UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
GRADUAÇÃO EM ENGENHARIA ELÉTRICA

RAFAEL JULIANO SCHOLTZ

SISTEMA DE COMUNICAÇÃO PARA AQUISIÇÃO DE DADOS COM O DSP F28379D E MICROCONTROLADOR ESP 32

JOINVILLE
2022

RAFAEL JULIANO SCHOLTZ

SISTEMA DE COMUNICAÇÃO PARA AQUISIÇÃO DE DADOS COM O DSP F28379D E MICROCONTROLADOR ESP 32

Trabalho de Conclusão de Curso submetido ao Bacharelado em Engenharia Elétrica do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina como requisito final para a obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Yales Rômulo de Novaes
Coorientador: Dr. Chrystian Lenon Remes

Joinville

2022

RAFAEL JULIANO SCHOLTZ

SISTEMA DE COMUNICAÇÃO PARA AQUISIÇÃO DE DADOS COM O DSP F28379D E MICROCONTROLADOR ESP 32

Trabalho de Conclusão de Curso submetido ao Bacharelado em Engenharia Elétrica do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina como requisito final para a obtenção do grau de Bacharel em Engenharia Elétrica.

Banca Examinadora:

Orientador:

Prof. Yales Rômulo de Novaes, Dr.
Universidade do Estado de Santa Catarina - UDESC

Coorientador:

Chrystian Lenon Remes, Dr.

Membros:

Prof. Ana Teruko Yokomizo Watanabe, Dra.
Universidade do Estado de Santa Catarina - UDESC

Christofer Schwartz, Dr.

Suplente:

Prof. Celso José Faria de Araujo, Dr.
Universidade do Estado de Santa Catarina - UDESC

Joinville, 09 de dezembro de 2022

AGRADECIMENTOS

O autor agradece ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e à FAPESC (Fundação de Amparo e à Pesquisa e Inovação do Estado de Santa Catarina) pelo financiamento parcial da pesquisa.

Ao meu orientador Dr. Yales Rômulo de Novaes, pelas oportunidades concedidas para aprofundar o meu conhecimento em eletrônica e também pela dedicação em acompanhar a elaboração deste trabalho.

Ao meu coorientador Dr. Chrystian Lenon Remes, que me instruiu com paciência sobre como lidar com as questões de documentação e verificação de *software*.

Aos meus pais, Elaine Maria Tonet Scholtz e Juliano Sadi Scholtz, pelo apoio e amor incondicional ao longo da minha vida.

Aos meus amigos da graduação, pelas memórias que criamos juntos.

Eu vos agradeço de coração.

RESUMO

Neste trabalho é proposto um sistema de comunicação destinado à transmissão de variáveis em velocidades de até 6,4 Mbps, com o intuito principal de possibilitar o cálculo de controladores digitais baseados em dados para conversores estáticos. O sistema utiliza os microcontroladores F28379D da *Texas Instruments* e ESP 32 da Espressif para executar o experimento de coleta de dados e transmiti-los via Wi-Fi respectivamente. A interface entre os microcontroladores funciona através do protocolo SPI, de maneira que o F28379D pode ser facilmente programado em linguagem C para transmitir as variáveis de interesse ao se utilizar as funções públicas desenvolvidas. Os dados são salvos no formato binário a partir da comunicação com um servidor TCP/IP escrito em Python e implementado em um computador pessoal. A validação do sistema é feita a partir da transmissão e comparação de dados pré-determinados, tal que 4 amostras de 16 bits são geradas em interrupções periódicas com frequência de 100 kHz. Dessa forma, seu funcionamento em um ambiente sem a presença de ruído é verificado a partir do recebimento esperado de todas as amostras em um período de até 5 segundos.

Palavras-chave: Armazenamento de dados. Conversores estáticos. Controle digital. Controle baseado em dados.

ABSTRACT

This work presents a communication system for logging information at speeds up to 6.4 Mbps, primarily motivated to enable the computation of data-driven discrete controllers for static converters. The system requires the use of the microcontrollers F28379D from Texas Instruments and ESP 32 from Espressif to perform the data collection experiment and transmit the generated information through Wi-Fi respectively. The interface between the microcontrollers is based on the SPI protocol, as such the F28379D can be programmed in C to transmit the variables of interest by using the developed public functions. A TCP/IP server written in Python and hosted in a personal computer saves the transferred data in binary format, so that it can be decoded in a more useful format by the user later. The system is tested by transmitting and comparing a pre-determined sequence of 4 samples, which are composed by 16 bits each and are generated at a frequency of 100 kHz. Thus, as it is verified that all the values received in a period of 5 seconds indicate no errors in an environment without significant noise, the communication system is successfully validated.

Keywords: Data-logging. Static converters. Digital control. Data-driven control.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACK	*Acknowledgement*
API	*Application Programming Interface*
AWGN	*Additive White Gaussian Noise*
BER	*Bit Error Rate*
CPU	*Central Processing Unit*
CS	*Chip Select*
DAC	*Digital-to-Analog Converter*
DBPSK	*Differential Binary Phase Shift Keying*
DCF	*Distributed Coordination Function*
DD	*Data-Driven*
DCF	*Distributed Coordination Function*
DFT	*Discrete Fourier Transform*
DSP	*Digital Signal Processor*
DSSS	*Direct Sequence Spread Spectrum*
FEC	*Forward Error Correction*
FFT	*Fast Fourier Transform*
FIFO	*First In First Out*
FHSS	*Frequency-Hopping Spread Spectrum*
GI	*Guard Interval*
GPIO	*General Purpose Input/Output*
HT-GF	*High-Throughput Greenfield*
HT-M	*High-Throughput Mixed*
IDFT	*Inverse Discrete Fourier Transform*
IoT	*Internet of Things*
IPV4	*Internet Protocol Version 4*

I2C	*Inter-Integrated Circuit*
MAC	*Media Access Control*
MCS	*Modulation Coding Scheme*
MCU	*Microcontroller Unit*
MISO	*Master Input Slave Output*
MOSI	*Master Output Slave Input*
NRZ	*Non-Return-to-Zero*
OFDM	*Orthogonal Frequency Division Multiplexing*
PHY	*Physical Layer*
PLCP	*Physical Layer Convergence Protocol*
PMD	*Physical Medium Dependent*
PPDU	*PLCP Protocol Data Unit*
PSDU	*PLCP Service Data Unit*
QAM	*Quadrature Amplitude Modulation*
RAM	*Random Access Memory*
RTOS	*Real Time Operating System*
SCI	*Serial Communications Interface*
SCLK	*Serial Clock*
SNR	*Signal-to-Noise Ratio*
SPI	*Serial Peripheral Interface*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UART	*Universal Asynchronous Receiver/Transmitter*
UML	*Universal Modeling Language*

LISTA DE SÍMBOLOS

f	Frequência
T	Período
$s_m(t)$	Forma de onda de um sinal modulado em função do tempo
X_k	Símbolo QAM
R	Taxa de código
N_{DBPS}	Número de bits de informação por símbolo OFDM
N_{BQAM}	Número de bits mapeados por símbolo QAM
T_{OFDM}	Intervalo de cada símbolo OFDM
SNR_{rx}	Relação sinal-ruído do lado do receptor
γ_b	SNR por bit
P_b	Probabilidade de erro de bit
P_{se}	Probabilidade de erro de símbolo
P_{de}	Probabilidade de erro no decodificador
P_{pe}	Probabilidade de erro na transmissão de um pacote
$P_{e,data}$	Probabilidade de erro na transmissão de um quadro de dados
$P_{e,ack}$	Probabilidade de erro na transmissão de um quadro de ACK
$P_{s,tx}$	Probabilidade de sucesso na transmissão de dados
$Q(x)$	Complemento da função distribuição cumulativa de um processo aleaório normal de média 0 e desvio padrão unitário
PL	Atenuação de um sinal em detrimento da distância percorrida
$kbps$	Kilobit por segundo

SUMÁRIO

1 INTRODUÇÃO

Conversores estáticos são circuitos compostos por semicondutores e elementos armazenadores de energia. Seu propósito é o processamento da energia elétrica, servindo como uma interface entre dois ou mais sistemas elétricos [1]. Neste contexto, há frequentemente a necessidade de controlar variáveis como tensão e corrente, de forma que diversas abordagens existem para tal finalidade. Um dos métodos mais utilizados é a utilização de controladores em malha fechada, os quais garantem que o comportamento desejado ocorra ao mudar a dinâmica do sistema inteiro.

A implementação destes controladores pode ocorrer de maneira analógica ou digital. As vantagens de se utilizar uma malha de controle analógica estão no custo reduzido e na simplicidade de projeto para uma planta linear. Entretanto, quando estratégias de controle mais complexas são necessárias, o controle digital permite um processo iterativo de projeto mais flexível, haja visto que basta mudar o *software* para mudar a lei de controle. Além disso, variações paramétricas de componentes eletrônicos podem ser significativas em um circuito de controle físico, o que não ocorre na versão digital visto que a lei de controle é gravada na memória de algum dispositivo [2].

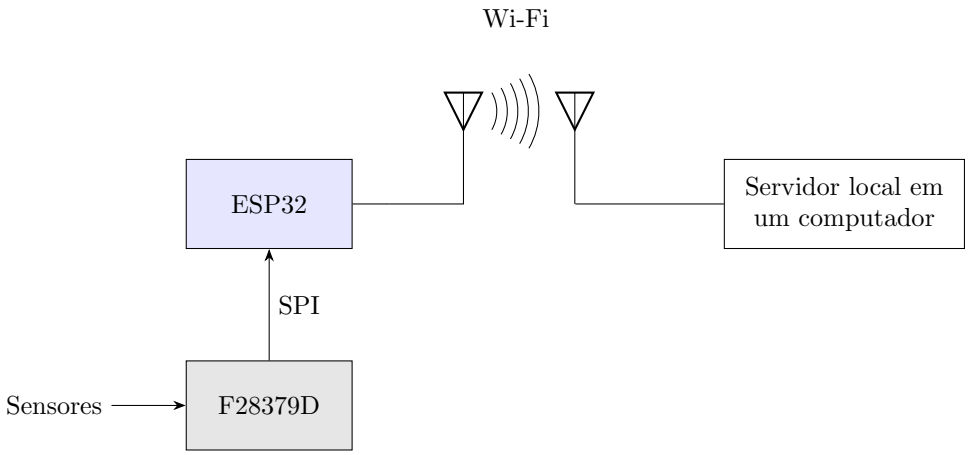
No âmbito de projeto de controladores, a vertente dos métodos baseados em dados (DD, do inglês *Data-Driven*), começou receber atenção no início da década de 1990 [3]. Ao contrário dos métodos baseados em modelo, que utilizam algum conhecimento prévio da planta para elaborar um modelo dinâmico representativo, a metodologia DD utiliza grandes quantidades de dados para a formação de controladores ótimos que atendam a certos critérios de desempenho especificados pelo projetista. Uma das principais vantagens do controle DD é a incorporação das características da planta nos dados de entrada dos algoritmos de cálculo dos controladores, tal que não é necessário realizar uma modelagem complexa para se obter um modelo representativo. Nesse sentido, surge a necessidade de se realizar experimentos para coletar uma grande sequência de variáveis do processo, como a referência, ação de controle e leituras de sensores.

Como conversores estáticos podem operar com altas frequências de comutação, chegando até a escala de MHz, surge a necessidade de um sistema que consiga transmitir os dados do experimento para um dispositivo com maior capacidade de armazenamento, visto que a memória em sistemas embarcados não é mais suficiente. Na literatura, Baba et al.[4] exploram a ideia de um sistema de conversores modulares integrados a uma rede de comunicação sem fio 5G, cuja interface é feita utilizando o microcontrolador (MCU, do inglês *Microcontroller Unit*) ESP 32 entre o conversor e um roteador 5G. Nesse sentido, a transmissão de dados do conversor é feita seguindo o protocolo SCI (Interface de Comunicação Serial, do inglês *Serial Communications Interface*) entre a placa de controle e o ESP 32, que é responsável pela comunicação com o roteador 5G via Wi-Fi. A aquisição de dados é feita com sucesso, mas problemas de interferência eletromagnética são relatados em algumas situações. Além disso, Silva et al.[5] implementam

tanto o controle de um conversor cc-cc *Flyback* quanto a comunicação com o usuário usando o mesmo módulo ESP 32. A comunicação é utilizada para monitorar os dados de tensão e corrente de entrada e saída de um módulo fotovoltaico, os quais ficam disponíveis para visualização através de uma página web.

Este trabalho propõe a elaboração de um sistema de comunicação que utiliza dois dispositivos microprocessados para separar as funções críticas referentes a alguma aplicação das funções de transmissão de dados, ilustrado pela Figura 1. Nesse aspecto, é desenvolvido um módulo em linguagem C para o DSP (Processador Digital de Sinais, do inglês *Digital Signal Processor*) F28379D da *Texas Instruments*, escolhido por disponibilidade e por ser um modelo popular em aplicações de controle e processamento digital de sinais. O módulo contém funções que garantem a transmissão de dados via Interface Periférica Serial (SPI, do inglês *Serial Peripheral Interface*) para o MCU ESP 32 da Espressif, que os transmite via Wi-Fi para um computador pessoal. Nesse aspecto, a comunicação sem fio também garante a isolação galvânica entre o computador e o restante do sistema. Adicionalmente, é feita uma análise da qualidade de comunicação sem fio do sistema, de maneira a identificar a distância máxima entre o ESP 32 e o dispositivo executando o servidor para armazenamento de dados. O sistema desenvolvido pode ser utilizado para a coleta de informações de um sistema qualquer, motivado principalmente pelos métodos de controle DD para conversores estáticos.

Figura 1 – Diagrama de blocos do sistema de aquisição de dados



Fonte: Elaborado pelo autor.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Projetar uma estrutura de comunicação capaz de transmitir dados referentes às variáveis de interesse para um servidor em um computador pessoal via rede sem fio.

1.1.2 Objetivos específicos

- Elaborar as funções de comunicação necessárias para a transferência dos dados;
- Verificar o desempenho do sistema mediante às configurações da camada física.

1.2 ESTRUTURA DOS CAPÍTULOS

Primeiramente, no Capítulo 2 é feita uma revisão bibliográfica dos protocolos de comunicação serial implementados em microcontroladores. Além disso, os protocolos de interesse do padrão IEEE 802.11-2012 são brevemente apresentados no Capítulo 3, que também aborda a teoria de comunicação referente ao cálculo da qualidade do link estabelecido entre 2 dispositivos no âmbito do Wi-Fi. Em seguida, é apresentada uma pesquisa de microcontroladores comumente utilizados para comunicação sem fio no Capítulo 4. Em sequência, no Capítulo 5 são mostrados os resultados obtidos para definir o perfil de qualidade das tecnologias de comunicação de interesse. É apresentado então, no Capítulo 6, a documentação do *software* para o sistema de aquisição de dados, tal que uma demonstração está disposta no Capítulo 7 para ilustrar como o conjunto pode ser utilizado e testado. Por fim, as conclusões sobre o trabalho são feitas no Capítulo 8.

2 REVISÃO DOS PROTOCOLOS DE COMUNICAÇÃO SERIAL

Neste capítulo é feita uma revisão bibliográfica dos tipos de protocolos de comunicação serial, a fim de se implementar o mais adequado como uma interface entre o MCU voltado à comunicação e o DSP de acionamento do conversor.

2.1 PROTOCOLOS DE COMUNICAÇÃO SERIAL

Em sistemas embarcados, é comum que os dados sejam transmitidos entre um microcontrolador central e os periféricos de maneira serial, visto que há um menor uso de fios para as conexões, além de que para frequências muito altas pode haver o problema de *crosstalking* entre os condutores, isto é, uma tensão indesejada em um pode ser induzida pelo campo eletromagnético gerado por outro [6].

Dessa maneira, no contexto dos protocolos de comunicação serial utilizados em microcontroladores, é possível destacar três que são comumente implementados via *hardware* por diversos fabricantes: SPI, I2C[1] e UART[2]. A fim de se determinar qual é o protocolo mais adequado à aplicação da exportação dos dados de medições e controle referentes a um conversor estático, é feita abaixo uma breve descrição de suas características, bem como uma comparação entre eles.

O protocolo I2C foi desenvolvido pela Phillips na década de 1980 a fim de permitir a comunicação entre diversos componentes em uma placa eletrônica de maneira simples e eficiente [6]. Inicialmente, a velocidade máxima de comunicação entre os dispositivos era de 100 kbps, mas em 2012 foi lançado o I2C 4.0, aumentando a velocidade de comunicação para 5 Mbps.

O I2C é um protocolo síncrono, ou seja, em que o sinal de *clock* é compartilhado entre os dispositivos, destinado para a conexão de diversos aparelhos na topologia de barramento. Dessa maneira, para cada componente com uma interface I2C, é necessário conectar somente os terminais referentes ao barramento de *clock* e ao barramento de dados. Apesar dessa versatilidade, o contrapeso é que o protocolo é *half-duplex*, de maneira que os dados podem fluir somente em 1 sentido por vez. Além disso, é padronizado de forma que cada transmissão de dados precisa ser de 8 bits.

Outra característica do I2C é a característica primário/secundário, que requer que, em uma rede de dispositivos, cada um tenha um identificador próprio para que os nós secundários ignorem o que não lhes for endereçado. A desvantagem disso é a ocupação do barramento de dados durante o período em que o endereço do nó secundário está sendo escrito, de forma a reduzir a máxima taxa de dados efetiva do sistema. Nesse sentido, recomenda-se o uso do I2C quando diversos componentes precisam se comunicar com uma baixa taxa de dados.

---

[1] Circuito Inter-Integrado, do inglês *Inter-Integrated Circuit*.

[2] Receptor/Transmissor Universal Assíncrono, do inglês *Universal Asynchronous Receiver/Transmitter*.

O protocolo SPI foi criado pela Motorola em 2000 para ser utilizado como uma interface síncrona serial de curta distância [6]. Uma das principais vantagens do SPI é que não é definida uma velocidade máxima de comunicação, de maneira que o fluxo de dados é limitado de maneira independente para cada dispositivo. Isso possibilita que certos fabricantes implementem em seus produtos suporte a SPI em frequências bem elevadas, chegando até a 80 MHz.

De maneira análoga ao I2C, o protocolo SPI adota a rede primário/secundário para propor a interação entre os dispositivos. Entretanto, enquanto o I2C permite a conexão de diversos nós primários, o protocolo SPI estabelece a presença de somente 1 primário centralizado no sistema. Além disso, para cada secundário conectado, é necessário 4 fios para que os dispositivos conversem na topologia mais simples [7]. Em contrapartida, a comunicação pode acontecer nos dois sentidos simultaneamente (*full-duplex*), desde que o par primário/secundário consiga trabalhar na velocidade e nas demais configurações especificadas.

