA DE REDES DE COMUNICAÇÃO

PUBLICAÇÃO - BRASÍLIA-DF, 28 DE NOVEMBRO DE 2019.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RECURSO DE COMUNICAÇÃO VIA BLUETOOTH PARA USO EM SISTEMAS DE AQUISIÇÃO DE SINAIS MÉDICOS

LUIZ FELIPE VILELA SANTOS DE ARAUJO

PROJETO FINAL DE GRADUAÇÃO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE GRADUAÇÃO EM ENGENHARIA DE REDES DE COMUNICAÇÃO.

APROVADA POR:

Prof. Dr. Adson Ferreira da Rocha, ENE/UnB
Orientador

Prof. Dr. Ricardo Zelenovsky, ENE/UnB
Examinador interno

Prof. Dr. João Luiz Azevedo de Carvalho, ENE/UnB
Examinador interno

BRASÍLIA, 28 DE NOVEMBRO DE 2019.

FICHA CATALOGRÁFICA
DE ARAUJO, LUIZ FELIPE VILELA SANTOS
Recurso de comunicação via *Bluetooth* para uso em sistemas de aquisição de sinais médicos
2019xv, 64p., 201x297 mm
(ENE/FT/UnB, Graduação, Engenharia de Redes de Comunicação, 2019)
Projeto Final de Graduação - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

DE ARAUJO, LUIZ FELIPE VILELA SANTOS (2019) Recurso de comunicação via *Bluetooth* para uso em sistemas de aquisição de sinais médicos. Projeto Final de Graduação em Engenharia de Redes de Comunicação, Publicação , Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 64p.

CESSÃO DE DIREITOS

AUTOR: Luiz Felipe Vilela Santos de Araujo
TÍTULO: Recurso de comunicação via *Bluetooth* para uso em sistemas de aquisição de sinais médicos.
GRAU: Graduação ANO: 2019

É concedida à Universidade de Brasília permissão para reproduzir cópias desta projeto final de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor se reserva a outros direitos de publicação e nenhuma parte desta projeto final de Graduação pode ser reproduzida sem a autorização por escrito do autor.

_____

Luiz Felipe Vilela Santos de Araujo

Agradecimentos

Agradecimentos a todos que contribuíram de forma positiva significativa durante toda graduação. Aqueles que não só perguntaram se estava tudo certo da boca pra fora ou por convenção social. Mas que tentaram ativamente entender e fazer a diferença positivamente. Seja no lado acadêmico, profissional ou até pessoal. Cada um de vocês ajudou a construir a pessoa que sou hoje.

RESUMO

Plataformas com microcontroladores como o Arduino surgiram como uma alternativa a contornar e reduzir complexidade de se trabalhar em arquiteturas programadas em baixo nível. Entretanto, mantendo acesso a essa arquitetura, é uma usada uma abstração e utilizando uma linguagem de programação mais amigável. Por outro lado, essa abstração limita as capacidades de desenvolvimento. Assim, o presente trabalho explora a programação em baixo nível do microcontrolador usado para a construção de um recurso de comunicação via Bluetooth para uso em sistemas de aquisição de dados médicos.

Em um contexto de evolução das tecnologias, algumas características se tornaram mais presentes em projetos e produtos, enquanto outras características passaram a ser até mandatórias. Por exemplo, a mobilidade, eficiência energética, miniaturização de componentes, uso de tecnologias sem fio e etc. Para isso, as áreas de eletrônica, telecomunicações e programação têm tido grande contribuição para alcançar estes objetivos, e não apenas se atendo as suas áreas de atuação, mas também trabalhando em conjunto com outras, e sempre com o compromisso de abrir mão apenas do desnecessário, otimizando recursos e assim fazer com que o projeto alcance novos patamares.

Diante disso, o trabalho apresenta uma abordagem focada em coleta e transferência de dados de um Arduino Nano para um computador e exibi-los em tempo real, em que essa transferência é feita utilizando um meio sem-feio com o uso da tecnologia Bluetooth. Em aplicações médicas isto é necessário, visto que nem sempre é possível armazenar os dados para verificação posterior. A visualização deve ser imediata. Os resultados das simulações mostram que o desempenho na aquisição e transferência de dados é satisfatório. Podendo assim ser melhorado e utilizado para disponibilização de dados precisos em tempo real com baixo custo.

ABSTRACT

Microcontrollers like Arduino have emerged as an alternative to reduce the complexity of working on low-level programmed architecture and still make use of this architecture. Through an abstraction using a friendlier programming language. However, this abstraction limits development capabilities. Thus, the present work explores the low-level program of the Arduino Nano microcontroller to build a Bluetooth data acquisition system for medical data acquisition.

In the context of evolving technologies, some features have become more present in projects and products. Other characteristics have even become mandatory. For example mobility, energy efficiency, miniaturization, use of wireless technologies and so on. To reach this, areas like electronics, telecommunications, and programming have made major contributions to achieve these goals. And not only taking care of their expertise areas but also working together with others. Always committed to giving up only what is necessary for the project and reaching new goals.

With that in mind, this work presents an approach focused on transferring data from an Arduino Nano to a computer and displaying it in real-time. Where the transfer is done using a wireless approach by using Bluetooth technology. In medical applications, it is necessary and it is not always possible to check the data shortly thereafter. Simulation results show that data acquisition and transfer performance is satisfactory. And it can be improved and used to provide accurate real-time data at a low cost.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE TERMOS E SIGLAS

A2DP	*Advanced Audio Distribution Profile*
ACME	*Analog Comparator Multiplexer Enable*
ADATE	*ADC Auto Trigger Enable*
ADC	*Analog-to-Digital Converter*
ADCH	*ADC Data Register High Byte*
ADCL	*ADC Data Register Low Byte*
ADCSR	*ADC Control and Status Register*
ADEN	*ADC Enable*
ADIE	*ADC Interrupt Enable*
ADIF	*ADC Interrupt Flag*
ADLAR	*ADC Left Adjust Result*
ADMUX	*ADC Multiplexer Select*
ADMUX	*ADC Multiplexer Select*
ADMUX	*Multiplexer Selection Register*
ADPS	*ADC Prescaler Select Bit*
ADSC	*ADC Start Conversion*
ADTS	*ADC Auto Trigger Source*
AFH	*Frequency-Hopping Spread Spectrum*
AREF	*Analog Reference*
ASCII	*American Standard Code for Information Interchange*

CS	*Clock Select*
DOR	*Data OverRun*
ECG	*Eletrocardiograma*
EDR	Enhanced Data Rate
EMG	Eletromiográfico
FE	*Frame Error*
HS	*High Speed*
I/O	Input Output
I2C	*Inter-Integrated Circuit*
ICF	*Input Capture Flag*
ICP	*Input Capture Pin*
ICR	*Input Capture Register*
IDE	Integrated Developmente Environment
IEEE	*Institute of Electrical and Electronics Engineers*
ISM	Industrial, Scientific and Medical
LMP	*Link Management Protocol*
MATLAB	*Matrix Laboratory*
MPCM	*Multi-Processor Communication Mode*
MUX	*Analog Channel Selection Bit*
OC	*Output Compare*
OCF	*Output Compare Flag*
OCR	*Output Compare Resiter*
OSI	*Open System Interconnection*
PCI	*Peripheral Component Interconect*
PWM	*Pulse-Width Modulation*
REFS	*Reference Selection Bit*
RFCOMM	Radio Frequency Communication

RX	*Receiver*
RXB8	*Receive Data Bit 8*
RXC	*USART Receive Complete*
RXCIE	*RX Complete Interrupt Enable*
RxD	*Receive Data*
RXE	*Receiver Enable*
SATA	*Serial Advanced Technology Attachment*
SIG	*Special Interest Group*
SPI	*Serial Peripheral Interface*
SPP	*Serial Port Profile*
TCCR	*Timer/Counter Control Resgiter*
TCNT	*Timer/Counter Register*
TIFR	*Timer/Counter Interrupt Flag Register*
TOV	*Timer/Counter Overflow Flag*
TX	*Transmitter*
TXB8	*Trasmit Data Bit 8*
TXC	*USART Transmit Complete*
TXCIE	*TX Complete Interrupt Enable*
TxD	*Transmit Data*
TXE	*Transmitter Enable*
U2X	*Double the USART Transmission Speed*
UBBR	*USART Baud Rate Register*
UCPOL	*Clock Polarity*
UCSR	*USART Control and Status Register*
UCSZ	*Character Size*
UDR	*USART I/O Data Register*
UDRE	*USART Data Register Empty*

UDRIE	*USART Data Register Empty Interrupt Enable*
UMSEL	*USART Mode Select*
UPE	*USART Parity Error*
UPM	*Parity Mode*
USAR	*Parity Error*
USART	*Universal Synchronous and Asynchronous Receiver-Transmitter*
USB	*Universal Serial Bus*
USBS	*Stop Bit Select*
Wi-Fi	*Wireless Fidelity*
XTAL	*Chip Clock Oscillator*

Capítulo 1

Introdução

1.1 Contextualização

O uso da tecnologia em diversas áreas nunca esteve tão em alta. Os desenvolvimentos em áreas adjacentes como tecnologia da informação, eletrônica, elétrica e telecomunicações tem auxiliado outras diversas áreas em seus desenvolvimentos tecnológicos. Dados e informações também estão abundantes como nunca antes. Plataformas e sistemas de telecomunicações atuais garantem o acesso instantâneo as informações, com a precisão necessária. Em várias áreas do conhecimento, e em aplicações médicas isso não é diferente, pois estes desenvolvimentos fornecem agilidade, rigor nos resultados e praticidade para o desenvolvimento da profissão, pesquisas, testes entre outros.

A presença ubíqua da Internet é um outro fator determinante, não apenas para demonstrar que tudo pode ser conectado todo o tempo, mas com inovações tecnológicas que foram e ainda estão sendo desenvolvidas de forma paralela. Transmissões com fio e sem fio cada vez mais robustas, confiáveis e ágeis permitem que novas aplicações sejam desenvolvidas e mais problemas sejam resolvidos de forma eficiente. Por outro lado, as áreas de eletrônica e programação também juntaram esforços para criar dispositivos eficientes o bastante para sustentar todo esse desenvolvimento. Um desses esforços foram refletidos em microcontroladores, presentes no dia-a-dia de toda população, em carros, hospitais, elevadores, meios de transporte, objetos inteligentes conectados entre outros.

O uso dos microcontroladores também nunca esteve tão acessível. Adaptações de hardware e software fizeram seu uso ser muito abrangente. Tanto especialistas quanto iniciantes podem fazer uso de tais plataformas. O Arduino, microcontrolador a ser explorado neste trabalho, é um deles. Este microcontrolador traz toda a complexidade de se trabalhar com linguagens de programação de baixo nível, pinos do processador e circuitos integrados que não são triviais. Entretanto, a adaptação para prover abundância de portas lógicas e bibliotecas lógicas intuitivas, foi o suficiente para massificar seu uso. Assim, sua aplicação pode ser dar em várias áreas do conhecimento e em vários níveis de complexidade. Neste trabalho, o

foco será em aplicações para aquisição de sinais médicos.

O desenvolvimento de sistemas para aplicações médicas exige que seus resultados sejam confiáveis. Assim que tais sistemas e aplicações atingem estes requisitos podem ser utilizadas por profissionais da área. Se tratando de desenvolvimento tecnológico com uso de microcontroladores, pode-se cumprir com sucesso não apenas esses, mas vários requisitos. Desenvolvimento este limitado apenas a criatividade, conhecimento técnico e hardware disponíveis. De maneira constante os equipamentos médicos que fazem algum tipo de medição e coleta de informações têm que tornada cada mais rápida, precisa, móvel, menos invasiva e miniaturizada. E a área de tecnologia tem tido papel fundamental em auxiliar este desenvolvimento. Assim, este conjunto de profissionais de área da saúde com profissionais da área de tecnologia conseguem ter um resultado com essas características.

O uso de conhecimentos de eletrônica, programação e telecomunicações abre muitas possibilidades em soluções de problemas. Como por exemplo através da digitalização de informações. Vamos supor que se quer saber o histórico da pressão sanguínea de um paciente. A maneira tradicional é obter essas medidas de pressão sanguínea de forma analógica, onde um profissional utiliza de seu aprendizado e seus sentidos para auferir a medida, sendo necessário o controle manual do equipamento. Ao final, esta medida precisa ser registrada em algum lugar, tradicionalmente físico ao invés de digital, para se ter o histórico desejado. Ou seja, a disponibilização dos dados para o paciente ou outro profissional está muito limitada, assim como consulta posterior a essas informações. Ao utilizarmos de conhecimentos de eletrônica, programação e telecomunicações, pode-se padronizar e aumentar a precisão de tal tarefa. Um equipamento de medida automatizado, que fosse capaz de extrair a medida sem intervenção humana, e exceto para colocação ou retirada do equipamento. Esta medida pode ser convertida digitalmente e exibida em um painel também digital. Reduzindo assim a imprecisão inerente do ser humano de operar dispositivos analógicos que dependem de coordenação e de ter que ler escalas analógicas por exemplo. Quanto ao armazenamento da informação da medida para histórico, esta pode ser registrada manualmente em um meio digital, ou até de forma automatizada, podendo ser sincronizada em outros dispositivos compatíveis. Este sincronismo pode ser em tempo real ou após as medições. Também pode-se escolher como esta informação será transmitida até o local desejado. Pode ser utilizado um cartão de memória, um cabo conectado a algum outro dispositivo. Ou ainda, que o equipamento esteja em rede. Seja uma rede local ou seja via Internet. Assim, o dado poderá ser disponibilizado em tempo real. Durante todo esse processo existirão hardware(s) e software(s) dedicados atuando simultaneamente.

Assim, atualmente, os esforços de desenvolvimento chegaram ao ponto de que pessoas sem conhecimento médico específico tenham a sua disposição alguns dados simples. Por exemplo, em tempo real, suas frequências de batimentos cardíacos a partir de um sensor óptico de frequência cardíaca a disposição em um relógio inteligente (*smartwatch*). Elevando a coleta e disponibilização de informação fisiológicos a um novo patamar. O uso massivo de *smartphones* permite que cada pessoa seja um ponto de centralização de informações mas

também um nó com potencial conexão com a Internet. [Masato 2010] Aplicações que fazem uso, conforme exemplo, de um sensor óptico de frequência cardíaco, podem ter seus dados disponibilizados em tempo real. Seja no dispositivo vestível, em um celular, um computador ou outro equipamento conectado compatível. A área de esportes e práticas de exercícios físicos também se beneficiam dos avanços citados, podendo auxiliar em aumento de desempenho de atletas ou, por exemplo, para prevenção de sobrecarga durante as atividades.

1.2 Definição do Problema

O uso de microcontroladores pode ser aplicado ao problema de adquisição de dados confiáveis, pois a natureza deste tipo de dispositivo auxilia na solução. Como por exemplo ter várias portas lógicas a disposição. Ter conversores analógico-digitais, digitais para analógicos, temporizadores (*timers*), memórias internas, um microprocessador, etc. Estes componentes podem ser programados para atingir diversos objetivo. Há duas abordagens para utilização de microcontroladores. A primeira é realizar a construção de uma placa própria de circuito impresso com todos os componentes necessários. A segunda é utilizar uma placa já existente que contempla todos os componentes e entradas lógicas desejados. O presente trabalho utilizará da segunda abordagem através do microcontrolador Arduino Nano. Além do hardware ter os requisitos necessários, há também três benefícios muito simples de características que auxiliam qualquer projeto: o tamanho miniaturizado do microcontrolador, o baixo custo e a facilidade de replicar a solução.

O microcontrolador pode ser uma maneira de agregar as várias formas de desenvolvimento tecnológico e conhecimento. Uma delas é ao aplicar em um sistema para estações médicas. Fixas ou móveis. Para aquisição de diversos tipos de sinais médicos, onde o profissional ou até o paciente, poderá ser monitorado em tempo real. Pois a características citadas anteriormente na tarefas de digitalização de informações, processamento e disponibilidade de informações podem ser aplicadas ao problema. Assim, informações como número de passos, frequência cardíaca e o uso de por exemplo, um sensor polar, dados de um ECG (eletrocardiograma) ou EMG (eletromiógrafo) podem ser lidos e disponibilizados em tempo-real, a baixo custo e de forma prática. E essa disponibilização poderá ser feita em uma rede local, sem-fio, com uso do protocolo Bluetooth.

1.3 Objetivo

O objetivo do presente trabalho é a criação de um sistema de comunicação que receba e exiba informações de sinais médicos. Esse sistema centralizará a informação coletada de uma ou várias fontes. O sinal deverá ser processado e as especificações deverão ser estudadas e explicadas, pois o sistema poderá ter características que refletirão no resultado final. Essa comunicação deverá ser feita utilizando um canal sem-fio. Escolheu-se o uso do padrão

Bluetooth (IEEE 802.15.1) para esta comunicação.

O sistema permitirá a exibição de informações em tempo real e com clareza. O que poderá auxiliar em um diagnóstico médico, uma vez que o foco é para coleta de um ou várias fontes de sinais médicos. O sistema também pode ser ser aplicado de forma genérica a outros sinais. Também poderão ser verificados alguns limites de software e hardware do microcontrolador escolhido (Arduino Nano), e do microprocessador utilizado para o estudo, neste caso o ATmega328P da Atmel.

1.4 Estrutura do Trabalho

O trabalho está organizado em 5 capítulos, iniciando com a presente introdução. Segue no Capítulo 2, a apresentação do microcontrolador escolhido, e toda fundamentação teórica que embasa as escolhas de especificação do sistema proposta. Suas características, aplicações e limitações. O que também dará informações necessárias para a solução do problema levantado. O Capítulo 3 apresenta a tecnologia Bluetooth e as propriedades do padrão. Este tem papel fundamental na transmissão e disponibilização dos dados. No Capítulo 4 é detalhado a solução técnica de coleta e transmissão de dados, suas características e propriedades, mostrando seus resultados. Por fim, no capítulo 5, serão feitas as conclusões finais do trabalho e possíveis propostas para desenvolvimentos futuros.

Capítulo 2

O Arduino

2.1 Introdução

Este capítulo apresentará de forma sucinta um embasamento teórico, passando por o que é a plataforma Arduino, um breve histórico e suas principais características. Também serão caracterizados aspectos do hardware e software que terão papel fundamental para o desenvolvimento do sistema proposto, como por exemplo estrutura de temporização, conversores, portas lógicas, alguns registradores, entre outros. O objetivo desse capítulo não é esgotar o assunto, mas fornecer o embasamento teórico suficiente para a entendimento do sistema proposto.

2.2 Plataforma Arduino

Uma das perspectivas possíveis de explicar a plataforma é através da sua composição, que pode ser dividida em duas partes. A placa Arduino e a IDE (do ingês, *Integrated Development Environment*) do Arduino. A placa é o local onde de fato trabalha-se no hardware do projeto, onde serão feitas as conexões lógicas para entrada e saída de dados. A IDE do Arduino é onde são criados os programas em linguagem muito similar ao C, compilados e carregados na placa Arduino. Este códigos são chamados de *sketchs*. Assim, diferente da forma clássica de desenvolvimento de circuitos, com diversos resistores, capacitores, indutores, lidar com fios e complexas ligações, usa-se o software para dar flexibilidade nos desenvolvimento. Ou seja, o software irá dizer ao microprocessador o que fazer, e quando fazer [Banzi 2009]. O hardware é construído uma vez só, com placas e conexões documentadas e padronizadas.

Destaca-se também que o plataforma Arduino é um projeto de código aberto (*open-source*). E, originalmente, a plataforma de hardware e software chamava-se Wiring. E teve sua origem na Itália, em 2005, resultado de uma tese de mestrado. Esta tese propôs construir uma plataforma de baixo custo, objetivando o ensino de programação e facilitava a prototi-

pagem de soluções de problemas, transferindo o foco para o desenvolvimento dos trabalhos. O foco que era em construir hardwares capazes de suportar as soluções, agora é a rápida implantação de ideias, com foco da qualidade e otimização das arquiteturas de soluções. [Barragán 2004]

2.3 Arduino Nano

Um dos *hardwares* escolhidos que faz parte dos sistema para coleta de dados é o Arduino Nano. A escolha deste modelo se justifica pela quantidade de memória suficiente para o *software* a ser escrito, disponibilidade de conexões lógicas em número suficiente para o sistema proposto, além de que a miniaturização de componentes é bem-vinda quando não é necessário mais robustez no projeto. A Fig. 2.1 mostra todas suas conexões lógicas, incluindo alguns registradores associados a estes. O presente trabalho fará o detalhamento conforme necessário ao longo dos capítulos as partes necessárias à construção do sistema pretendido.

