



UNIVERSIDADE DO ESTADO DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E DA TERRA
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

EVERALDO LIMA DE SOUZA JUNIOR

**ARQUITETURA DE MEMÓRIA DE BAIXO CUSTO PARA AQUISIÇÃO DE DADOS
EM MÉDIA E ALTA FREQUÊNCIA COM O ARDUINO**

SALVADOR

2021

EVERALDO LIMA DE SOUZA JUNIOR

ARQUITETURA DE MEMÓRIA DE BAIXO CUSTO PARA AQUISIÇÃO DE DADOS EM
MÉDIA E ALTA FREQUÊNCIA COM O ARDUINO

Monografia apresentada ao curso de Sistemas de Informação do Departamento de Ciências Exatas e da Terra da Universidade do Estado da Bahia - UNEB, como requisito à obtenção do grau de bacharel em Sistemas de Informação. Área de Concentração: Ciência da Computação

Orientador: Cláudio Alves de Amorim

Coorientador: Ernesto de Souza Massa Neto

SALVADOR

2021

FICHA CATALOGRÁFICA
Sistema de Bibliotecas da UNEB

L732a

Lima de Souza Junior, Everaldo

ARQUITETURA DE MEMÓRIA DE BAIXO CUSTO PARA
AQUISIÇÃO DE DADOS EM MÉDIA E ALTA FREQUÊNCIA COM O
ARDUINO / Everaldo Lima de Souza Junior. - Salvador, 2021.
76 fls : il.

Orientador(a): Cláudio Alves de Amorim.

Coorientador(a): Ernesto de Souza Massa Neto.

Inclui Referências

TCC (Graduação - Sistemas de Informação) - Universidade do
Estado da Bahia. Departamento de Ciências Exatas e da Terra. Campus
I. 2021.

1.Arduino. 2.Aquisição de dados. 3.Experimentos didáticos.

CDD: 005

EVERALDO LIMA DE SOUZA JUNIOR

ARQUITETURA DE MEMÓRIA DE BAIXO CUSTO PARA AQUISIÇÃO DE DADOS EM
MÉDIA E ALTA FREQUÊNCIA COM O ARDUINO

Monografia apresentada ao curso de Sistemas de Informação do Departamento de Ciências Exatas e da Terra da Universidade do Estado da Bahia - UNEB, como requisito à obtenção do grau de bacharel em Sistemas de Informação. Área de Concentração: Ciência da Computação

Aprovada em:

BANCA EXAMINADORA

Cláudio Alves de Amorim (Orientador)
Universidade do Estado da Bahia – UNEB

Ernesto de Souza Massa Neto (Coorientador)
Universidade do Estado da Bahia – UNEB

Diego Gervásio Frias Suarez
Universidade do Estado da Bahia – UNEB

AGRADECIMENTOS

Aos conselhos e os ensinamentos de meus orientadores.

Ao apoio e a compreensão da minha família e dos meus amigos.

A todos os professores que contribuíram, direta ou indiretamente, na realização deste trabalho.

RESUMO

Nas escolas e universidades, há interesse crescente pelos dispositivos experimentais de física de arquitetura aberta, mediados por microcontroladores, tendo em vista as suas virtudes didáticas aliadas ao baixo custo, quando comparados com os *kits* comerciais de arquitetura fechada. Por outro lado, não é trivial realizar experimentos com frequências de amostragem superiores a 100Hz, atendendo-se aos requisitos de integridade e permanência dos dados. Com base nessa demanda, o presente trabalho propõe uma arquitetura de armazenamento de baixo custo, não volátil, integrável ao Arduino, para aquisição de dados com frequências superiores às admitidas pelos populares módulos de cartões SD. Aplicou-se uma metodologia para selecionar arquiteturas candidatas, que foram avaliadas por meio de testes e de simulações de cenários experimentais. Por fim, obteve-se uma arquitetura de armazenamento baseada em memória *Flash* da série Winbond W25QXX com gravação de dados até 140 vezes mais rápida do que os módulos de cartões SD.

Palavras-chave: Arduino. experimentos didáticos. aquisição de dados. memória não volátil.

ABSTRACT

In schools, colleges and universities, there is growing interest in microcontroller-driven, open architecture physics experimental devices, because of their didactic virtues and low cost, in comparison with proprietary architecture kits. On the other hand, it is not trivial to carry out experiments with sampling rates above 100Hz, while meeting the requirements of data integrity and permanence. To address this problem, three low-cost, non-volatile storage architectures integrated to Arduino were compared. Candidate architectures were chosen based on the availability of cheap components, and then evaluated through simulations of experimental scenarios. A storage architecture based on Winbond W25QXX series Flash memory was shown to record data up to 52 times faster than SD card modules.

Keywords: Arduino. didactic experiments. data acquisition. datalogger. nonvolatile memory.

LISTA DE FIGURAS

Figura 1 – A arquitetura básica de um microcontrolador.	18
Figura 2 – Placa de circuito eletrônico desenvolvida com o PIC18F14K50.	20
Figura 3 – Plataforma de prototipagem facilitando a implementação de microcontroladores	20
Figura 4 – Características físicas de algumas das principais versões do Arduino.	22
Figura 5 – Alguns ambientes de desenvolvimento integrado para o Arduino	23
Figura 6 – Barramento do Protocolo de Comunicação I ² C	24
Figura 7 – Duas topologias com o barramento SPI	25
Figura 8 – Módulo Leitor de Cartões microSD	26
Figura 9 – Gráfico com V_{IL} e V_{IH} de alguns modelos do Arduino	28
Figura 10 – Exemplo de circuito divisor de tensão para as portas do módulo microSD . .	29
Figura 11 – Diagrama de bloco da arquitetura de um cartão microSD	29
Figura 12 – Aparato confeccionado em alternativa ao módulo leitor de cartões microSD	30
Figura 13 – Esquema elétrico da ligação do Arduino ao cartão microSD sem um módulo	31
Figura 14 – Dispositivos de armazenamento externo adquiridos	34
Figura 15 – Esquema elétrico do circuito temporizador com NE555.	35
Figura 16 – Fluxograma do Teste de Integridade	36
Figura 17 – Fluxograma de Medição do Tempo de Execução das Operações	37
Figura 18 – Alterações no Tamanho do <i>Buffer</i> das Bibliotecas nativas do Arduino	40
Figura 19 – Gráficos com o Desempenho da Memória EEPROM AT24C64A	41
Figura 20 – Aplicações da Memória <i>Flash</i> Winbond da Série W25Q	43
Figura 21 – Gerenciador de Bibliotecas do Arduino IDE	43
Figura 22 – Gráfico com os Resultados da W25Q64FV em cada Bateria de Testes	47
Figura 23 – Esquema formulado para a simulação	51
Figura 24 – Foto do módulo MPU-6050 utilizado nas simulações	51
Figura 25 – Diagrama de Blocos da Arquitetura do BMP-180	53
Figura 26 – Linhas contendo as funções de <i>delays</i> comentadas na biblioteca	55

LISTA DE TABELAS

Tabela 1 – Especificações de alguns dos principais modelos da Plataforma Arduino. . .	21
Tabela 2 – Resultado da Pesquisa pelos Dispositivos de Memória nas Lojas Virtuais. . .	33
Tabela 3 – Resultado dos Testes Sintéticos do Módulo Leitor de Cartões microSD . . .	38
Tabela 4 – Resultado dos Testes Sintéticos da AT24C64A com TWI_FREQ em 100kHz	40
Tabela 5 – Resultado dos Testes Sintéticos da AT24C64A com TWI_FREQ em 400kHz	41
Tabela 6 – Resultado da 1ª Bateria de Testes com a Memória <i>Flash</i> W25Q64FV	44
Tabela 7 – Resultado da 2ª Bateria de Testes com a Memória <i>Flash</i> W25Q64FV	45
Tabela 8 – Resultado da 3ª Bateria de Testes com a Memória <i>Flash</i> W25Q64FV	46
Tabela 9 – Resultado da 4ª Bateria de Testes com a Memória <i>Flash</i> W25Q64FV	47
Tabela 10 – Transmissão de Blocos de 8B nas Frequências Máximas de cada Dispositivo	48
Tabela 11 – Transmissão de Blocos de 8B em Frequências Moderadas de cada Dispositivo	49
Tabela 12 – Resultado dos Testes Preliminares de Leitura no Módulo MPU-6050	52
Tabela 13 – Resultado dos Testes Preliminares de Leitura no Módulo BMP-180	54
Tabela 14 – Resultado das Simulações com os Dispositivos de Memória	57

LISTA DE ABREVIATURAS E SIGLAS

BIOS	<i>Basic Input/Output System</i>
CS	<i>Chip Select</i>
DAQ	<i>Data Acquisition</i>
EEPROM	<i>Electrically-Erasable Programmable Read-Only Memory</i>
I ² C	<i>Inter-Integrated Circuit</i>
IDE	<i>Integrated Development Environment</i>
LCD	<i>Liquid Cristal Display</i>
MCU	<i>Microcontroller Unit</i>
MISO	<i>Master Input Slave Output</i>
MOSI	<i>Master Output Slave Input</i>
PCB	<i>Printed Circuit Board</i>
SCLK	<i>Serial Clock</i>
SPI	<i>Serial Peripheral Interface</i>
SRAM	<i>Static Random Access Memory</i>
SS	<i>Slave Select</i>
UART	<i>Universal Asynchrounous Receiver/Transmitter</i>
VANT	<i>Veículo Aéreo Não Tripulado</i>

LISTA DE SÍMBOLOS

F	<i>Farad</i>
V_{IL}	<i>Input Low Voltage</i>
V_{IH}	<i>Input High Voltage</i>
$k\Omega$	<i>Kiloohms</i>
μF	<i>Microfarad</i>
Ω	<i>Ohms</i>
π	Pi
V_{cc}	<i>Voltage Common Collector</i>

SUMÁRIO

1	INTRODUÇÃO	12
2	AQUISIÇÃO DE DADOS EM EXPERIMENTOS DIDÁTICOS	18
2.1	Placas de Prototipagem Eletrônica	19
2.2	O Arduino	21
2.3	Protocolos de Comunicação	23
2.4	Módulo Leitor de Cartões de Memória	26
2.4.1	Conversores Lógicos	27
2.4.2	Arquitetura do Módulo MicroSD	29
3	DESCRIÇÃO DO PROJETO	32
3.1	Seleção dos dispositivos	32
3.2	Testes Sintéticos	34
3.2.1	Cartão MicroSD	38
3.2.2	Memória EEPROM I ² C AT24C64A	39
3.2.3	Memória Flash SPI W25Q64FV	42
3.3	Análises Comparativas	48
3.4	Simulações	50
3.4.1	Módulo Acelerômetro MPU-6050	51
3.4.2	Módulo Barômetro BMP-180	53
3.4.3	Resultado das Simulações	57
4	CONSIDERAÇÕES FINAIS	59
4.1	Trabalhos Futuros	60
	REFERÊNCIAS	61
	APÊNDICES	64
	APÊNDICE A – Código-fonte	65
	APÊNDICE B – Extração de Dados via Putty	73
	APÊNDICE C – Extração de Dados via Python	74
	APÊNDICE D – Extração de Dados via Dispositivos Móveis	76

1 INTRODUÇÃO

As plataformas de prototipagem baseadas em microcontroladores vêm conquistando cada vez mais espaço e relevância dentro das instituições de ensino do país (1) (2), como ferramentas de baixo custo que auxiliam na construção do conhecimento, promovendo e viabilizando novos experimentos didáticos, bem como novas formas de realizar experimentos clássicos, especialmente nos laboratórios de física (2) (3) (4).

Os experimentos didáticos são atividades empíricas de cunho pedagógico, executadas sistematicamente, a fim de validar uma ou mais hipóteses. Quando bem conduzidos, melhoram a sobremaneira a compreensão dos fenômenos e a assimilação da teoria pertinente (2).

O estudante pode observar o fenômeno, prever o resultado, isto é, formular hipóteses, rapidamente comparar os resultados obtidos com os previstos pelo modelo teórico, explicar possíveis diferenças entre o previsto e o observado e ainda, reformular suas hipóteses, fazer ajustes experimentais e testá-las novamente (2, p. 2).

Os experimentos didáticos aumentam o engajamento dos estudantes, pois, no decorrer das atividades, eles desenvolvem as suas próprias conjecturas, ao interagir com as condições e dados concretos da experiência. Assim, são fomentadas discussões significativas acerca dos fenômenos estudados (4).

Apesar de existirem alguns *kits* didáticos no mercado, eles são relativamente caros para as instituições públicas de ensino do país (1) (5). Outra crítica frequente a tais *kits* é sua falta de flexibilidade, que limita a possibilidade de professores e estudantes reconfigurarem o *hardware* e o *software* dos aparatos experimentais, ficando assim, restritos aos modelos de experimentos previstos pelos fabricantes. Dessa forma, as possibilidades didáticas são limitadas.

Ainda, quando os equipamentos estão disponíveis e práticas experimentais são realizadas, comumente, estão construídas para servirem de simples ilustração teórica baseadas no modelo de laboratório estruturado (*cookbook*, em inglês). Nesse modelo o estudante recebe o equipamento e um roteiro para montar e/ou realizar os experimentos que os levam a respostas formatadas a fim de serem comparadas com o resultado teórico esperado sem que aconteçam elaboração de hipóteses, construção de modelos e discussões sobre suas limitações de aplicabilidade (4, p. 2).

Entre os itens inclusos nos *kits* didáticos estão os dispositivos de aquisição de dados, também conhecidos por *Data Acquisition* (DAQ), com a função de converter os sinais dos sensores em números digitais, para posterior processamento (6). Por proverem um método automatizado e uma maior precisão na obtenção dos dados, eles são equipamentos desejáveis e, muitas das vezes, necessários para a realização dos experimentos.

Quando não assistidos por um DAQ, alguns projetos se tornam impraticáveis devido à dificuldade de se coletar certos sinais: ora pela imprecisão dos aparatos manuais, ora pelas restrições físicas envolvidas no manuseio desses instrumentos. Exemplificando, por mais que um multímetro fosse capaz de medir a resistência de um determinado sensor com acurácia e taxa de amostragem ideais, se em sua medição ocorressem 100 variações de valores por segundo, seria impossível registrar todas essas leituras à mão. Outro exemplo pode ser observado no trabalho de Souza et al.(1). Após obter um tempo médio de 81,8 milissegundos no seu experimento de oscilações amortecidas, os autores declaram a dificuldade de se medir esse período: “por ser muito pequeno, seria muito difícil medi-lo manualmente, com auxílio de cronômetro” (1, p. 3).

Vários autores apontam o uso das plataformas de prototipagem como uma solução de baixo custo aos DAQs no desenvolvimento de experimentos didáticos dentro das instituições públicas de ensino (1) (2). “Neste contexto, a plataforma Arduino tem se mostrado uma ferramenta muito útil e importante para professores e estudantes de Física, principalmente por conta de sua versatilidade.” (3, p. 1)

Embora o Arduino seja pouco exigido nos experimentos mais simples, com poucos periféricos e baixas taxas de amostragem, são nos projetos arrojados que o seu microcontrolador de baixo custo é levado ao limite e posto em xeque. Em especial, nas atividades de aquisição de dados em média e alta frequências: a partir dos 100Hz, dependendo do número de eventos monitorados, pode-se enfrentar uma tarefa bastante desafiadora (7) (8) (9) (10). Isso acontece posto que essas tarefas levantam, simultaneamente, problemas relacionados ao poder de processamento do Arduino e a sua capacidade de armazenamento. Dessa maneira, de um modo geral, é encontrada uma situação na qual ao tentar sanar um dos pontos, agrava-se o outro.

Por exemplo, no intuito de contornar os poucos 2KB presentes na *Static Random Access Memory* (SRAM) da plataforma, os desenvolvedores podem optar por empregar uma abordagem de transmissão de dados similar ao realizado por Souza et al.(1), Cavalcante et al.(2) e Cid e Correa(4), uma espécie de monitoramento *online* do experimento. Nesse caso, o valor

obtido da leitura dos sensores é enviado imediatamente para a porta serial, sem efetuar qualquer armazenamento desse registro no Arduino. No entanto, como é destacado por Jr. et al.(3), ao fazer essa escolha o projeto é limitado a frequências muito menores de aquisição de dados, dado o impacto da comunicação serial sobre a performance do microcontrolador.

À medida que aumentamos a taxa de comunicação serial, a resolução do sinal obtido com o Arduino melhorava significativamente, confirmando assim nossa hipótese acerca do impacto da comunicação serial. Contudo, as taxas maiores que 115 kbps não apresentaram melhora significativa. Mesmo utilizando a maior taxa de comunicação possível (2 Mbps), o sinal reconstruído ainda continha pouca correlação com a onda original (3, p. 6)

Voltando ao exemplo, na tentativa de sanar essa questão da performance e elevar a taxa de amostragem do experimento, os desenvolvedores podem adotar algumas técnicas avançadas de otimização. Em Moreno et al.(6), é proposto um protocolo especial para a comunicação serial envolvendo dados numéricos, como são normalmente os valores lidos dos sensores durante a aquisição de dados. Basicamente, ao invés da transmissão convencional, que converte o número em uma *string* para o envio caractere a caractere usando a função *Serial.print*, a entrega é realizada byte a byte por meio da *Serial.write*. Dessa forma, o valor 155, ou melhor { '1', '5', '5' }, pode ser propagado em apenas um byte através do binário 10011011. Os autores apontam uma melhora de até 200% com essa estratégia. No entanto, indo além da explicação básica, esse procedimento implica no emprego de métodos de controle, verificação (*checksum*) e compressão dos registros, sendo, portanto, mais indicado aos usuários mais experientes em programação.

Então, nesse cenário hipotético, ante a complexidade na otimização da comunicação serial, esses desenvolvedores decidem mudar completamente a estratégia: desprender-se da técnica de transmissão de dados *online* e se enveredar numa abordagem centrada na gravação de dados em memória SRAM, algo equivalente ao apresentado no trabalho de Júnior et al.(11). Entretanto, o mais provável nesta situação é a rápida constatação de que o Arduino não terá capacidade de armazenamento suficiente para contemplar o registro de todo o ensaio experimental. Afinal, em uma aquisição de dados a 100Hz, que gere somente um byte por leitura, em 2 segundos já teríamos um total de 1600B, isto é, mais de 75% da memória principal do microcontrolador.

Além da capacidade, outro problema dessa última abordagem se refere à persistência dos dados coletados. Em uma falha que leve a interrupção no fornecimento de energia ao

dispositivo, todos os registros armazenados no decorrer do ensaio experimental serão perdidos, pois trata-se de uma memória volátil. Ao mesmo tempo que empregar a *Electrically-Erasable Programmable Read-Only Memory* (EEPROM) interna da plataforma não é uma opção muito viável, posto que normalmente possui uma capacidade de armazenamento ainda menor, por vezes, metade da SRAM, como é no caso do popular Arduino UNO, baseado no microcontrolador ATmega328P, com apenas 1KB.

É certo que há inúmeros microcontroladores no mercado que são não apenas mais robustos como também mais adequados às atividades de aquisição de dados em média e alta frequência, como o ARM Cortex-M3 utilizado no trabalho de Blanco-Claraco et al.(12). Todavia, usá-los sem uma plataforma de prototipagem implica no incremento descabido da complexidade do desenvolvimento desses projetos, devido à necessidade de integrá-los com outros componentes (3). Na maioria das vezes, essa abordagem resulta em um significativo e indesejável prejuízo no aspecto didático desses experimentos.

Por outro lado, existem algumas plataformas de prototipagem baseadas nesses microcontroladores mais poderosos. Ainda assim, o uso do Arduino é preferível diante de seu menor custo e seu enfoque alinhado à proposta didática, mesmo que seja necessário aplicar técnicas de otimização para se alcançar a performance exigida em alguns experimentos. Esta condição é retratada ao longo de todo o trabalho de Jr. et al.(3). De qualquer modo, se a plataforma de prototipagem escolhida não tiver uma memória não volátil, ela não solucionará os problemas encontrados em certos projetos.

Há experimentos em que incidentes são admissíveis e, por vezes, até mesmo esperados. Na ausência de um dispositivo adequado de memória não volátil, tanto para o Arduino quanto para as plataformas de prototipagem baseadas em microcontroladores mais robustos, ambos estarão igualmente sujeitos à perda dos dados, por exemplo, após uma súbita interrupção da fonte de alimentação durante os ensaios experimentais. Em outras palavras, apesar de operarem em frequências de *clock* mais elevadas ou terem maior capacidade de SRAM, ou seja, especificações superiores às do Arduino, o uso dessas plataformas, por si só, não garante o êxito desses projetos.

O módulo leitor de cartões SD, bem como sua variante microSD, é amplamente empregado na literatura ao lado do Arduino (13) (14), possivelmente, em virtude de sua alta capacidade de armazenamento (15) e conveniência no acesso e transferência dos dados (16).

Contudo, ele é encontrado também em atividades para as quais não é o mais adequado (8) (7), sendo aplicado tal como uma “bala de prata”, isto é, como se fosse a solução para todas as situações. Entretanto, González et al.(17) demonstraram em seu trabalho que esse módulo - e não Arduino - limitou a taxa de amostragem da bateria de testes para algo em torno dos 70Hz e, portanto, projetos que almejam frequências acima desse patamar devem desconsiderá-los.

In this case, it can be seen that the acquisition speed remained practically constant. Different numbers of sensors were acquired with the security of obtaining an almost constant acquisition speed. This was because it was the SD module which limited the acquisition speed and not the Arduino board microcontroller (17, p. 11).

Na tentativa de contornar essa limitação de frequência do módulo leitor de cartões SD, são implementadas técnicas de *cache* em memória SRAM. Um exemplo da aplicação deste recurso está no trabalho de Joris et al.(8), no qual o Arduino monitorou o voo de um foguete de água. O experimento foi executado de modo satisfatório até pouco antes do pouso, nesse momento o paraquedas não abriu como o esperado causando uma queda desastrosa. O cartão de memória foi recuperado em bom estado, mas perdeu-se uma parte relevante dos registros e, devido ao estrago provocado pelo incidente, novos ensaios não puderam ser realizados. Como não houve persistência dos dados, frustrou-se a análise descritiva do projeto.