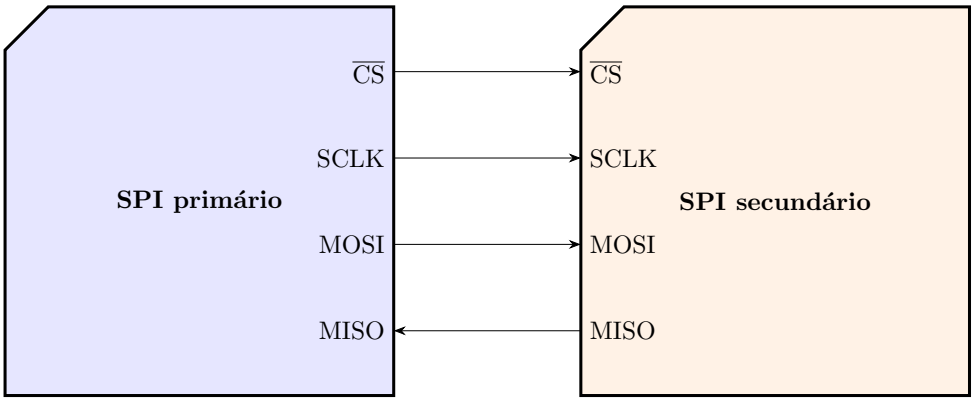
Por fim, o protocolo UART, também chamado de SCI por diversos fabricantes, é um método de comunicação assíncrona ponto a ponto que consiste em uma versão geralizada dos protocolos de camada física EIA, RS-232, RS-422 e RS-485, que especificam os cabos utilizados, níveis de tensão, etc [6]. Nestes protocolos é especificado o padrão de sinal Sem Retorno ao Zero (NRZ, do inglês *Non-Return-to-Zero*), que estabelece que um bit ‘1’ é representado por uma determinada faixa de valores de tensão com uma certa polaridade; e um bit ‘0’ é representado pela mesma faixa de valores, mas com polaridade oposta. Nesse aspecto, para que a comunicação aconteça, é necessário que um par de dispositivos compartilhe 2 conexões: uma para enviar dados e outra para recebê-los. Eles também precisam compartilhar o mesmo referencial. Além disso, ambos os dispositivos devem apresentar de antemão as mesmas configurações de codificação, fluxo de caracteres e taxa de transmissão para que a comunicação ocorra com sucesso, visto que se trata de um protocolo assíncrono. Dependendo do nível de tensão de uma conexão entre um microcontrolador e um equipamento externo, pode ser necessária a utilização de um adaptador de níveis de tensão. No caso do F28379D e do ESP 32 por exemplo, é necessário trabalhar com tensões de até 3,3 V.

Haja visto que a aplicação de coleta de dados já digitalizados referentes a um conversor estático requer uma alta taxa de dados, especificou-se o protocolo SPI como o método de troca de informação entre o DSP e o MCU de comunicação sem fio. Este protocolo é detalhado a seguir.

2.1.1 Protocolo SPI

No protocolo SPI, é especificado que há um dispositivo primário central que decide quando ocorre comunicação entre ele e algum dispositivo secundário. A Figura 2 apresenta as conexões necessárias para 1 componente secundário somente. Em geral, este protocolo é utilizado para a comunicação entre um MCU central e seus periféricos, mas ele também pode ser utilizado na comunicação entre dois dispositivos microprocessados [7].

Figura 2 – Comunicação SPI entre 2 chips



Fonte: Adaptado de [6].

Na Figura 2, o terminal $\overline{\text{CS}}$ (do inglês *Chip Select*) é utilizado pelo primário para selecionar quando o dispositivo secundário deve mandar ou receber dados. Dessa maneira, no início de uma transação, a tensão neste pino deve ir de nível lógico alto para baixo, indicando que uma transmissão de bits entre os componentes está começando. No final de uma transação, ele é colocado de volta em nível lógico alto. Como indicado na figura, este terminal representa uma ação de saída do SPI primário.

No pino SCLK (do inglês *Serial Clock*), o primário deve entrar com a frequência de *clock* utilizada na comunicação. Haja visto que os bits são amostrados, tanto pelo primário quanto pelo secundário, em bordas de transição do sinal de *clock*, esta frequência equivale à taxa de dados entre um par de dispositivos. Ressalta-se que o dispositivo secundário deve conseguir acompanhar a frequência imposta pelo primário, ou a comunicação falha.

O pino MOSI (do inglês *Master Output Slave Input*) é onde os dados de saída do primário são enviados. Já o pino MISO (do inglês *Master Input Slave Output*) é utilizado pelo dispositivo secundário para o envio de dados. Ambos devem estar configurados para ler o canal em uma determinada transição do sinal *clock* e atualizar o valor dos bits enviados na transição oposta. Por exemplo, caso a borda de subida seja utilizada para que o primário e secundário atualizem os valores de tensão nos pinos MOSI e MISO respectivamente, a borda de descida deve ser utilizada para que ambos leiam o que está sendo escrito nos respectivos terminas de entrada.

O protocolo não especifica nenhum esquema de endereçamento ou envio de bits próprios em uma sessão de comunicação, mas geralmente tem-se que há um máximo de bits que podem ser enviados por transação caso uma das partes o implemente via *hardware*. Isto ocorre, pois geralmente é utilizado um registrador de deslocamento para o envio síncrono de dados nos terminais MISO e MOSI com o *clock*, portanto o número máximo de bits enviados é limitado pelo tamanho desse registrador.

3 REVISÃO DOS PROTOCOLOS DE COMUNICAÇÃO SEM FIO

Neste capítulo é abordado o padrão IEEE 802.11-2012, a fim de se descrever como ocorre a troca de informações em um meio sem fio, bem como identificar os parâmetros mais importantes para o cálculo de fatores de qualidade de um canal. Também é apresentada a teoria que modela as probabilidades de erro de detecção de símbolo e de pacote. Estes parâmetros estão diretamente relacionados com a qualidade da comunicação e capacidade de um canal, e podem ser usados para projetar os parâmetros de potência do transmissor.

3.1 PROTOCOLOS DO PADRÃO IEEE 802.11-2012

O padrão IEEE 802.11-2012, comumente associado ao termo Wi-Fi, especifica os protocolos da Camada Física (PHY, do inglês *Physical Layer*) e da Subcamada de Controle de Acesso ao Meio (MAC[1]) em redes sem fio. Com o passar do tempo, foram propostas novas tecnologias que alteravam os padrões anteriores, a fim de se melhorar a qualidade de serviço da comunicação [8]. A Tabela 1 mostra uma comparação de alguns padrões com ênfase na frequência central da banda de comunicação, taxa máxima de dados e tipo de modulação, tais como Espalhamento Espectral por Sequência Direta (DSSS, do inglês *Direct Sequence Spread Spectrum*) e Multiplexação por Divisão em Frequências Ortogonais (OFDM, do inglês *Orthogonal Frequency Division Multiplexing*). A seguir, são descritos alguns detalhes das camadas física e MAC.

Tabela 1 – Comparação entre alguns padrões IEEE 802.11

Padrão	Data	Frequência (GHz)	Taxa de dados (Mbps)	Modulação
IEEE 802.11a	Out. de 1999	5	54	OFDM
IEEE 802.11b	Out. de 1999	2,4	11	DSSS
IEEE 802.11g	Jun. de 2003	2,4	54	OFDM
IEEE 802.11n	Out. de 2009	2,4 ou 5	600	OFDM

Fonte: Adaptado de [8]

3.1.1 Camada física

Nesta camada, são especificados os tipos de técnicas utilizadas para a troca de dados entre dois ou mais dispositivos. Dessa maneira, suas duas funções principais são executadas por duas subcamadas: uma destinada a adaptar qualquer recurso dependente da camada física à situação local (PLCP[2]); e outra destinada a definir as características de transmissão e recepção de dados (PMD[3]), tais como o tipo de modulação, a técnica de espalhamento espectral, etc.

As bandas de frequência usadas em aplicações industriais, científicas e médicas pelos protocolos são centralizadas em 2,4 e 5 GHz, sendo que a última foi introduzida posteriormente

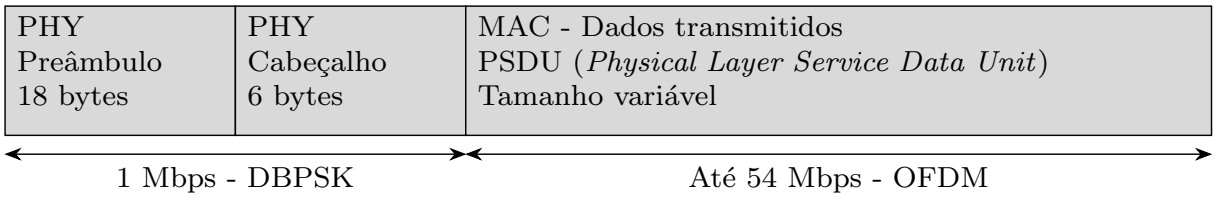
[1] Do inglês *Media Access Control*.
[2] Do inglês *Physical Layer Convergence Protocol*.
[3] Do inglês *Physical Medium Dependent*.

devido ao acúmulo de tráfego de outros sinais na banda de 2,4 GHz. Nesse aspecto, é possível citar as ondas de aparelhos microondas, redes *Bluetooth*, *ZigBee*, entre outras, como as principais fontes de interferência. Para dispositivos de baixa potência, não é necessária uma licença para se operar nessas faixas de frequência. Dito isso, é especificado no padrão um conjunto de 14 canais na frequência de 2,4 GHz para o uso de múltiplos usuários. A separação entre os canais é feita a cada 5 MHz a partir de 2,412 GHz, com exceção do canal 14, separado por 12 MHz do canal 13 [9].

Com relação às técnicas de transmissão utilizadas, são especificados os seguintes métodos de espalhamento espectral: Espalhamento espectral por saltos em frequência (FHSS[4]), DSSS e OFDM. Para cada um destes métodos, usa-se uma determinada técnica de modulação de acordo com a taxa de dados requerida.

A fim de se transmitir um pacote de dados entre dois dispositivos, são usadas diversas técnicas de modulação em uma mesma sessão de comunicação. Isso ocorre visto que um pacote de dados é dividido em várias partes. Por exemplo, o padrão IEEE 802.11g [9] define o formato de pacote longo de acordo com a Figura 3. Nesse aspecto, a transmissão do preâmbulo e cabeçalho da camada física é feita a 1 Mbps usando a modulação DBPSK (do inglês, *Differential Binary Phase Shift Keying*), enquanto o resto do pacote pode ser transmitido utilizando esquemas que permitem uma velocidade maior na comunicação. Isto é feito de forma a minimizar a probabilidade de erro nas informações críticas do pacote de dados, visto que, de maneira geral, as modulações de menor taxa de dados apresentam uma constelação menor de símbolos mais espaçados e, portanto, requerem uma maior interferência para que ocorra um erro na recepção da mensagem.

Figura 3 – Formato genérico de pacote longo de acordo com a IEEE 802.11g



Fonte: Elaborado pelo autor.

As técnicas de espalhamento espectral têm como objetivo aumentar o intervalo de frequência em que a informação é transmitida, visando amenizar possíveis efeitos de interferência que atinjam uma determinada banda estreita onde a mensagem estaria concentrada [10]. Nas primeiras versões da IEEE 802.11, a técnica originalmente adotada era o FHSS, de maneira que, ao se mudar a frequência da portadora em uma sequência pseudoaleatória conhecida pelo transmissor e pelo receptor, era possível reduzir a taxa de erros quando havia comunicação simultânea entre múltiplos usuários.

[4] Do inglês *Frequency-Hopping Spread Sprectrum*.

Já o DSSS funciona através da codificação da mensagem a ser enviada através de uma sequência periódica pseudoaleatória de espalhamento, que é gerada a uma taxa mais rápida do que os bits da mensagem. Esta sequência deve possuir uma autocorrelação similar a do ruído branco, mas, como é periódica, esta se repete em períodos fixos, o que por sua vez espalha o espectro da mensagem por um determinado fator. Com isso, tendo um canal utilizado por múltiplos usuários, desde que o receptor conheça as sequências utilizadas e que a correlação entre elas seja baixa, é possível decodificar os sinais recebidos corretamente.

A OFDM, por sua vez, pode ser entendida como uma forma de comunicação utilizando múltiplas subportadoras ortogonais simultaneamente, as quais operam com taxas de dados menores do que o cenário com somente uma portadora principal. Dessa maneira, o espectro da mensagem original é distribuído ao redor das frequências das subportadoras, entretanto, como estas são ortogonais, é possível recuperar corretamente o sinal transmitido através de demodulação coerente [11]. A grande vantagem disso é que, como cada portadora trabalha com uma taxa de dados reduzida, o tempo de símbolo OFDM é aumentado, o que por sua vez evita diversos problemas de interferência intersimbólica. Entretanto, mesmo com um maior tempo de símbolo, a taxa de dados resultante da comunicação ainda é alta, visto que diversas portadoras carregam informação.

3.1.2 Subcamada MAC

A subcamada MAC é responsável por, entre outros, indicar o endereço da rede para onde serão entregues os pacotes, determinar em um sistema multiusuário quem será o próximo a ter acesso ao canal, bem como coordená-los de maneira que todos possam acessá-lo individualmente. A IEEE 802.11-2012 especifica para isso um formato específico de quadro, chamado de unidade de dados do protocolo PLCP (PPDU[5]), que a camada física padroniza para ser transmitido, além de um conjunto de funções que coordenam o acesso ao meio.

É possível destacar a função de coordenação distribuída (DCF[6]), que é responsável por controlar o acesso ao meio via um protocolo de detecção de portadoras. Nesse aspecto, caso um usuário deseje utilizar o meio para transmitir algum dado, primeiramente ele deve detectar se o canal está sendo usado, a fim de evitar colisões no canal. Em caso afirmativo, é gerado um período de tempo aleatório que o usuário deve esperar para que ele possa tentar transmitir dados novamente. Este tempo aumenta exponencialmente caso o canal continue ocupado durante múltiplas tentativas de acesso [8].

Os padrões mais recentes da IEEE 802.11, como por exemplo a versão 802.11n, especificam também outros recursos como o retorno de um bloco de quadros de confirmação, transmissão de quadros de maneira agregada a fim de reduzir o tempo gasto ganhando acesso ao meio e criando uma seção de comunicação, etc [12].

[5] Do inglês *PLCP Protocol Data Unit*.
[6] Do inglês *Distributed Coordination Function*.

3.1.3 Padrão IEEE 802.11n

Em 2009 foi publicada uma emenda denominada de IEEE 802.11n, que visava principalmente elevar a velocidade de comunicação em redes locais [12]. Dentre as melhorias na camada física, destaca-se o novo suporte a sistemas de múltiplas entradas e saídas, permitindo a comunicação ponto a ponto através de múltiplas antenas. Além disso, foi possível dobrar a banda dos canais OFDM dos padrões anteriores de 20 MHz para 40 MHz. Vale destacar que, como a maioria dos canais são separados por 5 MHz entre si, há sobreposição da banda ocupada entre canais adjacentes.

Em se tratando do OFDM, esta versão utiliza as mesmas formas de modulação e codificação que as normas anteriores, com a adição de 4 subportadoras não utilizadas, totalizando 56, das quais 52 transportam dados e 4 servem para a calibração e estimação do canal de comunicação. Dentre as configurações citadas pela IEEE 802.11n, destaca-se na Tabela 2 os dados: índice de modulação e codificação (MCS[7]); tipo de modulação; a taxa de código R, que especifica a razão entre bits de informação e o total de bits transmitidos; o número de bits de informação por símbolo N_{DBPS}; e a taxa de dados referente ao MCS para dois Intervalos de Guarda (GI[8]), cuja função é prevenir a interferência intersimbólica causada pela transmissão de blocos de informação consecutivos.

Tabela 2 – Parâmetros do OFDM para 1 par de antenas e 20 MHz de banda de canal

MCS	Modulação	R	N_{DBPS}	Taxa de dados (Mbps)	
				GI de 800 ns	GI de 400 ns
0	BPSK	1/2	26	6,5	7,2
1	QPSK	1/2	52	13,0	14,4
2	QPSK	3/4	78	19,5	21,7
3	16-QAM	1/2	104	26,0	28,9
4	16-QAM	3/4	156	39,0	43,3
5	64-QAM	2/3	208	52,0	57,8
6	64-QAM	3/4	234	58,5	65,0
7	64-QAM	5/6	260	65,0	72,2

Fonte: Adaptado de [9].

Os MCSs mostrados na Tabela 2 indicam as possíveis combinações entre o tipo de modulação e o código corretor de erro utilizados. Estes códigos têm o objetivo de diminuir a probabilidade de um erro na recepção da mensagem, de maneira que a taxa de dados da comunicação seja maior do que no caso em que toda a mensagem fosse retransmitida um certo número de vezes. A norma recomenda o uso do código convolucional das versões anteriores, que garante um $R = 1/2$. Para aumentar o valor da taxa de código, geralmente a saída do

[7] Do inglês *Modulation Coding Scheme*.
[8] Do inglês *Guard Interval*.

codificador convolucional tem bits redundantes descartados através de um método denominado de puncionamento. Uma análise dos efeitos deste código corretor de erro é feita adiante.

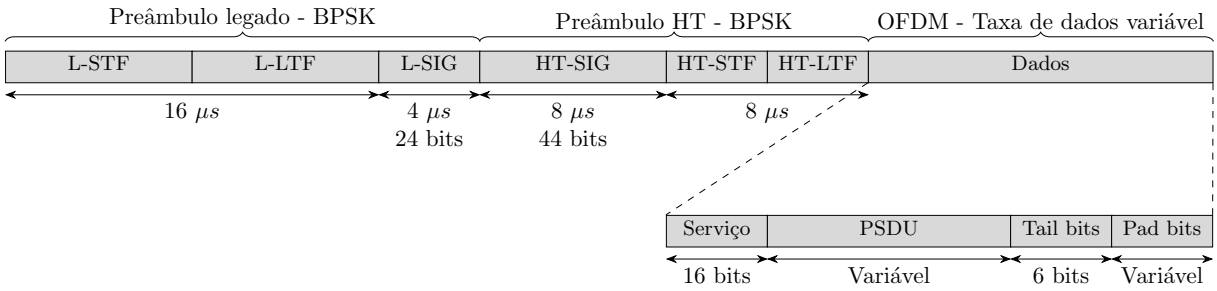
Quanto à camada física, a norma define três modos de transmissão de dados, cada um com um tipo de quadro PLCP diferente. São eles:

1. O modo de Alta Taxa de Dados Misto (HT-M, do inglês *High-Throughput Mixed*), cujo quadro PLCP é compatível com as versões anteriores da norma;
2. O modo legado, que usa as mesmas configurações que as normas anteriores (802.11a ou 802.11g);
3. O modo de Alta Taxa de Dados Greenfield (HT-GF, do inglês *High-Throughput Greenfield*), que permite comunicação somente entre dispositivos 802.11n [12].

Dentre os modos de transmissão citados, são obrigatórios o modo HT-M e o modo legado. A implementação do modo HT-GF acaba sendo usada somente quando a velocidade de comunicação é um fator crítico em um sistema, visto que uma quantidade menor de bits auxiliares é exigida do que no modo HT-M. Apesar disso, o aumento na taxa de dados efetiva do sistema acaba sendo pequeno, portanto usam-se mais os modos HT-M e legado, o último no caso da transmissão de pacotes menores como os de Confirmação (ACK, do inglês *Acknowledgement*) [12].

Dessa maneira, é mostrado como referência na Figura 4 o formato do quadro PPDU no modo HT-M. Este pode ser dividido em duas seções principais: preâmbulo e dados. O preâmbulo contém os campos de treinamento curto e longo (STF[9] e LTF[10]), utilizados para a calibração do receptor, bem como um campo de sinal (SIG), que descreve a taxa de dados utilizada e o tamanho do quadro de dados, denominado de unidade de dados de serviço do PLCP (PSDU[11]). É necessário haver uma seção de preâmbulo tanto para o modo legado quanto o modo de alta taxa de dados.

Figura 4 – Formato do quadro PPDU na IEEE 802.11n para 1 par de antenas



Fonte: Adaptado de [9].