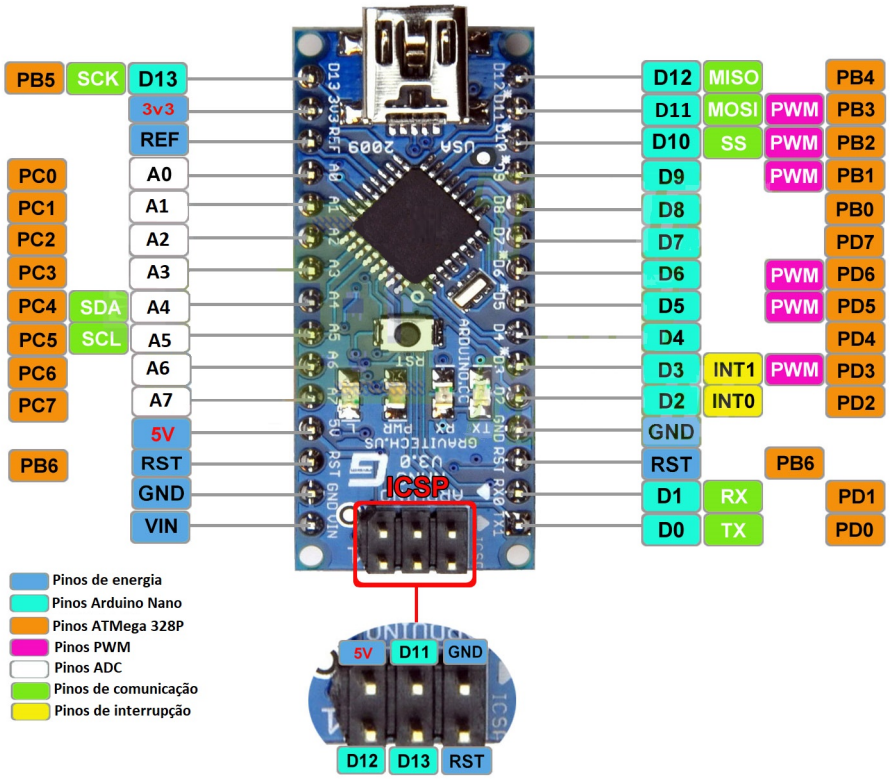

Figura 2.1: Arduino Nano e sua pinagem. Adaptado. [Techmemicro 2014]

O Arduino Nano é um dos membros da chamada família Arduino. Os membros dessa família se diferenciam principalmente por quantidade de portas e seu microprocessador. O microprocessador é fator determinante para quais e quantos recursos diferentes o Arduino terá. Por exemplo, há modelos com uma memória *flash* de 256 KB (Por exemplo o microprocessador ATmega2560), outros com 32 KB (Por exemplo o microprocessador ATmega328).

Tabela 2.1: Especificações do Arduino Nano com microprocessador ATmega328P. Adaptado. [Site Oficial do Arduíno 2019] e [Atmel Corporation 2015]

Componente	**Especificação**
Microcontrolador	ATmega328p
Arquitetura	AVR
Tensão de operação	5 V
Memória *flash*	32 KB
SRAM	2 KB
Frequência	16 MHz
Qtde. de Pinos Analógicos	8
EEPROM	1 KB
Tensão de entrada	7-12 V
Qtde. de Pinos Digitais	22 (6 são PWM)
Consumo	19 mA
Dimensões	18 x 45 mm
Peso	7 g

O Arduino Nano escolhido trabalha com o microprocessador ATmega328P, que é uma variação do ATmega328, no qual é usado o cabo micro USB ao invés do mini USB, e não há um conector para ligamos uma fonte externa. Esses fatores não influenciarão ativamente o projeto. [Site Oficial do Arduíno 2019]

O Arduino Nano, com microprocessador ATmega328P, possui características e capacidades detalhadas na Tabela 2.1.

2.4 Características do Arduino Nano a serem utilizadas

A seguir, serão destacadas as partes integrantes do hardware, registradores e abordagens via software necessárias para o desenvolvimento do sistema. Ressalta-se que a análise se manterá apenas nas área de conhecimento do projeto e não de todos os componentes da placa Arduino.

2.4.1 Interrupções

As interrupções são recursos importantes para o desenvolvimento do sistema. Com elas podemos interromper o fluxo do programa e dar atenção uma outra tarefa, que é prioritária naquele instante. O fluxo principal é retomado da sua finalização. Junta-se isso ao fato que é feito o uso de programação em cada registrador, uma vez que para o sistema proposto, as bibliotecas prontas do Arduino não cumprem os pré-requisitos. Serão utilizadas interrupções para o temporizador. Ou seja, para execução de subrotinas responsáveis marcação de tempo e subrotinas responsáveis pela conversão analógico-digital. Os vetores de interrupção rela-

Tabela 2.2: Lista de temporizadores disponíveis no ATmega328P [Atmel Corporation 2015]

Temporizador	Tamanho	Vetores de Interrupção	Interrupção
TIMER0	8 bits	TIMER0_COMPA	Por comparação
		TIMER0_COMPB	Por comparação
		TIMER0_OVF	Por estouro
TIMER1	16 bits	TIMER1_CAPT	Por captura
		TIMER1_COMPA	Por comparação
		TIMER1_COMPB	Por comparação
		TIMER1_OVF	Por estouro
TIMER2	8 bits	TIMER2_COMPA	Por comparação
		TIMER2_COMPB	Por comparação
		TIMER2_OVF	Por estouro

cionados a cada funcionalidade do Arduino podem ser verificados em detalhes no *datasheet* do microcontrolador.

2.4.2 Temporizadores

A família de microprocessadores AVR tem diferentes quantidades de temporizadores a depender do membro da família. Para o ATmega328P, há três temporizadores. TIMER0, TIMER1 e TIMER2. O TIMER0 e TIMER2 são temporizadores de 8 bits, sendo o TIMER1 de 16 bits. E pode-se dividir as interrupções para o temporizador em três grupos: as interrupções por comparação de registradores; interrupções por estouro do registradores e interrupções por captura. A Tabela 2.2 apresenta um resumo dos grupos de interrupções por temporizador e seus vetores responsáveis.

Também é necessário especificar os registradores responsáveis por estas operações. Destaca-se a seguir os principais. A notação a ser utilizada possui um valor $n = \{0, 1, 2\}$, que denota o contador de qual temporizador é a referência. A escolha dessa notação se justifica para que seja mantida a coerência com a Tabela 2.2. O TCNTn, (do inglês, *Timer/-Counter Register n*), é um contador de 8 bits (de 0x00 até 0xFF ou 0 a 255) ou 16 bits (de 0x0000 até 0xFFFF ou de forma equivalente, 0 a 65535). Ele é um contador de referência que incrementa a cada batida do relógio (do inglês, *clock*). Ciclicamente. Portanto, com reinicio automático. Há também uma opção em que pode ser especificado o valor de reinicio. Quando TCNTn atinge seu valor máximo, a *flag* TOVn é acionada. TOVn, (do inglês, *Timer/-Counter Overflow Flag*), é um bit do registrador TIFRn, (do inglês, *Timer/Counter Interrupt Flag Register*). É possível habilitar uma interrupção por estouro do contador. Uma forma mais utilizada é o temporizador por coincidência de comparação. Nessa modalidade usa-se os registradores de comparação OCRnA e OCRnB (do inglês, *Output Compare Registers*). Daqui em diante chamados de OCRnx também para manter a generalidade e conformidade com o *datasheet* do fabricante. Estes podem ser inicializados de acordo com o tempo esperado. Por exemplo, no TIMER1, quando TCNT1 = OCRnA, há a coincidência. Quando há

a coincidência, as *flags* OCFnA e OCFnB, (do inglês, *Output Compare Flag*) são acionadas. Ambas *flags* também fazem parte do registrador TIFRn. Nota-se que os valores de OCRnA e OCRnB devem ser inicializados. Uma das formas de reinicialização do TCNTn é feita quando há a coincidência por comparação, utilizando OCRnA ou OCRnB, o que simplifica a implementação. A forma de utilização dos temporizadores no sistema será explicada no Capítulo 4. O pino OC (do inglês, *Output Compare*), é a saída do circuito e fica disponível caso seja desejado seu uso, pois o mesmo representará mudanças síncronas, análogo a um gerador de forma de ondas.

Há também o modo de captura. Este modo é utilizado quando deseja medir quanto tempo um sinal se mantém em nível alto ou baixo. Assim, o sinal monitorado deve ser ligado na entrada via o pino ICP (do inglês, *Input Capture Pin*). O bloco de lógica de captura indica que devem ser configurados registradores que indicam como o pino será monitorado. Especifica-se se será monitorado o flanco de subida ou de descida. Deste modo, quando há a mudança de nível lógico no pino ICP, o valor de TCNTn é copiado para o registrador ICR (do inglês, *Input Capture Register*). Quando ocorrer o próximo flanco, é calculada a diferencia do novo valor de ICR com o valor anterior. Pode-se habilitar também a interrupção de captura na saída ICF (do inglês, *Input Capture Flag*). Assim é determinado o resultado de quanto tempo a entrada do pino ICP ficou em nível alto ou baixo. Valor é dado em termos de TCNTn. Apesar de disponível no ATmega328P, TIMER1, não será explorado no sistema proposto. [Zelenovsky 2019]

A Fig. 2.2 mostra um diagrama simplificado de todos registradores e seus bits anteriormente mencionados. Destacando onde acontecem as interrupções por coincidência de comparação, por estouro e demais entradas e saídas.

2.4.3 Pré-Escalonador

O pré-escalonador (do inglês, *prescaler*) é responsável por fazer a divisão da frequência do microprocessador para ser usado no acionamento dos temporizadores. A frequência de entrada é reduzida para um valor definido, compatibilizando esta frequência com as características dos circuitos e sistema almejados, normalmente mais lentos, que é o caso do sistema proposto. Alguns sinais trabalhados na área médica precisam de um relógio abaixo dos 16 MHz da CPU. O resultado da operação do pré-escalonador está conectado ao circuito dos temporizadores. Define-se $clk_{I/O}$ como a frequência base do microprocessador. Assim, o pré-escalonador irá mudar o passo do incremento do temporizador a partir de $clk_{I/O}$. O ATmega328P permite que o relógio seja dividido para o temporizador em cinco opções: 1, 8, 64, 256 e 1024. Portanto, é possível ter uma variedade de temporizadores com frequências inferiores a frequência base de 16 MHz. Sem pré-escalonador (fator igual a 1), o temporizador permanecerá contando em 16 MHz.

Para a programação do pré-escalonador é necessário o uso do registrador TCCRnB (do inglês, *Timer/ Counter Control Register*). E altera-se os bits 0, 1 e 2 (bits CS10, CS11 e CS12

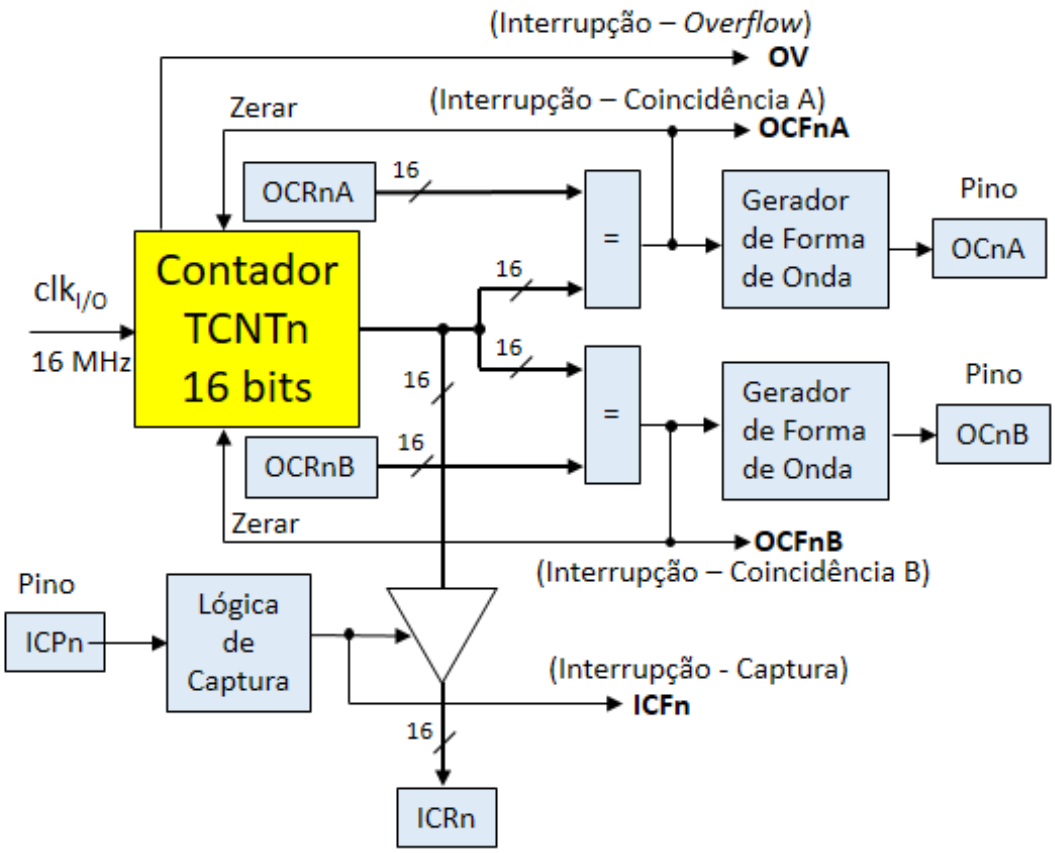

Figura 2.2: Diagrama simplificado de circuito comparador para funcionamento da temporização. Adaptado. [Zelenovsky 2019]

Tabela 2.3: Bits seletores de clock do registrador TCCRnB para configuração do pré-escalonador [Atmel Corporation 2015]

CSn2	**CSn1**	**CSn0**	**Descrição**
0	0	0	Sem fonte de clock. Temporizador desligado
0	0	1	$clk_{I/O}/1$ (sem pré-escalonador)
0	1	0	$clk_{I/O}/8$ (sem pré-escalonador)
0	1	1	$clk_{I/O}/64$ (sem pré-escalonador)
1	0	0	$clk_{I/O}/256$ (sem pré-escalonador)
1	0	1	$clk_{I/O}/1024$ (sem pré-escalonador)
1	1	0	*Clock* externo no pino Tn no flanco de descida
1	1	1	*Clock* externo no pino Tn no flanco de subida

respectivamente) deste registrador conforme indicado na tabela 2.3, onde CS vem do inglês *Clock Select* . A Fig. 2.3 também mostra um diagrama de blocos simplificado ilustrando as conexões do pré-escalonador. O resultado é definido como clk_{Tn} e é ilustrado como saída do diagrama mostrado na Fig. 2.3. Esta saída é utilizada pelos temporizadores, onde os bits mostrados na Tabela 2.3 são as entradas para seleção da saída de um multiplexador.

Agora é possível, através das informações referentes ao pré-escalador, ver o comportamento dos registradores TCNTn, OCRnx e OCFnx durante o funcionamento do microprocessador. A Fig. 2.4 mostra em forma de diagrama temporal de como é o comportamento

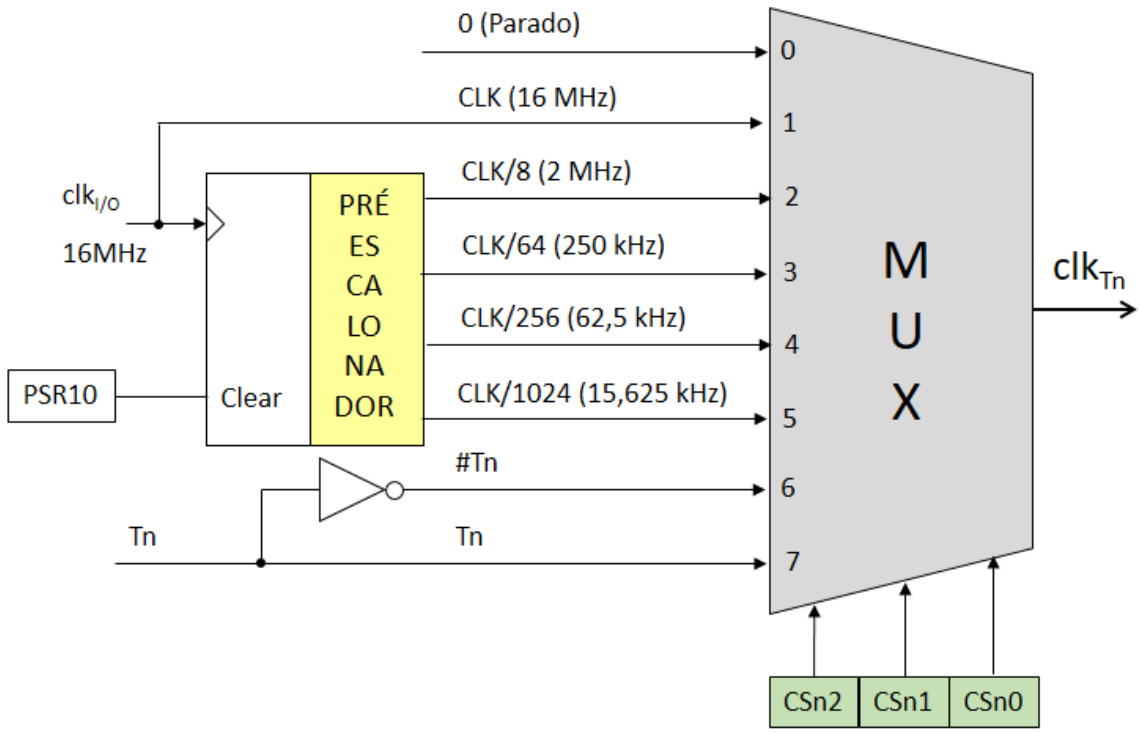

Figura 2.3: Circuito simplificado para a ilustrar a seleção de relógio divido e disponibilizado em clk_{Tn}. Adaptado. [Zelenovsky 2019]

sem um pré-escalonador. Ou seja, o registrador TCNTn incrementa a cada batida do relógio base de 16 MHz ($clk_{I/O}$). Na Fig. 2.5, o incremento ocorrerá a cada batida do relógio que é resultado do pré-escalonador, ou seja, clk_{Tn}. Nesse caso especifico, foi utilizada a divisão por oito ($clk_{Tn} = clk_{I/O}/8$). Quanto ao valor de OCRnx, não é necessária alteração pois o mesmo é programado manualmente, entretanto, o comportamento de OCFnx muda. Pode-se ver que na Fig. 2.4, a coincidência acontece a cada dois ciclos do relógio base $clk_{I/O}$. Já na 2.5, ocorre a cada dois ciclos do relógio de saída do pré-escalonador, clk_{Tn}. Que significa o mesmo que a coincidência acontecer a cada dezesseis períodos do relógio base.