The acceleration during takeoff (Fig. 11) was correctly stored on the memory card but no data concerning the height could be retrieved. A possible explanation for this error could be the data transmitting process where data was not flushed to the memory card until a data block was full. If the data block for the height measurement was not full yet at the moment of the crash, it would not have been transmitted to the memory card (8, p. 205).

Em resumo, quando as atividades de aquisição de dados exigem frequências mais elevadas, a abordagem de transmissão dos registros *online* é logo descartada por conta do seu impacto na performance dos microcontroladores de baixo custo. Nesses cenários, é aconselhável o emprego de um dispositivo de armazenamento externo, haja vista o pouco espaço que as memórias internas possuem nesses microcontroladores. Contudo, para uma seleção acertada deste componente alguns fatores determinantes precisam ser considerados. Por exemplo, enquanto o tamanho da memória limitará a duração máxima dos ensaios experimentais, o tempo de resposta da unidade de armazenamento limitará a frequência de amostragem do experimento.

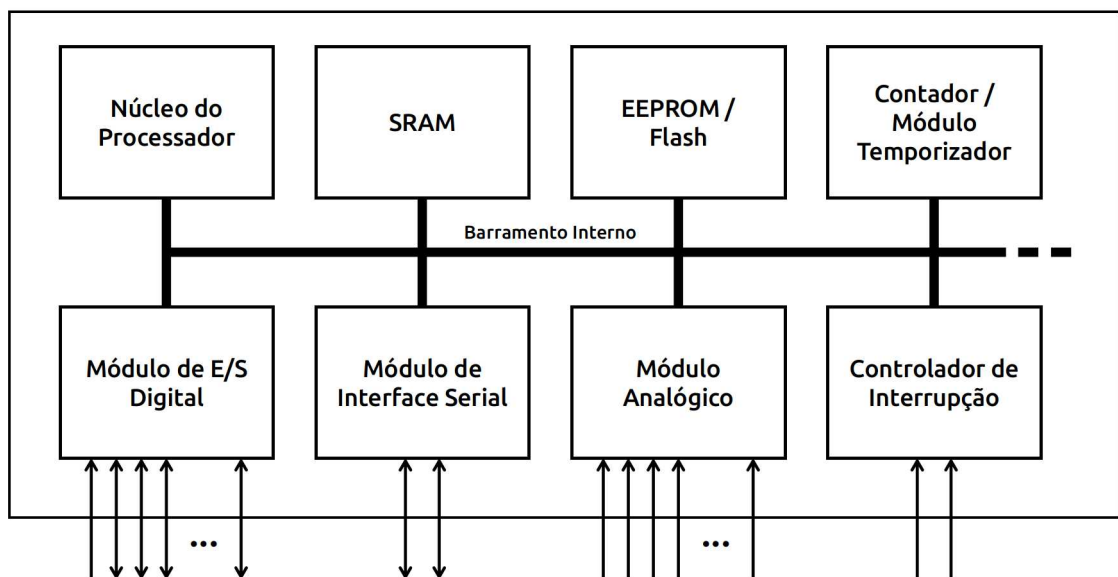
Partindo dessa situação-problema, o presente trabalho visa oferecer para os educadores uma alternativa de implementação simples e de baixo custo aos dispositivos comerciais de aquisição de dados. Assim sendo, principalmente, uma solução para os experimentos didáticos microcontrolados de física com taxas de amostragem acima dos 100Hz. Mais precisamente, o objetivo consiste em propor e avaliar uma arquitetura de armazenamento em memória não volátil, integrável à plataforma Arduino, que atenda aos requisitos, típicos ou presumíveis, de (i) capacidade, (ii) tempo de resposta e (iii) confiabilidade no processo de persistência dos registros, considerando a possibilidade de interrupções abruptas durante o ciclo de amostragem.

A realização deste trabalho resultará em um material de apoio para subsidiar os usuários da plataforma Arduino na implementação de experimentos didáticos de física. Mais especificamente, uma solução ao armazenamento de dados em memória não volátil que possibilitará não só atingir taxas de amostragem inalcançáveis aos módulos leitores de cartões SD, mas também com uma maior confiabilidade sobre o processo de persistência de seus registros.

2 AQUISIÇÃO DE DADOS EM EXPERIMENTOS DIDÁTICOS

Ao analisar as publicações em revistas nacionais de ensino de física, relacionadas à aquisição de dados, notamos uma tendência pelo desenvolvimento de experimentos didáticos por meio de sistemas de baixo custo e inteiramente livres (*open-source*) (2) (18) (11) (4). Em um primeiro momento, os microcontroladores surgem como essa alternativa aos DAQ contidos nos *kits* de experimentos didáticos. Eles são, basicamente, “um computador de chip único, onde temos encapsulados num circuito integrado a unidade lógica aritmética (ULA), memórias, barramentos de comunicação, além de periféricos programáveis para entrada e saída de dados” (3, p. 2). A arquitetura básica desse componente pode ser observada na Figura 1.

Figura 1 – A arquitetura básica de um microcontrolador.



Fonte – Adaptada de Gridling e Bettina(19).

Os primeiros microcontroladores, também conhecidos por *Microcontroller Unit* (MCU), foram concebidos no início da década de 70 pela Texas Instruments, uma empresa de expressiva presença no mercado de semicondutores¹. Atualmente, os microcontroladores são produzidos na escala de bilhões por ano e estão presentes em inúmeros equipamentos eletrônicos: do forno micro-ondas ao sistema de freios antitravamento dos automóveis (19). Essa ampla versatilidade provém das capacidades do microcontrolador de ler diversos sensores, escrever em

¹ <https://www.ti.com/>

diferentes classes de memória e se comunicar por variados protocolos. Logo, eles podem atuar nos experimentos de forma análoga aos dispositivos de arquitetura proprietária especializados em DAQ.

Na construção de experimentos didáticos, dois grupos de microcontroladores são amplamente referenciados na literatura em razão de seu baixo custo (18) e sua precisão satisfatória em atividades de cunho educacional (20), são eles: a família PIC e a série ATmega; atualmente, ambos pertencentes a Microchip, a empresa líder no fornecimento de soluções com microcontroladores². Essas duas famílias formam um conjunto com diversas opções de circuitos integrados. Neles, também encontramos uma variedade de características técnicas, como: diferentes encapsulamentos, dimensões físicas, capacidades de memória, quantidades de portas analógicas e digitais, entre outras particularidades que colaboram no desenvolvimento de soluções customizadas para problemas específicos.

2.1 PLACAS DE PROTOTIPAGEM ELETRÔNICA

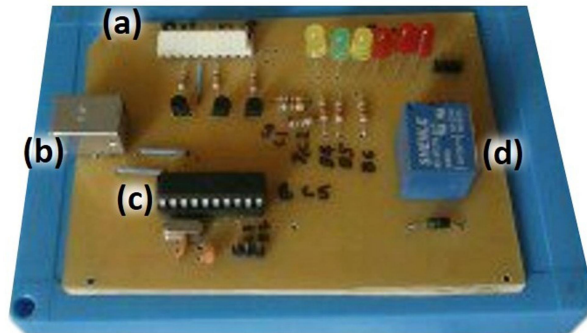
Em um segundo momento, é observado que a incorporação desses MCUs implica na montagem de minuciosos circuitos eletrônicos. Conforme é citado em (3, p. 2): “Para que os microcontroladores operem corretamente, é necessário um conjunto auxiliar de circuitos eletrônicos responsáveis por funções como regulação de tensão, geração de sinal de clock, proteção térmica, entre outras”. Introduzindo, portanto, um certo nível de complexidade no desenvolvimento dos experimentos ao passo que reduz o aspecto didático do projeto.

Seja soldado em uma placa de circuito impresso, também conhecidas por *Printed Circuit Board* (PCB), seja encaixado em uma placa de ensaio (*Protoboard*), em ambos será requerido do desenvolvedor um certo conhecimento na área de eletrônica, bem como habilidades manuais para a soldagem e manipulação dos componentes. Calderón et al.(18) documenta esse processo em seu trabalho cujo circuito é montado para atender as necessidade do microcontrolador PIC18F14K50, como pode ser visto na Figura 2.

Nesse sentido, por meio da abstração desses detalhes de *hardware* envoltos na implementação dos microcontroladores, as plataformas de prototipagem, que são pequenas placas de circuito eletrônico, surgem como uma ferramenta de assistência na construção dos experimentos didáticos. Conforme nota-se na Figura 3, com um microcontrolador ATmega328P à esquerda

² <https://www.microchip.com/>

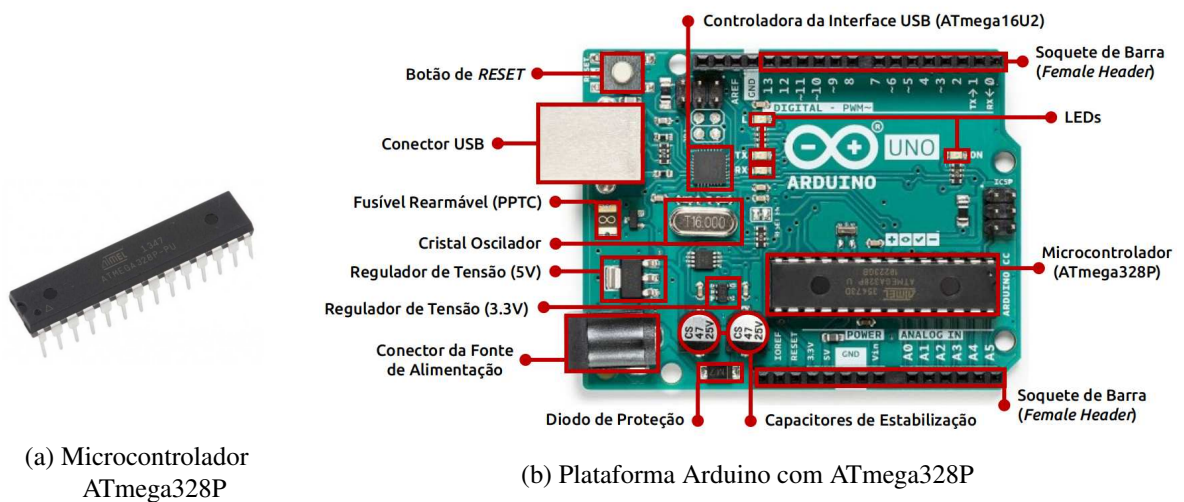
Figura 2 – Placa de circuito eletrônico desenvolvida com o PIC18F14K50.



Fonte – Retirado de Calderón et al.(18).

(3a) e ao seu lado uma plataforma de prototipagem baseada nele (3b), a provisão de uma arquitetura especificadamente preparada para a integração do MCU, com os diversos componentes requeridos por ele, resulta em uma PCB pronta para o uso. Desta forma, simplificando excepcionalmente o processo de desenvolvimento dos projetos, sobretudo, para os usuários mais leigos em eletrônica.

Figura 3 – Plataforma de prototipagem facilitando a implementação de microcontroladores



Fonte – Adaptadas do site da fabricante do Arduino.

Para ilustrar o uso dessas placas dentro dos laboratórios de física, temos o Arduino sendo empregado como dispositivo de aquisição de dados no experimento didático proposto por Cavalcante et al.(2). Em seu trabalho, a plataforma de prototipagem fez o monitoramento da tensão elétrica durante as fases de carga e descarga de um capacitor eletrolítico. Ao final dos ensaios, era gerado um gráfico com o registro temporal da variação de tensão em cada estágio desse componente eletrônico.

2.2 O ARDUINO

Em face das diversas plataformas disponíveis no mercado aptas às atividades de aquisição de dados, o Arduino é uma opção inteiramente *open-source* que se destaca pela sua extensa comunidade internacional, sua simplicidade de programação, seu fácil manuseio e seu baixo custo (1) (20). Originado na Itália, com o objetivo de facilitar a prototipagem de sistemas eletrônicos microcontrolados, esse projeto foi difundido ao redor do mundo tornando-se muito popular entre o público em geral (3) (20).

O Arduino dispõe atualmente de várias versões, cada qual com a sua proposta. Assim, a plataforma oferece uma ampla versatilidade aos desenvolvedores, conforme é percebido tanto nas especificações técnicas dos modelos apresentados na Tabela 1, quanto nas suas características físicas contempladas nas fotos da Figura 4. Por exemplo, o Arduino Micro pode ser mais propício num projeto que possua restrições de peso ou dimensões físicas, enquanto que a versão Mega pode ser uma escolha mais pertinente para lidar com trabalhos que exigem um maior número de portas, pois poupará o usuário de anexar circuitos de multiplexação adjacentes.

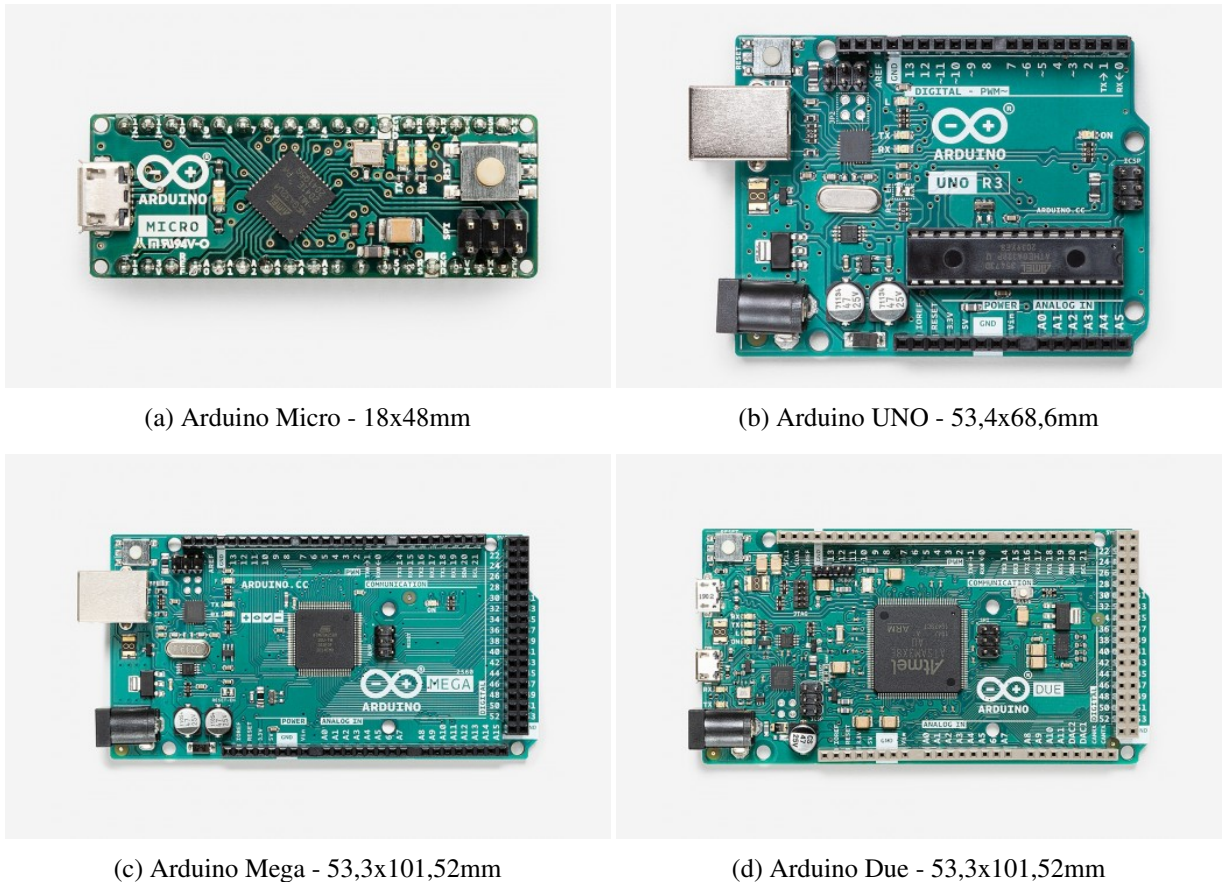
Tabela 1 – Especificações de alguns dos principais modelos da Plataforma Arduino.

Descrição	Micro	UNO	Mega	Due
Microcontrolador	ATmega32U4	ATmega328P	ATmega2560	AT91SAM3X8E
Tensão de Operação	5V	5V	5V	3.3V
Tensão de Alimentação	6-9V	6-20V	6-20V	6-16V
Frequência de Clock	16 MHz	16 MHz	16 MHz	84 MHz
Portas Digitais	20	14	54	54
Portas com PWM	7	6	15	12
Portas Analógicas	12	6	16	12
Memória Flash	32 KB	32 KB	256 KB	512 KB
Bootloader (Flash)	4 KB	0.5 KB	8 KB	–
SRAM	2.5 KB	2 KB	8 KB	96 KB
EEPROM	1 KB	1 KB	4 KB	–
Comprimento	48 mm	68.6 mm	101.52 mm	101.52 mm
Largura	18 mm	53.4 mm	53.3 mm	53.3 mm
Peso	13 g	25 g	37 g	36 g

Fonte – Adaptada do site da fabricante do Arduino.

Quanto ao *software*, a fabricante disponibiliza gratuitamente em seu site as bibliotecas do Arduino e um ambiente de desenvolvimento integrado (Figura 5a), mais conhecido como *Integrated Development Environment* (IDE), compatível com os sistemas operacionais:

Figura 4 – Características físicas de algumas das principais versões do Arduino.



(a) Arduino Micro - 18x48mm

(b) Arduino UNO - 53,4x68,6mm

(c) Arduino Mega - 53,3x101,52mm

(d) Arduino Due - 53,3x101,52mm

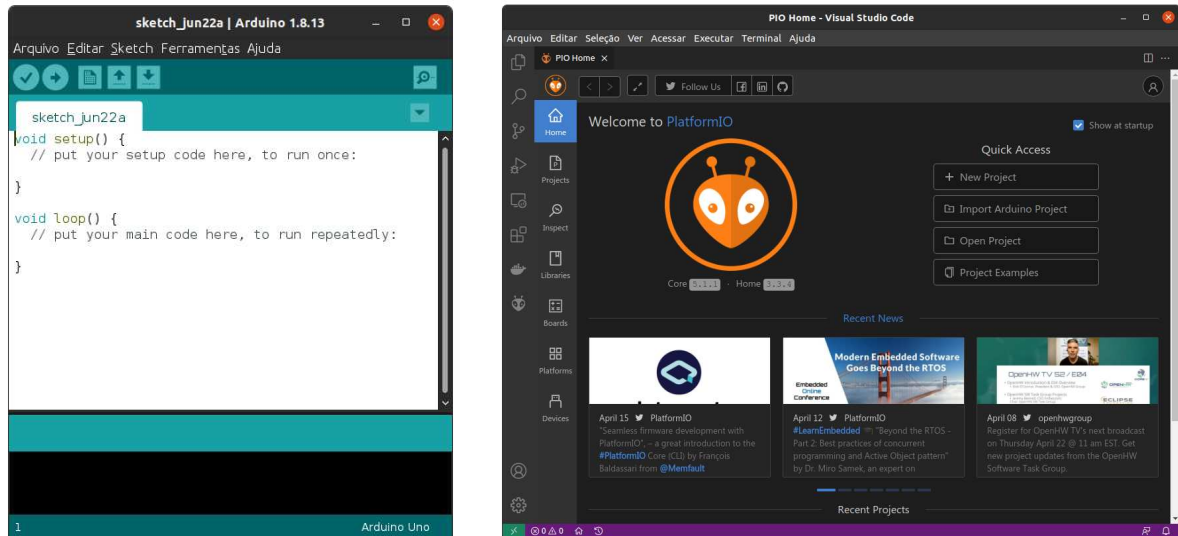
Fonte – Site da fabricante do Arduino.

Windows, Linux e Mac OS. Com esse ambiente, os usuários já têm o essencial tanto para criar e editar códigos-fonte como também para compilar e enviar programas a qualquer modelo da plataforma. Contudo, caso o desenvolvedor prefira trabalhar com uma IDE diferente, seja por questão de familiaridade, seja por necessitar de recursos específicos, as principais opções do mercado já oferecem compatibilidade mediante a instalação de *plugins*, é o caso do Microsoft Visual Studio Code (Figura 5b), IBM Eclipse, Apache Netbeans entre outras ferramentas.

Apesar de usar a linguagem C++, temida pelos calouros por conta dos seus conceitos de manipulação de ponteiros e alocação dinâmica da memória, o Arduino oferece uma experiência de programação bastante descomplicada e convidativa para os novos usuários da plataforma. Parte desse mérito deve-se à simplicidade de sua IDE, junto com a gama de exemplos contidos nela e as suas bibliotecas, as quais facilitam a implementação de diversos recursos.

Ademais, o Arduino é uma plataforma já preparada para trabalhar com alguns protocolos de comunicação, tais como: Receptor/Transmissor Universal Assíncrono ou *Universal Asynchronous Receiver/Transmitter* (UART), Circuito Inter-integrado ou *Inter-Integrated Circuit*

Figura 5 – Alguns ambientes de desenvolvimento integrado para o Arduino



(a) Arduino IDE

(b) Visual Studio Code com o *plugin* PlatformIO

Fonte – Elaborado pelo Autor (2021).

(I²C) e Interface Periférica Serial ou *Serial Peripheral Interface* (SPI). Esta aptidão proporciona a integração com uma infinidade de componentes, como sensores, atuadores, *displays*, módulos, *shields*, unidades de memórias entre outros periféricos que colaboram para a popularidade da plataforma.