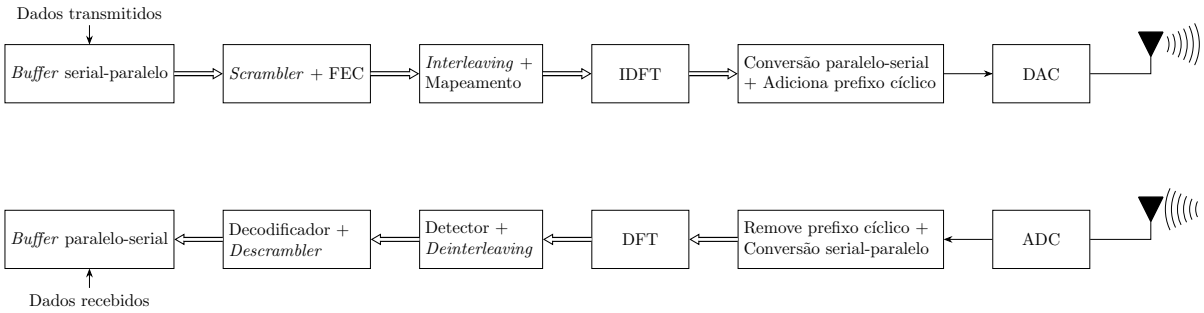
[9] Do inglês *Short Training Field*.
[10] Do inglês *Long Training Field*.
[11] Do inglês *PLCP Service Data Unit*.

A seção de dados contém a informação efetiva a ser transmitida no quadro PSDU, geralmente composto por um quadro MAC com cabeçalho. Além disso, são enviados os quadros de serviço, responsável por inicializar o *scrambler* (misturador), e os bits de cauda e preenchimento do quadro (*tail/pad* bits). O *scrambler* é um processo que embaralha os bits a serem transmitidos, de maneira a remover longas sequências de bits idênticos. Já os bits de cauda são utilizados para terminar corretamente o processo de codificação/decodificação quando se usa um codificador convolucional, que é recomendado pela norma IEEE 802.11n. Os bits de preenchimento servem para garantir que o pacote final enviado tenha um número par de símbolos OFDM [12].

3.2 VISÃO GERAL DA OFDM

A OFDM é um método de espalhamento espectral utilizado na IEEE 802.11n. Seu princípio básico de funcionamento é a utilização da banda do canal por diversas subportadoras ortogonais entre si [11]. Na Figura 5, é apresentado um diagrama de blocos da implementação da OFDM usando a Transformada Discreta de Fourier (DFT[12]) e a DFT Inversa (IDFT[13]), que podem ser calculadas de maneira eficiente pelo algoritmo da Transformada Rápida de Fourier (FFT, do inglês *Fast Fourier Transform*).

Figura 5 – Implementação da OFDM pela DFT



Fonte: Elaborado pelo autor.

No processo de transmissão digital de dados, primeiramente a mensagem a ser transmitida é dividida em "blocos" de bits de tamanho N_{BQAM} através do *buffer* serial-paralelo e do processo de codificação para correção de erros (FEC, do inglês *Forward Error Correction*). Como cada um destes blocos passa por um processo de Modulação de Amplitude em Quadratura (M-QAM[14]), tem-se que $N_{BQAM} = \log_2(M)$, onde M equivale ao número de possíveis símbolos QAM na mensagem. Assim, para o caso de 64-QAM os blocos armazenam 6 bits. Em seguida, a fim de reduzir a probabilidade de longas sequências de bits idênticos serem enviadas para o codificador FEC, os bits passam pelo *scrambler*. No caso da IEEE 802.11n, recomenda-se o uso de um codificador convolucional como FEC, que é detalhado nas próximas seções. A ortogonalidade entre cada subportadora da OFDM é obtida ao se fazer com que o espaçamento na frequência

[12] Do inglês *Discrete Fourier Transform*.
[13] Do inglês *Inverse Discrete Fourier Transform*
[14] Do inglês *Quadrature Amplitude Modulation*.

Δf seja igual à taxa de símbolos $1/T_{OFDM}$ do sistema [11], em que T_{OFDM} é a duração de cada símbolo OFDM sem o intervalo de guarda.

Em seguida, os blocos de dados passam por um processo de entrelaçamento (*interleaving*). Isto é feito através da troca de determinados bits entre os blocos, de maneira a evitar que a quantidade de erros em um deles ultrapasse o número máximo admitido pelo código corretor. Dessa forma, o entrelaçador distribui uma sequência de erros em diversos grupos. Como ponto negativo, o receptor precisa apresentar essa mesma divisão de blocos de bits em um *buffer* para realizar a decodificação, o que pode causar um atraso neste processo. Assim, cada bloco resultante de N_{BQAM} bits é mapeado a um símbolo QAM X_k de acordo com uma constelação específica. Um exemplo de constelação quadrada é mostrado adiante.

Determinando-se os símbolos QAM a serem enviados, estes passam pela IDFT, a fim de criar o sinal a ser transmitido no domínio do tempo. Na prática, usa-se um valor de pontos equivalente a uma potência de 2 para o cálculo da IDFT, a fim de possibilitar a implementação de um algoritmo mais simples de cálculo, além de alocar espaço para subportadoras piloto, que são necessárias para as sequências de treinamento entre o transmissor e receptor. Para o caso da OFDM com 52 subportadoras, usa-se 4 subportadoras piloto e uma IDFT de 64 pontos. Além disso, é comum a inserção de um prefixo à sequência de dados a fim de evitar interferência intersimbólica. Outro método utilizado é a inserção de um tempo de guarda entre transmissões sucessivas. Ambos são utilizados no padrão IEEE 802.11n.

Por fim, a sequência discreta final de dados passa por um conversor digital-analógico (DAC, do inglês *Digital-to-Analog Converter*) e é transmitida para o canal, que pode ser o ar livre. Os passos da recepção de uma mensagem são análogos ao da transmissão, mas em ordem reversa. Vale destacar que a propriedade de ortogonalidade entre as subportadoras diz que, dado um período de transmissão de símbolos OFDM T_{OFDM} em um sistema com N subportadoras [11]

$$\int_0^{T_{OFDM}} \cos(2\pi f_k t + \phi_k)\cos(2\pi f_j t + \phi_j)dt = 0, \tag{3.1}$$

onde $f_k - f_j = n/T_{OFDM},\ n = 1,2,...,N-1$. No processo de demodulação coerente, utiliza-se a propriedade de ortogonalidade para se identificar os símbolos QAM transmitidos em cada subportadora. Dessa maneira, calcula-se a integral durante o período T_{OFDM} entre o sinal recebido e duas funções de referência para cada subcanal. Na transmissão em quadratura, estas funções são

$$\psi_1(t) = \sqrt{\frac{2}{T_{OFDM}}}\cos(2\pi f_k t + \phi_k), \quad 0 \le t \le T_{OFDM} \tag{3.2}$$

$$\psi_2(t) = -\sqrt{\frac{2}{T_{OFDM}}}\sin(2\pi f_k t + \phi_k), \quad 0 \le t \le T_{OFDM}. \tag{3.3}$$

O sinal recebido no k-ésimo subcanal pode ser definido por

$$r_k(t) = \sqrt{\frac{2}{T_{OFDM}}}|C_k|A_{ki}\cos(2\pi f_k t + \phi_k) + \sqrt{\frac{2}{T_{OFDM}}}|C_k|A_{kq}\sin(2\pi f_k t + \phi_k) + \eta_k, \tag{3.4}$$

onde $|C_k|$ é a atenuação da k-ésima subportadora, A_{ki} e A_{kq} representam o símbolo QAM transmitido X_k através da relação $X_k = A_{ki} + jA_{kq}$, ϕ_k é o atraso de fase gerado pela transmissão da mensagem pelo canal e η_k é o ruído inserido na recepção do sinal $r_k(t)$. O conhecimento do receptor de C_k e ϕ_k pode ser obtido através da transmissão de sinais de treinamento durante os períodos de treinamento longo e curto da IEEE 802.11n. Na subseção 3.2.2 a forma de um sinal modulado em QAM e o método de mapeamento dos bits é visto com maiores detalhes.

Portanto, as integrais sobre o período T_{OFDM} do sinal recebido no k-ésimo subcanal e as funções de referência geram dois valores numéricos, que podem ser representados pelo símbolo complexo $Y_k = |C_k|X_k + \eta_k$. Dado que o receptor conhece a atenuação C_k, este fator pode ser compensado e o valor complexo resultante, isto é, o símbolo QAM transmitido X_k acrescido de ruído, é utilizado para identificar os bits mapeados na mensagem. Este processo é realizado de maneira discreta através da DFT para cada subcanal que carrega informação.

3.2.1 Probabilidade de erro de detecção de bit da OFDM

Haja visto que, em taxas de dados mais altas, diversas subportadoras moduladas em QAM são enviadas pelo canal, tem-se que a taxa de erro de bit (BER, do inglês *Bit Error Rate*) da OFDM pode ser diretamente correlacionada com a probabilidade de erro de transmissão de um símbolo QAM. Dito isso, nesta seção, são definidos alguns parâmetros de probabilidade de erro pertinentes na comunicação OFDM.

A probabilidade de erro de bit P_b da OFDM com N subportadoras que carregam informação pode ser definida por [10]

$$P_b = \frac{\sum_{i=1}^{N} K_i \cdot P_b[i]}{\sum_{i=1}^{N} K_i}, \tag{3.5}$$

em que K_i e $P_b[i]$ são o número de bits mapeados e a BER da i-ésima subportadora respectivamente. Nota-se que, para os casos em que todas as subportadoras utilizam o mesmo esquema de modulação, a expressão acima pode ser reescrita como

$$P_b = \frac{1}{N} \sum_{i=1}^{N} P_b[i], \tag{3.6}$$

tal que fica claro que, ao se considerar que as BERs de cada subportadora são iguais, a taxa de erro de bit da OFDM equivale à taxa de erro do tipo de modulação utilizada. Apesar de que geralmente este não é o caso, haja visto a presença de algoritmos que procuram alocar um nível ótimo de potência transmitida para cada subportadora a fim de minimizar a BER resultante [10], esta abordagem permite o cálculo simplificado da BER em função do tipo de modulação quando se desconhecem as características de frequência do canal, isto é, a atenuação de módulo e distorção de fase no domínio da frequência.

3.2.2 Definição da modulação M-QAM

O esquema de modulação QAM consiste em mapear grupos de bits em duas portadoras em quadratura. As M formas de onda de um sinal modulado em QAM $s_m(t)$ podem ser descritas por [11]

$$s_m(t) = \mathrm{Re}[(A_{mi} + jA_{mq})p(t)e^{j2\pi f_c t}] \tag{3.7}$$

$$= A_{mi}p(t)\cos(2\pi f_c t) - A_{mq}p(t)\sin(2\pi f_c t), \quad m = 1,2,...,M, \tag{3.8}$$

em que A_{mi}, A_{mq} representam as amplitudes nas quais os grupos de bits foram mapeados, $p(t)$ representa a forma de onda do pulso utilizado e f_c é a frequência das portadoras.

Este tipo de modulação pode ser interpretado como uma combinação de modulação em amplitude e fase, em que o mapeamento dos bits pode ser livremente definido. Este mapeamento é comumente ilustrado por uma constelação, que relaciona cada sequência de bits a serem transmitidos com cada possibilidade de módulo e fase referente à forma de onda s_m. Um exemplo de constelação quadrada de 64-QAM utilizado na IEEE 802.11-2012 é ilustrado na Figura 6, tal que $\log_2(64) = 6$ bits são mapeados nos pontos mostrados na imagem, que representam os símbolos QAM $X_k = A_{mi} + jA_{mq}$. Dentre as vantagens da constelação quadrada, destaca-se sua fácil implementação ao se utilizar dois moduladores de amplitude associados às duas portadoras em quadratura.

3.2.2.1 Probabilidade de erro de bit com 64-QAM usando constelação quadrada em canal AWGN

Nesta seção é apresentada a expressão de BER referente ao esquema de modulação 64-QAM utilizando constelação quadrada em um canal com Ruído Gaussiano Branco Aditivo (AWGN, do inglês *Additive White Gaussian Noise*). Não são analisados os métodos recepção ótimos, mas ressalta-se que este processo se dá pela detecção coerente utilizando filtros casados com as portadoras em quadratura [11].

De acordo com Proakis e Salehi[11], a probabilidade de erro de recepção de um símbolo M-QAM com constelação quadrada em um canal AWGN é dada por

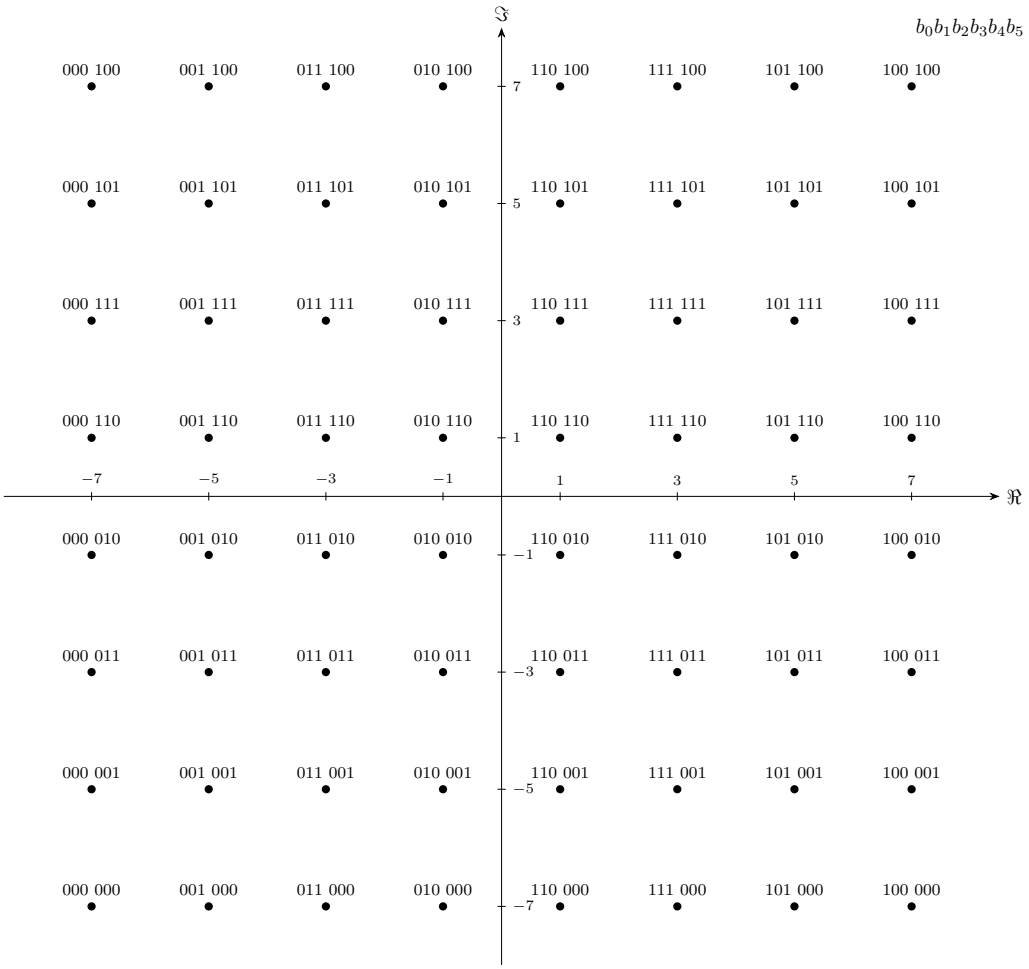
$$P_{se} = 4\left(1 - \frac{1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{3\log_2 M}{M-1}\gamma_b}\right) \times \left(1 - \left(1 - \frac{1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{3\log_2 M}{M-1}\gamma_b}\right)\right), \tag{3.9}$$

em que $Q(x)$ é o complemento da função distribuição cumulativa referente a uma variável aleatória gaussiana de média 0 e desvio padrão unitário; e γ_b é a relação sinal-ruído (SNR, do inglês *Signal-to-Noise Ratio*) por bit, que pode ser diretamente relacionada com a SNR do sistema.

Além disso, devido ao uso do código gray da constelação da Figura 6, a probabilidade de erro de bit pode ser aproximada para [13]

$$P_b \approx \frac{P_{se}}{\log_2 M}. \tag{3.10}$$

Figura 6 – Constelação quadrada de 64-QAM



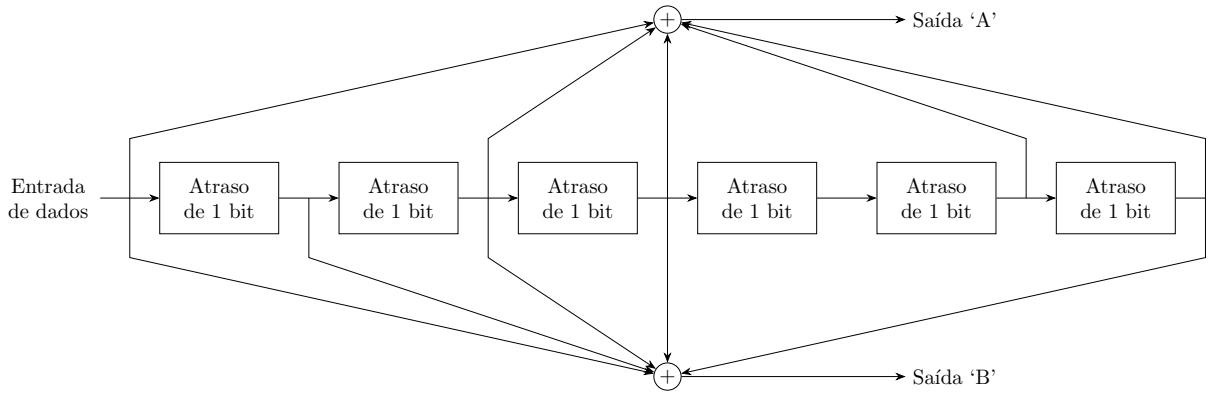
Fonte: Adaptado de [9].

3.2.3 Influência da codificação convolucional e do decodificador de Viterbi com o método de decisão rígida

A norma IEEE 802.11-2012 especifica o codificador convolucional da Figura 7 para reduzir a probabilidade de erro de recepção dos dados. A influência de se utilizar um codificador convolucional para esta finalidade não é descrita em detalhes, mas é dada uma breve explicação do que é feito e do motivo de usá-lo ao invés de simplesmente retransmitir os dados da mensagem. O funcionamento detalhado do processo de codificação convolucional e decodificação ótima pode ser encontrado na literatura [10].

O codificador da Figura 7 funciona como um registrador de deslocamento, tal que, à medida que novos bits são deslocados à direita 1 por vez, formam-se duas saídas para cada nova configuração de bits dentro do registrador. Estas saídas são transmitidas em ordem: primeiro a saída 'A' depois a 'B'. É notável que, como elas dependem até do 6º bit anterior, é introduzido um aspecto de memória nas sequências de bits 'AB' transmitidas. Dessa maneira, ao se utilizar um decodificador otimizado, tal como o decodificador de Viterbi, a performance do sistema aumenta em comparação com o caso em que a mensagem fosse transmitida duas vezes, visto

Figura 7 – Codificador convolucional da IEEE 802.11-2012



Fonte: Adaptado de [9].

que o receptor realiza a comparação entre diversas sequências de bits recebidos ao invés da comparação somente no momento em que os bits chegam.

A taxa de código do codificador da Figura 7 equivale a $R = 1/2$, visto que para cada bit de entrada cria-se 2 saídas, sendo este o esquema utilizado no MCS0, conforme mostra a Tabela 2. A fim de aumentar as taxas de códigos em situações que exigem maior taxa de dados, o mesmo codificador convolucional da Figura 7 é utilizado, mas utilizando um processo denominado *puncturing* (do inglês, perfuração). Nesse sentido, após uma sequência de dados de um tamanho específico ser gerada pelo codificador, alguns bits desta sequência são simplesmente descartados ao passo em que os demais são enviados. Apesar de que isso aumenta a probabilidade de erro do sistema, o uso do codificador convolucional e de um decodificador ótimo ainda compensa, desde que o receptor apresente uma SNR acima de um valor crítico.