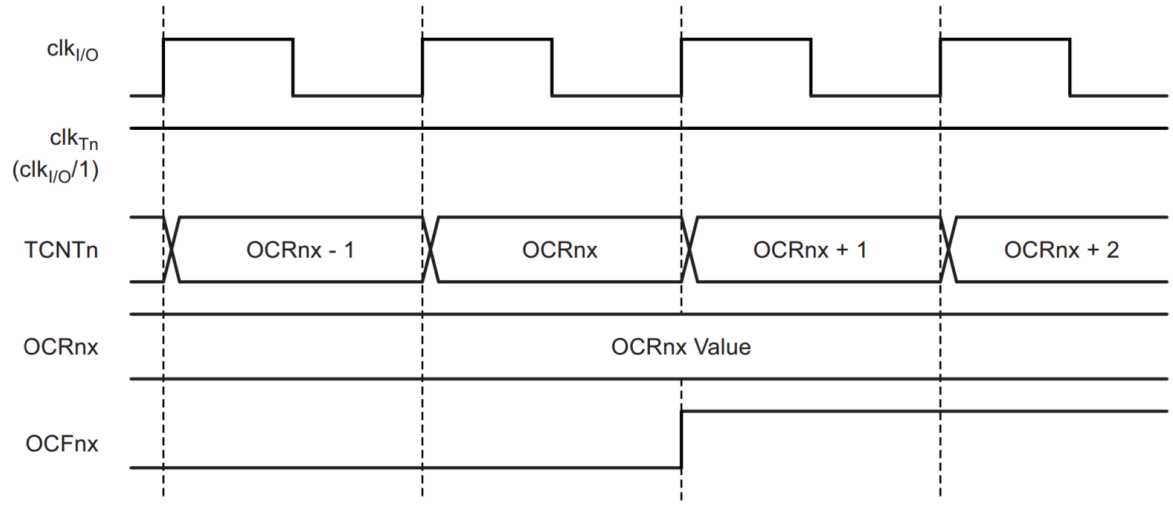

Figura 2.4: Diagrama temporal que ilustra o comportamento sem o pré-escalonador, destacando as variações dos registadores e *flags* TCNTn, OCRnx e OCFxn [Atmel Corporation 2015]

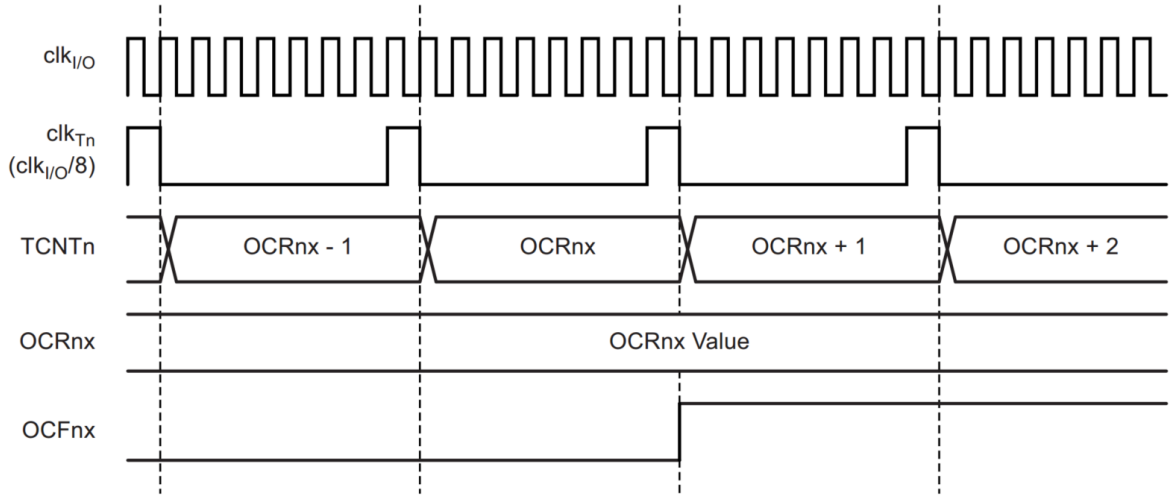

Figura 2.5: Diagrama temporal que ilustra o comportamento com o pré-escalonador com fator oito, destacando as variações dos registadores e *flags* TCNTn, OCRnx e OCFxn [Atmel Corporation 2015]

Por exemplo, vamos supor que se deseja programar o temporizador para executar uma interrupção a cada um segundo (1 Hz). Para isso usa-se a Equação 2.1. Neste exemplo é escolhido o TIMER1, ou seja, o contador e comparadores de 16 bits, onde $clk_{I/O}$ é a frequência base do microprocessador, T é o período desejado para a interrupção e N é o divisor de frequência. N pode assumir os valores 1, 8, 64, 256 ou 1024. Deve-se observar que o OCR1x é um valor válido apenas entre 0 e 65535, logo, deve ser feito o ajuste das variáveis T, $clk_{I/O}$ e N para o que intervalor de OCR1x seja respeitado.

$$OCR1x = \frac{T \times clk_{I/O}}{N} \tag{2.1}$$

Tabela 2.4: Registrador ADMUX (Registrador para a Seleção do Multiplexador do ADC) e breve descrição dos seu bits

Bit	**Nome do bit**	**Nome em inglês**	**Descrição**
7	REFS1	*Reference Selection Bits*	Bits de seleção para troca da tensão de referência
6	REFS0		
5	ADLAR	*ADC Left Adjust Result*	Justificação do ADCH e ADCL
4	-	-	Vazio
3	MUX3	*Analog Channel Selection Bits*	Seleção do canal analógico de entrada
2	MUX2		
1	MUX1		
0	MUX0		

2.4.4 Conversores Analógico-Digitais

A partir desse momento, o conversor analógico-digital será chamado de ADC por razões de simplicidade e coerência com o *datasheet* do fabricante. O ADC é escrito dessa forma devido a grafia em inglês, na forma *analog-to-digital converter*. Pressupõe-se que já exista conhecimentos prévios relacionados ao processo de amostragem e quantização de sinais. A solução de conversão neste microcontrolador envolve várias partes e vários componentes. Como por exemplo, alguns multiplexadores, um pré-escalonador próprio, pinos de referência, etc, além de alguns registradores dedicados.

O microprocessador ATmega328P possui um conversor de 10 bits, ou seja, a resolução de digitalização é de até 10 bits. Os resultados do processo de digitalização são armazenados em dois registradores, ADCH (do inglês, *ADC Data Register High Byte*) e ADCL (do inglês, *ADC Data Register Low Byte*). Como são utilizados apenas dois bits (os bits 0 e 1 do registrador ADCH), pode ser feito um deslocamento para a esquerda. E assim ter por exemplo um resultado de digitalização do ADC de oito bits. Há um bit registraor de controle do ADC que faz esta justificação. Ele é chamado de ADLAR (do inglês, *ADC Left Adjust Result*). Porém, para que este resultado esteja disponível para uso há vários passos anteriores e que serão detalhados a seguir.

Um dos registradores necessários para a configuração do ADC é o ADMUX (do inglês, ADC *Multiplexer Selection Register*). Nele é possível fazer três configurações conforme apresentado na Tabela 2.4. São elas:

- Configurações de tensão de referência no ADC;
- Configuração de alinhamento (justificação) para direita ou esquerda no resultado final, ou seja, o resultado nos registradores ADCH e ADCL (processo também é chamado de justificado a direita ou a esquerda);
- Quatro bits necessários para a seleção do canal de conversão.

Para entendimento do processo de digitalização, é necessário explorar questões relativas

à tensão de referência e aos modos disponíveis para sua configuração no Arduino Nano. A escolha da tensão de referência (V_{REF}) interferirá na resolução da medição. Pois esta tensão indicará o limite superior da faixa de conversão. Como a conversão é de 10 bits, são atribuídos valores de 0 a 1023 (0x3FF), e sendo cada nível representado por um nível de tensão diferente. Por exemplo, caso a tensão de referência for de 5 V, a diferença entre cada nível de tensão será de 5 V / 1024 = 4,88mV. Como mostrado na Tabela 2.4, os bits REFS1 e REFS0 do ADMUX fazem essa seleção. No ATmega328P há três modos de seleção de referência: utilizar o pino AREF como referência, ou seja, o uso de uma referência externa, Utilizar a referência interna AVcc, que possui 5 V dedicados a essa aplicação, ou Utilizar uma referência interna de 1,1 V. A Equação 2.2 mostra como é feita a escolha do nível de tensão. Onde $ADCn$ representa o valor em mV resultante da digitalização.

$$ADCn[mV] = \frac{{V_{REF}}^2}{V_{IN} \times 1024} \tag{2.2}$$

Ainda no registrador ADMUX é feita a escolha do canal analógico de entrada. Através dos bits MUX3, MUX2, MUX1 e MUX0. Canal este que compartilha uma entrada lógica do Arduino Nano. As oito entradas lógicas são representadas na placa com a denominação A0, A1, ... e A7. No multiplexador essas entradas são denominadas ADC1, ADC2, ... e ADC7.

Há também a preocupação com o ruído, que é inerente a um circuito alimentado e propagado por ondas eletromagnéticas. Caso o Arduino esteja sendo alimentado por um computador, este também é uma fonte de ruído. Assim, pode ser habilitado o Modo de Redução de Ruído a fim de mitigar este ruído. Para isso, o ADC deverá estar obrigatoriamente no modo de conversão único. O que implica que o ADC interrompe sua operação após uma conversão. Então cabe ao desenvolvedor habilitar e desabilitar o ADC a cada conversão.

Para as demais configurações de operação do ADC, é necessário conhecer mais dois registradores, o ADCSRA e ADCSRB (do inglês, *ADC Control and Status Register*). Ambos registradores estão resumidos nas Tabelas 2.5 e 2.6. Uma dessas configurações referem-se à forma de iniciar a conversão, que pode ser manual ou automática. Na manual deve-se configurar o bit ADEN para 0 e para o automática, o bit ADEN para 1. Dizer que o modo é automático é algo simplista, pois existem as chamadas fontes de disparo do ADC. As principais têm origem em eventos já conhecidos dos temporizadores. Como nos casos de coincidência por comparação ou por estouro do temporizador. A seleção dos modos é feita nos bits ADTS0, ADTS1 e ADTS2. Para maior detalhamento de cada fonte de interrupção, o *datasheet* do fabricante deverá ser consultado [Atmel Corporation 2015].

O bit ADIF representam uma *flag* de interrupção que é ativada ao final da conversão. Ela pode ser o início para outras ações. Por exemplo, ser a fonte de um novo disparo. Assim, o ADC fará conversões de forma sucessiva. Esse modo é chamado de modo de conversão contínua (do inglês, *free running*, que diverge do modo de conversão única, que é o modo onde o conversor encerra sua operação ao final da conversão. Para que se possa usar o bit ADIF, o bit ADIE (do inglês, ADC *Interrupt Enable*) deverá ser habilitado. O bit ADIE,

Tabela 2.5: Registrador ADCSRA (Registrador A de Controle e Estado do ADC) e breve descrição dos seus bits.

Bit	Nome do bit	Nome em inglês	Descrição
7	ADEN	*ADC Enable*	Habilita e desabilita ADC
6	ADSC	*ADC Start Convertion*	Inicia uma conversão
5	ADATE	*ADC Auto Trigger Enable*	Auto disparado do ADC no próximo flanco de subida ou disparo manual
4	ADIF	*ADC Interrupt Flag*	Flag de interrupção ao final da conversão
3	ADIE	*ADC Interrupt Enable*	Habilita uso da interrupção do ADC
2	ADPS2	*ADC Prescaler Select Bits*	Configuração da frequência do relógio do ADC clk_{ADC}
1	ADPS1		
0	ADPS0		

Tabela 2.6: Registrador ADCSRB (Registrador B de Controle e Estado do ADC) e breve descrição dos seus bits.

Bit	Nome do bit	Nome em inglês	Descrição
7	-	-	Vazio
6	ACME	*Analog Comparator Multiplexer Enable*	Bit para operação do comparador análogico. Não aplica ao ADC
5	-	-	Vazio
4	-	-	Vazio
3	-	-	Vazio
2	ADTS2	*ADC Auto Trigger Source*	Seleção a fonte do disparo automático ADC.
1	ADTS1		
0	ADTS0		

quando possui o valor 1, habilitará a o uso de interrupções do ADC, e assim o bit ADIF poderá ser usado.

Outro ponto importante é a frequência do relógio do ADC, clk_{ADC}. As conversões não acontecem na mesma frequência do *clock* base do processador ($clk_{I/O}$). O aumento demasiado da frequência de conversão pode fazer com que o resultado perca em resolução. Quanto maior a resolução desejada, menor deve ser a frequência clk_{ADC} a ser empregada. Para determinar essa frequência é necessário levar em consideração qual o requisito do circuito de aproximações sucessivas do ADC. No caso do ATmega328P, o fabricante recomenda valores de 50 kHz até 200 kHz. Entretanto, de acordo com [Zelenovsky 2019], de forma experimental, utilizar um relógio do ADC com frequências superiores a 200 kHz não gera um erro intolerável. De qualquer forma, para que seja mantida a premissa do fabricante, há a apenas uma opção disponível. Como o relógio base do processador é de 16 MHz, e a opções do divisor N do pré-escalonador do ADC são 2, 4, 8, 16, 32, 64 e 128, logo, o único divisor que reduz o clk_{ADC} para o intervalo indicado pelo fabricante é o 128. Assim, é obtido o valor de 125 KHz conforme a Equação 2.3. Com isso, o período de conversão é de 8 μs. Como a conversão dura 13 ciclos do relógio do ADC (clk_{ADC}), a taxa de conversão máxima a ser alcançada é de 9600 amostras por segundo. As configurações deverão ser feitas alterando

os valores dos bits ADPS2, ADPS1 e ADPS0 (do inglês, ADC *Prescaler Select Bits*) do registrador ADCSRA.

$$clk_{ADC} = \frac{clk_{I/O}}{N} = \frac{16 \times 10^6}{128} = 125 \text{ KHz} \tag{2.3}$$

A Fig. 2.6 resume em forma de diagrama de blocos o funcionamento do ADC. Iniciando pelo bit ADEN (do inglês, ADC *Enable*), onde é feita a habilitação do conversor analógico-digital. Outra configuração que deverá ser feita é a no bit ADATE (do inglês, ADC *Auto Trigger Enable*). O ADEN serve justamente para que os circuitos do conversor sejam ligados. Como é desejável fazer a conversão no tempo determinado e com o menos ruído possível, então papel de habilitar e desabilitar o ADC torna isso viável. O bit ADATE refere-se a fonte de disparo. Ressalta-se que a escolha da fonte de disparo não diz respeito ao modo de conversão (contínuo ou simples), e sim a qual evento que será o gatilho para início da conversão. Há uma grande variedade de fontes disponíveis, como por exemplo interrupções externas ou interrupções com origem no temporizador. O disparo pode ter origem em uma interrupção por comparação por exemplo. O ATmega328P possui oito opções de fontes de disparo que são selecionadas pelos bits ADTS2, ADTS1 e ADTS0. Para maiores detalhes de cada um, o *datasheet* do fabricante pode ser consultado. Então de acordo com a Fig. 2.6, existindo um flaco de subida e o bit ADATE em nível alto, o resultado é verdeiro e é iniciada uma conversão. Esse é o chamado modo automático. Como se vê na Fig. 2.6, ainda há uma porta lógica OU em seguida. Em que uma das entradas é o bit ADSC (do ingês, ADC *Start Conversion*). Este bit é que de fato pode iniciar manualmente uma conversão imediatamente, pois não depende de fontes de disparo de conversão. Então, quando o objetivo for utilizar utilizar um disparo manual, o bit ADSC deverá ter nível alto. Ao final da conversão, o bit é colocado automaticamente em nível baixo justamente como sinalização de fim da conversão.

O conversor possui várias formas de operação e variações dentro deles. Será explicado mais adiante quais componentes foram utilizadso e como cada componente foi programado no sistema.

2.4.5 Comunicação Serial

Há duas formas primordiais de envio de dados entre um transmissor e um receptor: uma comunicação em série e uma comunicação em paralelo. Em uma comunicação em paralelo, é necessário duas linhas extras para cada linha de comunicação paralela adicional, uma para transmissão e outra para recepção ou bidirecional para transmissão e recepção na mesma linha, e a outra para sincronia. Então há uma relação de compromisso entre a robustez do hardware (quantidade de linhas) e o custo do projeto. Uma vez que mais linhas, no geral, significa um projeto mais caro e menos miniaturizado. Historicamente a comunicação em série é explorada ao máximo. E a evolução desse tipo de comunicação levou a protocolos utilizados massivamente nos dias atuais, como por exemplo USB 2.0 e 3.0 (do inglês, *Uni-*

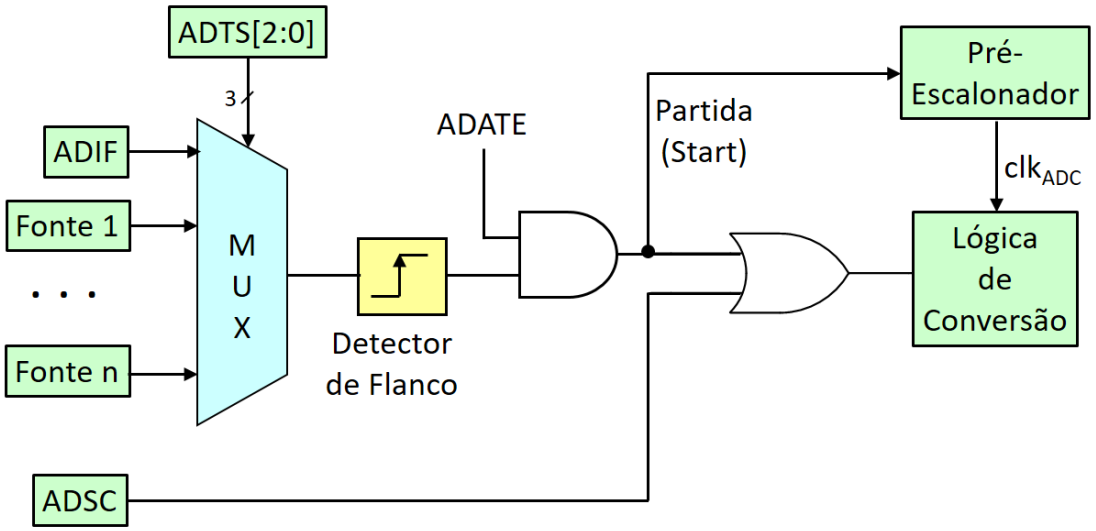

Figura 2.6: Diagrama de blocos parte do funcionamento do ADC, em destaque as forma de disparo de conversões. Adaptado. [Zelenovsky 2019]

versal Serial Bus), que em sua atualização para o modo USB 3.2 geração 2 tem potencial teórico de taxas de até 20 Gbit/s [Sanders 2017].

Outro exemplo é o PCI-e (do inglês, *Peripheral Component Interconnect*). Utilizado hoje como padrão de periféricos em microcomputadores, como por exemplo, placas de vídeo de alto desempenho. Também há o padrão SATA (do inglês, *Serial Advanced Technology Attachment*), usado em larga escala para conectar discos rígidos a placas-mãe de computadores, e também presente em *datacenters*. Outros protocolo mais simples como o I2C (do inglês, *Inter-Integrated Circuit*) fazem comunicação com apenas dois fios e de forma síncrona e o SPI (do inglês, *Serial Peripheral Interface*). Alguns desses protocolos fazer uso de mais linhas seriais para ter uma comunicação em paralela, entretanto, os fundamentos téoricos de uma comunicação serial continuam sendo aplicados. Já o Arduino Nano utiliza o protocolo USART (do inglês, *Universal Synchronous and Ssynchronous Receiver-Transmitter*). Ele também pode ser chamado de UART para quando o desejo for referir a ele como apenas a versão assíncrona do protocolo. Dado que o Arduino suporta este protocolo, o detalhamento das principais funções e quais que serão utilizados no sistema serão dadas mais adiante.

Os protocolos de comunicação serial trabalham de duas formas diferentes. Podemos dividir os protocolos em síncronos em assíncronos. Os protocolos síncronos, em sua forma mais simples, possuem três linhas. Uma para dados, uma para o relógio e outra para tensão de referência (terra). A transmissão é feita de forma invertida, ou seja, o bit menos significativo é enviado primeiro e logo em seguida os mais significativos. Já o relógio marca o envio de um bit. Podendo esta marcação ser no flanco de subida ou de descida. Quanto aos protocolos assíncronos, a sua forma mais simples também possui no mínimo três linhas: transmissão, recepção e tensão de referência. Sem a marcação através de um relógio. Para que mesmo

em modo assíncrono os dados consigam ser identificados pelo receptor é feito uso de outros mecanismos. O presente trabalho fará um maior detalhamento do USART em seu modo assíncrono devido ao seu uso ser mais presente em aplicações desse protocolo.

Um dos mecanismos utilizados na abordagem assíncrona é adotar uma taxa de transmissão comum entre transmissor e receptor. O transmissor e receptor devem combinar a taxa e o número de bits a ser enviado e recebido. Por exemplo, 9600 $baud$. Aqui cabe uma ressalva acerca do uso da unidade baud, que refere-se a símbolos por segundo. Se tratando de código de linha binários, e do envio de um bit a cada janela de tempo pré-determinado, o número de baud e bits por segundo são equivalentes. Outros dois mecanismos utilizados na transmissão assíncrona é a presença do bit de partida (do inglês, *start bit*) e do bit de parada, (do inglês, *stop bit*). Para entendimento desses dois mecanismos, é preciso ter em mente que a linha de transmissão quando ociosa, permanece em nível alto.