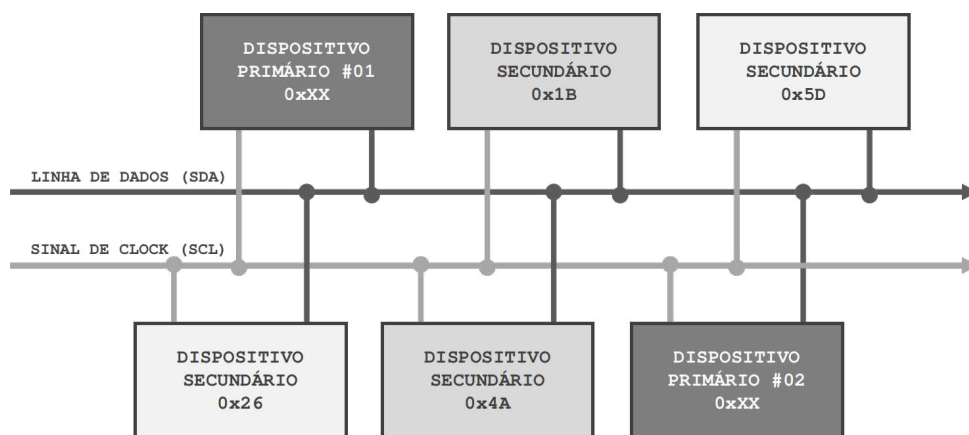
2.3 PROTOCOLOS DE COMUNICAÇÃO

Um protocolo de comunicação tem como objetivo estabelecer um conjunto de regras e procedimentos que devem ser seguidos em ambos os lados para possibilitar a conversa entre dispositivos (21). As unidades de memória externa encontradas com maior facilidade no mercado utilizam dois protocolos de comunicação: o I²C e o SPI. Esses são protocolos já conhecidos pelos desenvolvedores do Arduino, pois estão presentes em diversos periféricos do cotidiano, como nos módulos de *Liquid Cristal Display* (LCD) e nos *Ethernet Shields*. A plataforma já proporciona nativamente algumas bibliotecas a fim de amparar a implementação desses recursos, conservando, portanto, a simplicidade para os seus usuários.

Desenvolvido pela Philips, o I²C é um protocolo de comunicação síncrono *Half-Duplex*, ou seja, a sua conexão não permite o envio e recebimento simultâneo de dados. Conforme é apresentado na Figura 6, esse barramento permite a conexão com múltiplos dispositivos, tanto primários quanto secundários, com apenas dois fios: um destinado à linha de dados (SDA) e outro

para a difusão do sinal de *clock* (SCL) (22). Essa característica econômica é bastante conveniente aos microcontroladores de baixo custo, visto que não é necessário dedicar muitas portas na implementação dessa comunicação. Na ausência de um canal para seleção dos dispositivos, cada componente nesse barramento é identificado por um endereço, normalmente de 7 bits, possibilitando, assim, o correto tráfego dos dados entre eles. Embora o I²C esteja numa revisão que propicia alcançar taxas de transmissões na ordem dos *megabits* por segundo, a maioria dos equipamentos adotam as versões anteriores do protocolo, limitando-se, conseqüentemente, a velocidades menores, geralmente em torno dos 100 a 400 Kb/s, em troca de uma compatibilidade mais abrangente com outros sistemas (23).

Figura 6 – Barramento do Protocolo de Comunicação I²C



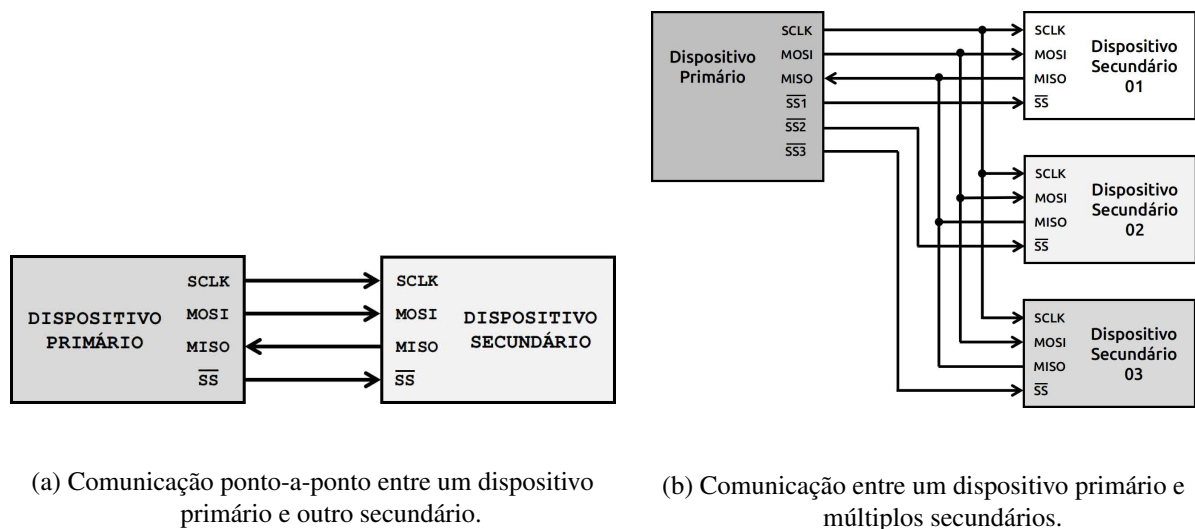
Fonte – Elaborada pelo Autor.

Já o SPI é um protocolo de comunicação serial síncrono *Full-Duplex*, isto é, uma conexão em que seus componentes podem enviar e receber dados simultaneamente. Desenvolvido pela Motorola, esse protocolo também permite interligar vários dispositivos, no entanto, somente um exercerá o papel de primário nessa conexão (24). Como pode ser visto na Figura 7a, a comunicação depende de um barramento contendo: uma porta de sinal de *clock* de responsabilidade do dispositivo primário, *Serial Clock* (SCLK); uma porta de envio de dados do dispositivo primário para os secundários, *Master Output Slave Input* (MOSI); outra para os dispositivos secundários enviarem os dados para o primário, *Master Input Slave Output* (MISO); e uma última porta para a seleção do dispositivo secundário durante a transmissão de dados, *Slave Select* (SS) ou *Chip Select* (CS).

Em resumo, comparando esses dois protocolos, o SPI começa em desvantagem por não proporcionar uma boa escalabilidade, dado que para cada dispositivo secundário inserido no

projeto é necessário dedicar uma nova porta de seleção no dispositivo primário, Figura 7b. Do outro lado, temos o I²C oferecendo não só um barramento mais simples como também capaz de trabalhar com múltiplos dispositivos primários, apresentando, dessa forma, uma topologia inegavelmente superior. Já no quesito velocidade, apesar do SPI não definir uma taxa de transmissão limite em sua documentação, frequentemente as suas implementações ultrapassam os 10 Mb/s, sendo, portanto, claramente superior aos 400 Kb/s, tipicamente encontrado nas fichas técnicas de componentes I²C, ou aos 3,4 Mb/s, alcançados apenas no modo mais rápido desse protocolo, o qual nem sempre está disponível e implica em uma possível perda de compatibilidade com outros equipamentos (22).

Figura 7 – Duas topologias com o barramento SPI



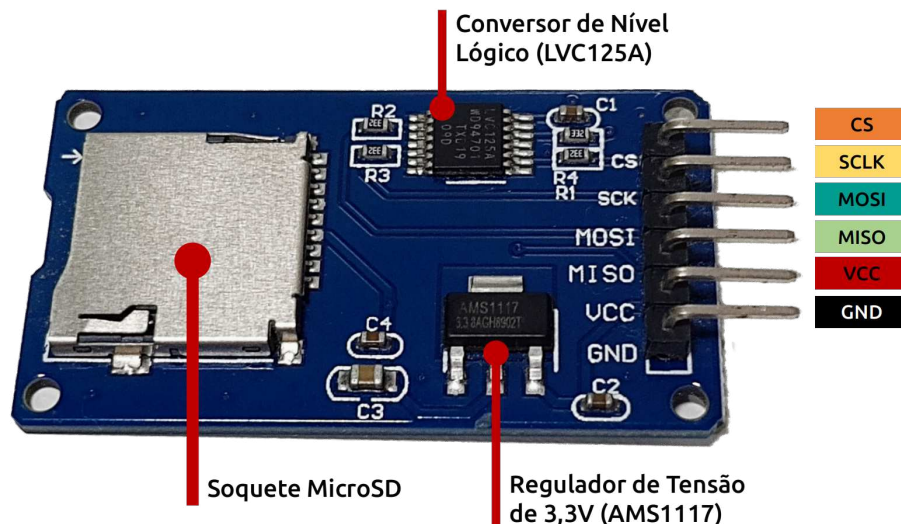
Fonte – Adaptada de Leens(22).

Entretanto, no ponto de vista da transferência de dados, somente essa comparação de taxas de transmissão não é o suficiente para assegurar alguma superioridade categórica aos dispositivos que usam determinado barramento, pois, há especificidades técnicas que vão além dos protocolos e precisam ser consideradas. Um grande exemplo dessa discordância entre as características técnicas do protocolo e o desempenho entregue pelo componente é notado nos experimentos didáticos envolvendo a aquisição de dados com os módulos leitores de cartões de memória. Apesar de empregarem uma comunicação de alta velocidade via barramento SPI, a arquitetura resultante não alcança taxas de amostragem acima dos 100Hz (17).

2.4 MÓDULO LEITOR DE CARTÕES DE MEMÓRIA

Os módulos leitores de cartões de memória são frequentes nos experimentos de aquisição de dados em virtude dos benefícios que oferecem por um baixíssimo custo (8) (25) (26). Uma das principais vantagens em seu uso é aumento substancial da capacidade de armazenamento do projeto, abrindo possibilidades como a coleta de dados de longo prazo (27) (13). Conforme é observado na Figura 8, além de alguns resistores e capacitores serigrafados na placa com as iniciais R e C respectivamente, sua PCB é composta basicamente por: um soquete para cartões microSD, um regulador de tensão de 3,3V (ASM1117) e um conversor de nível lógico (LVC125A).

Figura 8 – Módulo Leitor de Cartões microSD



Fonte – Elaborado pelo Autor (2021).

Em resumo, com praticidade, o soquete microSD efetua a junção do cartão de memória ao circuito do módulo. Os outros dois componentes são necessários para viabilizar a comunicação com microcontroladores de 5V, como os presentes no Arduino UNO e Mega, aos cartões que, via de regra, trabalham com 3,3V (28). Então, enquanto o regulador ASM1117 fica incumbido de ajustar a tensão de alimentação da placa, o LVC125A, um conversor de nível lógico de barramento quádruplo comum nas aplicações envolvendo sistemas de telecomunicação (29), é encarregado por adequar os sinais em ambos os lados do barramento SPI. Esse último elemento é bastante conveniente, contudo, perfeitamente dispensável – tanto que não aparece na versão SD do módulo.

2.4.1 Conversores Lógicos

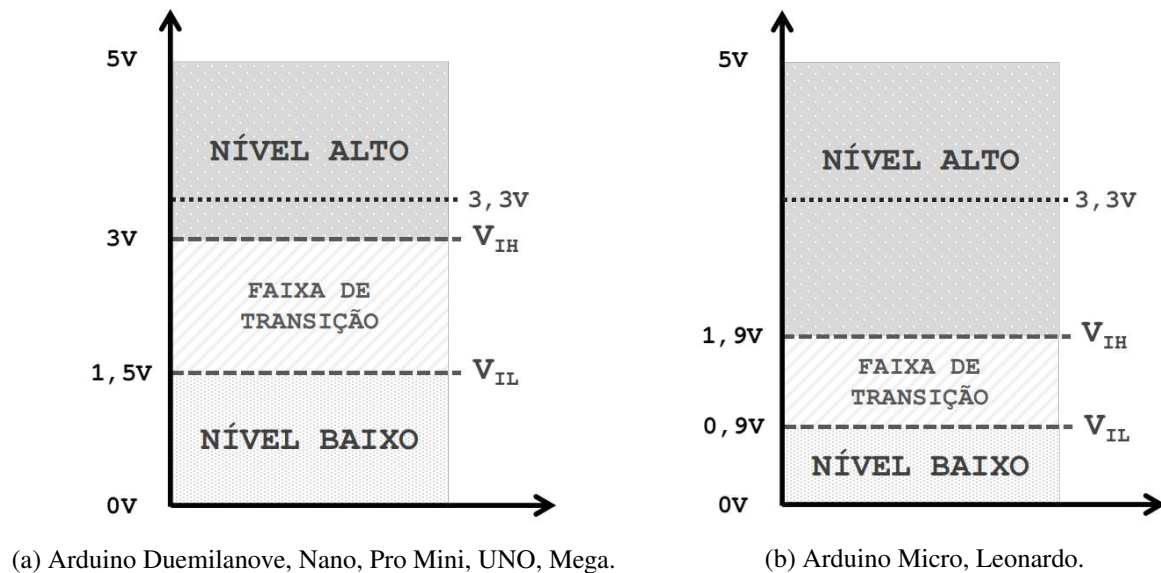
Nos microcontroladores, a leitura de uma certa tensão aplicada numa porta digital resulta em um binário: 0, também lido como nível lógico baixo; ou 1, nível lógico alto. Conhecer os parâmetros de tensão que definem os limites de cada resultado é fundamental para a correta decodificação dos sinais. Essas especificações podem ser encontradas na seção de “Características Elétricas” das folhas de dados dos dispositivos (30), normalmente, pelos dados de tensão de entrada baixa, mais conhecida como *Input Low Voltage* (V_{IL}), e tensão de entrada alta, denominada de *Input High Voltage* (V_{IH}). Geralmente, essas informações são expressas por meio de funções matemáticas embasadas na tensão de alimentação do MCU, ou melhor *Voltage Common Collector* (V_{cc}). Assim, a percepção do atual nível lógico se altera à medida que a tensão com que o componente é alimentado se modifica.

Enquanto que para determinar se o nível lógico de um sinal é baixo ele precisa ser menor ou igual ao V_{IL} , para defini-lo como alto, ele necessita ser igual ou maior ao V_{IH} (31). No universo da prototipagem com o Arduino é extremamente normal se encontrar dispositivos com tensões de alimentação de 3,3V e 5V contendo parâmetros de V_{IL} e V_{IH} distintos e, muitas vezes, incompatíveis para a estabelecer uma comunicação entre eles. Geralmente, isso acontece por conta de dois problemas: quando a tensão do sinal é insuficiente para atingir o V_{IH} do componente receptor, ou seja, encontra-se na faixa de transição, entre V_{IL} e V_{IH} , onde a leitura é indeterminada; ou, o oposto, a tensão do sinal excede à máxima suportada pelo dispositivo, ocasionando, desse modo, em danos nos seus circuitos internos.

A comunicação entre dois componentes com tensões divergentes é amparada por meio dos conversores de nível lógico, circuitos auxiliares encarregados na adequação desses sinais. Essa conversão pode ser simplesmente unidirecional, convertendo uma tensão maior para uma menor, porém não admitindo o contrário, como é feita através dos circuitos divisores e reguladores de tensão; ou bidirecionais, um pouco mais elaborada, permitindo o ajuste do sinal em ambas as direções, como a executada pelo LVC125A (29). Os conversores bidirecionais são usados, principalmente, nos cenários em que precisamos amplificar um sinal para que esse entre nos parâmetros V_{IH} do receptor e, dessa forma, saia de uma provável faixa de transição e seja compreendido como um sinal de nível lógico alto.

O LVC125A do módulo leitor de cartões microSD se torna dispensável a medida que analisamos o V_{IH} dos microcontroladores utilizados nas placas do Arduino. De acordo com as folhas de dados dos ATmega328 e ATmega2560, embarcados, dentre outras variantes, no Arduino UNO e Mega respectivamente, possuem um V_{IH} determinado em 60% de V_{cc} (30) (32). Como os MCU estão embarcados numa plataforma de prototipagem com tensão de alimentação regulada em 5V, isso resulta em um valor fixo de 3V para se reconhecer um sinal como nível lógico alto, Figura 9a. Outros modelos como o ATmega32U4, equipados no Arduino Micro e Leonardo, o V_{IH} é estabelecido em 20% de V_{cc} + 0.9V, totalizando em 1.9V (33), um valor ainda menor que as versões citadas anteriormente, Figura 9b.

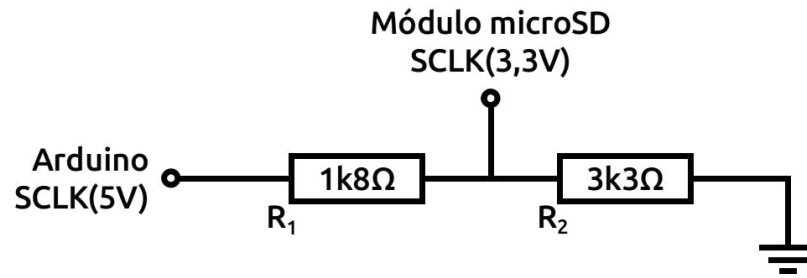
Figura 9 – Gráfico com V_{IL} e V_{IH} de alguns modelos do Arduino



Fonte – Elaborado pelo Autor (2021).

Dessa forma, conforme apresentado nos gráficos da Figura 9, qualquer conversão de sinais originados do módulo microSD, em 3.3V, com destino ao Arduino, mesmo para essas versões de 5V da plataforma, se tornaria uma tarefa dispensável, posto que a tensão dos sinais são suficientes para atingir o V_{IH} desses microcontroladores e, portanto, independentemente do uso ou não de conversores, teriam os seus níveis lógicos sendo interpretados corretamente. Já no caminho oposto, sinais de 5V enviados do Arduino para o módulo, estes sim, precisariam ser reduzidos a fim de proteger a integridade do dispositivo, no entanto, é uma situação de conversão unidirecional, logo, podem ser solucionadas apenas com um circuito divisor de tensão, como o da Figura 10.

Figura 10 – Exemplo de circuito divisor de tensão para as portas do módulo microSD

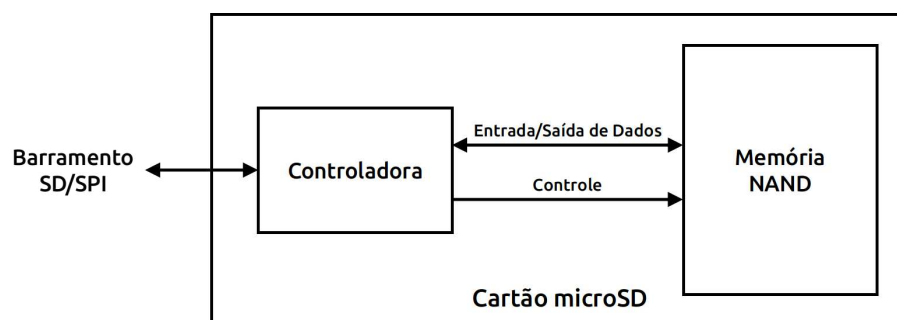


Fonte – Elaborado pelo Autor (2021).

2.4.2 Arquitetura do Módulo MicroSD

Essa investigação inicial na composição da PCB, verificando a disposição, interação e responsabilidade de cada elemento, teve um papel fundamental e bastante esclarecedor sobre o seu funcionamento. Diferentemente de outros, como o módulo WiFi ESP8266 ESP-1, o módulo microSD não dispõem de componentes desempenhando algum tipo de processamento, por exemplo, a conversão de dados para o modo serial. Na verdade, todo o trabalho fica sob responsabilidade de uma controladora embarcada nos próprios cartões (28). No diagrama de blocos disposto na Figura 11, podemos observar que ela, através de comandos de controle, é capaz de ler e escrever dados na memória NAND atendendo as demandas recebidas pelo barramento SPI.

Figura 11 – Diagrama de bloco da arquitetura de um cartão microSD



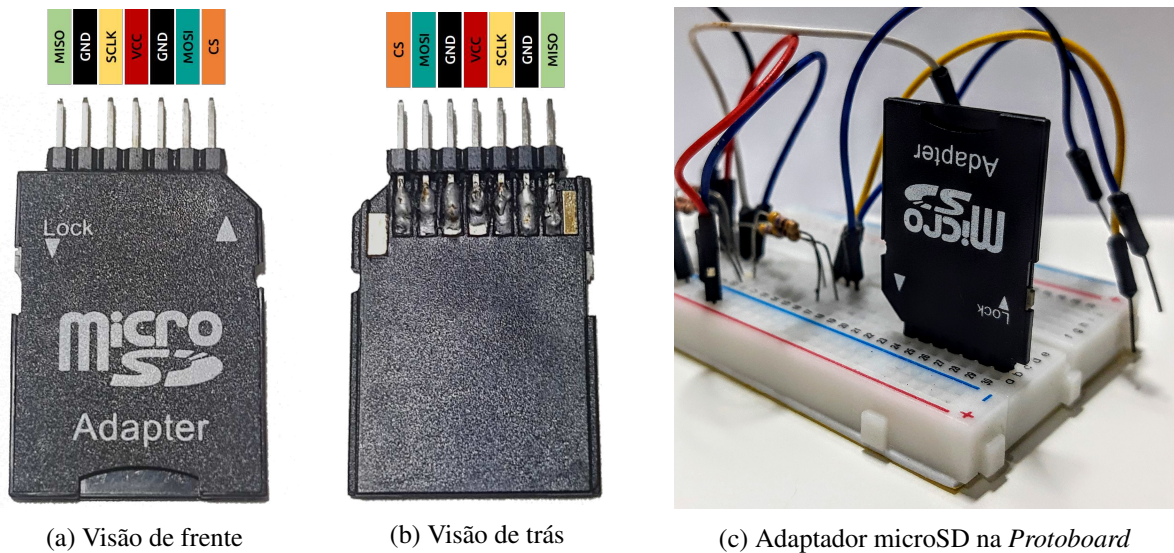
Fonte – Adaptado de SanDisk Corporation(28).

Portanto, sabendo-se que a controladora está presente dentro dos cartões, o módulo microSD se resume em apenas uma interface física. Esta é uma conclusão excepcionalmente relevante, pois, em sistemas baseados em microcontroladores de 3,3V, como nos experimentos conduzidos com o Arduino Due, além de um capacitor de desacoplamento, bastaria somente

possuir um soquete adequado ou qualquer outro elemento capaz de proporcionar a devida conexão dos contatos do cartão ao barramento SPI do microcontrolador. Certamente, também poderíamos estender essa solução as versões de 5V da plataforma, como a UNO e a Mega, simplesmente aproveitando o regulador de tensão interno dessas placas e um circuito de resistores estruturados como divisores de tensão.