3.2.3.1 Limite superior da probabilidade de erro com decodificador de Viterbi operando pelo método de decisão rígida

De acordo com Proakis e Salehi[11], a probabilidade de erro do primeiro evento, ou seja a probabilidade de erro no decodificador, ao se utilizar o algoritmo de Viterbi com o método de decisão rígida (*hard decision*) pode ser limitada por

$$P_{de} \leq \sum_{d=d_{free}}^{\infty} a_d P_2(d), \tag{3.11}$$

onde os coeficientes a_d dependem do codificador convolucional utilizado e podem ser calculados analiticamente, de maneira que já há resultados tabelados na literatura [14]. Já $P_2(d)$ pode ser calculado por

$$P_2(d) = \begin{cases} \sum_{k=(d+1)/2}^{d} \binom{d}{k} p^k (1-p)^{n-k} & \text{, para } d \text{ ímpar} \\ \frac{1}{2}\binom{d}{\frac{1}{2}d} p^{d/2}(1-p)^{d/2} + \sum_{k=d/2+1}^{d} \binom{d}{k} p^k (1-p)^{n-k} & \text{, para } d \text{ par} \end{cases}, \tag{3.12}$$

em que p representa a probabilidade de erro de bit para um tipo de modulação, como por exemplo P_b da Equação (3.10).

3.2.4 Modelagem do canal e obtenção da SNR por bit

A fim de se estimar o alcance em que um par de dispositivos podem se comunicar através de Wi-Fi, pode-se utilizar um modelo que caracterize a atenuação de um sinal transmitido no ambiente. Nesse aspecto, é importante que o modelo a ser escolhido seja adequado a situações de curta distância em ambientes fechados, que é o caso deste trabalho.

De acordo com Zamalloa e Krishnamachari[15], o modelo empírico log-normal descrito pela Equação (3.13) é capaz de descrever bem ambientes com linha de visada e curta distância com a presença de desvanecimento multipercurso. Este último efeito ocorre pelo recebimento de um sinal transmitido por diversos caminhos físicos, visto que a onda transmitida pode refletir em obstáculos. Nesse aspecto, estes sinais podem interagir entre si de maneira a prejudicar a comunicação.

$$PL(d) = PL(d_0) + 10\eta \log_{10}\left(\frac{d}{d_0}\right) + X_\sigma. \tag{3.13}$$

Na Equação (3.13), $PL(d)$ representa a atenuação do sinal em função da distância, $PL(d_0)$ é a atenuação média a uma distância de referência, η é o coeficiente de atenuação exponencial que o canal apresenta e X_σ é um processo aleatório de média 0 e desvio padrão σ a fim de representar desvanecimento multipercurso. Todos os termos citados podem ser obtidos empiricamente e são dados em dB.

Estimando-se o comportamento do canal, a relação sinal-ruído do lado do receptor pode ser proposta como

$$\text{SNR}_{rx,dB} = P_{tx,dB} - L_{0,dB} - L_{m,dB} - P_{n,dB}, \tag{3.14}$$

onde $P_{tx,dB}$ é a potência do transmissor, $L_{0,dB}$ são as perdas de atenuação do canal, $L_{m,dB}$ são perdas de redundância e $P_{n,dB}$ é a potência do ruído térmico do lado do receptor, todos os termos em dB. A potência do ruído térmico pode ser calculada por [13]

$$P_{n,dB} = 10\log_{10}((F-1)K_bT_0B), \tag{3.15}$$

em que K_b é a constante de Boltzmann, T_0 é a temperatura ambiente em Kelvin, B é a largura de banda do sistema em Hz e F é a figura de ruído do receptor, que consiste em um número associado ao calor gerado pelos seus circuitos eletrônicos e sistema de antenas.

Havendo a SNR do lado do receptor, é possível relacioná-la com a SNR por bit γ_b, necessária para o cálculo da probabilidade de erro na transmissão de símbolos. Para a OFDM, seguindo o modelo de Câmara e Hoefel[16], tem-se que

$$\text{SNR}_{rx} = \gamma_b \left(\frac{N_{data} + N_{pilot}}{N_{fft}}\right)(N_{BPSC}R), \tag{3.16}$$

onde N_{data} é o número de subportadoras que carregam informação, N_{pilot} é o número de subportadoras piloto, N_{fft} é o total de subportadoras geradas pela implementação da OFDM pela FFT, N_{BPSC} é o número de bits por subportadora e R é a taxa de código.

3.2.5 Probabilidade de sucesso de transmissão de quadros

Conhecendo-se a SNR por bit no lado do receptor, é possível calcular qual a probabilidade de que uma transação de quadros entre dois dispositivos ocorra com sucesso ao se utilizar os protocolos da IEEE 802.11n. Nessa análise, assumiu-se que ambos os dispositivos apresentam a mesma característica de potência transmitida e que são adaptados ao padrão 802.11n, trabalhando no modo HT-M.

Em Qiao, Choi e Shin[17], foi apresentado que a probabilidade de erro na recepção de um pacote de dados de h bytes, supondo o uso de um decodificador de Viterbi com o método de decisão rígida, pode ser limitada por

$$P_{pe}^{(m)}(h,\gamma_b) \leq 1-[1-P_{de}^{(m)}(\gamma_b)]^{8h}, \tag{3.17}$$

onde o sobrescrito (m) indica o MCS utilizado na transmissão do pacote. Ressalta-se que, em uma comunicação entre dois dispositivos, este índice muda dependendo do que se está transmitindo. Dessa maneira, o termo $P_{de}^{(m)}(\gamma_b)$ pode ser calculado utilizando a Equação (3.17) em função da SNR por bit a uma distância arbitrária.

Nesse aspecto, considerando que se deseja calcular a probabilidade de sucesso na transmissão de um quadro de dados de l bytes no MCS m, tem-se que tanto a transmissão do quadro PPDU da Figura 4 quanto a transmissão de um quadro ACK devem ocorrer com sucesso. De acordo com a IEEE 802.11-2012, a transmissão de quadros ACK deve ser realizada na taxa de dados menor e mais próxima o possível da taxa utilizada na sequência de dados inicial recebida. Apesar disso, a fim de apresentar compatibilidade com as versões anteriores da norma quando a maior taxa mandatória era reduzida, é comum que se utilize o modo de 24 Mbps da 802.11g (MCS3), como a maior taxa de dados de quadros de resposta [18].

Dito isso, o limitante superior da probabilidade de erro ao se enviar o quadro PPDU da Figura 4 com l bytes de dados da camada de aplicação pode ser definida, ao se utilizar a Equação (3.17) para cada trecho com um MCS (m) distinto, por

$$P_{e,data}(l,\gamma_b,m) = 1-[1-P_{pe}^{(0)}(68/8,\gamma_b)]\cdot[1-P_{pe}^{(m)}(22/8+l,\gamma_b)], \tag{3.18}$$

onde os termos $68/8$ e $22/8$ representam o número de bytes do preâmbulo e serviço mais cauda, respectivamente, do quadro PPDU. Ressalta-se que os campos do preâmbulo são sempre transmitidos com BPSK (MCS0).

De maneira análoga, o limitante superior da probabilidade de erro ao se enviar um quadro ACK com no MCS3 (16-QAM) pode ser calculado por

$$P_{e,ack}(\gamma_b) = 1-[1-P_{pe}^{(0)}(68/8,\gamma_b)]\cdot[1-P_{pe}^{(3)}(16.75,\gamma_b)], \tag{3.19}$$

visto que , no quadro ACK, são transmitidos 14 bytes de dados em adição aos 16+6 bits de serviço e cauda, totalizando 16.75 bytes [9].

Com os limites de probabilidade de erro dos dois tipos de quadro envolvidos calculados, a probabilidade de sucesso em uma seção de comunicação, em que l bytes são transmitidos entre dois dispositivos de características semelhantes e um quadro ACK é transmitido no MCS3, pode ser calculada por

$$P_{s,tx} = [1 - P_{e,data}(l, \gamma_b, m)] \cdot [1 - P_{e,ack}(\gamma_b)]. \tag{3.20}$$

3.2.6 Considerações finais

O equacionamento apresentado neste capítulo descreve a probabilidade de sucesso em uma seção de comunicação $P_{s,tx}$, em que l bytes são transmitidos entre dois dispositivos semelhantes se comunicando através do padrão IEEE 802.11n em um canal AWGN. Além disso, a modelagem log-normal pode ser utilizada para definir a atenuação dos sinais transmitidos e, como consequência, estabelecer uma relação entre $P_{s,tx}$ e a distância entre os dispositivos. Estas relações são usadas adiante para se definir um limite máximo de distância para a comunicação entre o ESP 32 e o computador pessoal executando o servidor TCP/IP[15] no processo de armazenamento de dados.

[15] Protocolo de controle de transmissão/protocolo de internet, do inglês *Transmission Control Protocol/Internet Protocol*.

4 MICROCONTROLADORES UTILIZADOS PARA COMUNICAÇÃO SEM FIO

Neste capítulo é feita uma revisão bibliográfica dos MCUs mais usados para aplicações com comunicação sem fio. É apresentado um comparativo entre eles, a fim de destacar suas limitações e identificar a tecnologia mais adequada ao projeto.

4.1 COMPARATIVO ENTRE MICROCONTROLADORES

Um microcontrolador é um sistema integrado que reúne em um único chip um microprocessador e diversos periféricos, tais como memória de dados e RAM[1], temporizadores, interfaces de comunicação, etc [19]. Com os avanços da microeletrônica, foi possível integrar nestes componentes cada vez mais recursos especializados, fazendo com que hoje existam diversos MCUs que sejam capazes de solucionar um mesmo problema.

A fim de selecionar um dispositivo capaz de realizar as funções de comunicação do projeto em tempo hábil, é necessário fazer uma pesquisa bibliográfica nos modelos mais utilizados em projetos hoje. Nesse sentido, os principais indicativos de qualidade dos MCUs estudados são:

- Presença de uma interface de comunicação serial compatível com o DSP de controle;
- Suporte à comunicação sem fio já integrada;
- Taxa de transmissão de dados máxima (caso haja um transreceptor embutido);
- Frequência máxima de operação;
- Custo.

Como conversores estáticos são capazes de operar com frequências na ordem de MHz [20], mesmo com uma pequena quantidade de dados a serem enviados por ciclo, correspondentes às medições e valores de referência de tensão e corrente do conversor, espera-se uma alta taxa de transmissão de dados, visto que se pretende salvar todas as variáveis geradas a cada ciclo de chaveamento. Dessa maneira, é fundamental analisar a maior taxa de dados dos MCUs pertinentes.

Além disso, um parâmetro secundário de qualidade no sistema projetado é a potência exigida pelo sistema digital, que normalmente é proporcional à frequência de *clock* do microprocessador, devido às perdas de comutação dos semicondutores [1]. Dessa maneira, caso seja possível utilizar uma frequência menor do que a máxima e ainda cumprir as funções desejadas, tem-se a possibilidade de reduzir o consumo de energia do sistema. A máxima frequência de *clock* também é um indicativo da maior velocidade de transmissão de dados em protocolos de comunicação serial.

[1] Memória de acesso aleatório, do inglês *Random Access Memory*.

Na literatura, Ojo et al.[21] fizeram uma revisão das plataformas de prototipagem mais utilizadas em sistemas de IoT[2]. As plataformas foram classificadas em baixo, médio e alto nível com base no poder de processamento, tamanho da memória disponível, etc. Foram selecionados alguns microcontroladores de plataformas de nível baixo e médio para serem discutidos a seguir. Os dados referentes aos MCUs e módulos *wireless* pesquisados são mostrados na Tabela 3, onde os preços dos dispositivos foram consultados *online* [3].

O CC2538 é um MCU fabricado pela *Texas Instruments* destinado a aplicações do padrão IEEE 802.15.4 [22], isto é, para redes com pouco tráfego de dados e baixa velocidade na comunicação. Dessa maneira, é um MCU ideal para aplicações em redes ZigBee, que é um protocolo baseado neste padrão da IEEE que resulta em um baixo consumo de energia [8]. O microprocessador é um Cortex-M3 com frequência de *clock* de até 32 MHz, sendo que o sistema já inclui um módulo de rádio apropriado para o padrão IEEE 802.15.4 na banda de 2,4 GHz. O consumo de corrente do microprocessador na máxima frequência de operação é de 13 mA.

Tabela 3 – Especificações de microcontroladores e módulos utilizados para comunicação sem fio

MCU/Módulo	Com. serial			Wireless int.	Taxa de dados	Freq. (MHz)	Preço ($)
	SCI	I2C	SPI				
CC2538	2	1	2	Sim	250 kbps	32	11,27
ATSAMR21-G18A	5 comuns			Sim	250 kbps	48	6,87
MSP430F16X	2	1	2	Não	-	8	26,61
ATSAMD21-G18	6 comuns			Não	-	48	4,33
ATmega256-RFR2	2	1	1	Sim	2 Mbps	16	7,85
ATSAMW25	1	1	1	Sim	72,2 Mbps	48	16,64
ESP8266EX	1	1	1	Sim	72,2 Mbps	160	1,60
ESP32	3	2	3	Sim	72,2 Mbps	240	2,00

Fonte: adaptado de [21]

O MSP430F16X, também fabricado pela *Texas Instruments*, apresenta uma arquitetura RISC de 16 bits com frequência de *clock* máxima de 8 MHz. A característica marcante deste MCU é o baixo consumo de corrente do sistema, tal que o sistema consome menos que 1 mA com 1 MHz de frequência de operação da CPU[4]. Em contrapartida, este CI não apresenta um módulo *wireless* integrado, que geralmente é o periférico que mais consome corrente em sistemas de comunicação.

A Atmel disponibiliza diversos MCUs voltados ao campo de IoT, dentre os quais é possível destacar: ATSAMR21G18A, ATSAMD21G18, ATmega256RFR2 e ATSAMW25. O

[2] Internet das coisas, do inglês *Internet of Things*.
[3] Consultado no endereço <https://br.mouser.com/> em 14 de dezembro de 2022.
[4] Unidade de processamento central, do inglês *Central Processing Unit*.

único modelo sem transreceptor já integrado no chip é o ATSAMD21G18, que é caracterizado pelo grande número de portas de entrada ou saída de uso genérico (GPIO [5]) [23]. Além disso, o MCU possui um multiplicador de *hardware*, possibilitando uma instrução de multiplicação em um único ciclo de *clock*.

O ATSAMR21G18A é uma versão do modelo ATSAMD21G18 mas com um transreceptor adequado ao padrão IEEE 802.15.4 embutido [24]. Além de manter as características do MCU anterior, destaca-se o fato do dispositivo apresentar um *Wake on Radio* que permite que a CPU seja ativada ao se receber um evento como o recebimento de uma mensagem pelo módulo de comunicação sem fio, permitindo economizar energia.

O ATmega256RFR2 é um modelo AVR, que é uma família de 8 bits de arquitetura RISC avançada. Dessa maneira, a maioria das instruções do microprocessador são executadas com apenas um ciclo de *clock*. A maior corrente, incluindo tanto a CPU quanto o módulo *wireless*, atingida pelo dispositivo é de 18,6 mA, o que constitui um valor bem baixo ao se considerar o baixo custo do MCU.

O último MCU da Atmel pesquisado é o ATSAMW25, que também se trata de uma versão do modelo ATSAMD21G18, mas com um transreceptor adequado ao padrão IEEE 802.11 b/g/n, isto é, Wi-Fi [25]. Dessa forma, o dispositivo é capaz de lidar com taxas de dados maiores, ao custo de um maior consumo de energia quando isto é necessário. Vale destacar que as maiores taxas de dados são atingidas, conforme especifica o padrão IEEE 802.11, através de mais bits por símbolo. No caso da taxa de 72,2 Mbps, é usada por exemplo a modulação 64 QAM, de maneira que 6 bits são codificados em um único símbolo a ser transmitido ou recebido.

Por fim, o ESP8266EX da Espressif é um módulo Wi-Fi de baixo custo que contém um microprocessador L106 de 32 bits RISC da Tensilica funcionando em até 160 MHz de frequência de *clock* [26]. A placa contém até 16 pinos de GPIO e tem suporte para até 3 interfaces de SCI, I2C e SPI. O módulo também apresenta outras versões com perfis de desempenho melhores, caso seja necessário para a aplicação. Além disso, a série de MCUs ESP 32 utiliza o mesmo microprocessador, mas com a arquitetura de dois núcleos e frequência de *clock* de até 240 MHz. Estão disponíveis para a aplicação três canais UART e SPI, além de dois canais para I2C [27].

Sendo assim, considerando-se os requisitos desejados e as características dos MCUs mencionados, verifica-se que o ESP 32 é aquele que atende melhor às necessidades do projeto.

[5] Do inglês, *General Purpose Input/Output*.

5 RESULTADOS PARA O PERFIL DE QUALIDADE DAS COMUNICAÇÕES SPI E WI-FI

Neste capítulo é apresentado um perfil de qualidade de comunicação entre dois dispositivos de acordo com o padrão IEEE 802.11n, além de resultados comprovando a realização da comunicação com os protocolos SPI e Wi-Fi para o projeto final.

5.1 PROBABILIDADE DE SUCESSO POR SEÇÃO DE COMUNICAÇÃO NO WI-FI

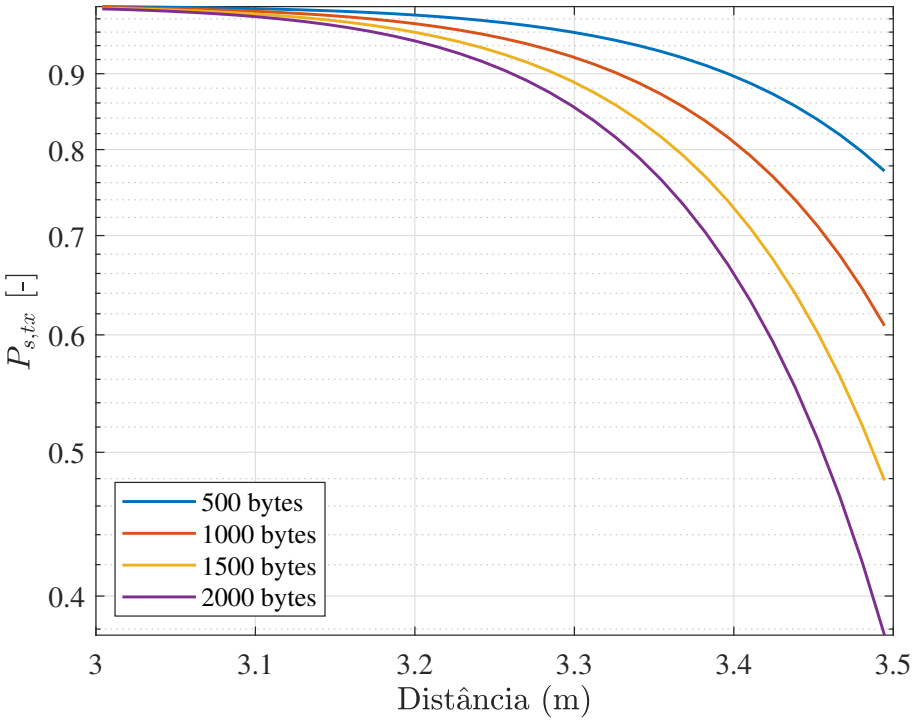
Ao se utilizar a Equação (3.20), foi possível traçar o gráfico da Figura 8 para 500, 1000, 1500 e 2000 bytes de dados transmitidos. Considerou-se a utilização do MCS7 para a transmissão do pacote de dados em OFDM e a transmissão dos quadros de ACK em MCS3. Como parâmetros do canal, utilizou-se a modelagem log-normal descrita na Equação (3.13) com os parâmetros $d_0 = 1\,\text{m}$, $\eta = 2,95$ e $PL(d_0) = 61,7\,\text{dB}$, obtidos por Sreedevi, Rao e Susila[28] em um ambiente fechado de apartamento. O termo X_σ da Equação (3.13), referente aos efeitos de desvanecimento multipercurso, foi desconsiderado por simplicidade. Além disso, foi considerada para o cálculo do ruído térmico na Equação (3.15) uma temperatura ambiente de 25 °C, bem como uma figura de ruído $F = 2$. Ressalta-se que a largura de banda utilizada foi de 20 MHz, apesar de que também é possível a operação em 40 MHz. Por fim, no cálculo da relação sinal-ruído vista pelo receptor da Equação (3.14), utilizou-se a potência transmitida típica do ESP32 de -17 dB [27] e 15 dB de perdas de redundância.