Para o bit de partida, o funcionamento é descrito a seguir. A linha responsável pela transmissão inicia em nível alto devida sua ociosidade. Quando há informações a serem enviadas, a linha é colocada em nível baixo por um ciclo de transmissão serial. Logo em seguida é enviado o primeiro bit contendo informação. Assim, para o receptor, é possível saber quando de fato começa a chegar os dados. Quanto ao bit de parada, este é adicionado após o último bit de informação a ser enviado. A linha permanece em nível alto por um ciclo (1/baud). Dado este evento há duas situações possíveis. A primeira é que se nada mais tiver que ser enviado, a linha permanece em nível alto. A segunda é que caso existirem mais dados para serem enviados, a diferença de níveis auxiliará a detecção dos bits de parada e de partida, que virão de forma sucessiva. Alguns protocolos usam mais de um bit de parada. Ressalta-se que a transmissão é orientada a *bytes*, onde os bits de cada *byte* são enviados sequencialmente. [Zelenovsky 2019]. Já o mecanismo de paridade funciona como um código detector de erros. O objetivo ao utilizar a opção com bit de paridade é detectar o quantidade de bits em nível alto. Seja ele par ou ímpar. Então, em caso de discrepância entre os valores, o trem de bits em questão deverá ser descartado. Entretanto, deve-se ressaltar que podem existir casos onde dois bits tiverem seus valores trocados, a paridade não funcionará como esperado. Pois mesmo com o erro, o trem de bits em questão permanecerá com a contagem de bits em nível alto original.

Há também a diferenciação por envio simultâneo de informações. São chamadas as transmissões *half-duplex* e *full-duplex*. No caso da *half-duplex*, seu uso é feito em sistema com uma linha de transmissão de dados apenas. Da mesma forma que a comunicação síncrona. Uma linha é responsável pela transmissão de dados e outra pelo relógio. E a transmissão e recepção simultânea não é possível. No sistema *full-duplex* há a presença de uma linha dedicada a transmissão e outra para recepção. Logo, a transmissão e recepção podem ser feitas de forma simultânea. Sendo mais adequada a transmissões assíncronas.

Na placa Arduino Nano, são usado os pinos RX0 e TX1 para a comunicação serial. Há os dois modos de operação (síncrono e assíncrono). Esses mesmos pinos são utilizados para a comunicação da IDE com a placa. Ou seja, em caso de uso da IDE juntamente com

uma aplicação que use a porta serial, cada operação deverá ser feita separadamente. Entre as várias placas da família Arduino, a principal diferença está na quantidade de interfaces seriais. Entretanto, os fundamentos teóricos ainda aplicam.

De posse dessa caracterização de uma comunicação serial genérica, vamos detalhar a comunicação serial do Arduino Nano. Para o Arduino Nano em específico e o microprocessador ATmega328P há apenas um controlador serial. Ou seja, é possível ter apenas um enlace de comunicação serial no dispositivo, e é feito uso do protocolo USART. A notação a ser usada é generalista e faz uso do nome do registrador acompanhado por um número inciado pela letra n. Como há apenas uma interface serial no Arduino Nano, n indicará neste caso o valor zero. O Arduino faz uso do protocolo USART. A seguir estão as principais características.

A primeira é referente a transmissão de dados. O USART do AVR usa uma comunicação *full-duplex*, que é uma comunicação nos dois sentidos simultaneamente. Quanto às questões relacionadas ao sincronismo, o AVR é bem versátil. Há quatro modos de operação no ATmega328P. Dois síncronos e dois assíncronos. A assíncrono normal (do inglês, *asynchronous normal mode*); o assíncrono com dupla velocidade (do inglês, *asynchronous double speed mode*); o mestre síncrono (do inglês, *synchronous master mode*), onde o Arduino Nano gera o relógio, e o escravo síncrono (do inglês, *synchronous slave mode*), onde o Arduino Nano lê o relógio.

Todos os modos trabalham com variações de componentes responsáveis pelo relógio e pelo sincronismo das transmissões. Para a escolha do modo, usam-se os bits UMSEL (do inglês, USART *Mode Select*) do registrador UCSRnC (do inglês, USART *Control and Status Register*). Para o entendimento do funcionamento da comunicação serial é necessário entendimento de configurações feitas pelos bits de quatro registradores diferentes: UDRn (do inglês, USART *I/O Data Registrer*), UCSRnA, UCSRnB e UCSRnC. Os registradores UCSRnA, UCSRnB e UCSRnC, e suas breves descrições estão disponíveis nas Tabelas 2.7, 2.8 e 2.9.

Como há vários componentes responsáveis pela transmissão e recepção, assim como registradores, serão citados e detalhados aqui apenas os principais e imprescindíveis para a transmissão. A notação é a mesma usada no *datasheet* do fabricante. O detalhamento completo também pode ser visto no manual do fabricante. Iniciamos por um dos principais registradores: o UDRn. Ele serve para o envio e recepção de dados da porta serial, e pode ser caracterizados por *buffers*. É o conteúdo desse registrador que é encaminhado ao pino TxDn para transmissão. Na recepção, o que é lido no pino RxDn é inserido nesse registrador. Este registrador trabalha em conjunto com o bit UDREn (do inglês, USART *Data Register Empty*) que funciona da seguinte forma. O bit UDREn permanece em nível alto enquanto não há transmissão de dados em andamento e em nível baixo quando o UDRn possui dados a serem transmitidos. Funcionamento parecido com os bits RXCn e TXCn, que sinalizam que os registradores estão de fato ocupados com dados. Ou seja, que ainda há conteúdo a ser lido ou que haja conteúdo a ser enviado. Os bits RXCn, TXCn e UDRE fazem parte do

Tabela 2.7: Registrador UCSRnA (Registrador de Estado e Controle da USART A) e breve descrição dos seus bits.

Bit	Nome do bit	Nome em inglês	Descrição
7	RXCn	*USART Receive Complete*	Sinaliza que ainda há (ou não) conteúdo no buffer a ser lido. Pode ser configurado com interrupção para sinalizar fim da transmissão.
6	TXCn	*USART Transmit Complete*	Sinaliza que ainda há (ou não) conteúdo no buffer a ser enviado.
5	UDREn	*USART Data Register Empty*	Sinaliza que o buffer do transmissor está pronto para receber mais dados.
4	FEn	*Frame Error*	Sinaliza que erro no *frame* ao verificar se o bit de parada subsequente é zero.
3	DORn	*Data OverRun*	Válido quando o buffer de recepção estiver cheio e um novo bit é recebido.
2	UPEn	*USART Parity Error*	Indica erro de paridade do buffer de recepção.
1	U2X	*Double the USART Transmisson Speed*	Indica a dobra de velocidade de transmissão
0	MPCMn	*Multi-Processor Communication Mode*	Habita comunicação multiprocessador. Assim, todos os frames recebidos terão que ser endereçados.

Tabela 2.8: Registrador UCSRnB (Registrador de Estado e Controle da USART B) e breve descrição dos seus bits.

Bit	Nome do bit	Nome em inglês	Descrição
7	RXCIEn	*RX Complete Interrupt Enable n*	Sinaliza fim da recepção e dispara interrupção.
6	TXCIEn	*TX Complete Interrupt Enable n*	Sinaliza fim da transmissão e dispara interrupção.
5	UDRIEn	*USART Data Register Empty Interrupt Enable n*	Sinalizar se registrador UDERn está vazio e dispara interupção.
4	RXEn	*Receiver Enable n*	Habilita e desabilita o receptor USART.
3	TXEn	*Transmiter Enable n*	Habilita e desabilita o transmissor USART.
2	UCSZn2	*Character Size n*	UCSZn0, 1 e 2 definem trabalho do quadro a ser recebido ou enviado.
1	RXB8n	*Receive Data Bit 8 n*	Nono bit de recepção
0	TXB8n	*Transmite Data Bit 8 n*	Nono bit de transmissão

Tabela 2.9: Registrador UCSRnC (Registrador de Estado e Controle da USART C) e breve descrição dos seus bits.

Bit	Nome do bit	Nome em inglês	Descrição
7	UMSELn1	*USART Mode Select*	Selecionam o modo de operação (sincrono e assíncrono)
6	UMSELn0	*USART Mode Select*	
5	UPMn1	*Parity Mode*	Seleciona o modo de paridade
4	UPMn0	*Parity Mode*	
3	USBSn	*Stop Bit Select*	Informa a quantidade de bits de parada
2	UCSZn1	*Character Size n*	UCSZn0, 1 e 2 definem tamanho do quadro a ser recebido ou enviado.
1	UCSZn0	*Character Size n*	UCSZn0, 1 e 2 definem tamanho do quadro a ser recebido ou enviado.
0	UCPOLn	*Clock Polarity*	Define o flanco que o dado será amostrado no modo síncrono

registrador UCSRnA.

A principal entrada dos circuitos responsáveis pela comunicação serial é o relógio base ($clk_{I/O}$). Como visto em seções anteriores, o relógio base opera com o oscilador de 16 MHz. Este valor é disponibilizado no pino XTAL (do inglês, *Chip Clock Oscillator*). Obviamente, a transmissão de dados não é feita nessa taxa, o que ocuparia todos os ciclos do relógio. Há o registrador de *baud rate* UBRRnH e UBRRnL (inglês, USART *baud rate registers*). Os dois registradores combinados possuem 12 bits destinados para a seleção do valor, e o pré-escalonador irá fornecer a taxa desejada. Utilizando do modo assíncrono normal, os valores a serem selecionadas seguem a relação da Equação 2.4. Onde D é o *baud rate*, $clk_{I/O}$ é o relógio base e o UBBRn registrador de *baud rate* e X é o dobrador de taxa de transmissão. O dobrador é definido pelo bit U2Xn (do inglês, *Double the USART Transmisson Speed*) do registrador UCSRnA. O dobrador X pode assumir os números 2 (modo mestre síncrono), 8 (modo assíncrono de dupla velocidade) e 16 (assíncrono normal). Ou seja, quando menor X no denominador, maior a taxa de transmissão.

$$D = \frac{clk_{I/O}}{X \times (UBRRn + 1)} \tag{2.4}$$

Outro ponto a ser definido são as opções de formatação de dados. Pois caso transmissor e receptor não estejam igualmente configurados, a comunicação de dados não funcionará. Obviamente, se igualmente configurados, já é uma confirmação que ambos equipamentos são compatíveis. Para isso é necessário fazer primeiro a configuração dos bits UCSZn2, UCSZn1 e UCSZn0. A combinação desses bits trará a informação se o quadro terá 5, 6, 7, 8, ou 9 bits de cumprimento. Em geral usa-se o comprimento de 8 bits, justamente porque é o equivalente a um *byte*. Outra configuração necessária é a configuração de bit de paridade. Há três opções: paridade ímpar, paridade par e sem paridade. O uso do bit de paridade é opcional. A configuração das opções de bits de paridade são feitas nos bits UPMn1 e

UPMn0 (do inglês, *Parity Mode*), do registrador UCSRnC. Outra configuração refere-se a quantidade de bits parada. As opções disponíveis são um e dois bits. O uso do bit de parada é obrigatório. A configuração dos bits de parada é feita no bit USBSn, (do inglês, *Stop Bit Select*), também no registrador UCSRnC. A Figura 2.7 resume essas características.

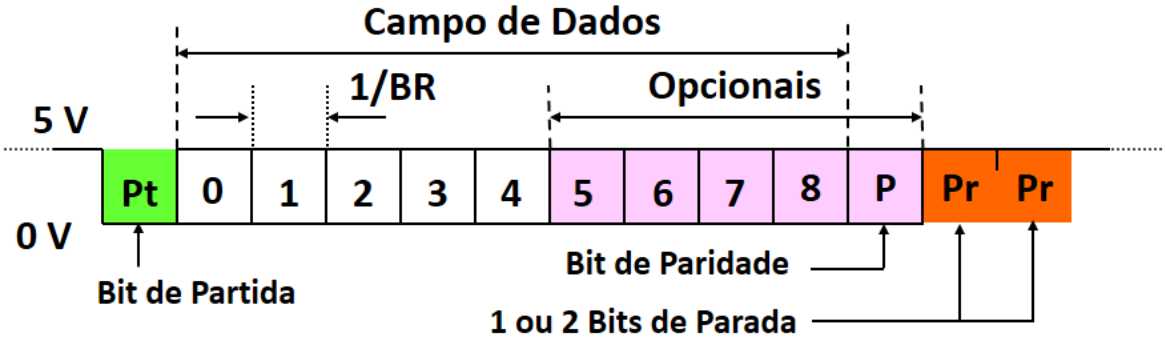

Figura 2.7: Trem de bits com todos bits possíveis para comunicação serial. [Zelenovsky 2019]

A comunicação serial do Arduino Nano também possui suporte a interrupções. Elas funcionam como resultados de três bits específicos: RXCIEn, TXCIEn e UDRIEn. Os bits estão descritos na Tabela 2.8. É possível ter interrupções ao final de uma recepção, ao final de transmissão e após o esvaziamento do *buffer* de transmissão. Cada interrupção acontece quando o respectivo bit vai para nível alto. Outro mecanismo importante do Arduino Nano é a capacidade de armazenar o equivalente a dois *bytes* no *buffer* de recepção, e também mais um *byte* no chamado registrador de descolamento. O registrador de descolamento é responsável por armazenar essas informações em uma espécie de fila. Quando o registrador de recepção UDRn estiver cheio, o registrador de descolamento pode ter uma informação à disposição. Assim, essa informação é copiada para o UDRn.

Entretanto, eventos na transmissão podem acontecer e a informações não serem processadas da forma explicada anteriormente. Para isso, existem três bits de controle de integridade de recepção. O controle de recepção é feito no momento que a informação é escrita é registrador UDRn. Então, antes mesmo do UDRn ser copiado para o registrador de destino, já é sabido se aquela informação possui algum comportamento inesperado. Os bits responsáveis pelo controle são: FEn, DORn e UPEn (do inglês, *Frame Error*, *Data Overrun* e USART *Parity Error* respectivamente), todos no registrador UCSRnA. O FEn refere-se a um erro de quadro, onde o bit de parada (nível alto) não é encontrado. A importância desse tipo de verificação se torna ainda mais importante quando é recebido vários *bytes* onde a representação da informação são bits 0. Ou seja, a única forma de manter a sincronização é pela leitura do bit de parada. Caso essa leitura não aconteça, a *flag* FEn é colocada em nível alto. O DORn refere-se a um erro de atropelamento de dados quando o *buffer* de recepção e o registrador de deslocamentos já estão ocupados. Caso esse cenário se concretize e for detectado um bit de partida, o dado mais antigo será descartado e a *flag* DORn irá para o nível alto. Já o bit UPEn refere-se ao caso onde o bit de paridade indicou uma inconsistência. Logo, o conteúdo do *buffer* que será lido no passo seguinte será descartado. Ressalta-se que estas

três *flags* também possuem vetores de interrupção associados.

2.5 Conclusão

Neste ponto já é possível observar as características principais do Arduino Nano. Em cada seção foram observados componentes e registradores que serão necessários para desenvolvimento do sistema. O temporizador, o ADC e a porta serial trabalharão de forma integrada. No Capítulo 4 será mostrado como essas características serão utilizadas. Entretanto, ainda resta uma tecnologia e um componente restante para que todo o embasamento teórico esteja completo. Pois uma vez coletado o dado no tempo determinado, convertido digitalmente, e disponibilizado para transmissão, este deve chegar até o receptor. Para isso é utilizada a tecnologia *Bluetooth* através de hardware compatível. Ambos serão detalhados no próximo capítulo.

Capítulo 3

Tecnologia Bluetooth

3.1 Introdução

Durante a concepção do sistema foi necessário fazer a escolha da forma de transmissão de informações. O caminho que vai da origem (dado coletado) ao destino (computador que processa os dados). A escolha da solução depende do problema a ser resolvido. Como objetivo é criar um sistema para aquisição de dados médicos, sejam eles quais quer, já é possível tirar algumas premissas a partir disso. Como por exemplo, que equipamentos médicos podem exigir certa mobilidade, e que a geração de dados e transmissão não exigem taxas muito elevadas para suprir o sistema. Outro ponto de observação é que a tecnologia deveria ser compatível com o Arduino Nano. Este possui uma vantagem que oferecer componentes modulares que podem ser adquiridos separadamente. Podendo ser adquiridos separadamente. Como por exemplo os chamados *shields*, que são placas construídas sob medida para determinado membro da família Arduino. Com tudo isso em consideração, a escolha foi o um módulo para a criação de um enlace Bluetooth. A seguir há um breve resumo teórico da tecnologia, sua pilha de protocolos, características e limitações.

3.2 Histórico

Baixo consumo energético, transmissão sem fio e com a maior distância possível entre transmissor receptor. Esse foi o desafio de pesquisadores ao longo da década de oitenta e noventa. Os primeiros registros são de patentes realizadas em 1989, com uma tentativa de fazer fones de ouvido sem fio. Entretanto, só em 1998 foi criado o Bluetooth SIG (do inglês, Bluetooth *Special Interest Group*). Iniciativa formada pelas empresas Ericsson, Intel, IBM, Nokia e Toshiba, onde seus departamentos de pesquisas tiveram seus esforços coordenados para atingir o objetivo de ter a tecnologia em forma de produto. Em 2001 ocorreu o primeiro lançamento de um telefone com a tecnologia e o primeiro laptop: o Ericsson T39 e o IBM ThinkPad A30. Durante o desenvolvimento da tecnologia Bluetooth o IEEE (*Institute*

of Electrical and Electronics Engineers), que é uma instituição sem fins lucrativos, assumiu a padronização do Bluetooth, fazendo com que o padrão atingisse mais fabricantes, retirando o viés lucrativo durante o desenvolvimento. Entretanto, foram apenas duas revisões do protocolo sob responsabilidade do IEEE. Uma em 2002 [IEEE 2002] e outra em 2005 [IEEE 2005]. Hoje, a responsabilidade da padronização retornou ao SIG, que possui mais de 35000 empresas que representam a SIG e sua padronização.

3.3 A tecnologia

O desenvolvimento da tecnologia Bluetooth obteve sucesso devido as suas premissas. Sendo assim, algumas tecnologias como infra-vermelho, onde é necessária a proximidade de dois dispositivos, se tornou quase que obsoleta. Outras tecnologias, como o Wi-Fi, permaneceram coexistindo. O Bluetooth se difere do Wi-Fi em vários aspectos, como por exemplo em potência transmissão e recepção. É mais factível a comparação entre Ethernet (IEEE 802.3) e o Wi-Fi (IEEE 802.11). Sendo o Wi-Fi uma forma adicional de acesso a uma rede local e o Bluetooth para aplicações variadas. No Bluetooth não é necessário configurar endereços ou permissões. Diferente de uma rede local. Entretanto, é necessário *hardware* dedicado - adaptador Bluetooth - para que o acesso seja realizado. O sistema usa umas das faixas não licenciadas do espectro. A chamada banda ISM (do inglês, *Industrial, Scientific and Medical*). Dentro dessa faixa, houve uma separação em setenta e nove canais Bluetooth de 1 MHz cada. Esse arranjo foi alterado com a especificação do Bluetooth 4.0, onde foram implementados quarenta canais de 2 MHz cada. Desde a especificação Bluetooth 1.2 [IEEE 2005] é usado o AFH (do inglês, *frequency-hopping spread spectrum* , ou seja, o módulo Bluetooth é capaz de alternar a transmissão entre seus canais na faixa ISM para enviar informação nos canais com menos ruídos e com menos interferências.

O protocolo se tornou o padrão de mercado para comunicação sem-fio a curto alcance, com destaque para dispositivos móveis. Podem-se citar como aplicações o próprio *smartphone*, fones de ouvido, controles de *video game*, transferência de arquivos, dispositivos de I/O variados para conectar a um computador. Também há algumas aplicações industriais em sua versão com maior uso de potência.

3.4 Especificações

A tecnologia Bluetooth possui sua própria pilha de protocolos, diferente do modelo OSI (do inglês, do inglês *Open System Interconnection*) por exemplo, o o Bluetooth representa um conjunto de tecnologias e protocolos. Também há algumas divisões de especificações, que facilitam o entendimento da tecnologia. Pode ser feita por exemplo uma divisão de classes. Existem as classes 1, 1.5, 2, 3 e 4. Esta divisão diz respeito a potência máxima transmitida e por conseguinte, a distância máxima de transmissão em um enlace com dispositivos

dessa classe. Por exemplo, dispositivos da classe 2, a mais comum em dispositivos móveis, podem transmitir em até 2.5 mW (4 dBm) e possuem alcance em um ambiente *indoor* de aproximadamente até 10 metros.