Em face dessas conclusões, com a finalidade de verificá-las, confeccionamos um aparato análogo ao módulo microSD, porém extremamente simples usando não mais que um adaptador de cartões com uma barra de pinos soldada nele, vide Figura 12. Esse modelo de adaptador é bem conhecido pelo público em geral, pois é um item frequentemente encontrado nas embalagens de venda dos cartões de memória. No entanto, ele foi particularmente escolhido devido a formidável área de contato de seus terminais, os quais facilitaram, desse modo, o processo de soldagem dos pinos. Note-se que, apesar de ordenada de maneira distinta, a pinagem resultante segue equivalente as contidas nos módulos, tendo apenas como detalhe a presença de uma porta GND a mais.

Figura 12 – Aparato confeccionado em alternativa ao módulo leitor de cartões microSD

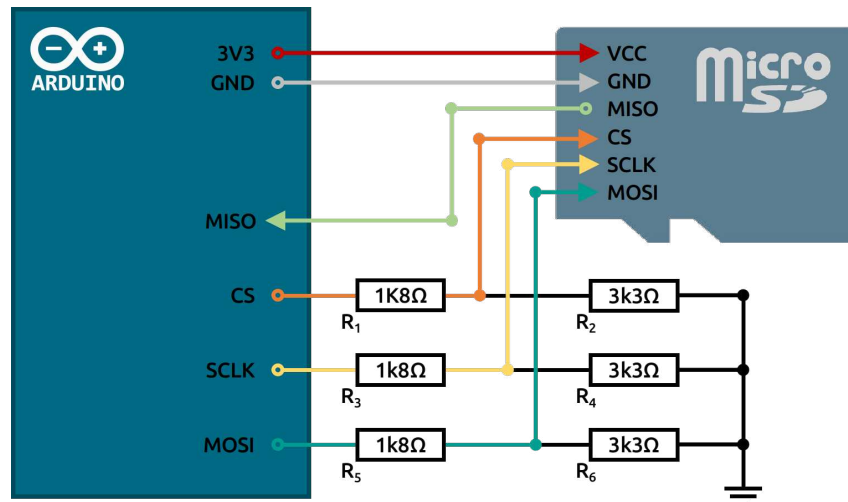


Fonte – Elaborado pelo Autor (2021).

Embora muito simples, conforme esquema elétrico da Figura 13, não apresentamos essa implementação com o intuito de propô-la como uma alternativa aos módulos, longe disso, já que são aparatos de baixíssimo custo e dispõe de um regulador de tensão de 3,3V mais robusto, podendo fornecer uma corrente de até 1A, em comparação com entregue pelo Arduino, com corrente máxima de 150mA. Entretanto, essa exposição serve para: (i) demonstrar que, além de

possível, é bem fácil reproduzi-la; (ii) o quão trivial é a implementação desses módulos, revelando, assim, a origem de seu baixo custo; e, por último, (iii) constatar que são dispositivos desprovidos de um componente destinado ao processamento de dados, tão pouco para viabilização da comunicação SPI, pois, como visto, tudo isso fica à cargo das controladoras embarcadas nos cartões de memória.

Figura 13 – Esquema elétrico da ligação do Arduino ao cartão microSD sem um módulo



Fonte – Elaborado pelo Autor (2021).

Dessa forma, não há fundamentos para imputar aos módulos leitores de cartões SD, e suas variantes, qualquer responsabilidade por alguma limitação de desempenho que, porventura, venha a ser detectada nos ensaios experimentais, ou seja, vamos de encontro com as conclusões do trabalho de González et al.(17). Mais precisamente, em oposição ao trecho que ele atribui o gargalo de performance encontrado em seus experimentos a esses dispositivos: “This was because it was the SD module which limited the acquisition speed and not the Arduino board microcontroller (17, p. 11)”.

De acordo com a nossa investigação, não podemos responsabilizar os módulos por essa limitação de desempenho. Como também, certamente, não podemos apontar para os cartões de memória, já que eles conseguem trabalhar em uma velocidade padrão de 25MHz, podendo chegar aos 100MHz em algumas configurações (28). Basta lembrar que celulares, *tablets*, computadores transferem, leem e reproduzem conteúdos em alta resolução por meio dessas unidades de armazenamento, atendendo às demandas em frequências que estão muito acima da barreira dos 80Hz encontrados nos experimentos (17).

3 DESCRIÇÃO DO PROJETO

A primeira etapa da metodologia deste trabalho visa selecionar os dispositivos de memória disponíveis no mercado melhor classificados para DAQ. Essas unidades de memória passarão por uma triagem baseada nas especificações técnicas fornecidas pelos seus fabricantes. Em seguida, foram avaliadas e categorizadas conforme o seu protocolo de comunicação. No final, além do módulo leitor de cartões microSD, elemento de referência para as posteriores comparações, haverá também, no mínimo, mais duas arquiteturas de armazenamento não volátil: uma I²C e outra SPI.

Após a triagem, se iniciará a bateria de testes sintéticos. O objetivo desta fase é mensurar a performance bem como o nível de confiabilidade no processo de persistência dos dados de cada uma das memórias. Para isso, dois testes foram aplicados: no primeiro, o Arduino medirá o tempo de execução das operações de escrita utilizando diversos tamanhos de blocos de dados; já no segundo, a plataforma transmitirá apenas blocos de 32B, porém, ao longo dos ensaios, interrupções sistemáticas ocorrerão na fonte de alimentação com o auxílio de um circuito temporizador. Neste estágio, o intuito é estressar os componentes a fim de identificar seus limites e possíveis perdas ou corrupções de dados.

Depois de testar e produzir os relatórios dos ensaios experimentais, se fará um confronto entre os resultados obtidos e os valores nominais encontrados nas especificações técnicas de cada dispositivo. Em seguida, essas memórias foram submetidas às análises comparativas, tanto entre as selecionadas na etapa anterior quanto entre as observadas ordinariamente na literatura. Dessa forma, este trabalho terá o embasamento desejável para propor a arquitetura de armazenamento detentora das melhores qualificações para as atividades de aquisição de dados desse escopo. Em conclusão, simulações foram formalizadas, desenvolvidas e executadas para validar essa proposta.

3.1 SELEÇÃO DOS DISPOSITIVOS

Para esta etapa de seleção das memórias foram considerados alguns critérios: (i) o item deve ter disponibilidade no mercado nacional; (ii) tratar-se de dispositivos de armazenamento

não volátil; (iii) lidar com comunicação I²C ou SPI, por serem protocolos com as maiores taxas de transmissão entre os disponíveis no Arduino; (iv) ter capacidade mínima de 8KB, o que corresponderia a pouco mais de 10 segundos de aquisição de dados em uma frequência de 100Hz com a utilização de registros de 6 Bytes, nos quais 2 são usados para guardar o resultado da conversão analógica-digital de um sensor e os outros 4 para armazenar o tempo retornado pelas funções *micros()* e *millis()*; e (v) ser capaz de operar com a alimentação de 3,3V, posto que essa tensão está presente em todos os modelos da plataforma, garantindo, portanto, uma maior compatibilidade, mesmo isso implicando no uso de circuitos conversores de nível lógico nas principais versões, como o Arduino Mega e UNO.

Durante a busca pelas unidades de memória, foram pesquisados sistematicamente os termos “memória”, “eeprom”, “flash”, “i2c”, “spi”, “sd”, “at24c” e “w25q” nos principais sites de *e-commerce* de componentes eletrônicos. A Tabela 2 expõe o levantamento dos dispositivos em cada uma das lojas consultadas. Dentre os resultados, além dos populares módulos leitores de cartões SD e microSD, também foram descobertas as unidades das séries AT24CXX, encontradas nos modelos de 1Kb a 256Kb com ou sem módulo, e Winbond W25QXX, visto apenas incorporados em módulos com capacidade de 32Mb e 64Mb.

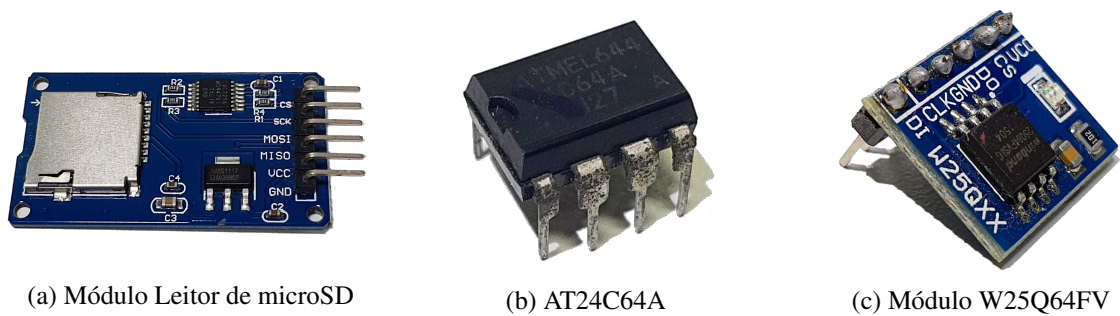
Tabela 2 – Resultado da Pesquisa pelos Dispositivos de Memória nas Lojas Virtuais.

Loja Virtual	Módulo microSD Card	EEPROM I ² C AT24CXX	Flash SPI W25QXX
Arduino Belém	X		
Arduinolândia			
AutoCore Robótica	X	X	X
Baú da Eletrônica	X	X	X
Casa da Robótica	X		
Curto Circuito	X	X	X
Eletrogate	X	X	X
FilipeFlop	X		
Instituto Digital	X	X	
Jasam	X		
Loja da Robótica			
Master Walker	X	X	
RoboCore	X		
RS Robótica	X		
Smart Projects	X		
Solda Fria	X		
UsinaInfo	X	X	X

Fonte – Elaborado pelo Autor (2021).

Finalmente, levando em consideração os critérios de seleção, de preço e dos prazos de entrega dos produtos, além do módulo leitor de cartões microSD, Figura 14a, ainda foram adquiridos um AT24C64A, Figura 14b, e um módulo W25Q64FV, Figura 14c. A versão da plataforma Arduino adotada para este trabalho foi a UNO, em função da sua popularidade, custo acessível e alta disponibilidade no mercado. Assim, estes 3 dispositivos seguem para a fase de testes sintéticos.

Figura 14 – Dispositivos de armazenamento externo adquiridos



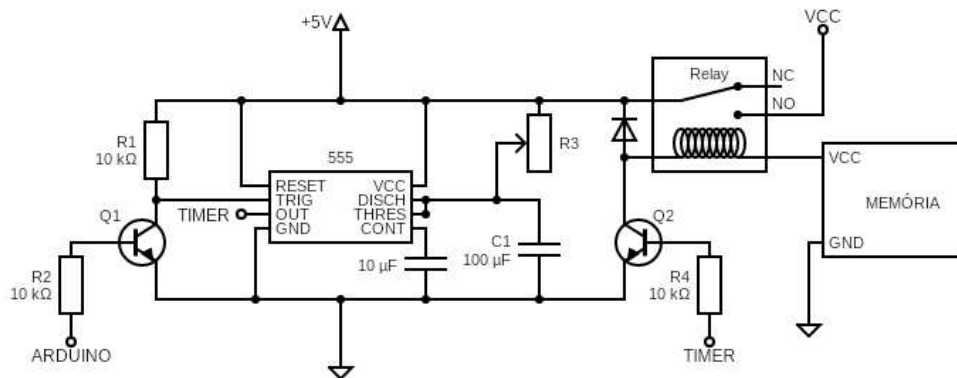
Fonte – Elaborado pelo Autor (2021).

3.2 TESTES SINTÉTICOS

Após selecionar e obter as unidades de armazenamento, o próximo passo era submetê-las aos testes sintéticos. Para isso, iniciou-se a etapa de planejamento e construção dos circuitos auxiliares, elementos necessários no amparo dos ensaios experimentais. Além dos divisores de tensão, destinados à conciliação dos sinais, também foi preciso implementar um temporizador monoestável com o NE555, conforme o esquema elétrico da Figura 15. Aliado a um relé, esse temporizador exercia o controle no chaveamento da alimentação da memória. Outro integrante importante deste arranjo era o potenciômetro R3, um componente eletrônico de resistência ajustável que serviu como mecanismo de regulação e, assim, tornou o sistema adaptável ao tempo de execução de cada dispositivo. Desta forma, podia-se simular as falhas de interrupção no fornecimento energético de modo preciso e sistemático.

Nesse esquema, o Arduino envia um pulso no resistor de base do transistor Q1 para iniciar o temporizador. Com isso, o relé é acionado permitindo a alimentação do dispositivo de memória e habilitando-o ao funcionamento. O tempo em que o circuito se manterá ativo pode ser facilmente calculado utilizando a capacitância de C1 em *Farads* (F) e a resistência do potenciômetro R3 em *Ohms* (Ω) na fórmula (34) a seguir:

Figura 15 – Esquema elétrico do circuito temporizador com NE555.



Fonte – Elaborado pelo Autor (2021).

$$T = 1.1 * C1 * R3$$

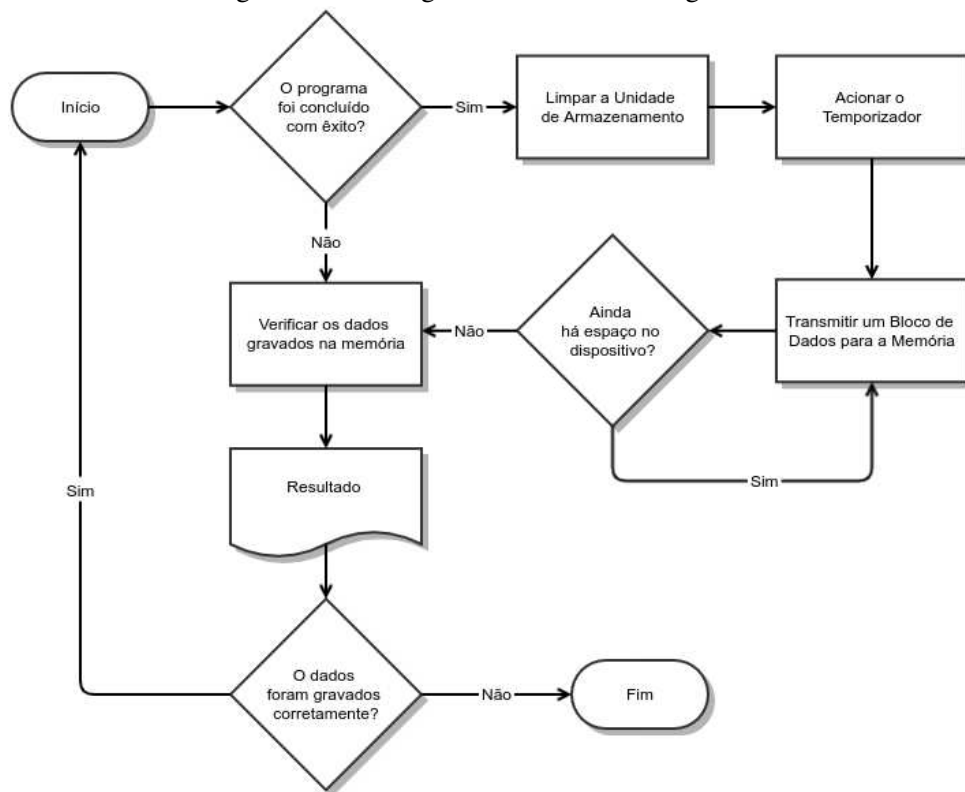
Como o potenciômetro tem uma resistência variável, então o seu valor deve ser verificado por meio de um multímetro toda vez que for regulado. Assim, após aplicar essas informações corretamente na fórmula, ela dará o tempo em segundos no qual o NE555 manterá um pulso alto na sua porta *OUTPUT*. Esse sinal será transmitido para o resistor de base do transistor Q2, que enfim acionará o relé. Neste ponto é importante lembrar que esse componente eletromecânico possui um *delay* de reação, portanto, isso também precisa ser considerado no código. Por exemplo, conforme ficha técnica (35), o tempo máximo de resposta do relé utilizado em nosso circuito é de 10 milissegundos. Dessa forma, devemos respeitar este período antes de tentar acessar a unidade de armazenamento pela primeira vez.

Depois de ajustar os parâmetros de tempo apropriadamente, tanto no *hardware* quanto no *software*, o relé estará pronto para desarmar no meio da transmissão de dados à unidade de armazenamento externo. Este circuito reproduzirá o efeito desejado de interrupção súbita da fonte de alimentação no dispositivo. Dessa maneira, permitirá a prática dos ensaios experimentais que testarão o comportamento das memórias não voláteis perante as situações de falha crítica durante o processo de persistência de dados.

O teste de integridade proposto por este trabalho consiste em simular mil vezes o fenômeno de falha ao longo da transferência de dados entre o Arduino e a unidade de armazenamento externo. Todos os ensaios usarão blocos de 32 *Bytes* para fornecer um parâmetro de comparação. Esse tamanho também é o maior comprimento de bloco presente em todas as arquiteturas de memórias aqui selecionadas. Após detectado o desligamento do dispositivo,

o sistema é reiniciado. Em seguida, o último estado da execução do programa é verificado através da EEPROM interna do Arduino. Então, ao identificar que houve um erro no decorrer da transmissão dos blocos, pula-se as etapas de limpeza e gravação executadas anteriormente na memória para se entrar no modo de verificação dos dados. O objetivo é examinar o conteúdo salvo, contabilizar os *Bytes* corrompidos e imprimir o resultado do teste. O fluxograma disponível na Figura 16 auxilia a compreensão da lógica deste programa.

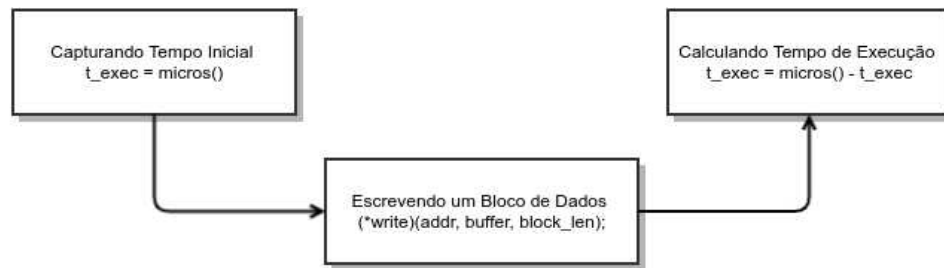
Figura 16 – Fluxograma do Teste de Integridade



Fonte – Elaborado pelo Autor (2021).

Contudo, esse temporizador apenas será empregado nos testes de integridade, devendo ser dispensado ou anulado durante os testes de desempenho. Com exceção dos divisores de tensão, nenhum outro circuito auxiliar se faz necessário para a aplicação desses testes. Eles consistem em mensurar o tempo de escrita da unidade de armazenamento. No código, o tempo atual é obtido com uma resolução de quatro microssegundos através do função *micros()* e o cálculo da duração desse ciclo é feito conforme é mostrado no fluxograma da Figura 17. Uma bateria com dez rodadas de testes é executada para cada comprimento de bloco de dados. Esse tamanho é dobrado entre as execuções, começando de um *Byte* e indo até o maior comprimento disponível pela arquitetura de memória em mãos, sendo: 32B para a EEPROM I²C AT24C64A (36), 256B para a *Flash* SPI W25Q64FV (37) e 512B para os cartões de memória microSD(28).

Figura 17 – Fluxograma de Medição do Tempo de Execução das Operações



Fonte – Elaborado pelo Autor (2021).

O conteúdo desses blocos é outro ponto bastante relevante: blocos totalmente preenchidos com os mesmos *Bytes*, como zeros ou uns, bem como sequências de zero a nove, não são abordagens interessantes. Pois, o microcontrolador do Arduino, assim como a unidade de armazenamento, estarão trabalhando em condições extremas e o risco de acontecer erros durante as operações não deve ser desprezado. Nessas circunstâncias, os dados enviados através do último bloco, os quais ainda estarão com os seus *Bytes* salvos no *buffer*, podem ser facilmente replicados para o restante da memória. Desse modo, a falha passaria despercebida nos testes devido a homogeneidade dos dados.

Assim, precisamos de dados que sejam pseudoaleatórios: aleatórios o suficiente para evitar que os componentes "trapaceiem" mas não tanto ao ponto de incapacitar a sua verificação. À vista disso, este trabalho adotou o uso de π , um número irracional bastante conhecido e que contém uma sequência de dígitos suficientemente diversificada. Então, conforme pode-se observar no Apêndice A, um vetor de 1.024 *Bytes* contendo os algoritmos de π foi anexado ao código-fonte do programa servindo de referência para o preenchimento dos blocos de dados que foram transmitidos ao longo dos testes.

Com os circuitos auxiliares devidamente construídos, programa contendo os testes de desempenho e integridade desenvolvido, como também o bloco de dados pseudoaleatórios definido, o próximo passo é iniciar os testes sintéticos. Cada dispositivo de armazenamento teve os seus resultados registrados em uma planilha eletrônica. Por meio dela, poderá-se verificar de maneira automatizada os tempos mínimos e máximos encontrados durante os ensaios, bem como os valores de tempos médios, desvios padrões, frequências médias e taxas de transmissão alcançadas já calculados. Por fim, as tabelas resultantes foram incorporadas neste trabalho.

3.2.1 Cartão MicroSD

O primeiro dispositivo a ser testado foi o cartão microSD, usando-se um módulo leitor de cartões. Esse produto está presente na maioria das lojas custando normalmente algo na faixa dos 7 aos 10 reais. Essa arquitetura, associando módulo e cartão de memória, foi selecionada para servir de parâmetro de referência, haja vista a sua recorrente implementação na literatura. Assim, ela será empregada neste trabalho visando servir de comparação com as demais soluções de armazenamento de dados.

Nos ensaios experimentais, utilizaremos um cartão de memória básico da SanDisk com capacidade de 1GB, previamente formatado com sistema de arquivos FAT32. O Arduino transmitirá os dados por meio do protocolo de comunicação SPI valendo-se das bibliotecas nativas da plataforma, como a SPI.h, SD.h e SdFat.h. Depois de inicializar corretamente o microSD, um arquivo era aberto no modo de gravação. Em seguida, começava-se o laço de repetição destinado à transmissão dos dados. Neste processo tinha-se: (i) os comandos para manipulação do *buffer*; (ii) os procedimentos para cálculo do tempo de execução das operações; e (iii) as funções para o envio dos dados à unidade de armazenamento.