A Figura 8 indica que a comunicação Wi-Fi entre o MCU ESP32 e outro dispositivo é factível até cerca de 3 m de distância. A partir desse ponto, a probabilidade de sucesso rapidamente diminui à medida que a distância aumenta, ao ponto que seriam necessárias múltiplas seções de comunicação para que uma determinada quantidade de bytes de informação fosse corretemente transmitida, efetivamente diminuindo a taxa de transmissão de dados do sistema. A sensibilidade da probabilidade de sucesso em relação à distância vai de acordo com outros resultados presentes na literatura [29, 30, 31, 32].

Um alcande maior poderia ser alcançado caso se utilizasse um método de transmissão menos sensível a ruído, como BPSK ou QPSK nos esquemas MCS0-2 da Tabela 2, entretanto a máxima taxa de dados do sistema seria consideravelmente reduzida. Isto, por sua vez, limita os contextos em que se poderia transmitir os dados a partir do Wi-Fi, visto que a banda utilizada para se transmitir valores de sensores seria menor, correspondendo por exemplo a uma menor frequência de amostragem.

Para a finalidade de coleta de dados, 3 metros é uma distância suficiente entre o ESP 32 e o computador executando o servidor. Em aplicações que exijam uma distância superior, uma possível solução seria utilizar uma placa contendo um módulo ESP 32 ligado a uma antena externa de maior ganho, a fim de se extender as curvas da Figura 8 para distâncias maiores.

Figura 8 – Probabilidade de sucesso por seção de comunicação em função da distância



Fonte: Elaborado pelo autor

5.2 VERIFICAÇÃO DA TAXA DE DADOS ATRAVÉS DA FERRAMENTA IPERF

A própria Espressif disponibiliza um exemplo que possibilita a medição da taxa de dados efetiva entre um ESP 32 e outro dispositivo em rede. Nesse aspecto, a ferramenta *iPerf* é um programa executável por linha de comando que coordena um teste com outro dispositivo para se medir a maior taxa de dados atingível na comunicação Wi-Fi [33]. Ao se utilizá-la em uma rede composta por um computador pessoal e um MCU ESP32 a 1 m de distância utilizando TCP/IP, constata-se uma mínima taxa de dados efetiva de 18,9 Mbps, com uma média de 22,17 Mbps. Dessa maneira, estima-se que o sistema com os MCUs e protocolos escolhidos é capaz de transmitir e armazenar dados sendo gerados a uma taxa máxima de 6,4 Mbps, que é um valor suficiente para se coletar até 4 variáveis de 16 bits em uma frequência de amostragem de 100 kHz. Apesar de ser conservadora, a estimativa é baseada no fato de que taxa de dados real do sistema cai razoavelmente devido à execução de processos internos relacionados à cópia dos *buffers* que contêm as informações a serem transmitidas.

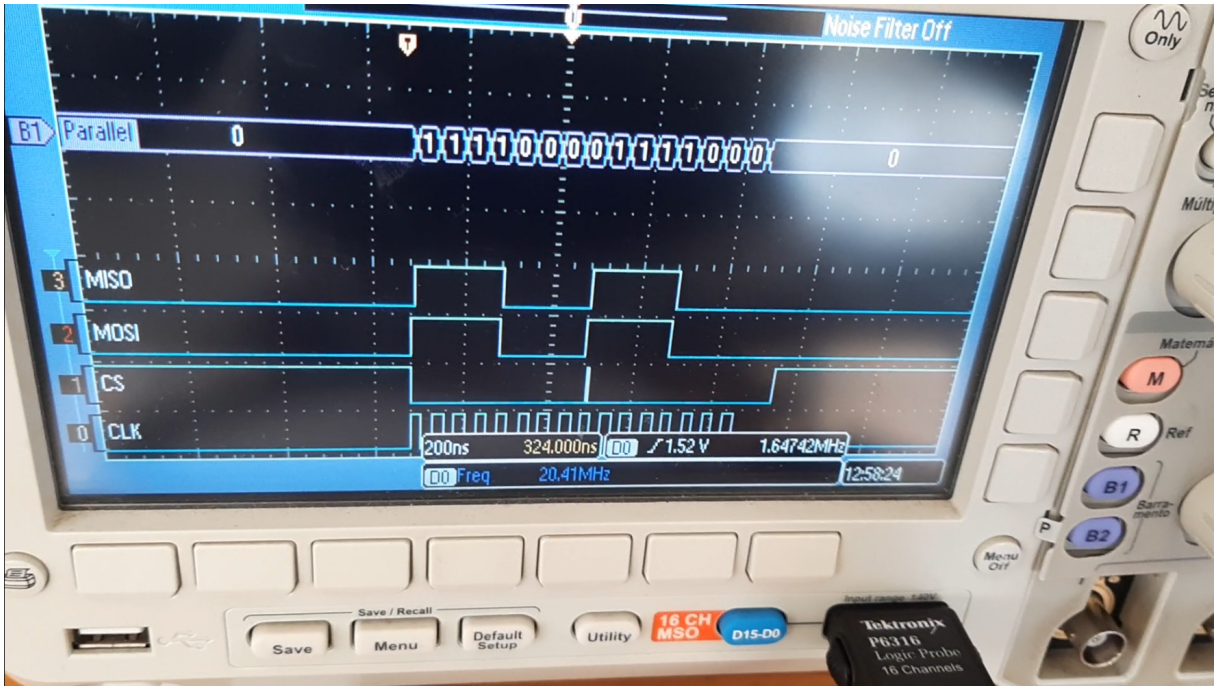
5.3 TESTE DA COMUNICAÇÃO SPI

A comunicação pelo protocolo SPI é testada para diversas configurações de conexão física entre os dispositivos. Como resultados finais, verificou-se que é possível implementá-la sem a constatação de erros na frequência de 20 MHz. Como configuração geral do protocolo,

ambos os dispositivos são programados de maneira que a escrita dos dados nos canais MISO e MOSI ocorra nas bordas de subida do sinal de *clock*, tal que a leitura ocorre nas bordas de descida. Além disso, o sinal de *clock* é definido para permanecer em nível lógico baixo quando não há transações em andamento.

A Figura 9 mostra as formas de onda obtidas a partir da ponteira lógica P6316, utilizada para se identificar o que é válido no domínio digital. Nesse aspecto, o par SPI primário/secundário foi implementado no ESP32 e F28379D respectivamente, tal que as conexões entre eles são feitas a partir de cabos *dupont*. Como forma de se validar a comunicação, especifica-se nos testes que o secundário sempre transmitisse a mesma sequência binária 0xF0F0 no canal MISO, enquanto que o primário retransmitisse o que fosse lido por ele durante o ciclo anterior no canal MOSI. Nesse aspecto, observa-se que a mensagem 0xF0F0 é corretamente enviada pelo ESP 32, ao passo que o DSP envia de volta a mesma mensagem equivalente recebida no período anterior. Nota-se um certo atraso de resposta do DSP devido ao tempo de propagação do sinal na linha CS. Além disso, apesar de haver certa interferência no canal CS devido à *crosstalk*, como ambas as sequências são equivalentes no momento em que o sinal CLK apresenta uma borda de descida, a comunicação SPI entre os dispositivos é válida e transmissão dos dados a 20 MHz é possível.

Figura 9 – Teste de comunicação SPI *full-duplex*



Fonte: Elaborado pelo autor

6 SISTEMA DE COMUNICAÇÃO PROPOSTO

Este capítulo apresenta a arquitetura proposta para o sistema de aquisição de dados para conversores estáticos projetado. Primeiramente são listados os dispositivos escolhidos para cada bloco fundamental do sistema. Em seguida, é explicado em detalhes o funcionamento da estrutura e da forma que ocorre a comunicação desde o MCU de controle ao servidor utilizado para armazenar os dados.

6.1 VISÃO GERAL DO SISTEMA

O sistema de coleta de dados é composto por três dispositivos que se comunicam entre si. Primeiramente, o DSP F28379D da *Texas Instruments* é utilizado para executar o código da camada de aplicação referente ao experimento de coleta de dados. Nos testes implementados, a placa de desenvolvimento LAUNCHXL-F28379D é utilizada por disponibilidade, além de ser ideal para aplicações relacionadas ao controle de conversores estáticos, haja visto os seus periféricos destinados à geração da ação de controle e conversores analógico-digitais com tempo de conversão na escala de ns. A camada de aplicação em si deste dispositivo não faz parte do escopo deste trabalho, de maneira que o DSP é tratado pelo resto do sistema simplesmente como um gerador de dados a serem armazenados. A configuração do periférico SPI e a lógica da comunicação foram programadas em linguagem C, de maneira que cabe ao usuário incluir no código da camada de aplicação quais variáveis devem ser salvas pelo sistema.

Comunicando-se diretamente com o DSP via protocolo SPI, o microcontrolador ESP 32 da Espressif é utilizado para transmitir os dados de interesse a um servidor TCP/IP sendo executado em um computador pessoal. Foi utilizada a placa de desenvolvimento ESP32 DEVKIT V1 - DOIT no contexto do protótipo do sistema. Vale destacar que, a escolha de se implementar a comunicação Wi-Fi em um microcontrolador dedicado foi feita considerando que o tempo de processamento utilizado pelo DSP para a comunicação deve ser o menor possível para não influenciar no restante da aplicação. Dessa maneira, o ESP 32 é responsável tanto pela coordenação do protocolo SPI, quanto pelas demais funcionalidades referentes à comunicação Wi-Fi direta com o servidor local. Destaca-se que, para o devido funcionamento do sistema com os equipamentos especificados, a distância entre o ESP 32 e o computador deve ser menor que 3 m, visto que o desempenho da comunicação Wi-Fi entre estes dispositivos começa a se degradar muito a partir deste limite. A programação no ESP 32 também foi feita em linguagem C, com auxílio da API (*Application Programming Interface*, do inglês Interface de Programação de Aplicação) elaborada pela Espressif [34].

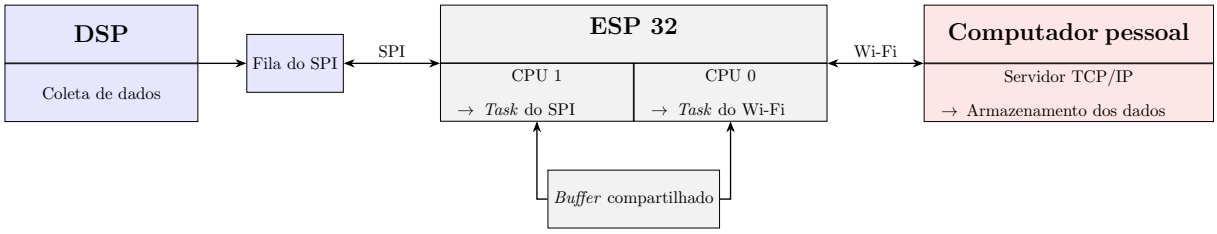
Por fim, um computador pessoal é utilizado para se executar um servidor TCP/IP destinado ao armazenamento dos dados. Nesse aspecto, foi elaborado um módulo de funções em Python para que as funcionalidades de inicialização do servidor e exportação dos dados em formato de texto possam ser acessadas via linha de comando. Os dados são salvos em formato

binário para que não se perca informação vinda do DSP, além de diminuir a memória necessária para o armazenamento de informação.

Dito isso, o diagrama de fluxo de sinais da Figura 10 ilustra como ocorre a coordenação entre os três dispositivos do sistema. Nesse aspecto, foi utilizada uma lógica de fila no DSP para que as transações da comunicação SPI ocorram em lotes. Isto é necessário, já que o tempo exigido pelo ESP 32 para leitura e escrita da memória dos registradores relacionados ao módulo SPI é significativo e não permite que os dados sejam transmitidos amostra por amostra.

Com relação ao ESP 32, é utilizado o RTOS (*Real Time Operating System*, do inglês Sistema Operacional de Tempo Real) adaptado pela Espressif para se executar as funcionalidades do SPI e Wi-Fi em duas *tasks*. Como o ESP 32 possui dois núcleos disponíveis, atribuiu-se à CPU 1 a coordenação do SPI e à CPU 0 a comunicação Wi-Fi. Na memória RAM compartilhada entre os dois núcleos, é utilizado um *buffer* para que os dados recebidos via SPI sejam salvos temporariamente. Quando o *buffer* é preenchido com dados, eles são copiados pela *task* do Wi-Fi e transmitidos para o servidor.

Figura 10 – Diagrama de fluxo de sinais do sistema completo



Fonte: Elaborado pelo autor.

Como diretivas fundamentais da estrutura proposta, além do seu funcionamento, definiu-se que o DSP deve ser alertado caso alguma falha ocorra durante o processo de armazenamento dos dados. Dessa maneira, o usuário pode incluir no código da camada de aplicação como essas falhas são tratadas, a fim de, por exemplo, desligar um conversor visto que nenhum dado gerado está sendo salvo. Os detalhes de como ocorre a comunicação entre cada dispositivo e de como é feita essa supervisão de falhas são mostrados em seguida.

6.1.1 Módulo de coleta de dados no DSP

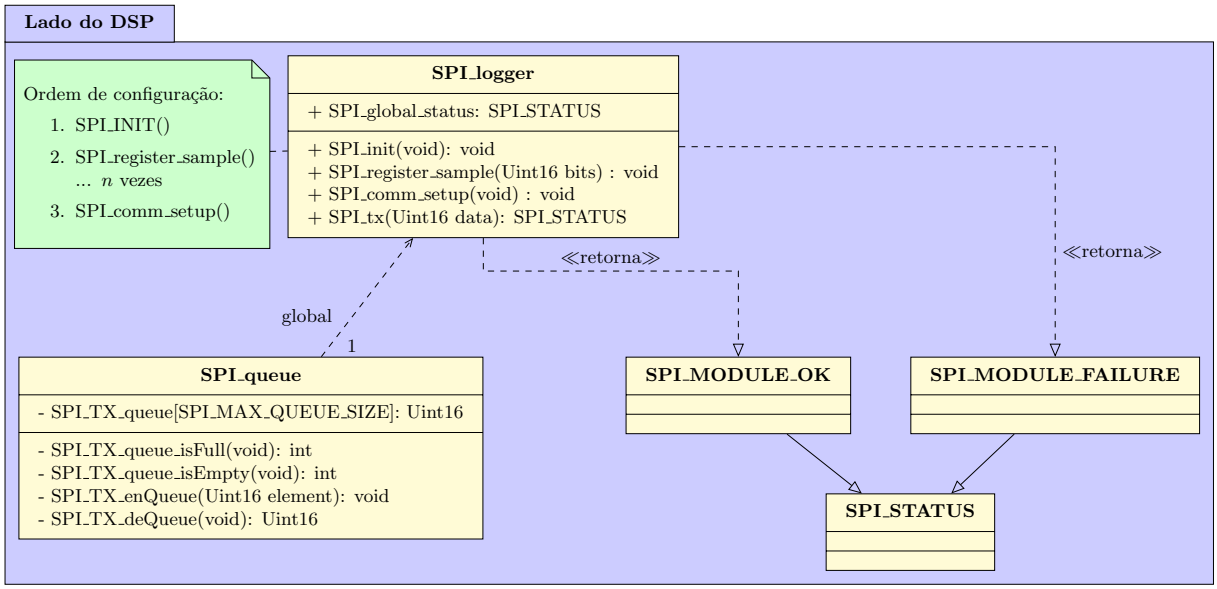
O código do lado do DSP é responsável por transmitir para o ESP 32 os dados de uma sequência de amostras via SPI. Dessa maneira, foi elaborado o diagrama de classes UML[1] da Figura 11 como documentação do código, que foi dividido em duas classes: `SPI_logger` e `SPI_queue`.

A classe `SPI_logger` é responsável por configurar o periférico do SPI e coordenar a transmissão de informação através de uma estrutura de fila. Nesse aspecto, as funções de

[1] *Universal Modeling Language*, do inglês Linguagem de Modelagem Unificada.

configuração `SPI_init()`, `SPI_register_sample()` e `SPI_comm_setup` devem ser chamadas em sequência, a fim de que o módulo funcione adequadamente. Esta etapa de configuração é necessária, visto que o resto do sistema precisa da informação de bits utilizados por cada variável transmitida para organizar os dados corretamente.

Figura 11 – Diagrama de classes do lado do DSP

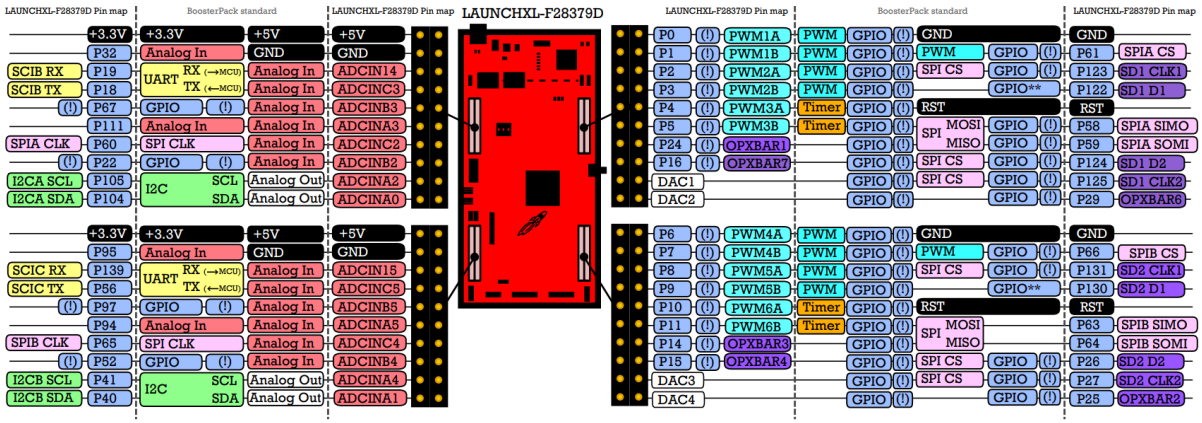


Fonte: Elaborado pelo autor.

Em se tratando da configuração do SPI, o DSP F28379D apresenta 4 módulos disponíveis para o usuário, cada um deles com diferentes pinos acessíveis. Escolheu-se o módulo SPI-A do DSP, que é roteado na configuração padrão aos pinos 58-61 do circuito integrado conforme mostra a Figura 12. Além disso, a função `SPI_init()` configura o módulo para o modo FIFO[2], que permite a utilização de até 16 registradores de 16 bits para a escrita automática dos dados no registrador de deslocamento do SPI. O módulo também foi configurado para atuar no modo secundário, utilizando todos os 16 bits do registrador de deslocamento. O sinal de *clock* foi definido para transmitir os dados em bordas de subida e realizar a leitura do canal em bordas de descida. Por fim, o método de comunicação implementado demanda a utilização de uma interrupção quando os registradores FIFO de recepção estiverem cheios, o que corresponde a 16 ciclos de transações SPI realizados.

[2] "Primeiro a entrar, primeiro a sair", do inglês *First In First Out*.