Outra divisão que pode ser feita é através de suas especificações. Que são Bluetooth: 1.0; 1.0B; 1.1; 1.2; 2.0 + EDR (do inglês, *Enhanced Data Rate*) ; 2.1 + EDR; 3.0 + HS (do inglês, *high speed*); 4.0 (também chamado de Bluetooth *Low Energy*; 4.1; 5.0 e 5.1. A cada nova especificação é feita alguma alteração ou inclusão de serviços, melhoria de algoritmos, inserção de novos códigos e atualizações. Por exemplo, na especificação 4.0, que foi um marco para o uso diferente do espectro, onde passou-se a utilizar canais de 2 MHz ao invés de 1 MHz. Outro exemplo é na especificação 5.0. Onde flexibilizou-se o uso de potência máxima para atingir grandes distâncias em algumas circunstancias. Assim, a especificação 5.0 possui um limite máximo teórico para ambiente *indoor* de até 40 metros.

Quanto aos protocolos, a Fig. 3.1 resume quais são protocolos e como estão relacionados. Não é o objetivo deste trabalho esgotar o assunto Bluetooth e suas características. Assim, será dada uma visão geral sobre os protocolos do padrão Bluetooth e mais adiante, com a mesma abordagem, sobre os perfis Bluetooth. Os protocolos Bluetooth estão divididos em dois grandes grupos: o *controller stack* (pilha da controladora) e *host stack* (pilha do hospedeiro). Os protocolos desses grupos têm objetivos similares. A pilha da controladora lida com os processos relacionados ao meio de transmissão, rádio, e etc. Enquanto a pilha do hospedeiro lida com a lógica de transmissão de dados e suas regras. Como por exemplo a protocolo LMP (do inglês, *Link Management Protocol*), onde é feito o estabelecimento de sessão de entre os rádios dos dispositivos, ajustes de potência de transmissão, entre outros. Um exemplo da pilha do hospedeiro é o protocolo RFCOMM (do inglês, *Radio Frequency Communication*). Onde é simulada uma porta serial, utilizando as regras do protocolo RS-232. Assim, é criada uma abstração onde é feito o transporte de informações via meio sem-fio de forma relativamente confiável. Devido a estruturas como bits de paridade em suas regras. A presença desses protocolo é primordial para a aplicação do Bluetooth ao sistema de adquisição de dados proposto. Entretanto, é possível verificar que há suporte em alguns protocolos da Internet como o TCP e UDP, e também protocolos para transferência de dados, entre outros.

Quanto aos perfis Bluetooth, também há várias destes presentes. O perfil Bluetooth pode ser visto de forma análoga ao serviço suportado. Então o perfil descreve quais e como cada protocolo será usado. Assim é possível criar soluções que usem os protocolos Bluetooth de forma padrão. Isso proporciona a construção de novas aplicações. Há perfis para impressão de arquivos, para dispositivos de saúde vestíveis, para criação de redes locais, para microfones sem fio e etc. Um destes perfis utilizados massivamente nos dias atuais é o perfil A2DP (do inglês, *Advanced Audio Distribution Profile*), que é uma presença unânime em *smartphones* com suporte a Bluetooth. Este perfil especifica a transferência de áudio para outro dispositivo. Isso pode ser um fone de ouvido, o aparelho de som de um carro, etc. Há inclusive o suporte para o som estéreo. Ou seja, dois canais de áudio são transferidos para

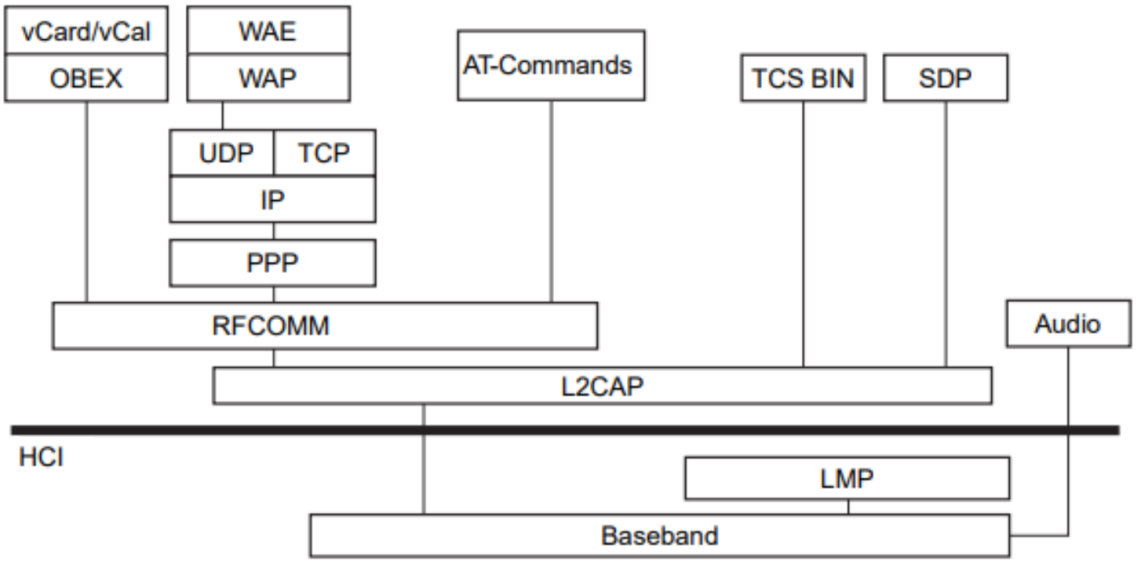

Figura 3.1: Protocolos da tecnologia Bluetooth. Adaptado [Sayali 2015]

exibição simultânea. Um serviço que também é imprescíndivel para o sistema proposto é o SPP (do inglês, *Serial Port Profile*), que faz uso do protocolo RFCOMM, que possui a lógica programada para criar uma porta virtual serial e conectar dois dispositivos que possuem o mesmo serviço suportado. O perfil SPP funciona como uma camada de abstração fazendo com que regras do meio cabeado sejam aplicadas ao meio sem-fio.

As Figs. 3.2 e 3.3 mostram exemplos de dispositivos reais e seus serviços suportados. As imagens foram obtidas em um computador com sistema operacional Windows onde os dispositivos com suporte Bluetooth foram conectados. Na Fig. 3.2 há o módulo Bluetooth HC-05, que é compatível com o Arduino e suporta o perfil SPP. Já na Fig. 3.3 é um *smartphone* Samsung J5 Pro, que possui o perfil A2DP como suportado.

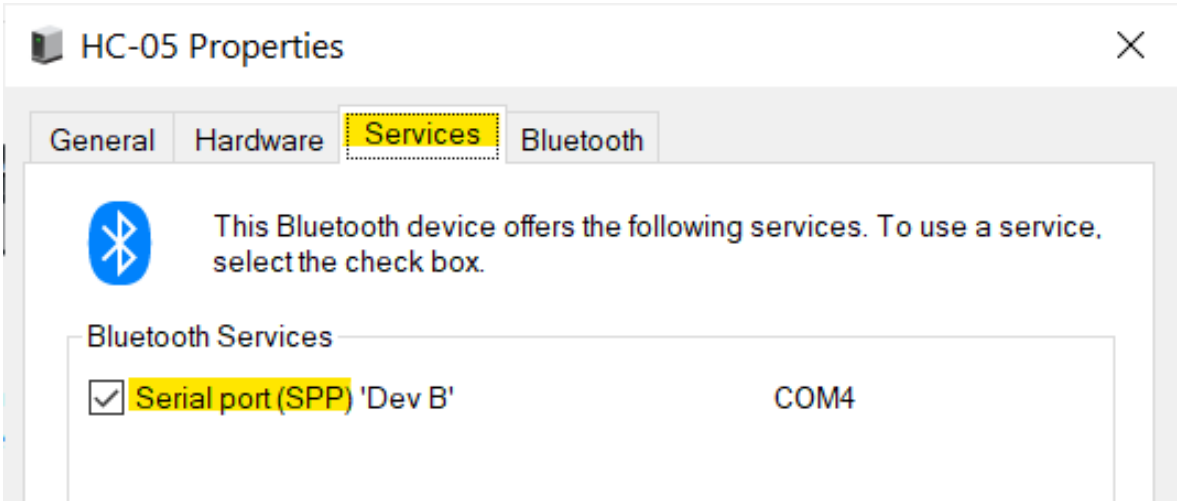

Figura 3.2: Lista de perfis suportados em módulo Bluetooth HC-05.

Para a conexão de pelo menos dois dispositivos com suporte a tecnologia Bluetooth é necessário que um dos dispositivo seja descoberto pelo outro. Para isso existe a classificação

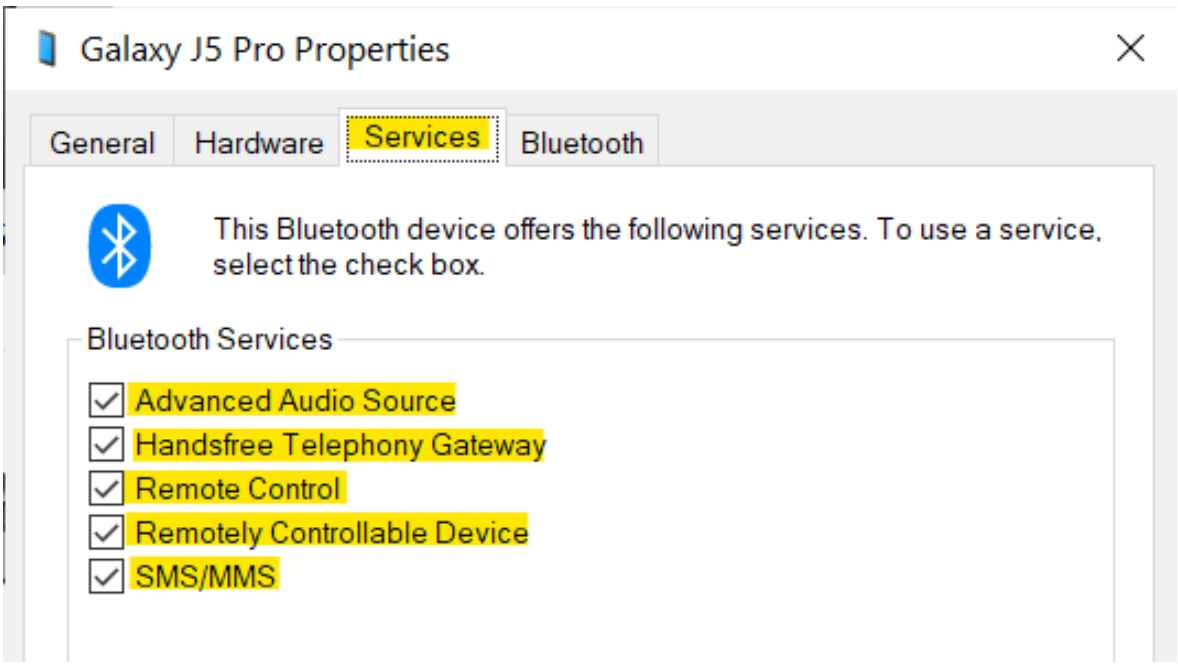

Figura 3.3: Lista de perfis suportados em um *smartphone* Samsung J5 Pro.

entre dispositivo mestres (do inglês, *master*) e dispositivos escravos (do inglês, *slave*). O dispositivo mestre pode ser proativo e solicitar uma conexão. Enquanto o escravo pode ser apenas descoberto. Destaca-se que a opção de descoberta por ser desabilitada em alguns dispositivos. O equipamento que deseja ser descoberto envia em formato *broadcast* seu nome, classe. serviços suportados (perfis) e informações complementares como por exemplo fabricante qual especificação Bluetooth ele pertence. Usualmente é observado o nome do dispositivo para ter a certeza de que se está conectando o dispositivo correto. Pois não é amigável a um usuário inserir um valor de 48 bits. Entretanto, algumas aplicações pode-se inserir o endereço diretamente. Uma vez que o dispositivo mestre possui o endereço, a conexão está estabelecida. Se um dispositivo de destino conhece o endereço do dispositivo mestre, ele sempre responderá as requisições.

O Bluetooth também suporta questões relativas a redes para conexão de vários dispositivos. É o suporte às chamadas picorredes (do inglês, *piconets*). Nela, um dispositivo mestre pode se comunicar com até sete outros dispositivos. Entretanto, é possível ter suporte para até 255 dispositivos. Desde que apenas oito deles estejam ativos por vez [Kurose 2010]. Uma *piconet* pode ser conectada a outra *piconet* e assim formarem a chamada *scatternet*. Entretanto o aprofundamento deste tópico foge do escopo do presente trabalho.

3.5 Módulo Bluetooth HC-05

O módulo Bluetooth HC-05 foi o escolhido para o sistema. Este é um módulo que suporta o serviço SPP comentado anteriormente. Logo, seus pinos de transmissão de recepção podem ser tratados como os de um cabo para conexão serial. Este módulo possui a especifica-

ção 2.0 do Bluetooth, ou seja, já possui o EDR (*Enhanced Data Rate*). As taxas suportadas são as seguintes: 9600, 19200, 38400, 57600, 115200, 230400 e 460800 baud. Portanto, compatível com o Arduino Nano. A alimentação também é de 5V. Então este módulo pode ser alimentado pela mesma fonte do Arduino. Este módulo também suporta as duas formas de conexão: mestre e escravo. O que significa que o módulo pode ser proativo em ser conectar a um dispositivo ou apenas receber tentativas de comunicação. Conforme mostrado na Fig. 3.2, o HC-05 (escravo) foi conectado a um computador (mestre) que solicitou a conexão.

Na Fig. 3.4 é mostrada a pinagem e o chip do módulo Bluetooth HC-05. As dimensões do chip são de 16.1mm x 37.5mm.

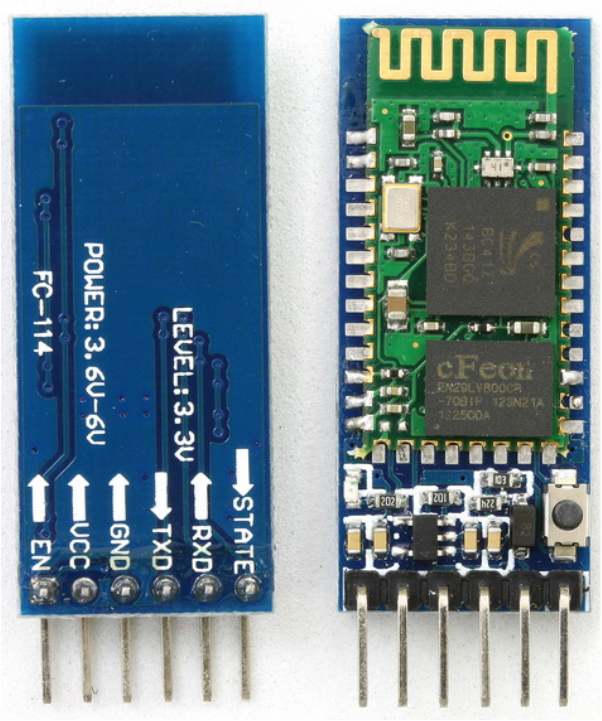

Figura 3.4: *Hardware* do módulo Bluetooth HC-05

3.6 Conclusão

Como visto ao longo do capítulo, o padrão Bluetooth, ou tecnologia Bluetooth, é um conjunto de especificações, que proporciona o uso nas mais diversas aplicações. O desenvolvimento das especificações vem trazendo melhoramentos a cada versão o que faz com que o padrão seja cada vez mais adotado. Para o sistema de aquisição dados médicos, a tecnologia Bluetooth mostra características que são de interesse, como a mobilidade e o baixo consumo de energia são prioridade. A integração com outros dispositivo como um computador também é usado no sistema proposto. No próximo capítulo será mostrado como que de fato usou-se o módulo Bluetooth HC-05 no sistema e como os fundamentos teóricos apresentados foram levados em consideração.

Capítulo 4

Sistema de aquisição

4.1 Hardware Utilizado

O *hardware* escolhido para o sistema levou em conta premissas como custo, dimensões, mobilidade, e uso de sistema de código aberto. Para isso, o Arduino Nano foi uma escolha natural neste processo. Com isso, foi necessário o uso de um módulo *Bluetooth* compatível. Para isso foi escolhido o HC-05. Há também a versão HC-06. Que é mais barata pois possui apenas o modo escravo. Porém, apenas o HC-05 foi encontrado no mercado local. Também foi separada uma alimentação externa de 5V para o Arduino para mitigar questões relacionadas ao ruído. Estas questões serão verificadas experimentalmente. Então ao invés de usar a porta USB de um computador, foi utilizado um *power bank*. A Figura 4.1 ilustra de forma simplificada o relacionamento entre as peças do *hardware*.

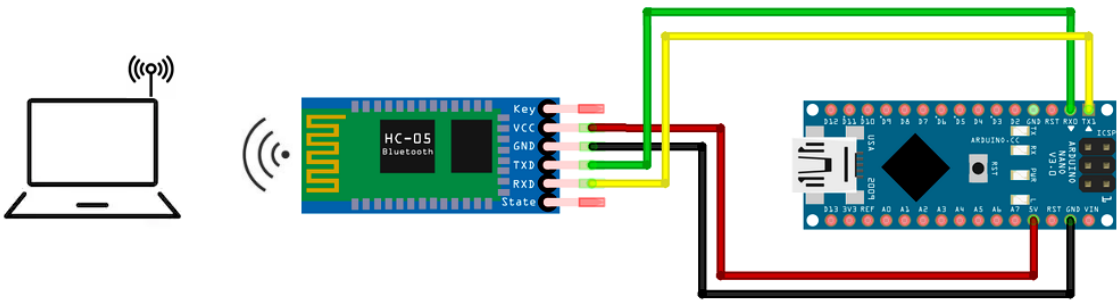

Figura 4.1: Topologia simplificada com o relacionamento de peças de *hardware*

4.2 Software Utilizado

4.2.1 Software no Computador

O computador executa funções importantes, pois receberá as informações via interface Bluetooth, processará esses dados que serão exibidos em um gráfico, em tempo real. O *software* construirá um gráfico dedicado para cada sinal de entrada utilizado. Para isso, foi utilizado o software MATLAB (do inglês, *Matrix Laboratory*) e sua linguagem de programação, em conjunto com a extensão *Instrument Control Toolbox*^{TM} [MathWorks 2019]. Os dados serão adquiridos através de sensores de captura conectados ao Arduino Nano e transmitidos ao computador. De forma geral, o código trabalha de maneira a criar uma conexão com o Módulo Bluetooth HC-05 através do adaptador Bluetooth do computador. Após isso permanecer em estado de espera até que seja recebido uma sinalização, que é um número de 4 *bytes* específico que indicará que o próximo inteiro é uma informação válida. Define-se com informação válida um valor inteiro após a sinalização que será colocada no gráfico em tempo- real. Isso é feito de forma repetida, onde alterna-se apenas em qual gráfico o dado ser será colocado de acordo com a sinalização enviada.

A Fig.4.2 ilustra e resume, via máquina de estados o comportamento do *software* no MATLAB.

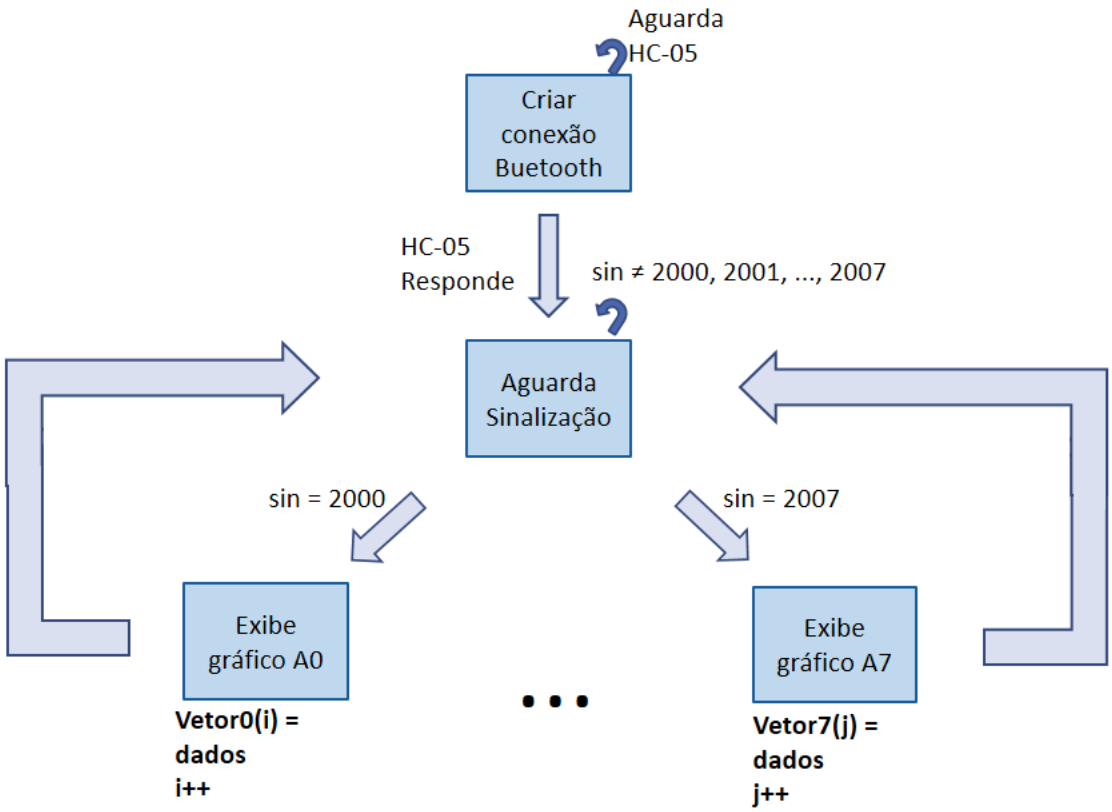

Figura 4.2: Comportamento resumindo do *software* no MATLAB através de máquina de estados finita.