Quanto as funções, tinha-se duas: *file.write()* e *file.flush()*. A primeira era empregada para a escrita dos dados no arquivo, enquanto que a segunda, aplicada para garantir que esses *Bytes* transmitidos tivessem sido salvos fisicamente no cartão de memória. Sem essa última, obtêm-se comportamentos inesperados assim como tempos inconsistentes. Os resultados da bateria de testes podem ser acompanhados por meio da Tabela 3.

Tabela 3 – Resultado dos Testes Sintéticos do Módulo Leitor de Cartões microSD

Tamanho do Bloco	Tempo Mínimo	Tempo Médio	Tempo Máximo	Desvio Padrão	Total de Operações	Frequência Média	Taxa de Transmissão
1B	13,7ms	16,8ms	168,6ms	10,8ms	2.560	60Hz	0,47Kb/s
2B	13,7ms	16,8ms	171,0ms	10,3ms	2.560	60Hz	0,93Kb/s
4B	12,1ms	16,9ms	170,6ms	10,8ms	2.560	59Hz	1,84Kb/s
8B	11,6ms	16,8ms	168,3ms	10,3ms	2.560	59Hz	3,72Kb/s
16B	11,7ms	17,1ms	155,6ms	10,6ms	2.560	58Hz	7,30Kb/s
32B	11,4ms	16,9ms	170,8ms	10,3ms	2.560	59Hz	14,76Kb/s
64B	11,4ms	16,9ms	178,2ms	10,8ms	2.560	59Hz	29,60Kb/s
128B	11,4ms	16,8ms	171,0ms	11,1ms	2.560	60Hz	59,67Kb/s
256B	11,2ms	16,4ms	178,8ms	11,4ms	2.560	61Hz	121,95Kb/s
512B	8,3ms	11,6ms	181,2ms	11,3ms	2.560	86Hz	343,90Kb/s

Fonte – Elaborado pelo Autor (2021).

Nessa unidade de armazenamento foram realizadas dez rodadas com 256 operações de escrita em cada tamanho de bloco. Entre os blocos de 1 a 256 *Bytes* não foi possível constatar uma variação significativa nos tempos médios de execução. Contudo, um ganho de performance de aproximadamente 40% foi identificado com o uso de blocos de 512B, o qual, coincidentemente, também é o tamanho de um setor de memória do cartão microSD. Com esses resultados, alcançou-se os tais 80Hz não ultrapassados no trabalho de González et al.(17), no entanto, ainda distante dos 100Hz desejados por Oliveira et al.(7).

Outro ponto a ser observado nessa arquitetura de armazenamento é em relação ao seu excessivo tempo máximo encontrado ao decorrer das execuções. Frequentemente esse valor era dez vezes superior ao tempo médio da unidade. Em todas as baterias de testes, esse pico foi detectado na primeira operação de escrita no arquivo. Apesar de ser um problema facilmente contornável, precisa-se levar em consideração esse aspecto ao lidar com projetos que envolvam a utilização desse dispositivo.

3.2.2 Memória EEPROM I²C AT24C64A

Posteriormente, ocorreram os testes com o AT24C64A. O AT24CXX é uma série de memórias EEPROM que utiliza o protocolo I²C em sua comunicação. Atualmente, ela é produzida pela Microchip, a mesma fabricante dos microcontroladores ATmega presentes na plataforma Arduino. Entre as suas especificações (36), destaca-se a capacidade de trabalhar em barramentos com frequências de até 400kHz (conhecido como *fast mode*), como também a possibilidade de gravar até 32 *Bytes* numa única transação. Todavia, há uma inconveniência nessas operações de escrita: o seu ciclo de gravação auto-cronometrado pode durar até 5 milissegundos, enquanto outras versões deste mesmo modelo podem chegar até aos 10ms (38). Nesse período outras gravações são intratáveis pela unidade, reduzindo expressivamente o seu desempenho.

Diferentemente das outras arquiteturas, o teste de desempenho na AT24C64A cobriu toda a capacidade de armazenamento do dispositivo. Essa amostra de dados mais ampla visa evidenciar, bem como melhor compreender, a influência do *delay* não determinístico sobre o ciclo de gravação da memória. Os registros dessa primeira bateria de testes, usando apenas os parâmetros padrões da biblioteca nativa *Wire.h*, estão dispostos na Tabela 4. No fim, a memória AT24C64A apresentou resultados interessantes: ela não só ultrapassou os 100Hz mas também

conseguiu ser até cinco vezes mais rápida nas operações de escrita que os cartões microSD.

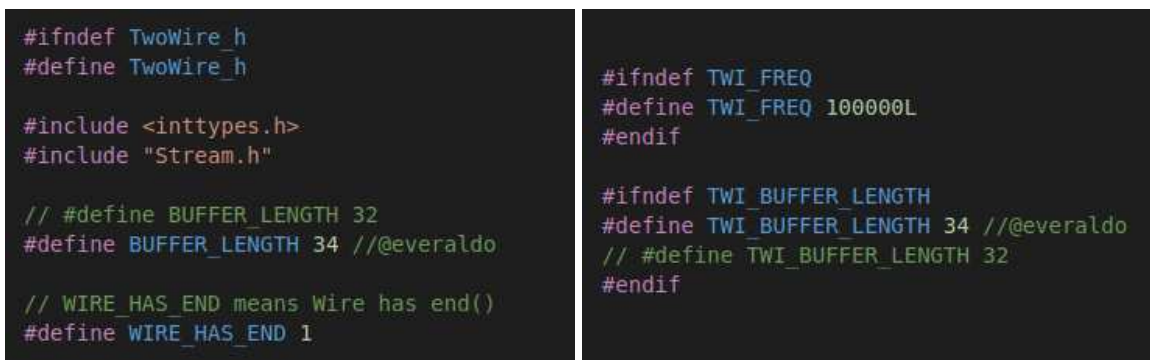
Tabela 4 – Resultado dos Testes Sintéticos da AT24C64A com TWI_FREQ em 100kHz

Tamanho do Bloco	Tempo Mínimo	Tempo Médio	Tempo Máximo	Desvio Padrão	Total de Operações	Frequência Média	Taxa de Transmissão
1B	2.456 μ s	2.594 μ s	2.736 μ s	7 μ s	81.920	385Hz	3,01Kb/s
2B	2.548 μ s	2.687 μ s	2.832 μ s	5 μ s	40.960	372Hz	5,81Kb/s
4B	2.756 μ s	2.895 μ s	2.904 μ s	6 μ s	20.480	345Hz	10,79Kb/s
8B	3.148 μ s	3.294 μ s	3.304 μ s	6 μ s	10.240	304Hz	18,97Kb/s
16B	3.944 μ s	4.089 μ s	4.108 μ s	15 μ s	5.120	245Hz	30,57Kb/s
32B	5.672 μ s	5.702 μ s	5.712 μ s	5 μ s	2.560	175Hz	43,85Kb/s

Fonte – Elaborado pelo Autor (2021).

A única ressalva nas configurações padrões do Arduino nessa primeira bateria de testes foi em relação ao tamanho concedido aos seus *buffers*. A princípio esse atributo é implementado com 32 *Bytes*. No entanto, dois deles são usados no endereçamento desta EEPROM, sobrando apenas 30 para o envio dos dados. Portanto, visando usufruir da transmissão de blocos de 32B, permitidos pela AT24C64A, deve-se alterar para 34 ambas as diretivas: BUFFER_LENGTH em Wire.h, Figura 18a, e TWI_BUFFER_LENGTH em twi.h, Figura 18b.

Figura 18 – Alterações no Tamanho do *Buffer* das Bibliotecas nativas do Arduino



(a) BUFFER_LENGTH em Wire.h

(b) TWI_BUFFER_LENGTH em twi.h

Fonte – Elaborado pelo Autor (2021).

Apesar dos bons resultados, ainda havia espaço para melhoria. Durante os testes, verificou-se outro ponto de ajuste nos parâmetros: a frequência de operação da interface I²C. O Arduino, posicionado como dispositivo principal do barramento, tem a responsabilidade de prover o sinal de *clock* que conduzirá o comportamento de seus componentes secundários. Ao passo que eleva-se essa frequência, minimiza-se o tempo de reposta da comunicação entre eles. Contudo, mesmo a plataforma sendo plenamente apta a trabalhar com pulsos de até 400kHz, de acordo com a ficha técnica de seu microcontrolador (30), o valor configurado nas diretivas da

biblioteca era de apenas 100kHz. Há razões para isso: essa é a velocidade padrão do protocolo, como tal, prover a compatibilidade com um maior número de equipamentos.

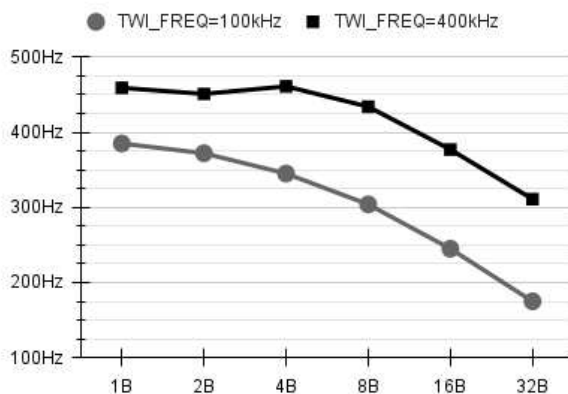
Sendo assim, realizou-se uma nova bateria de testes com o intuito de avaliar os impactos decorrentes da substituição do *clock* padrão pela frequência máxima de 400kHz permitida tanto pelo Arduino quanto pela EEPROM AT24C64A. Para esse fim, apenas precisou-se adicionar uma linha de comando chamando a função *Wire.setClock()*, na qual a frequência desejada é passada por parâmetro substituindo o valor da diretiva *TWI_FREQ* da *twi.h*. Como esperado, houve um ganho expressivo de performance começando aos 19% em transmissões *Byte a Byte* e chegando até aos 77% nos envios em blocos de 32B. Todos os ganhos podem ser conferidos na Tabela 5, como também percebidos nos gráficos da Figura 19.

Tabela 5 – Resultado dos Testes Sintéticos da AT24C64A com *TWI_FREQ* em 400kHz

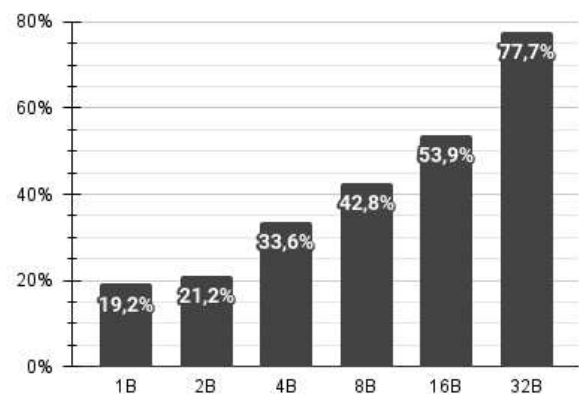
Tamanho do Bloco	Tempo Mínimo	Tempo Médio	Tempo Máximo	Desvio Padrão	Total de Operações	Frequência Média	Taxa de Transmissão
1B	2.108 μ s	2.180 μ s	2.232 μ s	24 μ s	81.920	459Hz	3,58Kb/s
2B	2.176 μ s	2.217 μ s	2.252 μ s	26 μ s	40.960	451Hz	7,05Kb/s
4B	2.040 μ s	2.168 μ s	2.332 μ s	48 μ s	20.480	461Hz	14,42Kb/s
8B	2.184 μ s	2.303 μ s	2.452 μ s	32 μ s	10.240	434Hz	27,14Kb/s
16B	2.588 μ s	2.650 μ s	2.708 μ s	17 μ s	5.120	377Hz	47,16Kb/s
32B	3.148 μ s	3.215 μ s	3.240 μ s	23 μ s	2.560	311Hz	77,76Kb/s

Fonte – Elaborado pelo Autor (2021).

Figura 19 – Gráficos com o Desempenho da Memória EEPROM AT24C64A



(a) Comparação de *TWI_FREQ* 100kHz vs 400kHz



(b) Ganhos usando *TWI_FREQ* em 400kHz

Fonte – Elaborado pelo Autor (2021).

Por fim, foram realizados os testes de integridade na AT24C64A. A unidade apresentou uma taxa de corrupção dos dados de 37,3% enquanto o *clock* de barramento I²C trabalhava a 100kHz e apenas 8,4% quando operava a 400kHz. Em todas as falhas ocorridas ao longo do

experimento, somente o último bloco de 32 *Bytes* era afetado. Entretanto, isso pode ser evitado inserindo um capacitor de 10 μ F entre os terminais de alimentação da memória, realizando algo semelhante ao que é encontrado em alguns módulos de memórias EEPROM. Dessa forma, o capacitor fornece um último suprimento energético para que o sistema possa concluir suas operações antes do completo desligamento, assim, funcionando de modo análogo àqueles dispositivos de *nobreak* encontrados junto a servidores.

3.2.3 Memória Flash SPI W25Q64FV

O último dispositivo selecionado para os testes sintéticos foi a memória W25Q64FV. A série W25QXX de memórias *Flash* é desenvolvida pela Winbond, uma empresa taiwanesa de semicondutores especializada em memórias¹. Dispondo-se do protocolo SPI, essas memórias se comunicam sob frequências de até 416MHz, porém isso dependerá da capacidade do dispositivo primário em trabalhar nesse *clock* e no modo de operação configurado pelo desenvolvedor (37). Essas unidades operam sob a tensão de alimentação de 2,7V até 3,6V, necessitando de algum circuito conversor de nível lógico para adequação dos sinais vindos de microcontroladores de 5V, como aqueles vistos nos Arduino Micro, Mega e UNO.

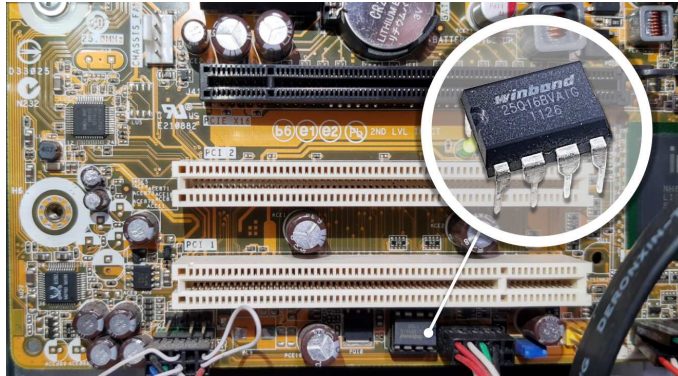
Embora menos frequente nas lojas, essa família de *Flash* é comumente utilizada no armazenamento de *softwares* embarcados, também conhecido como *firmwares*, em diversos aparelhos, inclusive armazenando o *Basic Input/Output System* (BIOS) das placas mãe de computadores (Figura 20a). Outro exemplo, agora mais específico, é o modelo W25Q80B contemplado nos módulos WiFi ESP8266 ESP-01, os quais são largamente usados para habilitar o Arduino às comunicações sem fio (Figura 20b).

Para dar início aos testes foi necessário instalar a biblioteca SPIMemory. Ela é livre e de código aberto, podendo ser facilmente obtida por meio do gerenciador de bibliotecas do próprio ambiente de desenvolvimento do Arduino (Figura 21) ou, para aqueles que utilizam outras IDEs, através de seu repositório no GitHub. Depois de sua instalação, algumas funções estarão a disposição do usuário, por exemplo: a inicialização do dispositivo, as verificações, assim como o controle dos estados da unidade de armazenamento, entre outras operações básicas de limpeza, leitura e escrita na memória.

Antes de iniciar os ensaios, levantou-se a possibilidade da execução de algumas

¹ <https://www.winbond.com>

Figura 20 – Aplicações da Memória *Flash* Winbond da Série W25Q



(a) Memória *Flash* W25Q16B de 16Mb com BIOS embarcada nas placas-mãe Intel DG41WV



(b) Memória *Flash* W25Q80B de 8Mb integrado ao módulo WiFi ESP8266

Fonte – Elaborado pelo Autor (2021).

Figura 21 – Gerenciador de Bibliotecas do Arduino IDE



Fonte – Elaborado pelo Autor (2021).

outras baterias de testes, uma vez que a biblioteca empregada oferecia uma razoável flexibilidade na configuração de seus parâmetros. Neste ensejo, além de examinar o comportamento dos variados ajustes disponíveis, o intuito era otimizar os resultados da memória W25Q64FV ao passo que registrava-se a influência de cada regulagem. Dessa forma, teria-se um relatório minucioso abrangendo as decisões, suas justificativas, os ganhos de desempenho obtidos bem como a evolução das frequências de aquisição ao longo dos experimentos.

Neste ponto, já era esperado uma maior taxa de amostragem em comparação com os demais dispositivos aqui selecionados. Isso muito em razão da frequência de operação que protocolo de comunicação SPI propicia, aliado com a simplicidade da camada lógica desta unidade de armazenamento. Diferentemente do cartão microSD, a arquitetura do W25Q64FV não possui um sistema de arquivos que torna mais lenta a manipulação de seus dados. Assim, essas operações são executadas diretamente na memória, de modo similar ao presenciado na

EEPROM AT24C64A testada anteriormente.

Em face disso, iniciou-se a primeira bateria de testes. Conforme observado na Tabela 6, a memória W25Q64FV já produziu bons resultados lidando apenas com os parâmetros padrão dispostos na biblioteca SPIMemory. Nesse cenário, ultrapassando os 5.000Hz, essa unidade foi amplamente superior aos demais dispositivos, sendo dez vezes mais rápida que o melhor resultado da EEPROM AT24C64A.

Tabela 6 – Resultado da 1ª Bateria de Testes com a Memória *Flash* W25Q64FV

Tamanho do Bloco	Tempo Mínimo	Tempo Médio	Tempo Máximo	Desvio Padrão	Total de Operações	Frequência Média	Taxa de Transmissão
1B	180 μ s	199 μ s	212 μ s	7 μ s	2.560	5.028Hz	39Kb/s
2B	188 μ s	191 μ s	200 μ s	3 μ s	2.560	5.243Hz	82Kb/s
4B	208 μ s	229 μ s	236 μ s	3 μ s	2.560	4.376Hz	137Kb/s
8B	244 μ s	262 μ s	268 μ s	3 μ s	2.560	3.811Hz	238Kb/s
16B	332 μ s	349 μ s	356 μ s	6 μ s	2.560	2.869Hz	359Kb/s
32B	484 μ s	507 μ s	512 μ s	4 μ s	2.560	1.971Hz	493Kb/s
64B	800 μ s	813 μ s	816 μ s	3 μ s	2.560	1.231Hz	615Kb/s
128B	1448 μ s	1460 μ s	1464 μ s	4 μ s	2.560	685Hz	685Kb/s
256B	2740 μ s	2745 μ s	2748 μ s	3 μ s	2.560	364Hz	729Kb/s

Fonte – Elaborado pelo Autor (2021).

Com a conclusão dessa primeira bateria de testes, a unidade da Winbond não só já havia ultrapassado com folga os objetivos de frequência determinados neste trabalho, como também superou o desempenho de todos os dispositivos testados anteriormente. Essa memória permite uma taxa de amostragem acima dos 1.000Hz com blocos de até 64 *Bytes*. Um resultado bastante expressivo, haja vista que a AT24C64A, o dispositivo mais rápido até então, não atingiu nem metade dessa velocidade, mesmo utilizando o menor tamanho de bloco à disposição.

Feita essa constatação, iniciou-se a segunda rodada de testes a fim de explorar as capacidades da W25Q64FV. Nessa bateria, o aprimoramento nos resultados foi alcançado de maneira bastante simples sem alterações no código-fonte da biblioteca, sendo, desse modo, uma otimização acessível aos usuários menos experientes da plataforma Arduino. Para tal, necessitou-se apenas modificar o valor padrão designado ao parâmetro *errorCheck* nas chamadas das operações de escrita. A *writeCharArray* é um exemplo desse tipo de função. Ela é aplicada na gravação de vetores de caracteres e possui a seguinte assinatura:

```
writeCharArray(uint32_t addr, char *dataBuffer, size_t bufferSize, bool errorCheck=true)
```

Ao empregar *true* para *errorCheck* em seu protótipo, a biblioteca torna esse argumento opcional para o usuário. Dessa forma, basta chamar a função passando as informações do endereço de memória, os dados a serem gravados e o tamanho total desse bloco que o valor padrão “*verdadeiro*” será assumido na quarta posição. No entanto, caso o desenvolvedor queira desabilitar essa verificação de erros, ele precisa passar um *false* em seu último parâmetro. Como pode ser visto na Tabela 7, a desativação desse recurso proveu um ganho expressivo na performance da unidade de armazenamento.

Tabela 7 – Resultado da 2ª Bateria de Testes com a Memória *Flash* W25Q64FV

Tamanho do Bloco	Tempo Mínimo	Tempo Médio	Tempo Máximo	Desvio Padrão	Total de Operações	Frequência Média	Taxa de Transmissão
1B	120 μ s	133 μ s	144 μ s	5 μ s	2.560	7.531Hz	59Kb/s
2B	124 μ s	139 μ s	148 μ s	5 μ s	2.560	7.191Hz	112Kb/s
4B	132 μ s	151 μ s	160 μ s	4 μ s	2.560	6.624Hz	207Kb/s
8B	156 μ s	174 μ s	180 μ s	3 μ s	2.560	5.760Hz	360Kb/s
16B	196 μ s	215 μ s	224 μ s	3 μ s	2.560	4.642Hz	580Kb/s
32B	276 μ s	298 μ s	304 μ s	3 μ s	2.560	3.361Hz	840Kb/s
64B	440 μ s	454 μ s	460 μ s	3 μ s	2.560	2.201Hz	1.101Kb/s
128B	772 μ s	783 μ s	788 μ s	6 μ s	2.560	1.276Hz	1.276Kb/s
256B	1432 μ s	1436 μ s	1444 μ s	3 μ s	2.560	696Hz	1.392Kb/s

Fonte – Elaborado pelo Autor (2021).