Figura 12 – Configuração de pinos da placa de desenvolvimento LAUCHXL-F28379D



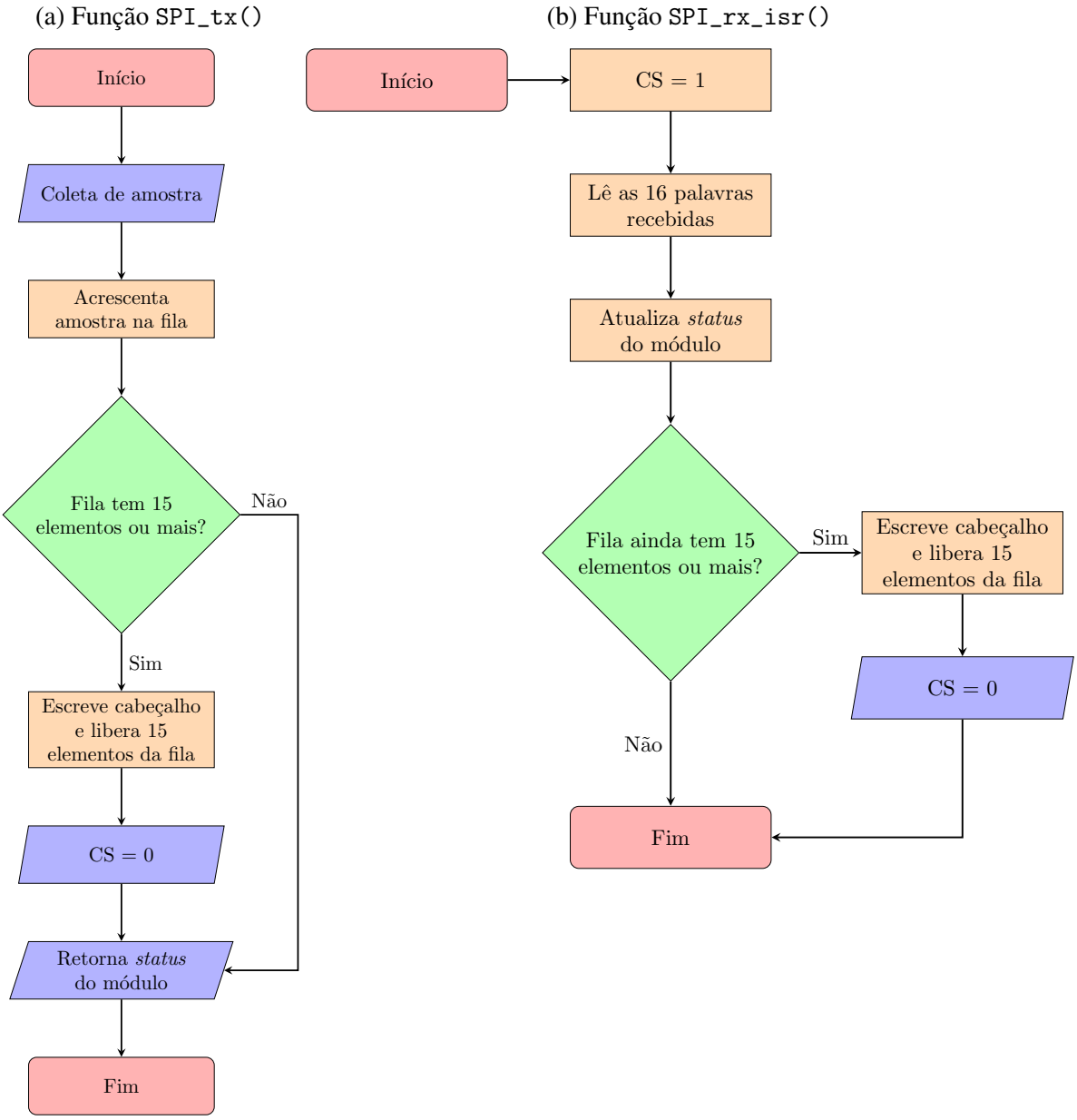
Fonte: Texas Instruments[35].

Além disso, a funcionalidade principal do módulo é implementada na função `SPI_tx()`, que possui como argumento uma variável de até 16 bits, que deve ser convertida para o tipo `Uint16` antes do chamado da função. Nesse aspecto, seu funcionamento é ilustrado pelo fluxograma na Figura 13a, tal que à medida que são coletadas amostras, a fila de dados coletados é populada. Quando o número de itens na fila é maior ou igual a 15, o DSP sinaliza para o ESP 32 o início de uma transação SPI, visto que foram coletados dados suficientes para uma rodada de transmissões. Um cabeçalho é escrito junto com as amostras nos 16 registradores FIFO de transmissão do SPI, para sinalizar para o ESP 32 que o conteúdo recebido nas próximas 15 palavras representa informação.

No âmbito da comunicação SPI implementada entre o DSP e o ESP, foi utilizada uma configuração de 3 pinos conectados entre si: SCLK, MOSI e MISO. Junto a eles, o terminal $\overline{\text{CS}}$ restante do módulo SPI foi utilizado como uma forma de comunicação externa para sinalizar o início de uma transação. Isto foi feito visto que a troca de mensagens entre os dispositivos é orientada na coleta de dados a partir do DSP, que precisa ser configurado como secundário devido às limitações do módulo SPI do ESP 32. Dessa maneira, o fluxo de mensagens ainda é comandado pelo DSP, mas é o ESP 32 que atua como dispositivo primário. Dito isso, assim que o DSP escreve nos registradores FIFO o cabeçalho e as 15 amostras de dados, ele sinaliza o início de uma transação SPI para o ESP escrevendo 0 V no pino $\overline{\text{CS}}$. Por fim, a função `SPI_tx()` retorna duas possíveis mensagens de erro para os casos em que não foi detectado nenhum erro e vice-versa: `SPI_MODULE_OK` e `SPI_MODULE_FAILURE` respectivamente. O tratamento de erros deve ser feito no código da camada de aplicação.

Em complemento à função `SPI_tx()`, é utilizada uma interrupção no DSP para possibilitar a leitura dos dados recebidos após uma transação SPI realizada. Nesse aspecto, a partir do momento em que 16 palavras são trocadas entre os dispositivos, ocorre a interrupção `SPI_rx_isr()`, cujo funcionamento é ilustrado pelo fluxograma da Figura 13b. Primeiramente, para sinalizar o fim de uma transação SPI, o terminal CS é acionado com 3,3 V. Em seguida,

Figura 13 – Fluxogramas para o funcionamento do SPI no DSP



Fonte: Elaborado pelo autor.

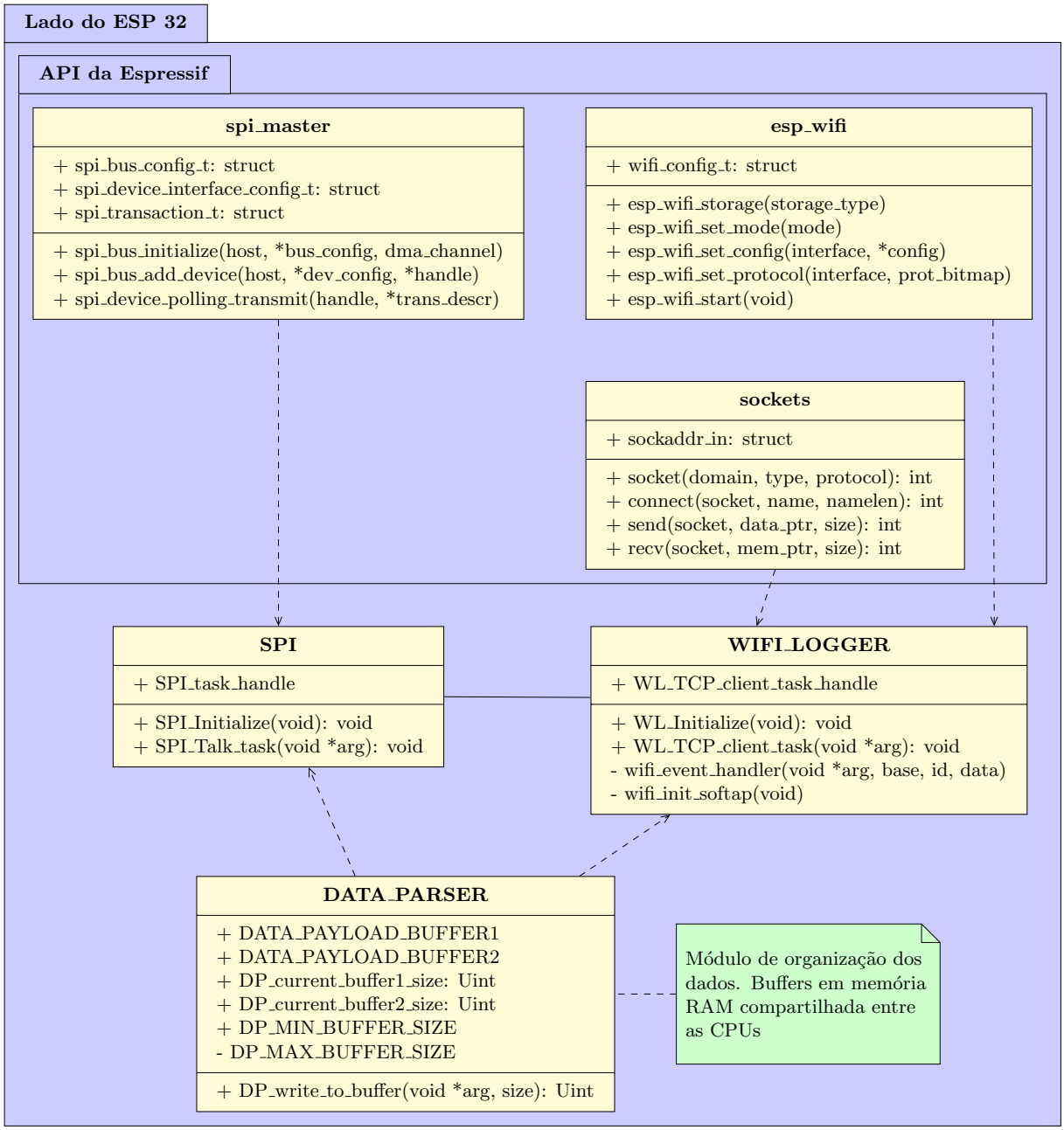
realiza-se a leitura dos dados armazenados nos registradores FIFO de leitura, a fim de se detectar a ocorrência de algum erro oriundo do resto do sistema. Caso um erro seja recebido, a variável de *status* interna ao módulo é alterada para `SPI_MODULE_FAILURE`, de maneira que o código da camada de aplicação seja capaz de lidar com a falha a partir da verificação da variável global `SPI_global_status`, que também é retornada pela função `SPI_tx()` por conveniência. Por fim, caso ainda haja dados suficientes para serem transmitidos, o mesmo processo de escrita das variáveis e cabeçalho descrito anteriormente é realizado, ao passo em que uma nova transação é sinalizada com 0 V no pino CS. Este processo de transmissão durante a interrupção foi implementado pois aumenta consideravelmente a velocidade da comunicação SPI, visto que o tempo

em que nenhuma transação ocorre é minimizado.

6.1.2 Módulo de coleta de dados e comunicação Wi-Fi no ESP 32

A programação no ESP 32 foi feita utilizando a API desenvolvida pela Espressif. Nesse aspecto, utilizou-se sobretudo o módulo `spi_master` para a configuração de `hardware` do SPI, além dos módulos `esp_wifi` e `sockets` para se configurar algumas funcionalidades do Wi-Fi. O diagrama de classes para o ESP é apresentado na Figura 14 e é detalhado a seguir.

Figura 14 – Diagrama de classes do lado do ESP 32



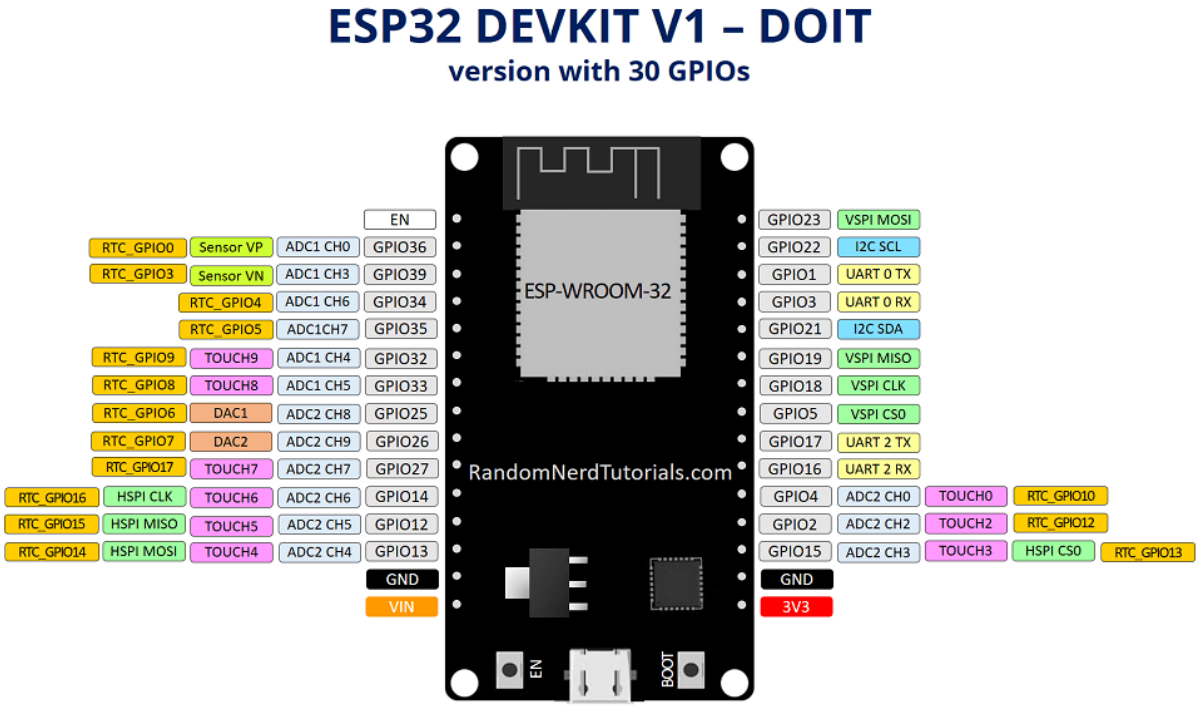
Fonte: Elaborado pelo autor.

Para se configurar o SPI, utilizam-se duas instâncias das estruturas `spi_bus_config_t`

e `spi_device_interface_config_t`. A primeira é responsável por definir quais pinos do ESP serão utilizados pelo barramento SPI, além do tamanho máximo de uma transação em *bytes*. As demais configurações de frequência da comunicação e funcionamento das bordas do sinal de *clock* são escritas em um objeto da segunda estrutura. Nesse aspecto, a frequência é fixada em 20 MHz e as configurações do *clock* são as mesmas do DSP. Como configuração de GPIO adicional, um *pull-up* interno em 3,3 V é feito, visto que o sinal de início de transação do DSP é feito em nível lógico baixo.

Com relação aos pinos utilizados, a placa de desenvolvimento do ESP 32 utilizada possui dois módulos SPI acessíveis ao usuário para a camada de aplicação, denominados HSPI e VSPI. Como não há diferença entre eles, especificou-se o módulo VSPI para a interface com o DSP, cuja configuração de pinos padrão da placa de desenvolvimento é apresentada na Figura 15.

Figura 15 – Configuração de pinos da placa de desenvolvimento ESP 32 DEVKIT V1 - DOIT



Fonte: RANDOM NERD TUTORIALS[36]

Na comunicação entre o ESP 32 e o servidor TCP/IP, primeiramente um ponto de acesso Wi-Fi é criado no microcontrolador. Dessa maneira, o módulo `esp_wifi` da API é utilizado para se configurar a inicialização do ponto de acesso, bem como outras características da camada física. Já as funções de comunicação em si são realizadas pelas funções do módulo `sockets` da API.

Para a configuração do ponto de acesso, as informações de identificador de rede, senha e número máximo de conexões são armazenadas em uma instância da estrutura `wifi_config_t`. Esse objeto é utilizado na função `esp_wifi_set_config()` para então realmente configurar o dispositivo. Além disso, utilizam-se os demais métodos da classe `esp_wifi` para se configurar o

tipo de armazenamento para a memória RAM, o modo de utilização para ponto de acesso e o protocolo de comunicação para IEEE 802.11n. Por fim, cria-se definitivamente o ponto de acesso com a função `esp_wifi_start()`.

A classe `sockets` da API cria os objetos e métodos necessários para se estabelecer a comunicação via protocolo TCP/IP. Nesse sentido, utiliza-se uma instância da estrutura `sockaddr_in` para se definir alguns parâmetros como endereço IP do *host*, família do endereço para IPV4[3] e porta utilizada para comunicação fixada em 9999. O método *socket()* é responsável por criar o objeto que atua como interface para as funções do módulo. Para isso, é preciso informar novamente a família do IPV4 como domínio, tipo de comunicação como TCP/IP e protocolo com o valor nulo. Dessa maneira, o objeto retornado pela função pode ser utilizado nos demais métodos *connect*, *send* e *recv* para, respectivamente, criar a conexão entre o ESP 32 e o servidor TCP/IP, mandar e receber dados.

Com o auxílio dos módulos da API apresentados, elaboraram-se três classes para a coordenação da comunicação SPI e Wi-Fi. Como o ESP 32 foi programado com RTOS, as classes `SPI` e `WIFI_LOGGER` dispõem de uma função *task* que é executada a partir do agendador do RTOS. Nesse aspecto, ambas essas funções dispõem de um *loop* infinito para a execução de código referente aos tipos de comunicação exigidos. Dito isso, as duas classes apresentam os *handles* dessas funções como objetos públicos, visto que a criação das *tasks* em si ocorre na função *main*, que é a primeira a ser executada após a inicialização do ESP 32. Além disso, as duas classes possuem métodos para inicialização de variáveis internas. A classe `WIFI_LOGGER` apresenta também dois métodos privados: `wifi_event_handler()`, que responde a certos eventos referentes ao módulo Wi-Fi, como por exemplo uma tentativa de conexão no ponto de acesso do ESP 32; e `wifi_init_softap()`, que é responsável por registrar os tipos de eventos relevantes, bem como executar as funções de configuração do Wi-Fi do módulo `esp_wifi`.