4.2.2 Software no Microcontrolador

Por outro lado, foi criado um código em Arduino, que é similar a linguagem C, para programação do microprocessador ATmega328P, além das demais funcionalidades do microcontrolador. Assim, este software gerencia toda a parte de adquisição de dados, formatação, configurações de interrupções, temporizadores, conversores, transmissão serial e frequências de adquisição. De forma geral o código é iniciado com uma extensa lista de parâmetros de inicialização. Usam-se algumas variáveis e constantes para a marcação de tempo via temporização, por exemplo. Também é feita a configuração inicial de como deverão ser as operações de digitalização, e demais variáveis de controle como um acumulador para cada canal também deve fazem parte deste escopo. Ao final da parte de preparação (*setup*), aguarda-se a contagem do temporizador para que sejam tomadas outras ações. Após a interrupção do temporizador é feita a coleta do dado em uma das entradas analógicas do Arduino. Após a coleta é feita a digitalização e em seguida o dado é colocado à disposição para envio via porta serial. Devido ao perfil Bluetooth SPP, a comunicação via sem-frio é transparente para o Arduino. Após o envio da informação o programa retorna ao seu estado de aguardar o temporizador novamente.

A Fig. 4.3 ilustra e resume, via máquina de estados o comportamento do *software* no Arduino.

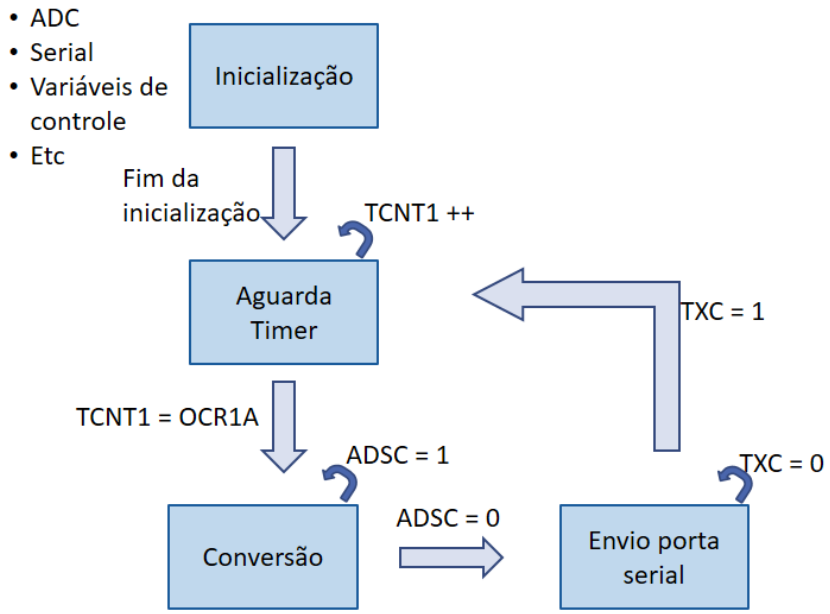

Figura 4.3: Comportamento resumindo do *software* no microcontrolador através de máquina de estados finita.

4.3 Características da Aquisição

Cada característica da adquisição deverá ser especificada, pois a variação de cada uma delas pode interferir na obtenção, precisão ou percepção dos dados adquiridos. Assim como suas limitações. Este detalhamento parte das escolhas de *hardware* e *software* descritas nas seções 4.1, 4.2. Assim, usa-se do embasamento teórico dos Capítulos 2 e 3 para tais escolhas. Cada uma das principais características serão detalhadas nas seções a seguir.

4.3.1 Frequências de aquisição

São necessárias algumas considerações a cerca do microprocessador para a aplicação dos temporizadores. O microprocessador ATmega328P possui uma frequência de 16 MHz, o que equivale de um ciclo de processador de duração 62,5 ns. Na Eq. 4.1 $clk_{I/O}$ é a frequência base do microprocessador e o T tempo para cada ciclo, ou seja, o período do relógio (*clock*).

$$T = \frac{1}{clk_{I/O}} = \frac{1}{16 \times 10^6} = 62,5 \text{ ns} \tag{4.1}$$

Como explicado no Capítulo 2, o ATmega328P possui três temporizadores: TIMER0, TIMER1 e TIMER2. Onde apenas o TIMER1 possui 16 bits. E este foi o escolhido para uso no sistema. Como precisamos de medidas na taxa de 1 Hz, a utilização do maior temporizador disponível e do pré-escalonador facilitarão a montagem da contagem de tempo. Também teremos o controle relacionado a possível atraso nas interrupções, o que será explicado em maiores detalhes na seção 4.3.2

O microprocessador do Arduino Nano também permite que seja feito um ajuste de frequência base ($clk_{I/O}$), como por exemplo para 1 MHz. Tal ajuste pode ser realizado em busca do aumento da autonomia energética, que pode se tornar um requisito do sistema caso se o desejo a implementação de uma alimentação externa. É possível imaginar uma aplicação modular do sistema, onde um *notebook* e o sistema pudessem ter mobilidade. Entretanto, tal opção não foi implementada e é sugerida na seção de proposta para trabalhos futuros.

Ao longo das subseções seguintes será usado um exemplo onde serão amostrados dois canais, com frequências de aquisição distintas. As escolhas, justificativas e ressalvas serão dadas a cada aplicação de componentes.

4.3.2 Temporização

Em várias aplicações é necessário marcar o intervalos de tempo com precisão. Assim, o ferramental desejado para isso são temporizadores. Para personalização de medidas de tempo descritas a seguir, é necessário programar diretamente os registradores do AVR ao

invés de fazer uso de bibliotecas prontas disponíveis no Arduino. Como o objetivo é fazer a aquisição de múltiplas fontes de dados, o processador deverá lidar com todas as interrupções sem que exista um atraso ou significativo atraso entre cada medida. Um atraso em alguma medida pode significar uma alteração da frequência de aquisição dos dados.

Um dos problemas encontrados é que deixar uma escolha livre de frequências de aquisição poderia levar as interrupções simultâneas. E de acordo com o funcionamento do microprocessador, há uma regra para tratamento desses casos. Em caso de várias interrupções sendo disparadas, elas são colocadas na pilha em ordem de chegada. Em seguida uma é executada após a outra com uma batida do relógio entre as interrupções. Em caso de simultaneidade, a CPU atende primeiro a interrupção mais prioritária. A interrupção é um evento assíncrono. Para a aplicação de aquisição de dados periódicos desejada são necessárias frequências definidas, ou seja, que os eventos de interrupção se aproximem ao máximo de um evento síncrono. Vamos supor a captura de dois sinais de entrada, onde cada captura é feita quando a interrupção pelo temporizador é iniciada. Há um temporizador e uma interrupção para cada captura. Nesse modelo, não é possível garantir quando uma interrupção irá coincidir com a outra. Assim, poderemos ter casos de coincidência de interrupção, atrasando a captura, e no próximo ciclo, não ter a coincidência, ou seja, este ciclo em especifico teve uma frequência menor do que foi especificado. Outra vantagem relacionada a essa abordagem é que não há temporizadores em quantidade suficiente para todos os canais do conversor analógico-digital. Considerando a hipótese de usar os três temporizadores, ou seja, incluir o TIMER0 e TIMER2 (8 bits), também não se mostra viável. Pois há oito canais de entrada. Para esse abordagem seria necessário ter oito temporizadores dedicados. Pré-requisito que não existe nem mesmo em outros membros mais robustos da família Arduino como o Arduino Mega.

Assim, para suplantar essa limitação, é feito uso de apenas um temporizador. Mesmo se tratando de uma coleta de dados de múltipla fontes. E para cada fonte adicional, podem ser usados múltiplos dessa frequência. Com esse mecanismo, assegura-se que o atraso em interrupções coincidentes será sempre igual, com algumas ressalvas. Estas interrupções são identificadas na Fig. 4.4 pelo Δt Uma vez que o atraso é sempre igual para todos os ciclos, a frequência para adquisição dos dados será constante. A Fig, 4.4 ilustra de forma visual este mecanismo.

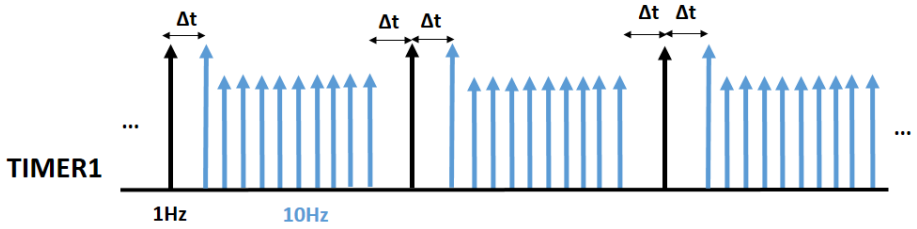

Figura 4.4: Ilustração do mecanismo de interrupção que gerará sempre o mesmo atraso controlado e mantém a frequência constante.

Inicialmente, é necessário identificar com que frequência de amostragem devem ser feitas

as aquisições, pois cada fonte possui uma peculiaridade. Cada aplicação também tem sua peculiaridade. Por exemplo, há casos onde uma medida periódica com intervalo de um segundo (1 Hz) ou dezenas de segundos são suficientes. Em outros casos é necessário o estudo do comportamento da onda, então deve-se transmitir dados em taxa superior para uma coleta mais granular, e possibilitar a reconstrução do sinal analógico. Como por exemplo um eletrocardiograma (ECG) ou um eletromiográfico (EMG) . Sendo mais valiosa a informação de um trem de *bytes* do que uma medição em específica. Para o exemplo teórico, supomos como a maior frequência de aquisição a taxa de 100Hz. Vamos denominar esta frequência como clk_{Aq}. Assim, como já mencionado, todos os canais deverão ter 100 Hz ou, para frequências exatas, em períodos múltiplos de 100 Hz ou divisões exatas em 100 Hz. Uma vez que para atingir frequências menores, como por exemplo, 25 Hz, é necessário esperar por quatro vezes a interrupção de 100 Hz para que uma amostra seja tomada. De acordo com a Eq. 2.1, pode-se escrever a Eq. 4.2 para que seja encontrado o valor a ser programado em OCR1x a partir do período T entre medições desejado. O valor de N referente ao pré-escalonador será detalhado na seção 4.3.3.

$$OCR1x = \frac{T \times clk_{I/O}}{N} = \frac{\frac{1}{100} \times 16 \times 10^6}{256} = 625 \tag{4.2}$$

4.3.3 Pré-Escalonador

O valor escolhido de divisor para pré-escalonador N, mostrado na Eq. 4.2, foi 256. Essa escolha se dá por dois motivos. É um valor que mantém a restrição de OCR1x estar entre 0 e 65535 e também resulta em um OCR1x inteiro, que é uma especificação do fabricante. O que também resulta em uma frequência teórica exata. Obviamente, em caso de outro valor de N, poderia-se abrir mão da precisão para que fosse feito um arredondamento. Assim, define-se uma frequência de aquisição máxima clk_{Aq}. Portanto, a partir da programação dessa definição, todas as frequências que são divisões exatas desta. Estes múltiplos são calculados a cada ciclo da interrupção que é executada a 100 Hz. Uma variável acumuladora Acc faz este papel. Por exemplo, é se necessário uma frequência de aquisição de 25 Hz, deve-se aguardar quatro interrupções de 100 Hz. Ou seja, uma condição onde um contador é igual a Acc. A Tabela 4.1 mostra todas frequências de números inteiros que podem ser programados. Nota-se que estes números inteiros são múltiplos exatos de 100 Hz. Para que este comportamento aconteça, é programado um acumulador (Acc) para registrar quantas vezes já se passou pela interrupção com frequência máxima de aquisição clk_{Aq} de 100Hz.

A configuração dos bits seletores do *clock* do Arduino Nano, referentes ao TIMER1, estão no registrador TCCR1B. Assim, os bits CS12, CS11 e CS10 foram configurados conforme Tabela 2.3 para o valor de $N = 256$ conforme a Equação 4.3. P é a frequência de

saída do pré-escalonador.

$$P = \frac{clk_{I/O}}{N} = \frac{16 \times 10^6}{256} = 62500\text{Hz} \tag{4.3}$$

Então, neste exemplo, a interrupção acontecerá a uma frequência de 100 Hz e o vetor de interrupção TIMER1_COMPA_vect é executado. O código inserido dentro desta sub-rotina tem algumas responsabilidades. A primeira é de saber se uma leitura do ADC deverá ser feita, disparar o ADC, verificar se é necessário mudar de canal e por último deverá se certificar que as variáveis dos temporizadores permaneçam corretamente inicializadas para as coletas seguintes.

Tabela 4.1: Valores de acumuladores a serem programados para que sejam alcançadas novas frequências desejadas nos canais de recepção de dados

clk_{Aq} [Hz]	*Acc*	Nova frequencia [Hz]	Intervalo entre medidas [s]
100	2	50	0,02
	4	25	0,04
	5	20	0,05
	10	10	0,1
	20	5	0,2
	25	4	0,25
	50	2	0,5
	100	1	1
	200	0,5	2
	...		
	1000	0,1	10
	...		

4.3.4 ADC

O ADC, conversor analógico-digital, também deve ser explorado de forma específica de maneira a atender os requisitos do sistema proposto. Como o Arduino Nano possui um ADC de até 10 bits, foi escolhido fazer uso da resolução máxima. Como já explicado na sessão anterior, a amostragem do sinal será feita nos tempos determinados no temporizador. Fisicamente, o sinal de entrada deverá ser ligado em uma das entradas lógicas da placa Arduino descritas como A0, A1, ..., A7. Ressalta-se que o sinal de entrada deverá ter tensão máxima de 5V. Caso a fonte tenha uma tensão superior, um divisor de tensão deverá ser aplicado para prevenir danos à placa. A seguir será explicado como cada componente foi configurado e por que foram feitas tais escolhas.

Para o funcionamento esperado do ADC do Arduino são necessárias várias configurações diferentes, em vários bits de alguns registradores. Alguns durante a fase inicial do programa (função `setup()`) e outras na função principal do Arduino (função `loop()`).

Na `setup()` teremos a configuração da tensão de referência, a habilitação do ADC, configuração de auto disparo, configuração do relógio do ADC e seleção do canal. Durante a execução das medições ocorrerão configurações de troca de canal e disparos de conversões.

Primeiramente, para a parte inserida no `setup()`, foi a escolhodia da tensão de referência, em que é usada a referência interna de 5V ($AVcc$). A escolha justifica-se por dois motivos: por não ter que se preocupar com mais um componente para servir de referência externa e por ter maior intervalo de valores possíveis para digitalização. Assim, é preferível a tensão de referência interna de 5V em detrimento da referência interna de 1.1V. Então os bits REFS1 e REFS0 do registrador ADMUX foram configurados com os valores 0 e 1 respectivamente para que $V_{REF} = AVcc = 5V$. Em seguida, para habilitação do ADC é necessária a configuração do bit ADEN, no registrador ADCSRA, que acontece de forma simples, onde é feita a operação equivalente a ADEN = 1. Para o auto disparo, é necessário indicar se ele será usado ou não. Visto que optou-se pelo sistema que faz uso da conversão do tipo única (ADC interrompe a operação a cada conversão), então o bit ADATE do registrador ADCSRA deverá ser ADATE = 0. Ainda no `setup()`, é necessária a configuração do relógio do ADC (clk_{ADC}). Conforme definido na Eq. 2.3, há apenas uma opção disponível onde são respeitadas as especificações do fabricante, que orienta que clk_{ADC} fique no intervalo entre 50 kHz a 200 kHz. Logo, o pré-escalonador do *clock* do ADC foi configurado para uma divisão de 128. Assim, os bits responsáveis foram configurados na forma ADPS2 = 1, ADPS1 = 1 e ADPS0 = 1, no registrador ADCSRA. Por último, é necessário escolher o canal do multiplexador do ADC que será utilizado. No Arduino Nano as portas lógicas de entrada estão em ordem crescente. Então a porta lógica A0 representa a escolha dos bits MUX3:0 em zero (equivalente a 0000). A porta lógica A1 o equivalente ao 0001. E assim sucessivamente.

Durante a execução do `loop()` é necessária a troca do canal em alguma ocasiões. Então a depender da interrupção do temporizador é mantida a escolha ou é feita a troca. A Figura 4.5 ilustra os pontos em que ocorrerão as trocas de canal. A troca é feita nos bits MUX2:0. Deve-se observar que a troca de canal significa aguardar o tempo de uma primeira conversão novamente. Porém, como já comentado na seção 4.3.2, uma vez que o atraso é sempre igual, a frequência de adquisição não é alterada. Então o tempo total de conversão leva em conta o tempo de primeira conversão relacionada a estabilização do retentor (13,5 ciclos do clk_{ADC}), mais o tempo de conversão em si (25 ciclos do clk_{ADC}). Logo, após a troca de canal, a conversão tem tempo total de 38,5 ciclos do clk_{ADC}. Quando for uma conversão onde não é necessária a troca do canal, o tempo total é de 15 ciclos do clk_{ADC} (1,5 ciclos referentes ao retentor e 13 referentes ao tempo de conversão).

Outra questão referente ao `loop()` é que as conversões precisam ser disparadas manualmente devido a escolha da conversão única ao invés da conversão contínua (*free running*). Então após a troca de canal, é necessário iniciar a conversão fazendo ADSC = 1. Ao fim da conversão, ADSC retorna a zero automaticamente.