Com esses resultados, era inevitável o exame do código-fonte da função *writeCharArray()* a fim de entender o que de fato o recurso de verificação de erros executava “por trás dos panos”. Após essa análise, seria possível determinar se a desativação desse artifício seria ou não algo prudente. Um total de 23.040 blocos de dados de diversos tamanhos foram gravados ao longo das dez baterias de testes e em nenhuma das execuções apresentou-se falhas na transmissão de dados.

Ao abrir o código, constatou-se que não havia nenhum algoritmo de *checksum* ou qualquer outro meio complexo de apuração da integridade dos dados transmitidos. Na verdade, o método de verificação de erros da biblioteca SPIMemory era simples, direto e custoso: após gravar os dados, eles eram lidos do dispositivo e, em seguida, comparados. Assim, se o dado lido na memória fosse divergente do que foi enviado à função para ser gravado, então tinha-se um erro de gravação. Portanto, para cada operação de escrita de *n Bytes* estava-se realizando outra de leitura dos mesmos *n Bytes*, desse modo, prejudicando seriamente os resultados de desempenho da W25Q64FV.

Justificado os ganhos significativos provenientes da desativação da verificação de erros nas operações de escrita, o próximo passo era testar um modo disponível na biblioteca chamado de *HIGHSPEED*. Para isso, só era preciso descomentar a diretiva de pré-processamento que o definia no arquivo *SPIMemory.h*. Analisando o código, nota-se que a ativação desse recurso incide na supressão das chamadas a função *_notPrevWritten()*. Nas memórias *Flash*, uma operação de limpeza deve ser realizada antes de se gravar os dados em uma determinada área da unidade de armazenamento. A *_notPrevWritten()* cuidava de efetuar essa confirmação por meio de uma operação leitura no dispositivo. Dessa forma, depois de ler todo o local selecionado e constatando que este estava limpo, então autorizava-se a gravação de dados.

Independentemente se a função *_notPrevWritten()* está ativa ou não no programa, o desenvolvedor precisará efetuar manualmente a devida limpeza do local de memória que será alvo das operações de gravação de dados. A diferença prática é ter ou não um retorno de erro ao tentar escrever em uma área ainda preenchida, porém, pelo custo de uma considerável redução de desempenho. Assim, chegando muito próximo a frequência média dos 10.000Hz, a terceira bateria de testes com o modo *HIGHSPEED* habilitado proporcionou outra expressiva evolução nos resultados, os quais estão contemplados na Tabela 8. Um ganho decorrente da remoção de outra dispendiosa operação de leitura em meio ao processo de transmissão dos dados.

Tabela 8 – Resultado da 3ª Bateria de Testes com a Memória *Flash* W25Q64FV

Tamanho do Bloco	Tempo Mínimo	Tempo Médio	Tempo Máximo	Desvio Padrão	Total de Operações	Frequência Média	Taxa de Transmissão
1B	96 μ s	108 μ s	120 μ s	4 μ s	2.560	9.289Hz	73Kb/s
2B	100 μ s	110 μ s	124 μ s	3 μ s	2.560	9.094Hz	142Kb/s
4B	104 μ s	116 μ s	128 μ s	4 μ s	2.560	8.649Hz	270Kb/s
8B	116 μ s	128 μ s	140 μ s	5 μ s	2.560	7.815Hz	488Kb/s
16B	136 μ s	154 μ s	160 μ s	3 μ s	2.560	6.508Hz	813Kb/s
32B	176 μ s	197 μ s	204 μ s	3 μ s	2.560	5.088Hz	1.272Kb/s
64B	264 μ s	273 μ s	280 μ s	3 μ s	2.560	3.656Hz	1.828Kb/s
128B	428 μ s	441 μ s	448 μ s	5 μ s	2.560	2.266Hz	2.266Kb/s
256B	764 μ s	772 μ s	776 μ s	3 μ s	2.560	1.295Hz	2.590Kb/s

Fonte – Elaborado pelo Autor (2021).

Por fim, a última bateria de testes. Essa foi a única que exigiu uma modificação mais ativa na biblioteca, na qual desabilitou-se as chamadas a função *_notBusy()* antes de cada gravação. A *_notBusy()* checa, mediante uma operação de leitura, o estado de um registrador da memória responsável por informar se o dispositivo está ou não ocupado naquele momento.

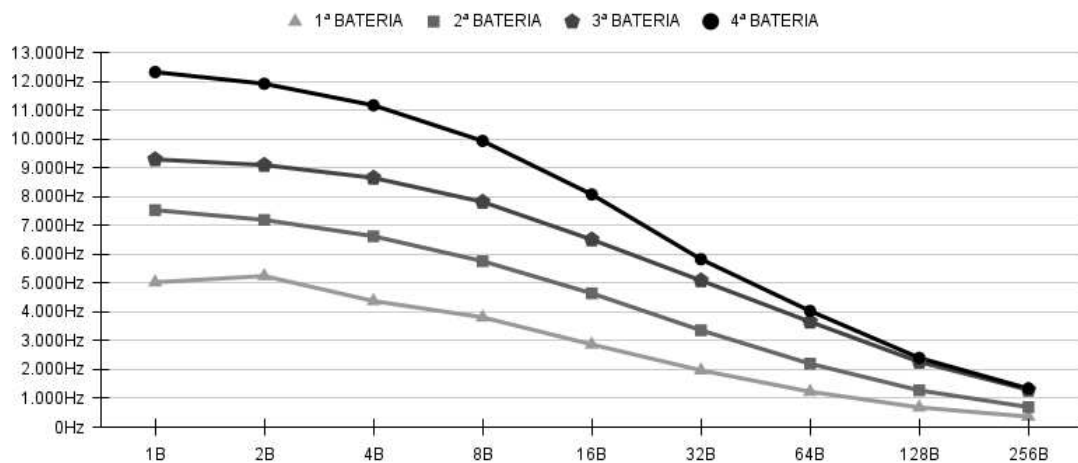
Uma vez que o Arduino possui apenas um microcontrolador procedural de núcleo único e a W25Q64FV está em um sistema diretamente ligado a ele, então presume-se que esta verificação é dispensável neste cenário. Dessa forma, ultrapassamos a marca dos 12.000Hz. Os resultados podem ser conferidos com um maior detalhe na Tabela 9, enquanto que a evolução das otimizações ao longo das baterias está disponível no gráfico da Figura 22.

Tabela 9 – Resultado da 4ª Bateria de Testes com a Memória *Flash* W25Q64FV

Tamanho do Bloco	Tempo Mínimo	Tempo Médio	Tempo Máximo	Desvio Padrão	Total de Operações	Frequência Média	Taxa de Transmissão
1B	72 μ s	81 μ s	96 μ s	3 μ s	2.560	12.319Hz	96 Kb/s
2B	72 μ s	84 μ s	96 μ s	2 μ s	2.560	11.914Hz	186 Kb/s
4B	80 μ s	90 μ s	104 μ s	3 μ s	2.560	11.166Hz	349 Kb/s
8B	88 μ s	101 μ s	116 μ s	3 μ s	2.560	9.930Hz	621 Kb/s
16B	108 μ s	124 μ s	136 μ s	5 μ s	2.560	8.074Hz	1.009 Kb/s
32B	152 μ s	172 μ s	180 μ s	3 μ s	2.560	5.826Hz	1.456 Kb/s
64B	236 μ s	248 μ s	256 μ s	3 μ s	2.560	4.031Hz	2.015 Kb/s
128B	404 μ s	416 μ s	424 μ s	5 μ s	2.560	2.404Hz	2.404 Kb/s
256B	740 μ s	747 μ s	752 μ s	3 μ s	2.560	1.339Hz	2.678 Kb/s

Fonte – Elaborado pelo Autor (2021).

Figura 22 – Gráfico com os Resultados da W25Q64FV em cada Bateria de Testes



Fonte – Elaborado pelo Autor (2021).

Finalmente, realizou-se os testes de integridade na memória *Flash* W25Q64FV. Após uma bateria com mil rodadas de testes, não identificou-se nenhum bloco de dados corrompido. Presume-se que o capacitor de capacitância desconhecida disposto entre os terminais de alimentação do módulo contribui para esse resultado. Assim, conclui-se essa minuciosa etapa de testes sintéticos com os dados que darão alicerce à análise comparativa e determinação da arquitetura de armazenamento não volátil melhor avaliada.

3.3 ANÁLISES COMPARATIVAS

Na etapa anterior foram executadas 58 baterias de testes, totalizando mais de 440 mil operações de escrita. Esses dados possibilitaram mensurar o desempenho de cada dispositivo de memória para variados tamanhos de blocos em cenários ideais. Com base nesses resultados, pode-se construir diversas análises descritivas como também paralelos entre essas arquiteturas de armazenamento. Dessa forma, os usuários da plataforma podem verificar qual das soluções melhor atende aos requisitos de seu projeto.

Um exemplo de análise construída através dos resultados apresentados nos testes sintéticos está compreendido na Tabela 10. Blocos de 8 *Bytes* são transmitidos na frequência máxima de cada arquitetura de armazenamento. Em razão desse cenário extremo, os resultados referentes a duração máxima dos experimentos foram os menores possíveis, para esse comprimento de registro. Dessa forma, esse primeiro ensaio expõe um pouco acerca da influência que a capacidade de armazenamento possui sobre as atividades de aquisição de dados.

Tabela 10 – Transmissão de Blocos de 8B nas Frequências Máximas de cada Dispositivo

	EEPROM I ² C AT24C64A	Flash SPI W25Q64FV	SanDisk microSD 1014204120XF
Capacidade de Armazenamento	8 KB	8 MB	1 GB
Tamanho do Bloco de Dados	8B	8B	8B
Frequência de Aquisição de Dados	434Hz	9.930Hz	59Hz
Quantidade Máxima de Registros	1.024	1.048.576	16.777.216
Taxa de Transmissão dos Dados	27,13Kb/s	620,63Kb/s	3,69Kb/s
Duração Máxima do Experimento	2,35 segundos	1,76 minutos	26,3 dias

Fonte – Elaborado pelo Autor (2021).

Buscando cenários mais verossímeis e menos extremos, isto é, alterando os parâmetros para taxas de amostragem mais moderadas, obtém-se uma maior duração nos ensaios experimentais. Como é notado na Tabela 11, para a EEPROM AT24C64A determinou-se uma frequência de 100Hz, almejada no trabalho de Oliveira et al.(7). Contudo, a capacidade de armazenamento dessa memória limitaria o tempo de voo do *Veículo Aéreo Não Tripulado* (VANT) em apenas dez segundos nesta conjuntura. Enquanto isso, para a *Flash* W25Q64FV foi assumido o valor de 1.000Hz, o qual era desejado em Joris et al.(8). Nesse caso, alcançou-se os dezessete minutos e meio de aquisição de dados, tempo mais que suficiente para contemplar todo o experimento com o foguete de água.

Tabela 11 – Transmissão de Blocos de 8B em Frequências Moderadas de cada Dispositivo

	EEPROM I ² C AT24C64A	Flash SPI W25Q64FV	SanDisk microSD 1014204120XF
Capacidade de Armazenamento	8 KB	8 MB	1 GB
Tamanho do Bloco de Dados	8B	8B	8B
Frequência de Aquisição de Dados	100Hz	1.000Hz	59Hz
Quantidade Máxima de Registros	1.024	1.048.576	16.777.216
Taxa de Transmissão dos Dados	6,25Kb/s	62,50Kb/s	3,69Kb/s
Duração Máxima do Experimento	10,24 segundos	17,5 minutos	26,3 dias

Fonte – Elaborado pelo Autor (2021).

Como visto nas duas tabelas anteriores, os cartões de memória são uma ótima opção para experimentos de baixa frequência em virtude da sua ampla capacidade de armazenamento. Principalmente nos projetos que destinam-se ao monitoramento de eventos de longa duração. Como é o caso da aquisição de dados empregada nos minissatélites abordados no trabalho de Geeroms et al.(13). Os modelos mais básicos de microSD vendidos atualmente já oferecem *gigaBytes* de espaço para a alocação dos registros.

No entanto, em projetos que visam a aquisição de dados em média e alta frequências, as unidades *Flash* da série Winbond conseguem proporcionar uma taxa de amostragem consideravelmente superior e com uma capacidade de armazenamento satisfatória. Dado que o modelo de 8MB, usado neste trabalho, já possibilita ensaios experimentais de mais de um minuto e meio em sua frequência máxima, como também ultrapassa os 30 minutos em 500Hz. Contudo, ainda há outras variantes disponíveis no mercado, como a versão W25Q128FV provendo o dobro do espaço em memória (39).

A EEPROM AT24C64A não dispõe nem de uma capacidade de armazenamento favorável nem de um tempo de escrita atrativo. Apesar da família AT24CXX oferecer outras opções com mais espaço em memória, o maior problema identificado nos testes sintéticos foi o seu ciclo de gravação. Ele produz um *delay* não determinístico (36) de tempo bem significativo, tornando-se o principal fator prejudicial ao desempenho do dispositivo nas transmissões contínuas de blocos de dados. A indicação dessa solução de arquitetura fica limitada aos experimentos mais curtos, de blocos menores ou com baixa taxa de amostragem. Portanto, não atendendo por completo a proposta abordada no escopo deste trabalho.

3.4 SIMULAÇÕES

As etapas anteriores de testes sintéticos foram executadas com o enfoque em extrair a máxima performance de cada dispositivo, portanto submetendo-os a cenários ideais. Seus resultados forneceram informações relevantes sobre o desempenho dessas unidades de memória. Os dados obtidos, por si só, já possibilitam várias análises criteriosas sobre quais condições de projeto certa solução deve ou não ser considerada. Desse modo, esta nova fase de simulações visa avaliar os reais impactos da implementação de cada uma dessas arquiteturas de armazenamento.

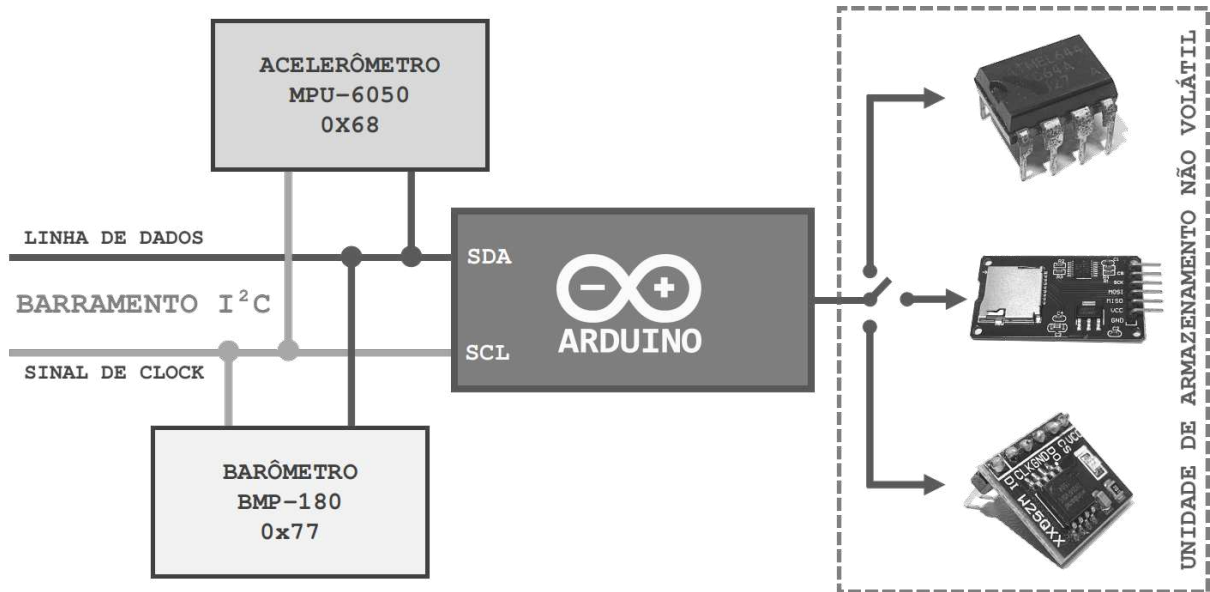
A simulação foi formalizada baseando-se no artigo de Joris et al.(8), no qual um foguete de água é construído e lançado. Os dados de decolagem, voo, bem como as informações de pouso são monitorados por meio de um Arduino Nano, um módulo microSD, um módulo acelerômetro ADXL001 e um módulo altímetro MS5607. Note-se que essa plataforma do Arduino compartilha do mesmo microcontrolador ATmega328 presente na versão UNO deste trabalho, porém noutro encapsulamento mais compacto.

Para simular o experimento, tanto a montagem do foguete quanto o uso estrito dos componentes relacionados são elementos dispensáveis: basta elaborar um cenário verossímil, que exija uma elevada frequência na aquisição de dados, afim de avaliar o desempenho do Arduino frente as diferentes arquiteturas de armazenamento aqui analisadas. Portanto, utilizou-se os módulos disponíveis atualmente no mercado nacional que pudessem fornecer os registros de aceleração e altitude necessários para este ensaio experimental.

Nesse sentido, adquiriu-se o acelerômetro MPU-6050 e o barômetro BMP-180. Eles foram implementados em uma *protoboard* seguindo o esquema formalizado na Figura 23. De fato, os valores lidos ao longo da simulação não irão variar tanto quanto ocorreria em um ensaio real, porém, essa variabilidade dos dados é irrelevante para a finalidade desta fase do trabalho. Pois, neste momento, o foco do experimento está na interação do Arduino com os demais componentes, assim, demonstrando como os dispositivos de memória podem influenciar na taxa de amostragem final da aquisição de dados.

Devido as divergências nas especificações técnicas dos sensores deste trabalho e os de Joris et al.(8), um paralelo entre eles torna-se incabível. Contudo, estas simulações possibilitarão avaliar o quanto que a superioridade da memória W25Q64FV, verificada nos testes sintéticos, refletirá em ganhos reais nos ensaios experimentais diante das arquiteturas concorrentes.

Figura 23 – Esquema formulado para a simulação

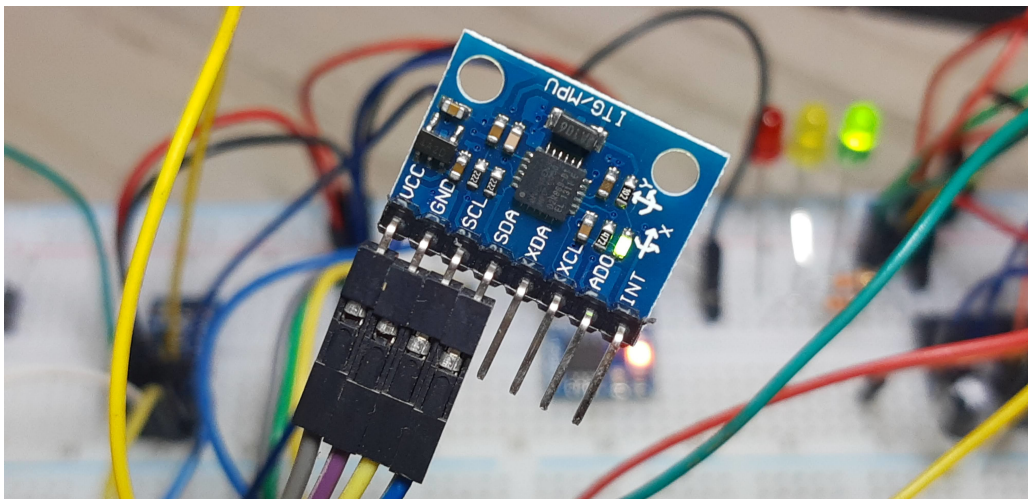


Fonte – Elaborado pelo Autor (2021).

3.4.1 Módulo Acelerômetro MPU-6050

O MPU-6050 foi o primeiro dispositivo a ser associado à simulação formalizada neste trabalho. Além de um sensor de temperatura, esse módulo é constituído por um acelerômetro e um giroscópio, ambos sensores digitais de três eixos. Os registros por eles gerados são armazenados em endereços específicos da memória EEPROM integrada ao componente. A obtenção desses dados é realizada através da interface de comunicação I²C, cujo *clock* de operação admite velocidades de até 400kHz. Por fim, pode-se observar na Figura 24 o dispositivo já conectado ao barramento do Arduino pelos pinos *SCL* e *SDA*.

Figura 24 – Foto do módulo MPU-6050 utilizado nas simulações



Fonte – Elaborado pelo Autor (2021).

Segundo a folha de dados do MPU-6050 (40), seu giroscópio opera numa taxa de amostragem de 8kHz, e seu acelerômetro possui uma frequência de atualização de 1kHz. Visando validar essas informações, bem como verificar as particularidades na implementação desse dispositivo, executou-se uma bateria de testes preliminares que resultou na Tabela 12. Revezando entre *clocks* de 100kHz e 400kHz no barramento I²C, dois cenários foram levantados: no primeiro, lia-se todos os sensores contidos no módulo; enquanto no segundo, considerava-se apenas a leitura da aceleração, assim sendo mais próximo ao trabalho de Joris et al.(8).

Tabela 12 – Resultado dos Testes Preliminares de Leitura no Módulo MPU-6050

Sensor Acel	Sensor Temp	Sensor Giro	Arduino I ² C Clock	Tempo Mínimo	Tempo Médio	Tempo Máximo	Desvio Padrão	Frequência Média
Lido	Lido	Lido	100kHz	1.712μs	1.716μs	1.728μs	3μs	583Hz
Lido	-	-	100kHz	928μs	930μs	936μs	2μs	1.075Hz
Lido	Lido	Lido	400kHz	544μs	551μs	564μs	5μs	1.816Hz
Lido	-	-	400kHz	304μs	311μs	324μs	4μs	3.218Hz

Fonte – Elaborado pelo Autor (2021).