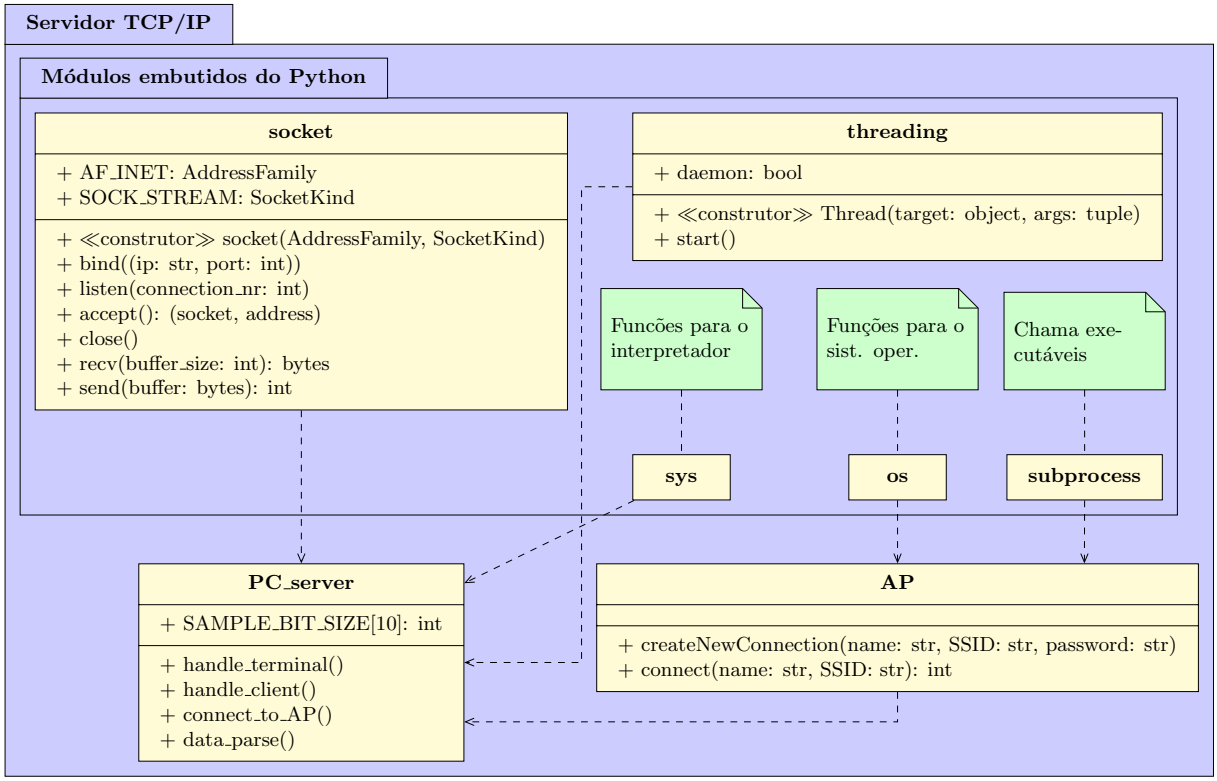
Por fim, a classe `DATA_PARSER` contém os objetos públicos dos *buffers* que são populados com os dados transmitidos via Wi-Fi. São utilizados dois *buffers*, de forma que ambos os núcleos do ESP possam trabalhar individualmente com cada um, sem haver conflito de acesso simultâneo à memória RAM compartilhada entre eles. Nesse aspecto, o *buffer* 1 é preenchido à medida que chegam dados via protocolo SPI. Quando ele é populado com dados suficientes, copia-se seu conteúdo para o *buffer* 2, que passa a ser trabalhado pelo módulo Wi-Fi, enquanto o *buffer* 1 volta ao estado vazio para acumular mais dados do DSP. Isto ocorre no método `SPI_Talk_task()` da classe `SPI`. O limite mínimo de tamanho utilizado para iniciar a cópia do *buffer* 1 é de 2500 bytes, obtido por tentativa e erro para se obter a melhor taxa de dados. Junto a eles, são disponibilizados também o tamanho atual de cada *buffer*. Para realizar a escrita no *buffer* 1, utiliza-se o método `DP_write_to_buffer()`, que copia os dados em um endereço de memória ao *buffer* e retorna o número de bytes utilizados por ele.

[3] *Internet Protocol Version 4*, do inglês "Protocolo de Internet Versão 4".

6.1.3 Servidor TCP/IP para armazenamento de dados

Como receptor das mensagens transmitidas Wi-Fi, o servidor TCP/IP desempenha o papel de inicializar automaticamente a conexão ao ponto de acesso criado pelo ESP 32, além de atuar também no processo de inicialização do sistema para a coleta dos parâmetros de tamanho em bits de cada amostra. Dito isso, utilizam-se módulos nativos do Python análogos aos fornecidos pela API da Espressif no ESP 32. A Figura 16 ilustra o diagrama de classes elaborado para o servidor executado em um computador pessoal.

Figura 16 – Diagrama de classes do servidor TCP/IP



Fonte: Elaborado pelo autor.

Para realizar a interface de comunicação de baixo nível, é utilizado o módulo `socket` assim como no ESP 32. Nesse aspecto, utilizam-se as propriedades `AF_INET` e `SOCK_STREAM` para indicar que o tipo de comunicação Wi-Fi utilizado é através do protocolo TCP/IP utilizando IPV4. Além disso, tem-se que os métodos `recv()` e `send()` desempenham as mesmas funções que os métodos de mesmo nome utilizados no ESP 32. Nesse aspecto, como diferenças no uso dessas funções, tem-se que `recv()` retorna diretamente os bytes recebidos via Wi-Fi e requer como argumento o número de bytes máximos a serem recebidos de uma vez. O método `send()`, analogamente, requer como argumento o pacote em bytes a ser transferido e retorna o número de bytes transmitidos.

O método `bind()` é utilizado para se configurar o endereço IPV4 e a porta utilizados na comunicação. O argumento da função é uma tupla que contém estes dois parâmetros respectivamente, com o detalhe que é fornecida uma *string* vazia para indicar para o servidor receber

transmissões de qualquer endereço conectado. A configuração de porta utilizada é equivalente à do ESP 32, tal que esta foi fixada em 9999. Além disso, utiliza-se o método `listen()` para se definir o número de conexões simultâneas ativas e permitir que o servidor aceite requisitos de conexão. Por fim, os métodos `accept()` e `close()` são utilizados para se aceitar e fechar uma conexão Wi-Fi respectivamente, de maneira que o objeto `socket` referente à conexão e o endereço são retornados pelo método `accept()`.

Além do módulo `socket`, são utilizados os módulos nativos `threading`, `sys`, `os` e `subprocess` no código do servidor. O primeiro é utilizado para possibilitar a execução simultânea dos métodos do servidor em si, haja visto que são utilizadas duas `threads` para as funções de comunicação e leitura de comandos recebidos pelo terminal. Nesse aspecto, cria-se as *threads* com o construtor `Thread()`, que começam a ser executadas após a utilização do método `start()`. A propriedade `daemon` indica que o programa do servidor em si é encerrado após a *thread* inicial, ou seja a `main()`, é finalizada. Os três módulos restantes são utilizados para utilizar os métodos `sys.exit()`, `os.system()` e `subprocess.call()`, que são chamados para, respectivamente, encerrar a *thread* principal; executar um comando adequado à linha de comando do sistema operacional; e chamar um arquivo executável externo.

O servidor TCP/IP em si pode ser dividido em duas classes: `PC_server` e `AP`. O método `AP.createNewConnection` é chamado para se registrar uma nova configuração de ponto de acesso Wi-Fi e exige como argumentos o nome e senha deste, que são fixados no código do ESP 32. Dessa maneira, utiliza-se o método `connect()` para se executar um *scan* das redes Wi-Fi disponíveis e requisitar uma conexão com a rede do ESP 32. O código em Python para o registro das configurações do ponto de acesso e o executável que inicia o *scan* das redes foram adaptados de [37] e [38] respectivamente.

Finalmente, a classe `PC_server` contém o objeto `SAMPLE_BIT_SIZE[10]` que armazena a quantidade de bits de cada amostra, que é necessária para separar corretamente os dados recebidos. Além disso, as funções executadas nas *threads* de leitura do terminal e comunicação com o ESP 32 são definidas nesta classe e consistem nos métodos `handle_terminal()` e `handle_client()` respectivamente. O método `connect_to_AP()` é chamado no início do processo e requisita uma tentativa de busca e conexão ao ponto de acesso do ESP 32 até encontrá-lo. Já o método `data_parse()` organiza e salva os dados coletados em arquivos separados por amostra, de maneira que a interpretação destes possa ser feita pelo usuário posteriormente. Ressalta-se que a execução destes métodos é coordenada pela função `handle_terminal`, que recebe as entradas do usuário via linha de comando e chama os métodos apropriados.

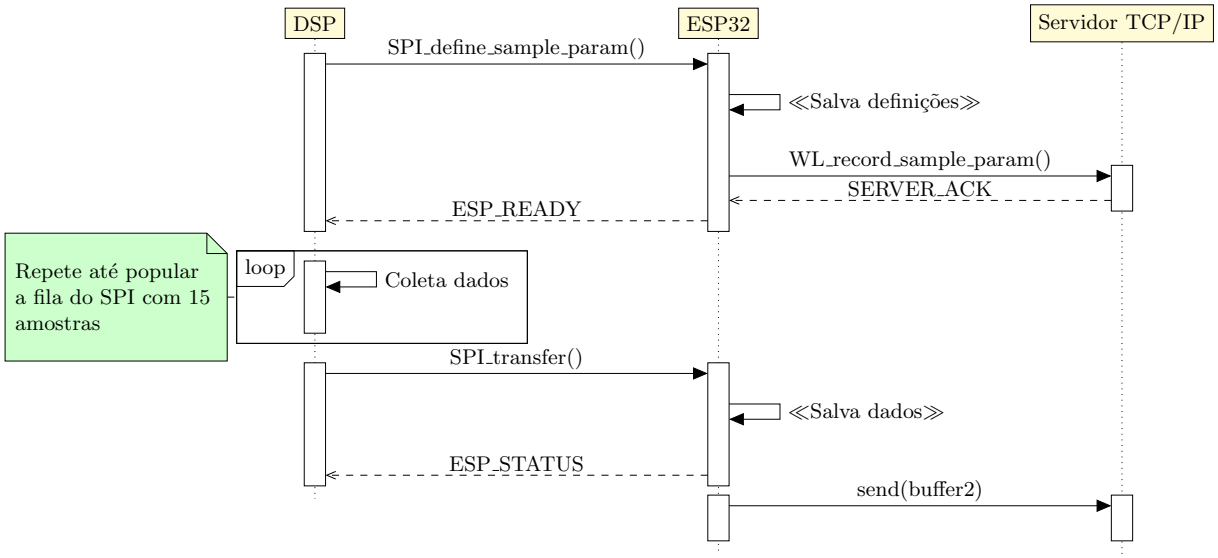
6.1.4 Comunicação do sistema em detalhes

Como o sistema proposto pode ser descrito através da transmissão de dados do DSP ao servidor pela interface de um microcontrolador ESP 32, a troca de mensagens pode ser descrita ao se analisar como ele atua como interface entre ambos os lados. Nesse aspecto, a Figura

17 contém um diagrama de sequência ilustrando a troca de mensagens durante as rotinas de inicialização e coleta de dados do sistema integrado. O processo inteiro pode ser dividido em 4 etapas:

1. Inicialização e gravação dos parâmetros de amostras;
2. Conexão com o servidor e transferência dos parâmetros salvos;
3. Coleta de dados e preenchimento do *buffer* de transmissão;
4. Envio do *buffer* com as amostras cíclicas via Wi-Fi.

Figura 17 – Diagrama de sequência representando a comunicação entre o DSP, ESP 32 e servidor durante uma coleta de dados



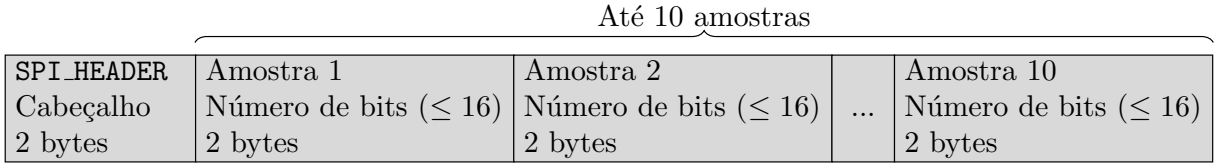
Fonte: Elaborado pelo autor.

O primeiro passo após a definição dos parâmetros de bit por amostra no DSP é a gravação deles no ESP 32. Durante essa etapa as interrupções são desabilitadas, visto que o processo ocorre de maneira sequencial e não deve ser interrompido. Dessa maneira, a troca de mensagens por SPI ocorre de maneira convencional, mas o controle do fim de uma transação no DSP é feito ao se consultar diretamente o número de palavras recebidas nos registradores FIFO de leitura ao invés da utilização de uma interrupção. Isso implica diretamente que esta parte do processo exige uma parada da execução de código e deve ser feita estritamente durante a inicialização do DSP em si, a fim de não interromper a execução de outras instruções no dispositivo.

A gravação dos parâmetros é feita através da função `SPI_define_sample_param()`, que os transmite para o ESP 32 via SPI. A estrutura do pacote de dados de configuração que o DSP envia é ilustrada na Figura 18 e é sempre composta por 16 palavras de 16 bits, tal que a primeira representa um cabeçalho que indica se a mensagem advém da lógica do protocolo de comunicação em si, ou se simplesmente representa dados coletados. As posições de amostras

não utilizadas e o resto do pacote são preenchidos com zeros. As representações literais dos elementos vistos pelo DSP que compõem um pacote SPI são apresentadas na Tabela 4.

Figura 18 – Formato de pacote de inicialização do SPI



Fonte: Elaborado pelo autor.

Tabela 4 – Representação das mensagens usadas pelo sistema inteiro

Mensagem	Representação
`SPI_HEADER`	`SPI_DATA` - 10 (0x000A)
	`SPI_PROTOCOL` - 200 (0x00C8)
`SPI_ESP_RESPONSE`	`SPI_ESP_READY` - 100 (0x0064)
	`SPI_ESP_OK` - 200 (0x00C8)
	`SPI_ESP_FAILURE` - 300 (0x012C)
`WL_MESSAGE`	`WL_SAMPLE_PARAM` - 400 (0x0190)
	`WL_SERVER_ACK` - 500 (0x01F4)
	`WL_SERVER_FAILURE` - 600 (0x0258)

Fonte: Elaborado pelo autor

Ao receber as informações, o ESP 32 espera 10 segundos para receber uma conexão do servidor TCP/IP, caso já não tenha recebido. Caso isso não ocorra, uma exceção de tempo esgotado é apontada e a resposta `SPI_ESP_FAILURE` é mandada para o DSP. Se a conexão for feita sem demais problemas, o ESP 32 envia os parâmetros salvos através do método `WL_record_sample_param()` e espera por uma mensagem de recebimento do servidor. A estrutura das mensagens usadas para a comunicação Wi-Fi também é mostrada na Tabela 4 como a classe `WL_MESSAGE`.

Por fim, o ESP 32 retorna a mensagem `ESP_READY` para o DSP, indicando o término do processo de inicialização. As mensagens do ESP 32 também são compostas por 16 palavras, mas somente a primeira contém os 2 bytes da mensagem em si enquanto as demais são nulas. Em seguida, à medida que se coletam amostras, estas são enviadas para o DSP em grupos de 15 através do método `SPI_transfer()`. Como a comunicação SPI é *full-duplex*, ao fim de cada transação o DSP recebe a mensagem `ESP_STATUS`, que indica se houve algum erro detectado pelo ESP 32 em algum momento anterior. Além disso, com o recebimento dos dados e o preenchimento do *buffer* de transmissão Wi-Fi, este é eventualmente enviado para o servidor quando atinge o tamanho mínimo correspondente.

7 DEMONSTRAÇÃO DE USO DO SISTEMA

Neste capítulo é mostrada uma demonstração de utilização do sistema integrado de aquisição de dados. Nesse sentido, gerou-se um conjunto de dados conhecidos a uma taxa de dados representativa de conversores estáticos no DSP, que são então transmitidos para o servidor e comparados com a resposta esperada. O código dos módulos escritos para o DSP, ESP 32 e servidor em Python, junto com esta demonstração, estão disponibilizados no Github [39].

7.1 UTILIZAÇÃO DO SISTEMA NO DSP

O código fonte para o DSP foi desenvolvido em linguagem C e inclui a biblioteca `F28x_Project.h` elaborada pela TI para acessar as posições de memória dos registradores do SPI e API de GPIO [40]. O exemplo apresentado neste capítulo consiste na geração de 4 amostras de 16 bits cada a uma frequência de 100 kHz, resultando em uma taxa de dados de 6,4 Mbps.

De forma geral, a inicialização do DSP F28379D requer a execução de 4 passos em ordem. A Figura 19 contém um pseudocódigo para inserir as rotinas de inicialização do módulo e como configurar as interrupções necessárias para seu funcionamento. Na figura, as instruções que utilizam rotinas de configuração incluídas pelo módulo estão destacadas em azul. O código do usuário pode ser acrescentado, desde que respeite a ordem de configuração apresentada.

Primeiramente, deve-se incluir o cabeçalho `SPI_logger.h` para se utilizar as funções desenvolvidas. Dentre os passos de configuração do DSP, configura-se de início o PLL, *clock* dos periféricos e *Watch-Dog*. Em seguida, configuram-se os pinos de GPIO, de maneira que a função `SPI_HS_GPIO_config` deve ser chamada para configurar as portas 58-61 do DSP, correspondentes aos terminais do periférico SPI-A. Após isso, configura-se os registradores e parâmetros de interrupção relacionados ao SPI em si, o que é feito através da função `SPI_init()`. A função `SPI_register_sample()` deve ser executada em seguida o número de vezes que for necessário, a fim de gravar o número de amostras coletadas no teste e a quantidade de bits usados por cada uma. Como último passo da configuração do SPI, chama-se a função `SPI_comm_setup()` para se definir o início da coleta de dados para o DSP e servidor TCP/IP. Essa função retorna o *status* do módulo, de maneira que é recomendado realizar um tratamento das condições de erro, caso a inicialização do sistema não seja bem sucedida.

Além de se configurar a parte de GPIO e a configuração do módulo em si, deve-se registrar a interrupção que comanda a linha CS do módulo `SPI_rx_isr()`. Este passo deve ser feito após a inicialização do módulo SPI, junto das demais interrupções utilizadas pelo resto da aplicação. Além disso, deve-se habilitar a interrupção atrelada ao FIFO de recepção do periférico SPI-A, que está contida no grupo 6 de interrupções, o qual também deve ser habilitado. No final do processo, todas as interrupções devem ser habilitadas.

Após a inicialização do sistema e periféricos, a função `SPI_tx()` pode ser utilizada no

Figura 19 – Pseudocódigo para inicialização do DSP com o módulo `SPI_logger.h`

```
1  Inclua o módulo SPI_logger.h
2  Passo 1 - Inicialização do sistema:
3      Inicialize PLL
4      Inicialize clock dos periféricos
5      Inicialize WatchDog
6  Passo 2 - Inicialização de GPIOs:
7      Inicialize as portas de alta velocidade do SPI-A
8  Passo 3 - Inicialização do módulo SPI e registro de amostras:
9      Inicialize os registradores do SPI
10     para cada amostra coletada faça
11         Registre número de bits
12     Configure o link de comunicação DSP-ESP-PC
13     se erro detectado então
14         Trate o erro
15 Passo 4 - Inicialização de interrupções:
16     Desabilite todas as interrupções
17     Inicialize os registradores de controle dos eventos de interrupção de periféricos
18     Desabilite as interrupções da CPU e descarte as suas flags de interrupção
19     Inicialize a tabela de vetores de interrupção com as rotinas padrão da TI
20     Registra rotina de interrupção FIFO RX do SPI-A
21     Defina as interrupções de periféricos
22     Ative a interrupção 1 do bloco 6
23     Ative o bloco 6 de interrupções
24     Habilite todas as interrupções
```

Fonte: Elaborado pelo autor.