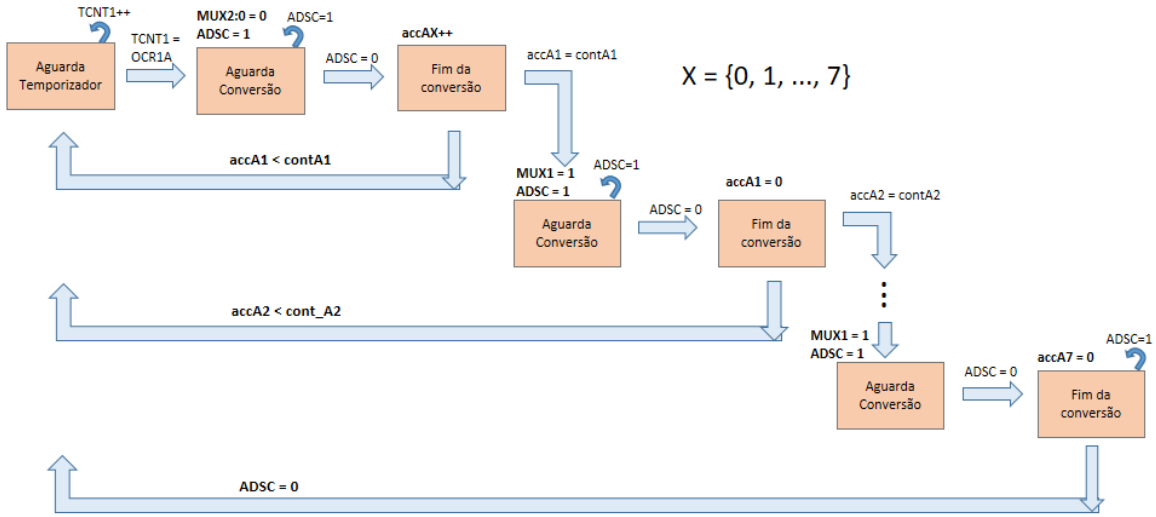

Figura 4.5: Ilustração via máquina de estados para melhor identificação de mudança nos registradores

4.3.5 Comunicação Serial via Bluetooth

A Arduino Nano possui apenas uma porta para a comunicação serial. Então de aqui em diante, a notação genéria usando a letra n será substituída pelo registrador real equivalente. Por exemplo: UBRRn será o UBRR0. Para fins práticos e por limitação de tempo para desenvolvimento do projeto, a comunicação serial foi desenvolvida com uma abordagem diferente. Via funções da biblioteca padrão ao invés de programação diretamente nos registradores do AVR, e também utilizando de funções de leitura disponíveis no MATLAB. A seguir alguns dos problemas encontrados neste cenário onde o transmissor é um Arduino e o a recepção é o código de processamento do MATLAB. É utilizado um exemplo onde deseja-se enviar o valor 5.01:

- A função `serial.print()` na transmissão (Arduino) e `fread()` na recepção (MATLAB). A comunicação serial via esta função permite nesse formato que cada caractere seja enviado separadamente para receptor, e cada caractere também é convertido para um item da tabela ASCII. Caso for necessário enviar o valor 5.01. Será enviado o cinco (5), depois o ponto (.), depois o zero (0) e depois o um (1), convertidos em itens da ASCII. Então gera uma complexidade muito maior no receptor. Pois teremos que tratar quatro variáveis separadamente. E não é da natureza do código de MATLAB o tratamento bit-a-bit ou byte-a-byte para construção de um valor válido. O mais próximo disso é a função `fread()`, onde pode ser delimitado quantos bits deverão ser lido por vez. Entretanto, o tratamento para criar um valor a partir dessa leitura continua complexo;

- Função `serial.print()` na transmissão (Arduino) e `fscanf()` na recepção (MATLAB). Também há o mesmo problema. Pois a receber os valores convertidos em tabela ASCII. Este valores serão imediatamente convertidos pela segunda vez para

o tipo indicado na função fscanf(). Persistindo assim o problema de retirar um valor válido a partir do exemplo ilustrado anteriormente;

- Programação nos registradores do AVR na transmissão e função `fread()` na recepção. Neste cenário, é necessária a criação de um padrão de valores. Por exemplo: valores de 0 a 9,99, com duas casas de precisão. E então, no receptor no MATLAB, pode ser feito o tratamento para apenas um tipo. A solução é menos flexível, porém o suficiente para algumas soluções;

- Função *serial.println()* na transmissão e `fscanf()` o receptor. No transmissor, será enviada qualquer quantidade de dados e em seguida o caractere na tabela ASCII referente a pular linha (do inglês, *line feed*). E no receptor, a leitura de qualquer dado também é o `line feed`. Então, em caso de número com diferentes quantidades de casas decimais, ou valores maiores de 9.99, 99.99, etc, não limitam o programa. E como a solução visa obter dados de várias fontes diferentes. É uma escolha plausível ter esta flexibilidade.

No exemplo de transmissão do último exemplo utilizou-se duas funções de uma das bibliotecas padrão da plataforma Arduino: `serial.begin()` e `serial.println()`, e na recepção no MATLAB, `fscanf()`. Entretanto, tais funções representam várias configurações de registradores do AVR que foram referenciados na seção 2.4.5. A seguir a será detalhado quais escolhas foram feitas no projeto quais são os registradores utilizados implicitamente por essas duas funções.

Fisicamente, as conexões são simples. O pino TX1, referente a transmissão da placa Arduino Nano, é ligado na recepção do módulo *Bluetooth* (pino RXD). E o contrário também acontece. Pino TXD do módulo *Bluetooth* no pino RX0 do Arduino. Para o *software* Arduino não faz diferença existir o módulo *Bluetooth* ou ser uma comunicação serial tradicional por um cabo devido as abstrações do próprio *Bluetooth*. Dessa forma, o código deverá definir a taxa de transmissão de dados (*baud rate*, formato de transmissão, configurar opções desejadas os três registradores de status e controle (UCSR0A, UCSR0B e UCSR0C), disponibilizar os dados de envio e realizar adequações necessárias para uma comunicação sucessiva sem erros.

Assim como no ADC, é necessário inserir algumas das configurações já no `setup()`. Com o uso da biblioteca padrão do Arduino, a função `serial.begin(9600)` traz consigo a configurações de formato de transmissão de *baud rate* de 9600, sem bits de paridade e um bit de parada. Durante a execução do código, a amostra retirada da fonte analógica é um inteiro (8 bits), porém ele é convertido em um ponto flutuante (32 bits). Então são necessários quatro ciclos da comunicação serial para o envio de um dado válido. De forma análoga a função `serial.begin(9600)`, os registradores a ser utilizados para essa operação são UBRR0H e UBRR0L (ou seja, a representação em 16 bits do valor desejado) de acordo com a Eq. 4.4. Onde D é o *baud rate* e X é fator relacionado ao dobrador de velocidade explicado na Seção 2.4.5. Este valor deverá ser substituído na Eq. 2.4 para verificar a taxa de 9600

bps. Quanto às configurações implícitas relacionadas a paridades e bits de parada, os bits UPM00:1, do registrador UCSZ0C, deverão estar configurados para nível respectivo a esta taxa. Pois essa configuração diz respeito a desabilitar o bit de paridade. Para a uso de um bit de parada, o bit USBS0 deverá estar em nível baixo para ter esse formato. A

$$UBRR0 = \frac{clk_{I/O}}{X \times D} - 1 = \frac{16 \times 10^6}{16 \times 9600} - 1 = 103 \tag{4.4}$$

Portanto, todo o registrador UCSR0A pode ser zerado no `setup()`. No registrador UCSR0B, apenas o bit TXE0 deverá ser configurado em nível alto. Para que a transmissão de dados na porta 0 seja habilitada. E para garantir uma transmissão de 8 bits de comprimento, os bits UCSZ00:2 dos registradores UCSR0B e UCSR0C deve assumir os valores 110 respectivamente.

Já durante o `loop()`, a única função a ser chamada é `serial.println()`. Que na prática colocará os dados a disposição da porta serial pra envio. De forma análoga, é o mesmo que escrever no registrador UDR0 para que o dado seja colocar no *buffer* de envio. E também para aguardar o final da transmissão ao verificar o bit TXC0 do registrador UCSR0A. Após o envio esta *flag* ficará em nível alto após o envio concluído e deverá ser colocado em zero para estar pronto para o próximo envio. Deve-se observar que a função insere o decimal 10 da tabela ASCII ao final da informação colocada em `serial.println()`. Isso facilitará o processado do dado na recepção, uma vez que o conteúdo útil é toda informação que antecede o decimal 10. O decimal 10 na tabela ASCII equivale a mudança de linha (*Line Feed*).

Um dos desafios encontrados durante a construção do sistema foi a identificação da origem dos sinais. Pois, em caso de uso de dois ou mais canais, é necessário separá-los para que cada fonte tenha seu gráfico dedicado construído separadamente. Para isso foi criada uma sinalização, onde para cada dado enviado, é transmitido um número inteiro (dois *bytes)* específico. Por exemplo, ao iniciar a amostragem e transmissão dos dados do canal A0, a primeira informação a ser disponibilizada na porta serial é o número 2000. Depois disso é enviado próximo valor do tipo inteiro transmissor. Ou seja, uma sinalização para cada inteiro enviado. Em seguida, porta A1 deverá ser utilizada. Nesse momento é enviado o inteiro 2001. O receptor, ao detectar o cada valor de 2001, irá separar este dado para um outro segmento do programa. Assim será possível ter os dados separados. Essa solução também é escalável para os oito canais disponíveis.

Dessa forma podemos observar que há uma taxa efetiva de transmissão, pois a cada inteiro enviado ao receptor é necessário o uso de 16 bits. Aproximadamente 32 bits se não forem considerados os bits de partida, parada e paridade. Então, excluindo a sinalização criada, a taxa de efetiva transmissão de dados é de 4800 bps. Então como é necessário enviar em rajadas de 32 bits as informações digitalizadas, existem 300 amostras a serem enviadas por segundo. Sendo 150 para informação e 150 para sinalização. Por exemplo, se cada um dos oito canais amostrarem a 37 vezes por segundo, teremos a taxa agregada efetiva

excluindo a sinalização será aproximadamente 300 amostras. Que é o equivalente a 9600 bps. Limite teórico aproximado do sistema.

4.4 Resultados

De forma a exemplificar o uso de toda conceituação realizada neste trabalho, foi utilizada uma abordagem experimental para a demonstração de resultados. Para isso, todo o sistema foi implementado conforme descrido ao longo do Capítulo 4.

Para a demonstração de forma experimental, o sistema foi levado a um laboratório onde foi possível ter acesso a um gerador de funções e a um osciloscópio. O gerador de funções tem o papel de gerar um sinal previsível, com frequência conhecida, para assim termos o controle da aquisição. O osciloscópio serve como um controle para validação. Pois espera-se que os resultados com a digitalização do sistemas usando o Arduino Nano sejam visualmente similares aos que são vistos no osciloscópio.

O primeiro teste diz respeito ao ruído, onde era necessário eliminar a possibilidade da fonte de alimentação do Arduino contribuir com um ruído significativamente grande em relação as medidas. O Arduino foi inicializado e foi colocada uma ponta de prova ligada ao osciloscópio para possível identificação do ruído. A Fig. 4.6 mostra um dos instantes de captura com a maior amplitude de ruído, que foi de aproximadamente 0,60 mV de tensão máxima. Foi feita a tentativa de troca a alimentação do sistema. Feito uma teste utilizando a alimentação externa via da porta USB do computador e outro teste com uma alimentação externa de 5V e 2A através de um carregador portátil. Não foi possível identificar alguma diferença significativa entre as medidas. Isso significa que ambos influenciaram de forma similiar as medidas experimentais. Como ambas medidas são equivalentes, optou-se pelo próprio computador para as demais medidas.

A simulação de colocação de sinais de entrada, digitalização e transferência de dados forai feita com os mesmos parâmetros. O sinal injetado foi uma onda senoidal de 1 Hz, tensão pico-a-pico de 3 V e com *offset* de 1,5V. O *offset* foi utilizado pois o Arduino foi programado para utilizar um canal simples ao invés de canal diferencial. Fig. 4.7 mostra a onda de controle para validação medida através do osciloscópio.

Neste primeiro cenário, utilizou-se apenas uma entrada analógica do Arduino. Entretanto, são utilizadas duas taxas de amostragens diferentes. Quanto ao resultado, foi observada a saída do sistema no MATLAB. A Fig. 4.8 representa este resultado. Verificou-se que há uma pequena diferença de tensão para cima. A linhas em azul no gráfico referem-se a amostragem de 20 Hz e de vermelho a 100 Hz. Observa-se também as em frequências mais altas de amostragem, o ruído é mais detalhadamente amostrado

Os testes a seguir mostram a utilização de dois canais de entrada no Arduino, para se verifique o efeito de trocas de canais do ADC. O comportamento geral da digitalização se

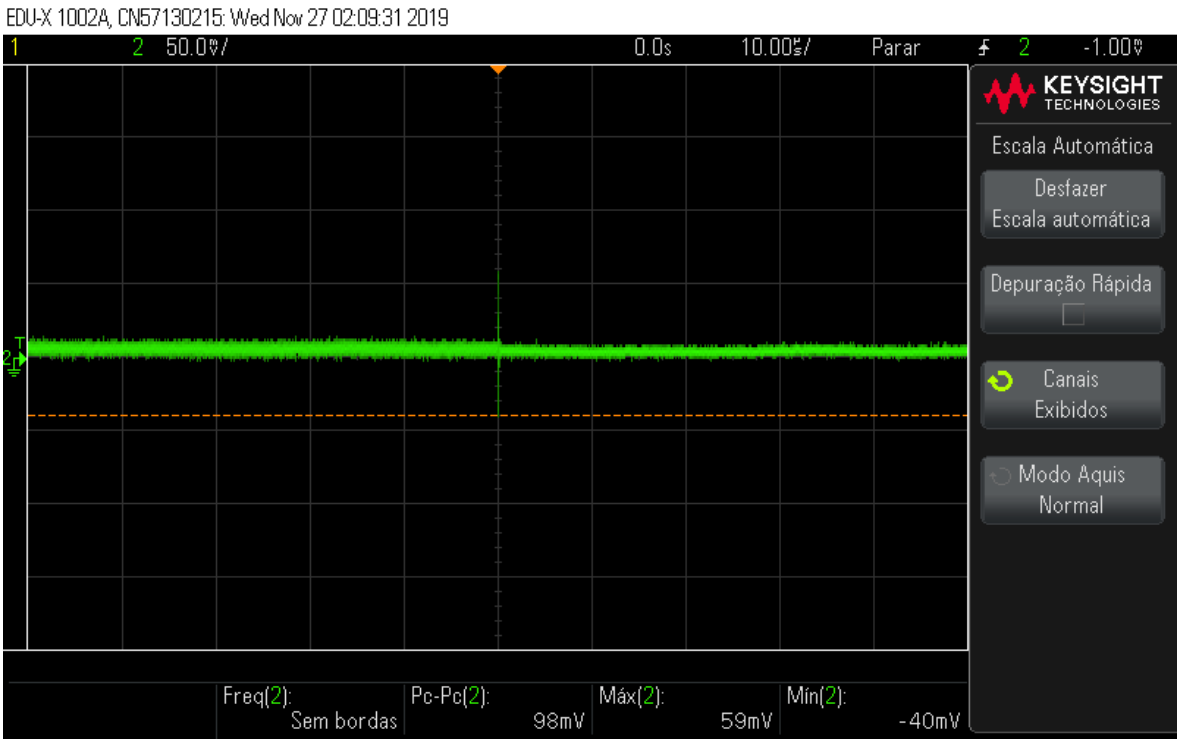

Figura 4.6: Perfil de ruído capturado na porta analógica do Arduino Nano.

manteve. Entretanto, observou-se um ruído mais acentuado na coleta de 100 Hz. Assim com o presença de uma tensão de uma componente DC. Pois pela inspeção visual da Fig. 4.9, o sinal não toca a linha inferior de 0 V, diferente da Fig. 4.8. Ressalta-se que há uma mudança abrupta no gráfico nas primeiras amostras. Essa mudança acontece pois a medida nova sobrescreve a medida antiga a cada quinhentas amostras. Então a imagem reflete o momento da sobrescrita naquele instante.

De forma a ilustrar a variedade de sinais possíveis nos canais de entradas foram feitas mais coletas: uma com uma onda quadrada e outra com uma onda dente de serra. As Figs. 4.10 e 4.11 mostram a onda de controle para validação medida pelo osciloscópio.

As mesmas observações feitas para o caso da onda senoidal podem ser feitas para a dente de serra e quadrada conforme esperado. Foi mantida a tensão em 3 Vpp, tensão de *offset* de 1.5 V e frequência de 1 Hz. Também foram mantidas as taxas de amostragem em 5 Hz e 20 Hz. Os resultados estão representados nas Figs. 4.12 e 4.13.

Por último foi elevada a frequência do sinal de entrada para 3 Hz. O sinal foi uma onda dente de serra, e as tensões foram mantidas conforme testes anteriores. A representação deste resultado se encontra na Fig. 4.14. Observou-se que as questões relativas à tensão DC permaneceram. Quanto à amostragem a 5 Hz da Fig. 4.14, a onda dente de serra se aproximou de uma onda triangular. Pois com a taxa de amostragem selecionada algumas mudanças abruptas do sinal não são percebidas. Para ajuste da taxa de amostragem correta, deve-se levar em consideração o Teorema da amostragem de Nyquist, que em resumo descreve que para a recuperação do sinal original após a digitalização, é necessário amostrar com o dobro

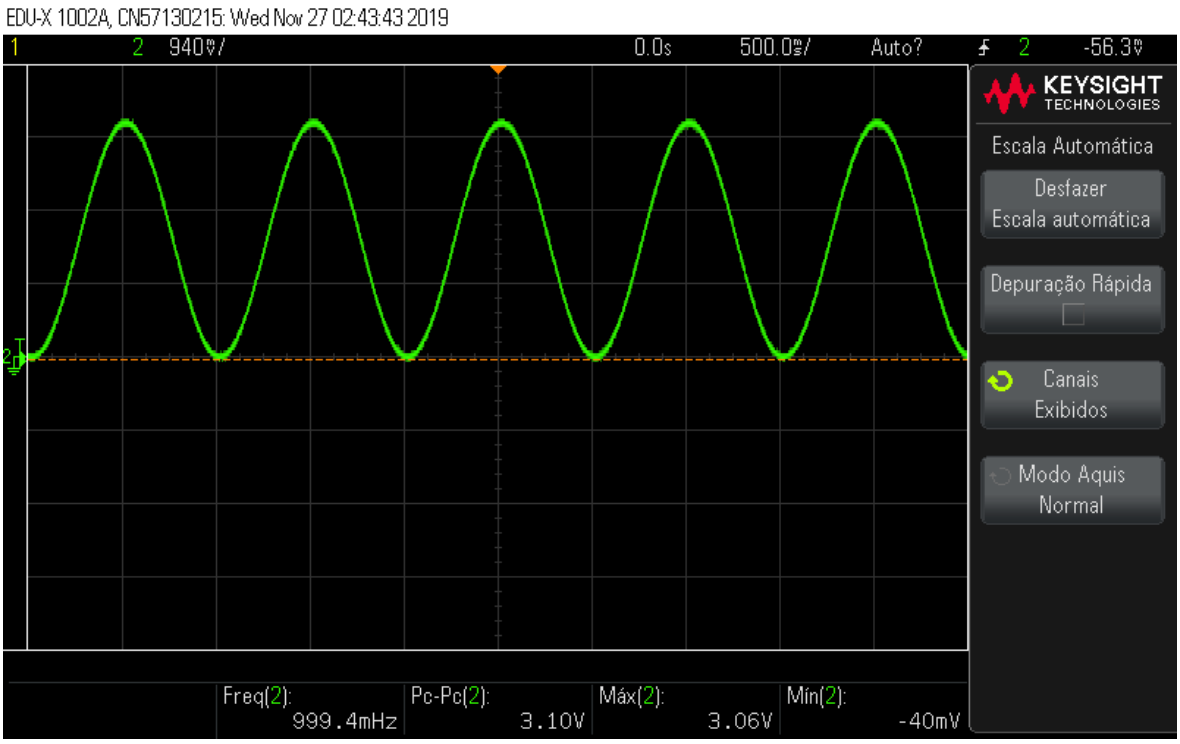

Figura 4.7: Sinal de entrada capturado no osciloscópio da onda senoidal.

da taxa da maior frequência daquele sinal. Esta taxa foi observada.

É possível prever outras formas de dados de entrada. Por exemplo, uma fonte de dados médicos compatível com o Arduino, ou mais ainda, fazer com que esse sinal seja de fato compatível. Assim, pode-se programar o Arduino para receber este dado. Seja para estudar o formato da onda ou para enviar os dados em outros formatos. Por exemplo a transmissão de dados via ponto flutuante também é possível. Entretanto, deve-se observar que o ponto flutuante tem o tamanho de 32 bits e que a taxa efetiva diminuirá conforme exemplificado no final da seção 4.3.5.

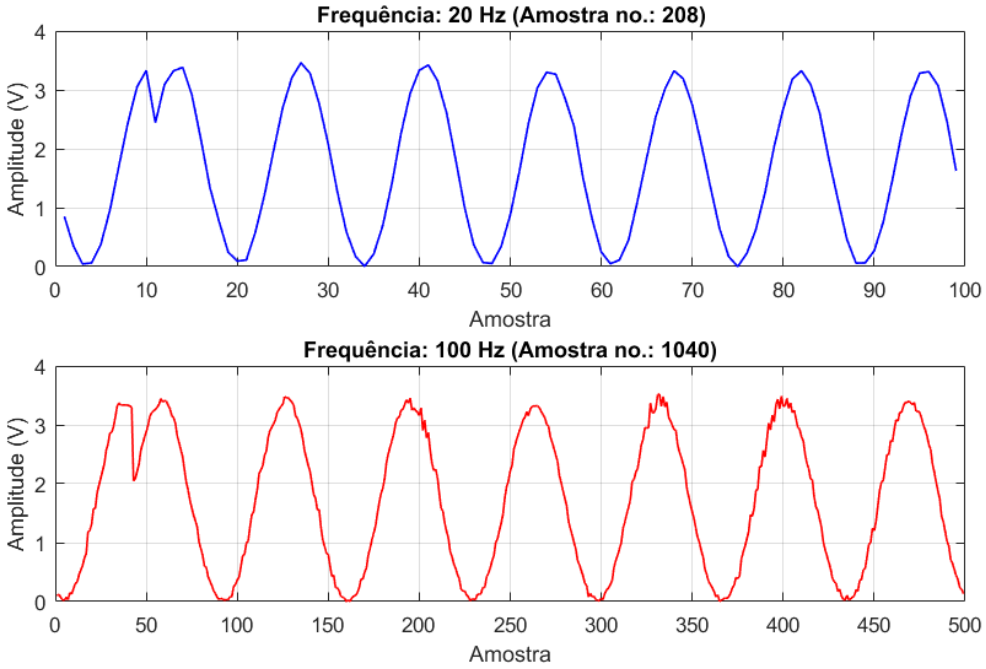

Figura 4.8: Sinal observado após a digitalização. Sinal de origem é um seno de 1 Hz de frequência, 3 Vpp e 1.5 V de *offset*. Figura criada em tempo real no MATLAB.