Posto que o objetivo nesses ensaios preliminares era a obtenção dos tempos de leitura do módulo, o armazenamento dos registros foi um ponto ignorado nestas dez rodadas de testes com mil execuções cada. Dessa maneira, o Arduino UNO alcançou frequências de aquisição superiores à taxa de atualização do acelerômetro. A coletar de dados duplicados não é um problema, pelo contrário, denota uma certa margem de desempenho para a associação de outros sensores no laço de monitoramento. Acerca da comunicação com o MPU-6050, exigiu-se apenas o uso da biblioteca nativa *Wire.h* e baseou-se, basicamente, no trecho de código abaixo:

```

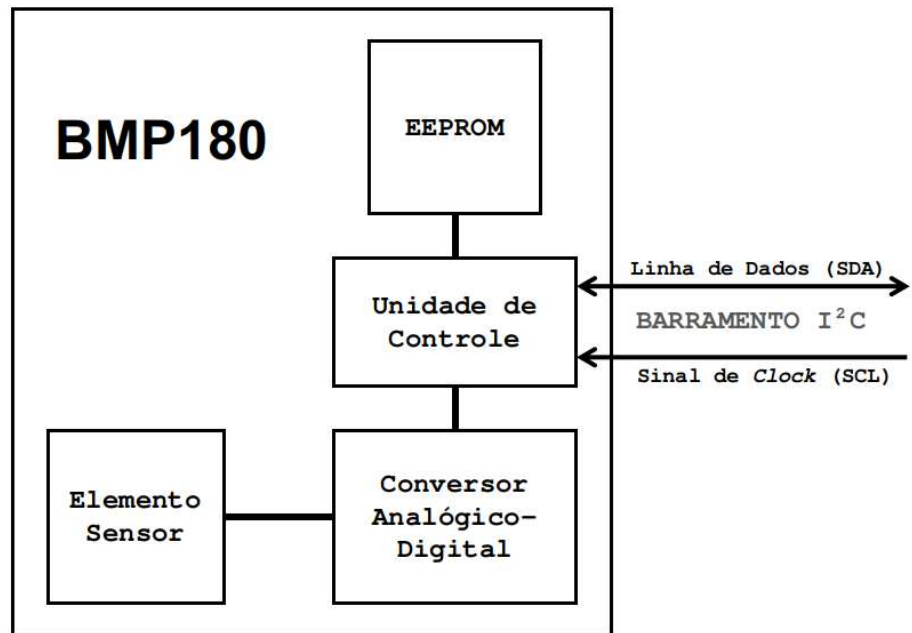
1  t_exec = micros(); // Obtendo o Tempo Inicial
2
3  // Inicializando a Leitura do Módulo MPU-6050
4  Wire.beginTransmission(MPU_I2C_ADDRESS);
5  Wire.write(0x3B); // Registrador ACCEL_XOUT_H
6  Wire.endTransmission(false);
7
8  Wire.requestFrom(MPU_I2C_ADDRESS, 14, true); // Solicitando 14 Bytes do Módulo
9
10 //Armazenando o valor dos sensores nas variáveis correspondentes:
11 AcX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
12 AcY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
13 AcZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
14 Tmp = Wire.read() << 8 | Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
15 GyX = Wire.read() << 8 | Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
16 GyY = Wire.read() << 8 | Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
17 GyZ = Wire.read() << 8 | Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
18
19 t_exec = micros() - t_exec; // Calculando e Armazenando o Tempo de Execução

```

3.4.2 Módulo Barômetro BMP-180

O segundo componente adicionado nesta simulação foi o barômetro digital BMP-180. Esse dispositivo compõe a nova geração sucessora e compatível do BMP-085, razão pela qual suas bibliotecas são mutuamente compartilháveis. De acordo com a sua ficha técnica (41), ele é constituído de um sensor piezoelétrico resistivo, um conversor analógico-digital e um microcontrolador com memória EEPROM, conforme é mostrado no diagrama da Figura 25. Como o módulo MPU-6050, ele também possui um sensor de temperatura incorporado, contudo, irrelevante ao experimento. Por fim, a leitura dos dados é viabilizada pela comunicação I²C admitindo-se *clocks* de até 3.4MHz no barramento.

Figura 25 – Diagrama de Blocos da Arquitetura do BMP-180



Fonte – Adaptado de Bosch Sensortec(41).

Neste barômetro, a altitude é obtida por meio de operações aritméticas considerando a pressão atmosférica lida através do módulo. A fórmula reproduzida logo abaixo foi retirada da ficha técnica do BMP-180 (41), onde p é a pressão medida e p_0 é a pressão ao nível do mar.

$$altitude = 44330 * \left(1 - \left(\frac{p}{p_0} \right)^{\frac{1}{5.255}} \right)$$

A biblioteca de código aberto *Adafruit_BMP085* pode ser obtida diretamente no Arduino IDE ou pelo repositório da Adafruit no Github. Ela simplifica a inicialização do BMP-180 bem como a leitura de seus dados. Uma vez que a sua implementação disponibiliza constantes para os endereços dos registradores, procedimentos para a calibração dos sensores e métodos para o tratamento de dados brutos. Um exemplo disso é a função *readAltitude()*: ela lê o módulo, calcula a altitude usando a fórmula apresentada anteriormente e, por fim, já retorna o seu valor em metros.

Embora poder registrar a altitude em tempo de execução seja uma prática conveniente no ponto de vista do usuário, uma estratégia mais eficiente, em termos de taxa de amostragem, é o monitoramento dos dados brutos. Nesta abordagem, durante os ensaios o Arduino dedica-se unicamente à tarefa de armazenar, de modo cru, os valores lidos do sensor com a finalidade de tratá-los somente após a conclusão do experimento. Assim, os cálculos de ponto flutuante, entre outros procedimentos subsidiários, são removidos do laço principal do programa. Desta maneira, poupa-se a carga de processamento do microcontrolador para a elevação da frequência de aquisição dos dados.

Conforme a ficha técnica do BMP-180 (41), ele é capaz de entregar uma taxa de atualização de até 125 medições por segundos em seu modo padrão de execução. Essa quantidade varia em função do modo de operação configurado, sendo que cada qual tem o seu consumo de corrente, a sua taxa de amostragem e o seu nível de acurácia. Então executou-se uma bateria de testes preliminares com o intuito de avaliar o desempenho inicial desse módulo, analisar a biblioteca *Adafruit_BMP085* assim como examinar as especificidades de sua implementação. Os resultados estão contidos na Tabela 13 e contemplam a performance do sensor sob diferentes condições.

Tabela 13 – Resultado dos Testes Preliminares de Leitura no Módulo BMP-180

Descrição do Teste	Arduino I ² C Clock	Tempo Médio	Desvio Padrão	Frequência Média
Leitura usando <i>readAltitude()</i> com <i>Delays</i>	100kHz	15.358 μ s	5 μ s	65Hz
Leitura usando <i>readAltitude()</i>	100kHz	7.351 μ s	5 μ s	136Hz
Leitura usando <i>readAltitude()</i>	400kHz	5.892 μ s	4 μ s	170Hz
Leitura usando <i>readPressure()</i>	400kHz	5.881 μ s	4 μ s	170Hz
Leitura usando <i>readRawPressure()</i>	400kHz	541 μ s	3 μ s	1.849Hz
Leitura Manual de Dados Brutos	400kHz	345 μ s	3 μ s	2.900Hz

Fonte – Elaborado pelo Autor (2021).

Seguindo o praticado no módulo anterior, foram realizadas dez rodadas com mil execuções para cada teste no BMP-180. Na ordem, a primeira bateria de testes consistia em verificar o desempenho do dispositivo quando configurado somente com os parâmetros padrões, tanto pela sua biblioteca quanto pela plataforma Arduino. Desse modo, o resultado obtido neste cenário foi uma frequência de aquisição média de 65Hz, uma taxa de amostragem admissível até para algumas arquiteturas de armazenamento com cartões microSD.

Após a conclusão dessa primeira bateria de testes, verificou-se que neste cenário o módulo BMP-180 não foi capaz de entregar as 125 medições por segundo anunciadas em sua ficha técnica. Diante desse péssimo resultado, analisou-se o código-fonte da biblioteca *Adafruit_BMP085* a fim de encontrar pontos passíveis de otimização nela. Assim, percebeu-se que ela aplicava *delays* correspondentes com o modo de operação selecionado para garantir a leitura de dados atualizados no módulo. Dessa maneira, trava-se o microcontrolador por períodos significativos influenciando no desempenho final do experimento.

Conforme já discutido durante o estudo do módulo MPU-6050, não há contrariedades em se ler dados duplicados nos sensores em prol de se obter taxas de amostragens mais elevadas na aquisição de dados do experimento. Então, alterou-se o código-fonte da biblioteca com o propósito de ignorar esses *delays* implementados por ela. O produto dessa modificação, exibida na Figura 26, foi um ganho expressivo de 109% na frequência média de aquisição dos dados.

Figura 26 – Linhas contendo as funções de *delays* comentadas na biblioteca

```

113
114  uint32_t Adafruit_BMP085::readRawPressure(void) {
115      uint32_t raw;
116
117      write8(BMP085_CONTROL, BMP085_READPRESSURECMD + (oversampling << 6));
118
119      // if (oversampling == BMP085_ULTRALOWPOWER)
120      //     delay(5);
121      // else if (oversampling == BMP085_STANDARD)
122      //     delay(8);
123      // else if (oversampling == BMP085_HIGHRES)
124      //     delay(14);
125      // else
126      //     delay(26);
127
128      raw = read16(BMP085_PRESSUREDATA);
129
130      raw <= 8;
131      raw |= read8(BMP085_PRESSUREDATA + 2);
132      raw >>= (8 - oversampling);
133

```

Fonte – Elaborado pelo Autor (2021).

Na terceira bateria de testes, considerou-se a utilização do *clock* de 400kHz no barramento I²C. Uma configuração simples que promoveu mais um ganho de 25% sobre o último resultado. Já na quarta bateria, substituiu-se a função de leitura de *readAltitude()* para a *readPressure()* visando reduzir o processamento de cálculos. Entretanto, essa modificação refletiu em uma diferença mínima: apenas onze microssegundos a menos que o tempo médio de execução do método empregado anteriormente.

Essa pequena diferença nos tempos médios de execução entre a terceira e a quarta bateria de testes são explanados por meio dos resultados obtidos na quinta. Pois, alcançou-se nela frequências dez vezes superiores as anteriores, devido a substituição da função de leitura *readPressure()* para a *readRawPressure()*. Assim, dando indícios de que os principais e mais custosos cálculos já estavam presentes em *readPressure()*, como constatou-se posteriormente. Desse modo, sobrava muito pouco, em termos de operações aritméticas, para serem resolvidas em *readAltitude()*, justificando há de ganhos significativos.

Por fim, buscando o máximo desempenho do módulo BMP-180, implementou-se a leitura manual dos dados brutos na sexta e última bateria de testes. Uma alternativa à função *readRawPressure()*, porém com uma performance até 57% superior. Note-se que a biblioteca não precisa ser excluída do programa, pois, ela ainda fornece diversos recursos que podem ser aproveitados, como os métodos de calibração bem como as constantes definidas para diversos registradores do sensor. Abaixo, o trecho de código que proporcionou o resultado apresentado:

```

1  t_exec = micros(); // Obtendo o Tempo Inicial
2
3  // LEITURA MANUAL DE DADOS BRUTOS NO BMP-180
4  // Solicitando a Medição da Pressão Atmosférica ao Módulo
5  Wire.beginTransmission(BMP085_I2CADDR);
6  Wire.write(BMP085_CONTROL);
7  Wire.write(BMP085_READPRESSURECMD + (oversampling << 6));
8  Wire.endTransmission(true);
9
10 // Solicitando o Resultado do Sensor
11 Wire.beginTransmission(BMP085_I2CADDR);
12 Wire.write(BMP085_PRESSUREDATA);
13 Wire.endTransmission(true);
14
15 // Lendo e Armazenando corretamente o Resultado de 19 bits
16 Wire.requestFrom(BMP085_I2CADDR, 3);
17 altitude = (((int32_t) Wire.read() << 16) | \
18             (((int32_t) Wire.read() << 8) | \
19             (((int32_t) Wire.read())) >> (8 - oversampling));
20
21 t_exec = micros() - t_exec; // Calculando e Armazenando o Tempo de Execução

```

3.4.3 Resultado das Simulações

Depois de concluir todas as baterias de testes preliminares, tanto no acelerômetro quanto no barômetro, obteve-se a aptidão necessária para se iniciar a construção bem como a execução das simulações. Assim, a partir dos resultados alcançados e da compreensão das particularidades de cada dispositivo, elaborou-se três cenários para os ensaios experimentais, onde: no primeiro, utilizou-se o *clock* de operação do barramento I²C em seu valor padrão de 100kHz; no segundo, apenas ajustou-se esse parâmetro para 400kHz, sendo a frequência máxima desse sistema; e no terceiro, manteve-se essa velocidade na interface, porém foram desprezadas as leituras dos sensores de temperatura e giroscópio do módulo MPU-6050, portanto, aproximando-se mais do trabalho de Joris et al.(8).

Após cinco mil simulações por arquitetura de armazenamento para cada cenário formalizado, obteveram-se os resultados dispostos na Tabela 14. Em todas as baterias, o módulo BMP-180 trabalhou no modo padrão, com leituras manuais dos dados brutos. Respeitando os aspectos definidos nos ensaios, todos os dispositivos de memória foram configurados para entregarem o máximo de desempenho apresentado nos seus respectivos testes sintéticos.

Tabela 14 – Resultado das Simulações com os Dispositivos de Memória

BMP Pressão	MPU Acel	MPU Temp	MPU Giro	Tam. Bloco	Arduino twi_freq	Memória Externa	Frequência Média	Duração Máxima
Lido	Lido	Lido	Lido	22B	100kHz	Cartão SD	52Hz	2617 horas
Lido	Lido	Lido	Lido	22B	100kHz	AT24C64A	135Hz	2,8 segs
Lido	Lido	Lido	Lido	22B	100kHz	W25Q64FV	352Hz	18,1 mins
Lido	Lido	Lido	Lido	22B	400kHz	Cartão SD	57Hz	2379 horas
Lido	Lido	Lido	Lido	22B	400kHz	AT24C64A	264Hz	1,4 segs
Lido	Lido	Lido	Lido	22B	400kHz	W25Q64FV	954Hz	6,7 mins
Lido	Lido	-	-	12B	400kHz	Cartão SD	58Hz	3666 horas
Lido	Lido	-	-	12B	400kHz	AT24C64A	300Hz	2,0 segs
Lido	Lido	-	-	12B	400kHz	W25Q64FV	1.309Hz	7,6 mins

Fonte – Elaborado pelo Autor (2021).

Os resultados dessas simulações possibilitaram diversas análises sobre o desempenho das arquiteturas de armazenamento nos projetos de aquisição de dados. A primeira é em relação aos cartões de memória microSD, houve novamente a constatação da mesma faixa de frequência encontrada na literatura (6) (7) (17), independentemente do cenário imposto a ele. Também destaque-se na tabela uma relevante influência do *clock* da interface I²C sobre a taxa de amostragem final do experimento. Dessa forma, tornando-se um ajuste importante e

extremamente recomendável para essas atividades. Por fim, outro registro que chama atenção é o tempo de duração máxima dos ensaios com a AT24C64A. Eles são inferiores à marca de três segundos em todas as baterias realizadas.

As simulações evidenciaram a esperada superioridade da W25Q64FV em todos os cenários. Em dado momento, a unidade da Winbond proporcionou leituras mais de 1.000Hz acima da frequência média da AT24C64A. Assim, demonstrando a importância, bem como os benefícios, do emprego de memórias com um bom tempo de resposta para as atividades de aquisição de dados.

Essa etapa também serviu para validar os testes sintéticos com as memórias assim como os ensaios preliminares com os sensores. Por meio de seus dados, é possível calcular taxas de amostragem aproximadas para determinado cenário. Para exemplificar isso, calculou-se o valor aproximado da frequência de aquisição da W25Q64FV para a terceira bateria de simulações da seguinte forma:

$$\begin{aligned}
 tempo_total &= W25Q64FV_16B + ACCEL_MPU_400kHz + RAW_BMP_400kHz \\
 tempo_total &= 116\mu s + 311\mu s + 345\mu s = 772\mu s \\
 taxa_amostragem &= \frac{1s}{tempo_total} = \frac{1s}{772\mu s} = \frac{1}{772 * 10^{-6}} = 1.295Hz
 \end{aligned}$$

O cálculo consiste em somar os tempos médios das operação de cada componente. Como não há informações de performance para blocos de 12 *Bytes* nos testes sintéticos com a W25Q64FV, utilizou-se o tamanho mais próximo, neste caso, o de 16B (116 μ s). Depois, acrescentou-se os tempos do acelerômetro (311 μ s) e do barômetro (345 μ s), usando as mesmas condições impostas na terceira bateria de simulações, isto é: barramento I²C trabalhando em 400kHz, lendo a pressão de forma manual no BMP-180 e apenas a aceleração no MPU-6050. Dessa forma, teria-se o valor de 772 μ s, ou seja, uma frequência de aquisição de 1.295Hz.

Também pode-se aplicar uma regressão linear sobre os resultados de tempo médio da W25Q64FV para se obter o tempo dos blocos de 12 *Bytes* (112 μ s), desse modo, encontrando a taxa de amostragem de 1.302Hz. Enfim, ambos os resultados são muito próximos aos 1.309Hz produzidos nos ensaios experimentais, assim, demonstrando a meticulosidade e o rigor técnico empregados nas baterias de testes executadas ao longo deste trabalho.

4 CONSIDERAÇÕES FINAIS

Levando em consideração as análises realizadas ao longo deste trabalho, a unidade *Flash* W25Q64FV apresentou-se como a candidata mais pertinente dentre as arquiteturas aqui selecionadas. Quanto à capacidade, permite mais de 32 mil registros com blocos de 256 *bytes*. Em relação ao tempo de escrita é até 25 vezes mais rápida que a EEPROM I²C AT24C64A e passando de 140x quando comparado ao cartão microSD testado. Sobre a confiabilidade, trouxe resultados impecáveis ao não corromper nenhum *byte* mesmo após mil execuções do teste de integridade.

Nos cenários semelhantes aos encontrados nos artigos de Joris et al.(8) e Oliveira et al.(7), embora com uma menor capacidade de memória, o nosso trabalho contribui diretamente com uma solução mais simples e mais segura para o armazenamento de registros derivados da aquisição de dados em frequências além das suportadas pelos cartões microSD. Em suma, mais simples por não depender de técnicas avançadas de otimização para se alcançar as taxas de amostragem desejadas, assim como mais segura em razão de prover a gravação dos dados imediatamente após a sua leitura nos sensores, ou seja, sem a necessidade de guardá-los temporariamente em alguma espécie de *cache* em memória volátil – passível de perdas.

A comodidade do acesso aos dados é uma vantagem preeminente das unidades de microSD sobre o uso da W25Q64FV. Nesse sentido, os cartões pontuam por serem uma tecnologia presente em nosso cotidiano. Assim, eles são compatíveis com dispositivos de uso comum, como *tablets* e *notebooks*. Essa condição possibilita que, ao fim dos ensaios, o estudante desconecte o microSD do projeto e insira-o diretamente em seu celular. Em seguida, carregue todos os registros recém capturados em uma aplicativo *mobile* gratuito de planilha eletrônica. Dessa forma, em poucos minutos, ele já obtém os valores mínimos, máximos, médios, assim como os desvios padrões de seu experimento.

Enquanto isso, nas memórias *Flash* da série Winbond W25QXX, um processo de extração de dados é necessário e comumente realizado através da comunicação serial: seja copiando os registros manualmente do monitor serial de sua IDE, seja exportando o buffer do terminal diretamente para um arquivo. Para isso, há diversos *softwares open-source* de

comunicação serial que facilitam o processo, como o *Putty* vide Apêndice B. Contudo, em poucas linhas de código em *Python* alcança-se a mesma funcionalidade desses programas, como é demonstrado no script do Apêndice C. A conexão direta com um dispositivo móvel também é possível, porém, exigirá um aplicativo *mobile* e um adaptador de cabos adequados, no Apêndice D mostra-se o monitoramento dos testes sintéticos a partir de um *tablet*.

A memória *Flash* da Winbond atendeu a todos os requisitos estabelecidos no objetivo deste trabalho. Ela é uma unidade de armazenamento não volátil, integrável ao Arduino e de baixo custo que satisfaz os critérios de capacidade, tempo de resposta e confiabilidade. Quanto à taxa de amostragem alcançada, a W25Q64FV ultrapassou a marca dos 12.000Hz nos testes sintéticos. Enquanto que nas simulações, ela evidenciou sua superioridade frente as demais arquiteturas quando alcançou mais de mil *hertz* só na diferença entre sua frequência e a do dispositivo com o segundo melhor resultado. Dessa forma, essa memória demonstrou ser uma solução mais adequada nos experimentos de aquisição de dados, em média e alta frequências, com o Arduino do que os populares cartões microSD.

4.1 TRABALHOS FUTUROS

Como visto ao longo deste trabalho, os cartões microSD são memórias *Flash* que não atingem o desempenho da Winbond W25Q64FV, ainda que ambos tenham de especificações similares em relação aos seus protocolos e *clocks* máximos de operação, conforme discriminado nas suas fichas técnicas (28) (37). Foi verificado que essa limitação não origina-se dos módulos leitores de cartões, pois, o problema persiste mesmo após a ligação direta do microSD ao barramento do microcontrolador. Então, seria interessante investigar as razões pelas quais esse fenômeno ocorre no Arduino. Assim como, como também verificar as possibilidades de otimizar esta comunicação. Esses estudos resultariam em um material extremamente relevante para a comunidade de usuários da plataforma.