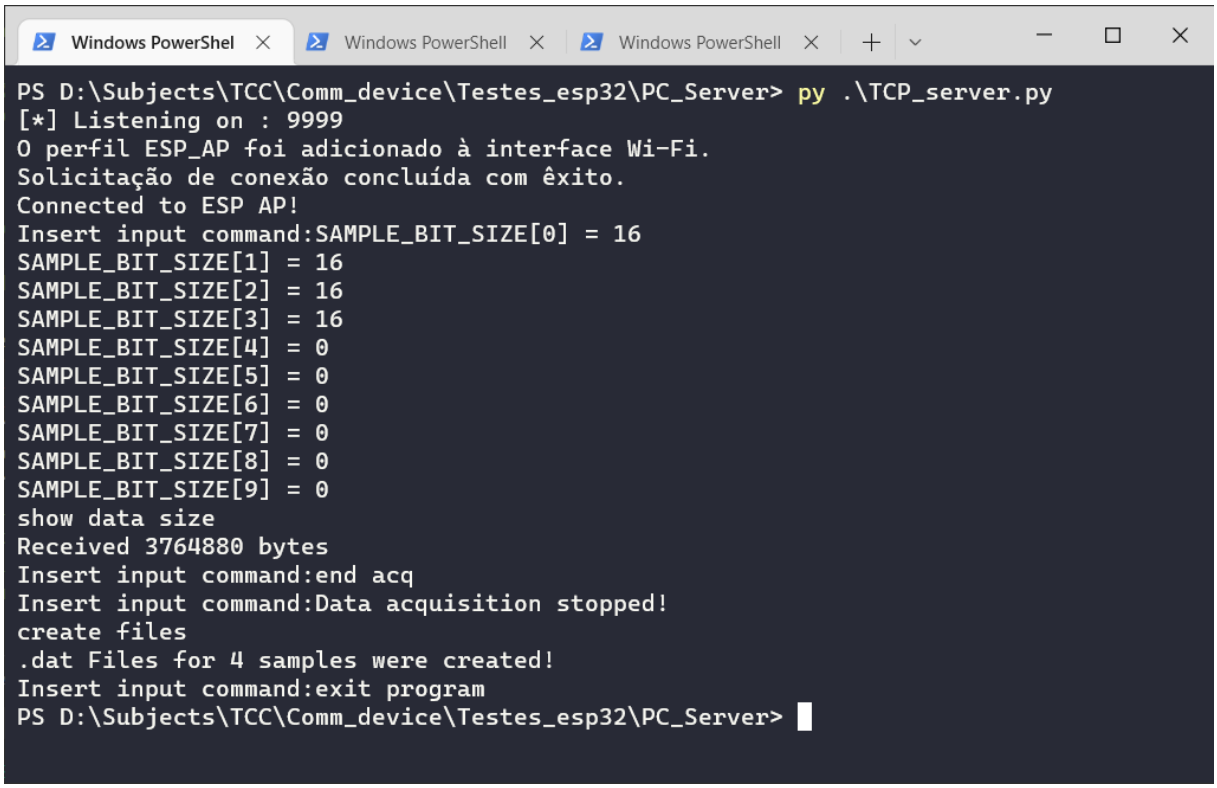
código de aplicação para sinalizar a transferência de amostras. Seu uso recomendado é em rotinas de interrupção atreladas à leitura de conversões do ADC. Destaca-se que o valor de retorno desse método é o *status* do módulo, de modo que é recomendado tratar os possíveis erros de retorno adequadamente, analogamente ao que ocorre com a função `SPI_comm_setup()`.

7.2 UTILIZAÇÃO DO SISTEMA EM UM COMPUTADOR PESSOAL

O servidor foi desenvolvido em Python e pode ser executado simplesmente ao se chamar o *script* `TCP_server.py` por um recurso de terminal do Windows, por exemplo o Windows PowerShell®. A Figura 20 ilustra a janela do processo do servidor sendo executado. Dito isso,

primeiramente o servidor realiza uma busca pelo ponto de acesso do ESP 32 até encontrá-lo. Em seguida, é permitido ao usuário entrar com comandos no terminal para, por exemplo, encerrar o teste ou gravar os dados coletados em arquivos separados. A Tabela 5 mostra os comandos aceitos e as suas funcionalidades.

Figura 20 – Visão da interface do servidor acessado pelo terminal do Windows



Fonte: Elaborado pelo autor.

Tabela 5 – Comandos acessíveis para interagir com o servidor via linha de comando

Comando	Descrição
`show data size`	Mostra a quantidade de bytes recebidos até o momento
`end acq`	Encerra a coleta de dados
`create files`	Cria arquivos '.dat' para cada amostra registrada
`exit program`	Encerra o servidor e sai do programa

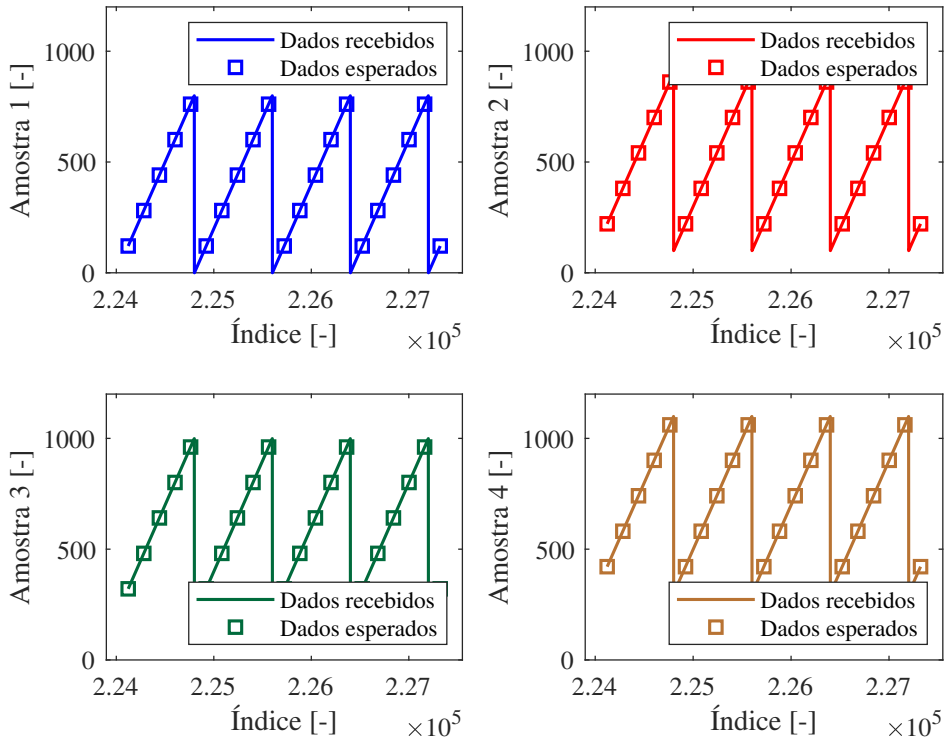
Fonte: Elaborado pelo autor

Após o processo de inicialização do sistema, os valores de bits por amostra são apresentados no terminal logo em seguida. Para esta demonstração, o experimento de coleta de dados foi executado por cerca de 5 segundos. Para automatizar os testes, elaborou-se um *script* separado em Python, também disponibilizado no GitHub [39], que faz a verificação dos dados salvos em cada arquivo com os valores esperados.

7.3 RESULTADOS OBTIDOS COM SINAL DE TESTE DENTE DE SERRA

Os casos de teste abordados nesta seção se referem à geração e transmissão de 4 amostras geradas pelo DSP ao servidor. Estas são geradas a uma frequência fixa de 100 kHz, o que foi obtido ao se implementar uma rotina de interrupção atrelada aos periféricos de geração de comando por largura de pulso do DSP. As 4 amostras ocupam 16 bits cada, portanto a taxa de dados máxima testada equivale a $4 \cdot 16 \cdot 100\,\text{kHz} = 6{,}4\,\text{Mbps}$. As amostras 1 a 4 equivalem ao mesmo sinal dente de serra somado às constantes 0, 100, 200 e 300 respectivamente, a fim de diferenciá-los no processo de validação. A Figura 21 mostra uma comparação gráfica dos dados recebidos com os esperados, sendo que a marcação dos dados esperados foi mostrada em com um espaçamento maior entre pontos para melhorar a visualização. É mostrado apenas um certo trecho do teste, que foi executado durante um tempo total de aproximadamente 5 segundos. Dessa maneira, verifica-se que a taxa de erro dos testes é nula, validando o funcionamento do sistema.

Figura 21 – Dados recebidos e de referência para o teste do sistema



Fonte: Elaborado pelo autor.

8 CONCLUSÃO

Este trabalho apresentou uma possível abordagem para um sistema de aquisição de dados destinado a aplicações de conversores estáticos. Apesar de diversos serviços para a gravação de dados serem disponibilizados no mercado, muitos deles são destinados a aplicações que geram menos informação, o que pode não ser adequado para conversores. Dito isso, procurou-se criar um sistema apropriado a partir de dois microcontroladores: um destinado à interação direta com o conversor e outro dedicado à comunicação Wi-Fi.

Para se estimar alguns fatores de qualidade referentes ao sistema, fez-se uma revisão bibliográfica nos Capítulos 2 e 3 a fim de se identificar os melhores protocolos existentes para a interface de comunicação serial entre os microcontroladores e de transmissão dos dados. Como resultado, observou-se um grande potencial para o uso do protocolo SPI e do padrão IEEE 802.11n, visto que são compatíveis com o escopo de alta geração de dados. Dessa maneira, as soluções implementadas foram baseadas fundamentalmente nestes dois protocolos.

Ainda no Capítulo 3 foi apresentada uma revisão bibliográfica abordando o cálculo teórico da probabilidade de sucesso em uma seção de comunicação Wi-Fi entre dois dispositivos. No âmbito de comunicação sem fio, foi feita também no Capítulo 4 uma busca dos microcontroladores mais utilizados para este tipo de aplicação, de maneira a identificar um possível modelo adequado ao contexto do trabalho. Dessa forma, concluiu-se que o ESP 32 apresenta tanto as características de *hardware* necessárias para a aplicação, quanto um bom suporte ao desenvolvimento de *software* por meio de APIs oficiais da própria fabricante.

A partir da definição dos protocolos de comunicação e dispositivos do sistema, foi possível estimar no Capítulo 4 que a distância esperada para operação válida do conjunto é de cerca de 3 m. Além disso, verificou-se no Capítulo 5 com o auxílio de ferramentas de *software* que a taxa de dados referente à comunicação Wi-Fi em condições ótimas fica em torno de 20 Mbps, o que implica que as tecnologias escolhidas foram apropriadas.

O Capítulo 6 apresenta toda a documentação elaborada para o *software* desenvolvido para cada dispositivo utilizado no sistema. Foram apresentados diagramas de classe no padrão UML que sintetizam as funcionalidades de cada dispositivo, bem como a relação entre as classes escritas para cada um deles. Também foi apresentada a lógica de comunicação para a inicialização e coleta de dados do sistema, tal que os objetivos de coleta de dados e tratamento de erros foram abordados pelo código desenvolvido para o sistema integrado. Destaca-se que, durante o processo de desenvolvimento, diversas funcionalidades tiveram que ser repensadas após a constatação de erros no processo de validação, amparando a ideia de *design* baseado em testes.

Como validação do trabalho, verificou-se no Capítulo 7 que o sistema passa no teste de demonstração proposto. Nesse aspecto, as maiores condições de *stress* da estrutura se referem a uma taxa de dados de 6,4 Mbps, de maneira que nenhum erro foi constatado ao se coletar dados durante alguns segundos de operação.

Por fim, baseado no desenvolvimento deste trabalho, sugerem-se como trabalhos futuros os seguintes temas: i) a construção do *hardware* que integre o DSP com o ESP 32 e um sistema de condicionamento de sinais para medição das variáveis de conversores estáticos, melhorando a integridade dos sinais SPI; ii) a implementação de uma interface gráfica para o sistema.

REFERÊNCIAS

1 BARBI, Ivo. Eletrônica de potência. Ed. do Autor, 2006. Citado 2 vezes nas páginas 13 e 33.

2 FADALI, M Sami; VISIOLI, Antonio. **Digital control engineering: analysis and design**. [S.l.]: Academic Press, 2012. Citado na página 13.

3 BAZANELLA, Alexandre Sanfelice; CAMPESTRINI, Lucíola; ECKHARD, Diego. **Data-driven controller design: the H2 approach**. [S.l.]: Springer Science & Business Media, 2011. Citado na página 13.

4 BABA, Sebastian et al. Evaluation of modular power converter integrated with 5g network. **Energies**, Multidisciplinary Digital Publishing Institute, v. 14, n. 21, p. 7355, 2021. Citado na página 13.

5 SILVA, Antonio Wallace Neres da et al. Control and monitoring of a flyback dc-dc converter for photovoltaic applications using embedded iot system. **IEEE Latin America Transactions**, IEEE, v. 18, n. 11, p. 1892–1899, 2020. Citado na página 13.

6 ALMEIDA, Rodrigo Maximiano Antunes de; MORAES, Carlos Henrique Valério de; SERAPHIM, Thatyana de Faria Piola. **Programação de Sistemas Embarcados: Desenvolvendo Software para Microcontroladores em Linguagem C**. [S.l.]: Elsevier Brasil, 2017. Citado 3 vezes nas páginas 16, 17 e 18.

7 LEENS, Frédéric. An introduction to i2c and spi protocols. **IEEE Instrumentation & Measurement Magazine**, IEEE, v. 12, n. 1, p. 8–13, 2009. Citado na página 17.

8 WILAMOWSKI, Bogdan M; IRWIN, J David. **Industrial communication systems**. [S.l.]: CRC Press, 2018. Citado 3 vezes nas páginas 19, 21 e 34.

9 IEEE COMPUTER SOCIETY. Ieee standard part 11: Wireless lan mac and phy specifications. **IEEE Std 802.11-2012**, 2012. Citado 6 vezes nas páginas 20, 22, 23, 28, 29 e 31.

10 LATHI, B P; DING, Zhi. **Modern digital and analog communication systems**. [S.l.]: Oxford University Press, 2010. Citado 3 vezes nas páginas 20, 26 e 28.

11 PROAKIS, John G; SALEHI, Masoud. **Digital communications**. 5. ed. [S.l.]: McGraw-hill New York, 2008. Citado 5 vezes nas páginas 21, 24, 25, 27 e 29.

12 GAST, Matthew S. **802.11n: A Survival Guide**. [S.l.]: O'Reilly Media, Inc., 2012. Citado 4 vezes nas páginas 21, 22, 23 e 24.

13 ANDERSON, Harry R. **Fixed broadband wireless system design**. [S.l.]: John Wiley & Sons, 2003. Citado 2 vezes nas páginas 27 e 30.

14 HACCOUN, David; BEGIN, Guy. High-rate punctured convolutional codes for viterbi and sequential decoding. **IEEE transactions on communications**, v. 37, p. 1113–1125, 1989. Citado na página 29.

15 ZAMALLOA, Marco Zúñiga; KRISHNAMACHARI, Bhaskar. An analysis of unreliability and asymmetry in low-power wireless links. **ACM Transactions on Sensor Networks (TOSN)**, ACM New York, NY, USA, v. 3, n. 2, p. 7–es, 2007. Citado na página 30.

16 CÂMARA, André Michelin; HOEFEL, Roger Pierre Fabris. On the performance of ieee 802.11n: Analytical and simulations results. In: **XXIX Brazilian Symposium of Telecommunications, SBrT 2011**. [S.l.: s.n.], 2011. Citado na página 30.

17 QIAO, Daji; CHOI, Sunghyun; SHIN, Kang G. Goodput analysis and link adaptation for ieee 802.11a wireless lans. **IEEE transactions on Mobile Computing**, IEEE, v. 1, n. 4, p. 278–292, 2002. Citado na página 31.

18 GAST, Matthew. **802.11 Wireless Networks: The Definitive Guide**. [S.l.]: "O'Reilly Media, Inc.", 2005. Citado na página 31.

19 LIMA, Charles B de; VILLAÇA, Marco VM. Avr e arduino–técnicas de projeto. 2ª edição. **Editora Clube de Autores**, 2012. Citado na página 33.

20 RASHID, Muhammad H. **Power Electronics Handbook**. [S.l.]: Butterworth-Heinemann, 2017. Citado na página 33.

21 OJO, Mike O. et al. A review of low-end, middle-end, and high-end iot devices. **IEEE Access**, v. 6, p. 70528–70554, 2018. Citado na página 34.

22 TEXAS INSTRUMENTS. **CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee ® Applications**. 2015. Disponível em: <https://www.ti.com/product/CC2538>. Acesso em: 28 fev. 2022. Citado na página 34.

23 ATMEL. **Low-Power, 32-bit Cortex-M0+ MCU with Advanced Analog and PWM**. 2021. Disponível em: <https://www.microchip.com/en-us/product/ATsamd21g18>. Acesso em: 28 fev. 2022. Citado na página 35.

24 ATMEL. **SMART ARM-Based Wireless Microcontroller**. 2016. Disponível em: <https://www.microchip.com/en-us/product/ATSAMR21G18A>. Acesso em: 28 fev. 2022. Citado na página 35.

25 ATMEL. **IEEE 802.11 b/g/n SmartConnect Wi-Fi Module**. 2016. Disponível em: <https://www.microchip.com/en-us/product/ATSAMW25>. Acesso em: 29 fev. 2022. Citado na página 35.

26 ESPRESSIF. **ESP8266EX Datasheet**. 2020. Disponível em: <https://www.espressif.com/en/products/socs/esp8266>. Acesso em: 03 mar. 2022. Citado na página 35.

27 ESPRESSIF. **ESP32 Series Datasheet**. 2022. Disponível em: <https://www.espressif.com/en/products/socs/esp32/resources>. Acesso em: 27 jun. 2022. Citado 2 vezes nas páginas 35 e 36.

28 SREEDEVI, AG; RAO, T Rama; SUSILA, M. Measurements at 2.4, 3.4, 5.2, 28 and 60 ghz for device-to-device wireless communications. **Wireless Personal Communications**, Springer, v. 108, n. 3, p. 1733–1743, 2019. Citado na página 36.

29 MILOS, Jiri; POLAK, Ladislav; SLANINA, Martin. Performance analysis of ieee 802.11 ac/ax wlan technologies under the presence of cfo. In: IEEE. **2017 27th International Conference Radioelektronika (RADIOELEKTRONIKA)**. [S.l.], 2017. p. 1–4. Citado na página 36.

30 MTIBAA, Abderrahmen et al. On practical device-to-device wireless communication: A measurement driven study. In: IEEE. **2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)**. [S.l.], 2017. p. 409–414. Citado na página 36.

31 CHEN, Yin; TERZIS, Andreas. On the implications of the log-normal path loss model: an efficient method to deploy and move sensor motes. In: **Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems**. [S.l.: s.n.], 2011. p. 26–39. Citado na página 36.

32 TIAN, Le; FAMAEY, Jeroen; LATRÉ, Steven. Evaluation of the ieee 802.11 ah restricted access window mechanism for dense iot networks. In: IEEE. **2016 IEEE 17th international symposium on a world of wireless, mobile and multimedia networks (WoWMoM)**. [S.l.], 2016. p. 1–9. Citado na página 36.

33 ESNET. **iPerf - The ultimate speed test tool for TCP, UDP and SCTP**. 2022. Disponível em: <https://iperf.fr>. Acesso em: 04 jul. 2022. Citado na página 37.

34 ESPRESSIF. **ESP-IDF Programming Guide**. 2022. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/release-v4.1/index.html>. Acesso em: 06 nov. 2022. Citado na página 39.

35 TEXAS INSTRUMENTS. **LAUNCHXL-F28379D Overview**. 2016. Disponível em: <https://www.ti.com/tool/LAUNCHXL-F28379D>. Acesso em: 27 jun. 2022. Citado na página 42.

36 RANDOM NERD TUTORIALS. **ESP32-DOIT-DEVKIT-V1-Board-Pinout-30-GPIOs**. 2022. Disponível em: <https://randomnerdtutorials.com/esp32-doit-devkit-v1-board-pinout-30-gpios-copy/>. Acesso em: 06 nov. 2022. Citado na página 45.

37 GEEKSFORGEEKS. **How to connect WiFi using Python?** 2022. Disponível em: <https://www.geeksforgeeks.org/how-to-connect-wifi-using-python/>. Acesso em: 13 nov. 2022. Citado na página 48.

38 SUPERUSER. **Force refresh (re-scan) wireless networks from command line?** 2022. Disponível em: <https://superuser.com/questions/889414/force-refresh-re-scan-wireless-networks-from-command-line>. Acesso em: 13 nov. 2022. Citado na página 48.

39 GITHUB. **Código fonte - Sistema de Aquisição de Dados para Conversores Estáticos**. 2022. Disponível em: <https://github.com/pod042/Data_Acq_System>. Acesso em: 14 nov. 2022. Citado 2 vezes nas páginas 51 e 53.

40 TEXAS INSTRUMENTS. **C2000Ware for C2000 MCUs**. 2019. Disponível em: <https://www.ti.com/tool/C2000WARE#downloads>. Acesso em: 14 nov. 2022. Citado na página 51.