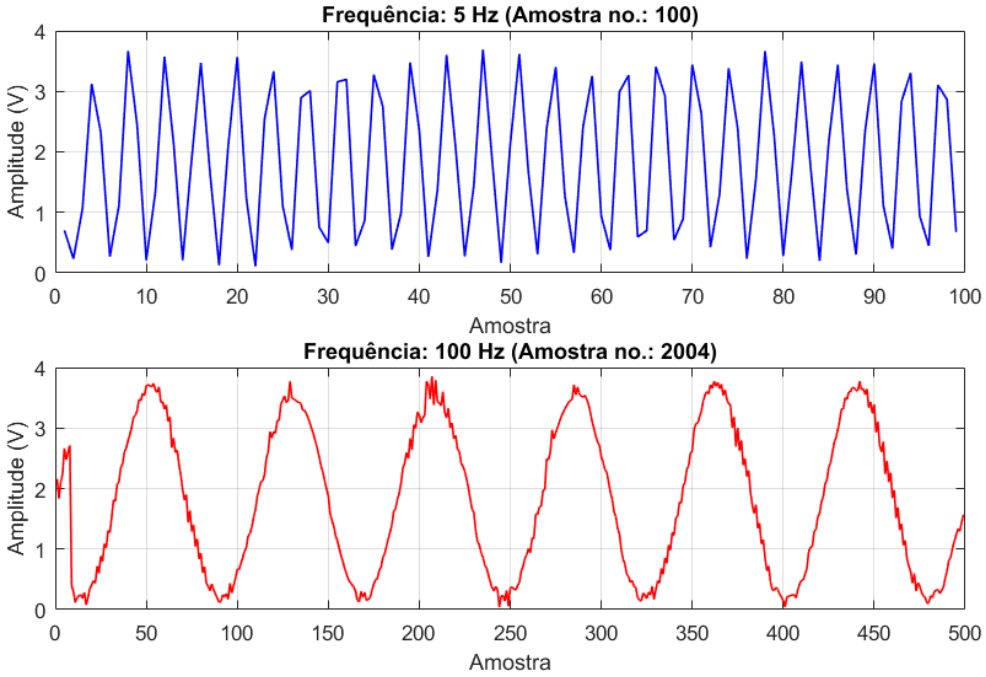

Figura 4.9: Sinal observado após a digitalização utilizando dois canais diferentes. Figura criada em tempo real no MATLAB.

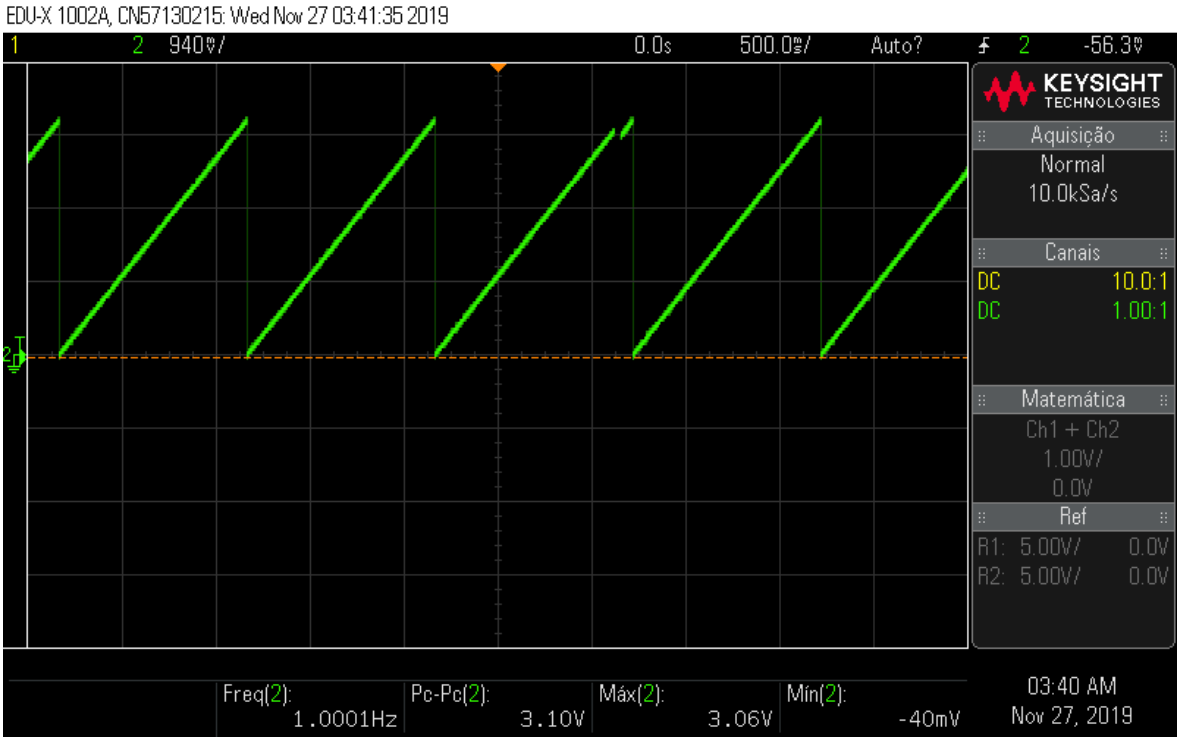

Figura 4.10: Sinal de entrada capturado no osciloscópio da onda dente de serra.

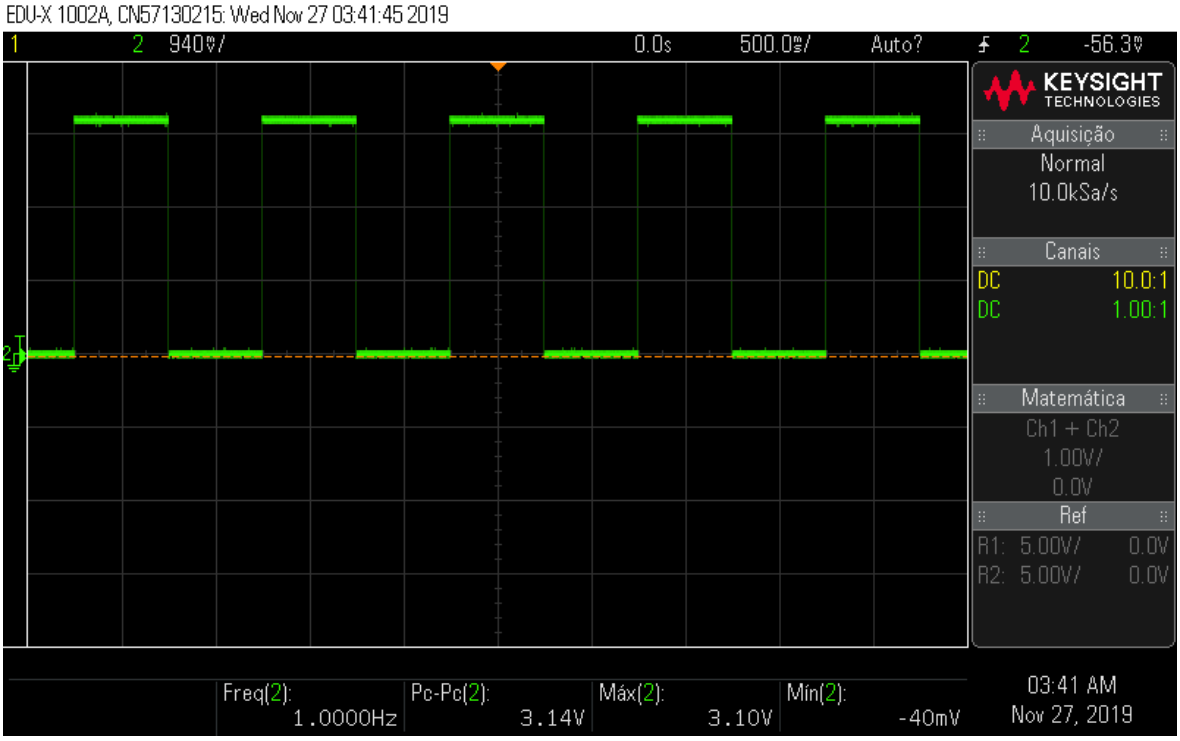

Figura 4.11: Sinal de entrada capturado no osciloscópio da onda quadrada.

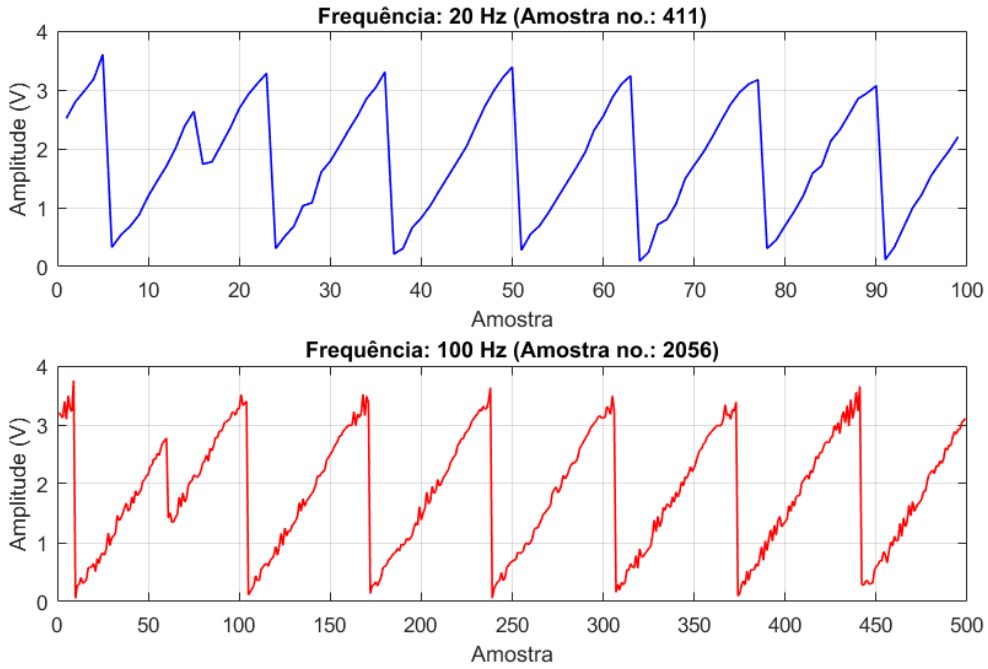

Figura 4.12: Sinal de entrada digitalizado pelo Arduino e criado em tempo real no MATLAB. Onda triangular de 1 Hz.

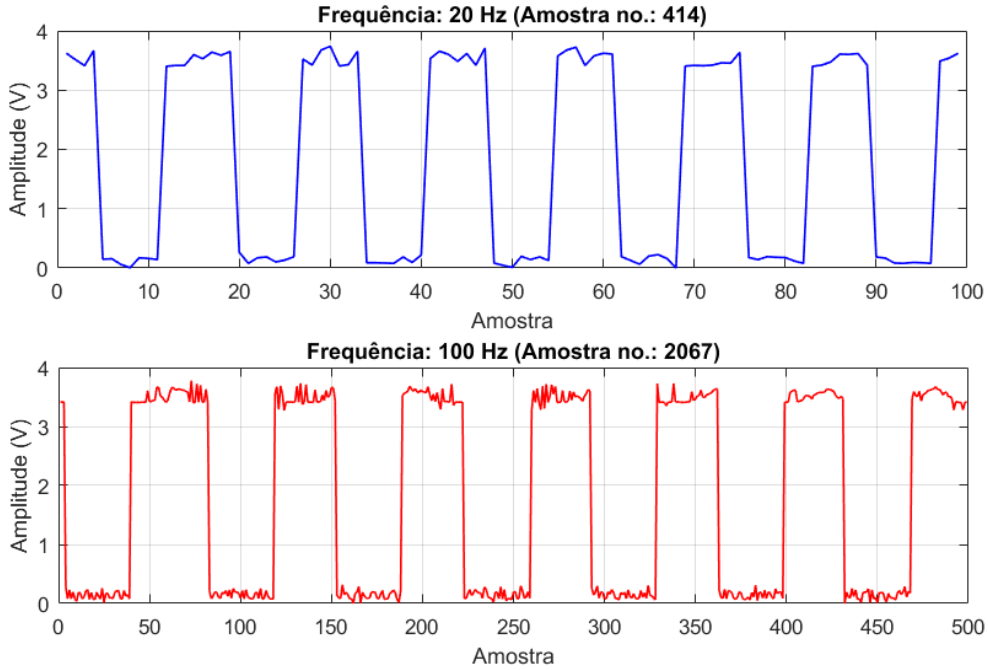

Figura 4.13: Sinal de entrada digitalizado pelo Arduino e criado em tempo real no MATLAB. Onda quadrada de 1 Hz.

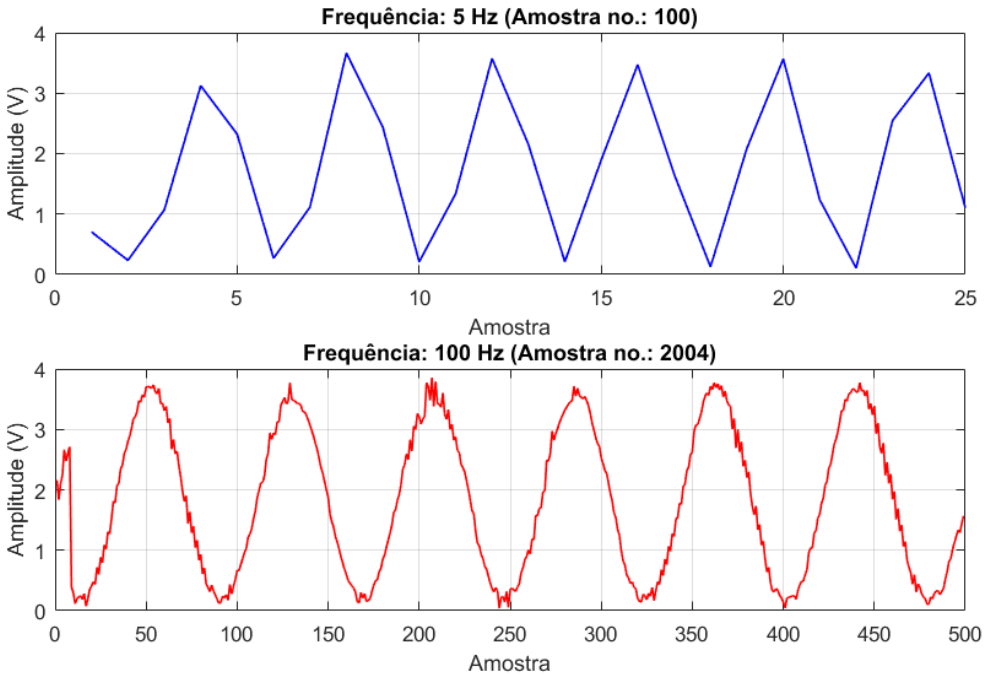

Figura 4.14: Sinal de entrada digitalizado pelo Arduino e criado em tempo real no MATLAB. Sinal de origem é uma onda dente de serra de 3Hz e senóide de 1 Hz

Capítulo 5

Conclusão

O presente trabalho mostra que uma abordagem onde são programados os registradores do microprocessador diretamente tem várias vantagens. Então o uso de conceito de micro-eletrônica abre portas para a utilização dessa abordagem. Que é uma opção em relação a implementação via bibliotecas prontas construídas por terceiros. O estudo teórico constatou que algumas funções como controle de temporização e conversão analógico-digital são suficientemente precisas se construídas levando em consideração as especificações corretamente. Quando aos resultados experimentais, foi possível observar no laboratório que para que seja possível ter resultados suficientemente precisos, é necessário investigação mais aprofundada da diferença do sinal de entrada de saída. Onde a digitalização de várias fontes de fato é possível. A transmissão de dados sem-fio através da tecnologia Bluetooth foi possível graças a abstrações criadas nos chamados perfis Bluetooth. Os dados atravessaram o meio sem-fio e foram exibidos em tempo real num computador. O computador que funcionou com receptor e possuía o código de processamento via MATLAB exibiu estes dados.

Foi observado que o uso de microcontroladores pode ser usado como um centralizador de dados de múltiplas fontes, com múltiplas frequências de adquisição e múltiplas naturezas. E que o microcontrolador é capaz de transferir estes dados através de um meio sem-fio para um dispositivo que é capaz de processar estes dados em tempo real.

5.1 Proposta para futuros trabalhos

Durante a realização do trabalho foram vários aspectos que ainda poderiam ser estudados a partir do que foi desenvolvido. Algumas propostas para que o trabalho possa ser estendido são apresentados abaixo:

- Desenvolvimento utilizando outros microcontroladores;
- Desenvolvimento utilizando outras tecnologias e padrões para transmissão de dados. Como por exemplo Ethernet (IEEE 802.3), Wi-fi (IEEE 802.11), Zigbee (IEEE

802.15.4) e etc;

- Desenvolvimento visando mais eficiência energética e modularidade, como por exemplo diminuindo a frequência do processador;
- Verificação de forma mais aprofundada do ruído e tensão DC encontradas nos experimentos realizados. Como por exemplo utilização de alguns processos de filtragem;
- Testes envolvendo frequências mais altas de digitalização, para encontrar o limite experimental do microcontrolador neste contexto.

Referências Bibliográficas

[Atmel Corporation 2015] Atmel Corporation (2015). ATmega328P Datasheet - 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash.

[Banzi 2009] Banzi, M. (2009). Getting Started With Arduino. *O'Rilley*, pages 5–19.

[Barragán 2004] Barragán, H. (2004). Wiring: Prototyping Physical Interaction Design. *Interaction Design Institute Ivrea*.

[IEEE 2002] IEEE (2002). IEEE 802.15.1-2002 - IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN - Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs). <*https://ieeexplore.ieee.org/document/1016473*>.

[IEEE 2005] IEEE (2005). IEEE 802.15.1-2005 - IEEE Standard for Information technology - Local and metropolitan area networks– Specific requirements– Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN). <*https://ieeexplore.ieee.org/document/1490827*>.

[Kurose 2010] Kurose, James F. e Rosse, K. W. (2010). Redes de Computadores e a Internet. *Ed. Pearson*, 5a Edição.

[Masato 2010] Masato, I. e Ichiro, M. (2010). Rapid Prototyping for Control Education using Arduino and Open-Source Technologies. *IFAC Proceedings Volumes*, 42:317–321.

[MathWorks 2019] MathWorks (2019). Instrument Control Toolbox. <*https://www.mathworks.com/help/instrument/index.html*>.

[Sanders 2017] Sanders, B. e Nardozza, L. (2017). Specification defines doubling bandwidth to extend existing USB Type-C™ cable performance. <*https://www.usb.org*>.

[Sayali 2015] Sayali, B. (2015). Bluetooth Protocol Stack. <*https://www.ques10.com/p/2700/bluetooth-protocol-stack-1/*>.

[Site Oficial do Arduíno 2019] Site Oficial do Arduíno (2019). Arduino Nano. <*https://store.arduino.cc/usa/arduino-nano*>.

[Techmemicro 2014] Techmemicro (2014). Arduino NANO Pinout Diagramt. <*https://www.teachmemicro.com/arduino-nano-pinout-diagram/*>.

[Zelenovsky 2019] Zelenovsky, R. e Mendonça, A. (2019). Arduino Guia Avançado para Projetos. *Editora Interciência*, 1:127–130.