REFERÊNCIAS

- 1 SOUZA, A. R. d.; PAIXÃO, A. C.; UZÊDA, D. D.; DIAS, M. A.; DUARTE, S.; AMORIM, H. S. d. A placa Arduino: uma opção de baixo custo para experiências de física assistidas pelo PC. **Revista Brasileira de Ensino de Física**, v. 33, p. 01–05, 03 2011.
- 2 CAVALCANTE, M. A.; TAVOLARO, C. R. C.; MOLISANI, E. Física com Arduino para iniciantes. **Revista Brasileira de Ensino de Física**, v. 33, p. 4503–4503, 12 2011.
- 3 JR., I. V. d. S.; MIRANDA, J. O. S.; NASCIMENTO, A. C. S.; ARAÚJO, F. R. V. Física experimental com Arduino: ondas em uma corda tensionada. **Revista Brasileira de Ensino de Física**, v. 42, 2020.
- 4 CID, A.; CORREA, T. Venturino: análise da variação de pressão em um tubo de Venturi utilizando Arduino e sensor de pressão. **Revista Brasileira de Ensino de Física**, v. 41, 02 2019.
- 5 ALEXANDRE, G. B.; AZEVÊDO, W. V. da S.; VIEIRA, E. F. L.; RODRIGUES, J. H. de A.; BARROS, T. A. A. de; SANTANA, V. B. Projeto e montagem de uma bancada didática híbrida, eólica e solar. In: **VII Congresso Brasileiro de Energia Solar-CBENS 2018**. [S.l.: s.n.], 2020.
- 6 MORENO, C.; GONZÁLEZ, A.; OLAZAGOITIA, J. L.; VINOLAS, J. The acquisition rate and soundness of a low-cost data acquisition system (lc-daq) for high frequency applications. **Sensors**, v. 20, n. 2, 2020. ISSN 1424-8220.
- 7 OLIVEIRA, W. D. S.; LACERDA, A. C. S.; VALE, M. R. B. G. Desenvolvimento de um sistema embarcado para aquisição de dados em um veículo aéreo não tripulado. **Revista Eletrônica de Engenharia Elétrica e Engenharia Mecânica**, v. 1, n. 1, p. 158–172, 2019.
- 8 JORIS, T.; THYSEN, J.-H.; ROEVE, M. D.; GEEROMS, D.; BERTHO, S. Data logging a water rocket with arduino. In: OUWEHAND, L. [S.l.]: European Space Agency, 2013. v. 721, p. 201–205. ISBN 978-92-9092-285-8. ISSN 0379-6566.
- 9 SILVA, A. L.; VARANIS, M.; MERELES, A. G.; OLIVEIRA, C.; BALTHAZAR, J. M. A study of strain and deformation measurement using the Arduino microcontroller and strain gauges devices. **Revista Brasileira de Ensino de Física**, v. 41, 12 2018.
- 10 VIDAL-PARDO, A.; PINDADO, S. Design and development of a 5-channel arduino-based data acquisition system (abdass) for experimental aerodynamics research. **Sensors**, v. 18, n. 7, 2018. ISSN 1424-8220.
- 11 JÚNIOR, J. F. N.; BORGES, V. E. S.; NASCIMENTO, R. M. M. F. Descrição temporal de forças de colisão: um modelo didático para laboratório de física assistido por sistema embarcado. **Revista Brasileira de Ensino de Física**, v. 41, 12 2018.
- 12 BLANCO-CLARACO, J. L.; LÓPEZ-MARTÍNEZ, J.; TORRES-MORENO, J. L.; GIMÉNEZ-FERNÁNDEZ, A. A low-cost modular platform for heterogeneous data acquisition with accurate interchannel synchronization. **Sensors**, v. 15, n. 10, p. 27374–27392, 2015. ISSN 1424-8220.

- 13 GEEROMS, D.; BERTHO, S.; ROEVE, M. D.; LEMPENS, R.; ORDIES, M.; PROOTH, J. ARDUSAT, an Arduino-Based CubeSat Providing Students with the Opportunity to Create their own Satellite Experiment and Collect Real-World Space Data. v. 730, p. 643, set. 2015.
- 14 BRAHMANA, K. The design of data collector to calculate the efficiency of dc to ac inverter based on atmega328 and micro sd as storage media. **Journal of Technomaterials Physics**, v. 2, n. 2, p. 129–138, 2020.
- 15 ARAÚJO, M. C. A. D.; ARAÚJO, F. J. C. Confecção alternativa de pluviômetro basculante utilizando materiais de fácil acesso. **Congresso Técnico Científico da Engenharia e da Agronomia – CONTECC**, 2021. ISSN 2178-8308.
- 16 PAULA, P. M. C. de; SECCO, R. A. M.; MOLENTO, C. F. M.; FILHO, P. L. de P. Construction of a prototype for tracking dogs in the streets using arduino. **Acta Veterinaria Brasilica**, v. 14, n. 2, 2020.
- 17 GONZÁLEZ, A.; OLAZAGOITIA, J. L.; VINOLAS, J. A low-cost data acquisition system for automobile dynamics applications. **Sensors**, v. 18, n. 2, 2018. ISSN 1424-8220.
- 18 CALDERÓN, G.; MUÑOZ, J. H.; RIVERA, J. Y. Dispositivo para medir tiempo y temperatura usando un microcontrolador. **Revista Brasileira de Ensino de Física**, v. 40, 11 2017.
- 19 GRIDLING, G.; BETTINA, W. Introduction to microcontrollers. **Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group**, 2007.
- 20 VARANIS, M.; SILVA, A. L.; BRUNETTO, P. H. A.; GREGOLIN, R. F. Instrumentation for mechanical vibrations analysis in the time domain and frequency domain using the Arduino platform. **Revista Brasileira de Ensino de Física**, v. 38, 04 2016.
- 21 SILVA, R. A. Programando microcontroladores pic: linguagem c. In: _____. [S.l.]: Ensino Profissional, 2006. cap. Comunicação, p. 105.
- 22 LEENS, F. An introduction to i2c and spi protocols. **IEEE Instrumentation Measurement Magazine**, v. 12, n. 1, p. 8–13, 2009.
- 23 PEREIRA, F. Microcontroladores msp430: Teoria e prática. In: _____. [S.l.]: Editora Érica Ltda, 2005.
- 24 SOUZA, A. A. de; CASTRO, H. A.; CARRIJO, R. S. Proposta de uma solução alternativa aplicada em rede de sensores sem fio no padrão ieee 802.15.4. **XIII Conferência de Estudos em Engenharia Elétrica**, 2015. ISSN 2178-8308.
- 25 BARDAJI, R.; SÁNCHEZ, A.-M.; SIMON, C.; WERNAND, M. R.; PIERA, J. Estimating the underwater diffuse attenuation coefficient with a low-cost instrument: The kduino diy buoy. **Sensors**, v. 16, n. 3, 2016. ISSN 1424-8220.
- 26 ASSENDELFT, R. S.; MEERVELD, H. J. I. van. A low-cost, multi-sensor system to monitor temporary stream dynamics in mountainous headwater catchments. **Sensors**, v. 19, n. 21, 2019. ISSN 1424-8220.
- 27 ALI, A. S.; ZANZINGER, Z.; DEBOSE, D.; STEPHENS, B. Open source building science sensors (osbss): A low-cost arduino-based platform for long-term indoor environmental data collection. **Building and Environment**, Elsevier, v. 100, p. 114–126, 2016.

- 28 SanDisk Corporation. **SanDisk Industrial microSD TM Card - Datasheet**. [S.l.], 2015.
- 29 Texas Instruments. **SN74LVC125A Quadruple Bus Buffer Gate With 3-State Outputs**. [S.l.], 2015.
- 30 Atmel Corporation. **ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash - DATASHEET**. [S.l.], 2015.
- 31 YEE, O. C.; KING, L. S. Simulation study on different logic families of not gate transistor level circuits implemented using nano-mosfets. **Journal of Telecommunication, Electronic and Computer Engineering (JTEC)**, v. 8, n. 5, p. 61–67, 2016.
- 32 Atmel Corporation. **Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V - 8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash - DATASHEET**. [S.l.], 2014.
- 33 Atmel Corporation. **ATmega16U4/ATmega32U4 8-bit Microcontroller with 16/32K bytes of ISP Flash and USB Controller - DATASHEET**. [S.l.], 2016.
- 34 Texas Instruments. **xx555 Precision Timers**. [S.l.], 2014.
- 35 Songle Relay. **Songle Relay Relay SRD-05VDC-SL - DATASHEET**. [S.l.], 2015.
- 36 Atmel Corporation. **2-Wire Serial EEPROM - AT24C32A, AT24C64A**. [S.l.], 2008.
- 37 Winbond Electronics Corporation. **3V 64M-BIT Serial Flash Memory with Dual/Quad SPI & QPI**. [S.l.], 2013.
- 38 Atmel Corporation. **2-Wire Serial EEPROM - AT24C32, AT24C64**. [S.l.], 2003.
- 39 Winbond Electronics Corporation. **3V 128M-BIT Serial Flash Memory with Dual/Quad SPI & QPI**. [S.l.], 2012.
- 40 InvenSense Inc. **MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2**. [S.l.], 2013.
- 41 Bosch Sensortec. **BMP180 Digital Pressure Sensor - DATASHEET**. [S.l.], 2013.

APÊNDICES

APÊNDICE A – Código-fonte

```

1  #include <Arduino.h>                // Biblioteca Básica do Arduino
2  #include <EEPROM.h>                // Para Comunicação com a EEPROM Interna
3  #define SERIAL_BAUDRATE      115200 // Taxa de Transmissão de Dados via Serial
4  #define DEBUG                true   // Quando true apresenta os prints de debug
5  #define W25Q64                // Define o uso da Memória Flash W25Q64
6
7  #ifndef AT24C64
8  // Informações da Memória EEPROM ATmega AT24C64A
9  #include "Wire.h"                // Para Comunicação com a I2C com a Memória
10 #define EEPROM_ADDR          0x50  // Endereço do Dispositivo no Barramento I2C
11 #define EEPROM_WRITE_CYCLE    5000  // Tempo Máximo do Ciclo de Gravação (us)
12 #define EEPROM_LEN            8192  // Capacidade de Armazenamento da Memória
13 #define EEPROM_BLOCK          16    // Tamanho Máximo do Bloco de Dados em Bytes
14 #define EEPROM_BUF_MAX        32    // Tamanho Máximo do Buffer
15 #define END_OF_DATA           ((char)0x00) // Byte utilizado para Limpar a Memória
16 #endif
17
18 #ifndef SDCARD
19 // Informações do Cartão de Memória microSD
20 #include "SD.h"                // Para Comunicação com o Cartão SD
21 #define SD_CS                  10    // Pino CS do Módulo Leitor de Cartões SD
22 #define SD_LEN                 8388608 // Tam. Máximo para o Arquivo (8MB em Bytes)
23 #define SD_BUF_MAX             512   // Tamanho Máximo do Buffer
24 #define FILENAME               "data.txt" // Nome do Arquivo de Teste
25 File file;
26 #endif
27
28 #ifndef W25Q64
29 // Informações da Memória Flash Winbond W25Q64FV
30 #include "SPIMemory.h"          // Para Comunicação com a SPI com a Memória
31 #define FLASH_LEN              8388608 // Tamanho da Memória (64Mib em Bytes)
32 #define FLASH_BUF_MAX          256    // Tamanho Máximo do Buffer
33 #define END_OF_DATA            ((char)0xFF) // Byte utilizado para Limpar a Memória
34 SPIFlash flash;                // Criando um objeto da Classe SPIFlash
35 #endif
36
37 #ifndef END_OF_DATA
38 #define END_OF_DATA            ((char)0x00) // Então será definido como caractere nulo
39 #endif
40
41 /* Definindo um tipo para ponteiros de funções usadas no acesso à memória e que
42  * compartilham da mesma estrutura, como os procedimentos de write e read.
43  *
44  * Parâmetros:
45  * - addr: Endereço de Memória
46  * - buffer: Ponteiro para Buffer
47  * - buffer_len: Tamanho do Buffer
48  *
49  * Retorno:
50  *   Inteiro com Código de Erro ou Zero para execuções bem sucedidas.
51  */
52 typedef int8_t (*access_func)(uint32_t addr, char* buffer, size_t buffer_len);

```

```

53
54 // Nos testes, o último estado do programa será registrado na EEPROM do Arduino
55 #define LAST_STATE    0x00 // Endereço da Memória EEPROM onde será armazenado.
56
57 /*
58  * LISTA DE ESTADOS DO PROGRAMA
59  * Exemplo de uso: Ao reiniciar após uma interrupção em estado de WRITING, o pro-
60  * grama será inicializado no modo de CHECKING para verificar os dados gravados na
61  * execução anterior a fim de identificar bytes corrompidos devido à interrupção.
62  */
63 enum STATES {
64     FINISHED = 0,           /* Após a conclusão de todos os processos. */
65     CLEARING,               /* Durante a Limpeza da Memória. */
66     WRITING,                /* Durante a Gravação de Dados na Memória. */
67     CHECKING,               /* Durante a Verificação dos Dados. */
68 };
69
70
71 /*
72  * LISTA DE ERROS
73  * Código de Erro < 0: Erros críticos, levam ao fim da execução do programa.
74  * Código de Erro = 0: Sem erros durante a execução.
75  * Código de Erro > 0: Erros contornáveis, podem ser tratados de alguma forma.
76  */
77 enum ERRORS {
78     MEM_ERROR = -1,         /* Erro na Alocação Dinâmica na Memória. */
79     NO_ERRORS,              /* Sem erros durante a execução. */
80     READ_ERROR,             /* Erro de Leitura na Unidade de Armazenamento. */
81     WRITE_ERROR             /* Erro de Gravação na Unidade de Armazenamento. */
82 };
83
84
85 // LISTA DE LEDs (Sinalização Luminosa do Programa)
86 enum LEDS {
87     LED_RED = 5,            /* Acenderá ao falhar no teste de integridade. */
88     LED_YELLOW,             /* Acenderá durante a gravação de dados. */
89     LED_GREEN               /* Acenderá durante a leitura dos dados. */
90 };
91
92
93 // Macros para identificação de Maior e Menor
94 #define MAIOR(a,b) (((a)>(b))?(a):(b))
95 #define MENOR(a,b) (((a)<(b))?(a):(b))
96
97
98 // Algumas Macros para Manipular os LEDs
99 #define toggle(led)         digitalWrite(led, !digitalRead(led))
100 #define blink(led, ms)      toggle(led), delay(ms), toggle(led), delay(ms)
101 #define blink_red(ms)       blink(LED_RED, ms)
102 #define blink_green(ms)     blink(LED_GREEN, ms)
103 #define blink_yellow(ms)    blink(LED_YELLOW, ms)
104 #define digitalWriteLEDs(val) PORTD = val ? PORTD|B11100000 : PORTD&B00011111
105
106
107 // Macro para "Reiniciar o Programa", mais especificamente, ir p/ índice 0 da memória
108 #define RESTART_PROGRAM asm volatile ("jmp 0");
109
110

```

```

111
112 // BLOCO DE DADOS
113 #define DATA_LEN      1024          // Tamanho do Bloco de Dados
114
115 // Temos abaixo os primeiros 1024 dígitos de Pi.
116 const char data[DATA_LEN]={ '3','1','4','1','5','9','2','6','5','3','5','8','9',
117 '7','9','3','2','3','8','4','6','2','6','4','3','3','8','3','2','7','9','5','0',
118 '2','8','8','4','1','9','7','1','6','9','3','9','9','3','7','5','1','0','5','8',
119 '2','0','9','7','4','9','4','5','9','2','3','0','7','8','1','6','4','0','6','2',
120 '8','6','2','0','8','9','9','8','6','2','8','0','3','4','8','2','5','3','4','2',
121 '1','1','7','0','6','7','9','8','2','1','4','8','0','8','6','5','1','3','2','8',
122 '2','3','0','6','6','4','7','0','9','3','8','4','4','6','0','9','5','5','0','5',
123 '8','2','2','3','1','2','5','3','5','9','4','0','8','1','2','8','4','8','1','1',
124 '1','7','4','5','0','2','8','4','1','0','2','7','0','1','9','3','8','5','2','1',
125 '1','0','5','5','5','9','6','4','4','6','2','2','9','4','8','9','5','4','9','3',
126 '0','3','8','1','9','6','4','4','2','8','8','1','0','9','7','5','6','6','5','9',
127 '3','3','4','6','1','2','8','4','7','5','6','4','8','2','3','3','7','8','6','7',
128 '8','3','1','6','5','2','7','1','2','0','1','9','0','9','1','4','5','6','4','8',
129 '5','6','6','9','2','3','4','6','0','3','4','8','6','1','0','4','5','4','3','2',
130 '6','6','4','8','2','1','3','3','9','3','6','0','7','2','6','0','2','4','9','1',
131 '4','2','7','3','7','2','4','5','8','7','0','0','6','6','0','6','3','1','5','5',
132 '8','8','1','7','4','8','8','1','5','2','0','9','2','0','9','6','2','8','2','9',
133 '2','5','4','0','9','1','7','1','5','3','6','4','3','6','7','8','9','2','5','9',
134 '0','3','6','0','0','1','1','3','3','0','5','3','0','5','4','8','8','2','0','6',
135 '6','5','2','1','3','8','4','1','4','6','9','5','1','9','4','1','5','1','1','6',
136 '0','9','4','3','3','0','5','7','2','7','0','3','6','5','7','5','9','5','9','1',
137 '9','5','3','0','9','2','1','8','6','1','1','7','3','8','1','9','3','2','6','1',
138 '1','7','9','3','1','0','5','1','1','8','5','4','8','0','7','4','4','2','3','7',
139 '9','9','6','2','7','4','9','5','6','7','3','5','1','8','8','5','7','5','2','7',
140 '2','4','8','9','1','2','2','7','9','3','8','1','8','3','0','1','1','9','4','9',
141 '1','2','9','8','3','3','6','7','3','3','6','2','4','4','0','6','5','6','6','4',
142 '3','0','8','6','0','2','1','3','9','4','9','4','6','3','9','2','2','4','7','3',
143 '7','1','9','0','7','0','2','1','7','9','8','6','0','9','4','3','7','0','2','7',
144 '7','0','5','3','9','2','1','7','1','7','6','2','9','3','1','7','6','7','5','2',
145 '3','8','4','6','7','4','8','1','8','4','6','7','6','6','9','4','0','5','1','3',
146 '2','0','0','0','5','6','8','1','2','7','1','4','5','6','3','5','6','0','8','2',
147 '7','7','8','5','7','7','1','3','4','2','7','5','7','7','8','9','6','0','9','1',
148 '7','3','6','3','7','1','7','8','7','2','1','4','6','8','4','4','0','9','0','1',
149 '2','2','4','9','5','3','4','3','0','1','4','6','5','4','9','5','8','5','3','7',
150 '1','0','5','0','7','9','2','2','7','9','6','9','2','5','8','9','2','3','5','4',
151 '2','0','1','9','9','5','6','1','1','2','1','2','9','0','2','1','9','6','0','8',
152 '6','4','0','3','4','4','1','8','1','5','9','8','1','3','6','2','9','7','7','4',
153 '7','7','1','3','0','9','9','6','0','5','1','8','7','0','7','2','1','1','3','4',
154 '9','9','9','9','9','9','8','3','7','9','7','8','0','4','9','9','5','1','0','5',
155 '9','7','3','1','7','3','2','8','1','6','0','9','6','3','1','8','5','9','5','0',
156 '2','4','4','5','9','4','5','5','3','4','6','9','0','8','3','0','2','6','4','2',
157 '5','2','2','3','0','8','2','5','3','3','4','4','6','8','5','0','3','5','2','6',
158 '1','9','3','1','1','8','8','7','1','0','1','0','0','0','3','1','3','7','8','3',
159 '8','7','5','2','8','8','6','5','8','7','5','3','3','2','0','8','3','8','1','4',
160 '2','0','6','1','7','1','7','7','6','6','9','1','4','7','3','0','3','5','9','8',
161 '2','5','3','4','9','0','4','2','8','7','5','5','4','6','8','7','3','1','1','5',
162 '9','5','6','2','8','3','8','8','2','3','5','3','7','8','7','5','9','3','7','5',
163 '1','9','5','7','7','8','1','8','5','7','7','8','0','5','3','2','1','7','1','2',
164 '2','6','8','0','6','6','1','3','0','0','1','9','2','7','8','7','6','6','1','1',
165 '1','9','5','9','0','9','2','1','6','4','2','0','1','9','8','9','3','8','0','9',
166 '5','2','5','2','0','1','0','6','5','4','8','5','8','6','3','2','7','8' };
167
168

```

```

169
170 #ifndef W25Q64
171 // FUNÇÕES DE LEITURA E ESCRITA NA MEMÓRIA FLASH W25Q64
172
173 int8_t w25_read(uint32_t addr, char* buffer, size_t block_len){
174     return !flash.readCharArray(addr, buffer, block_len);
175 }
176
177 int8_t w25_write(uint32_t addr, char* buffer, size_t block_len){
178     return !flash.writeCharArray(addr, buffer, block_len, false);
179 }
180 #endif
181
182 #ifndef AT24C64
183 // FUNÇÕES DE LEITURA, ESCRITA E LIMPEZA DA MEMÓRIA EEPROM AT24C64
184
185 int8_t at24_write(uint32_t addr, char* buffer, size_t len){
186     /* Esta função grava os dados na memória EEPROM externa. */
187
188     char is_busy;           // Flag de Disponibilidade de Escrita na Memória EEPROM.
189     unsigned long int end_write_cycle = 100;    // Timeout do Ciclo de Gravação.
190
191     Wire.beginTransmission(EEPROM_ADDR); // Iniciando a Transmissão de Dados.
192     Wire.write(highByte(addr));           // Enviando o MSB do Endereço de Memória.
193     Wire.write(lowByte(addr));            // Enviando o LSB do Endereço de Memória.
194     Wire.write(buffer, len);              // Enviando os Dados do Buffer.
195     Wire.endTransmission();               // Finalizando Transmissão de Dados.
196
197     // Determinando o tempo máximo de espera do ciclo de gravação
198     end_write_cycle += millis() + EEPROM_WRITE_CYCLE;
199
200     do { // Verificando se a EEPROM já está disponível para Escrita novamente...
201         Wire.beginTransmission(EEPROM_ADDR);
202         is_busy = Wire.endTransmission(); // Diferente de 0 é "EEPROM Ocupada"
203     } while(is_busy && millis() < end_write_cycle);
204
205     // Se após o timeout do Ciclo de Gravação a EEPROM ainda estiver ocupada,
206     // então temos um erro de gravação. Caso contrário, deu tudo certo.
207     return is_busy ? WRITE_ERROR : NO_ERRORS;
208 }
209
210 int8_t at24_read(uint32_t addr, char* buffer, size_t len){
211     /* Esta função ler os dados da memória EEPROM externa. */
212
213     Wire.beginTransmission(EEPROM_ADDR); // Iniciando a Transmissão de Dados.
214     Wire.write(highByte(addr));           // Enviando o MSB do Endereço de Memória.
215     Wire.write(lowByte(addr));            // Enviando o LSB do Endereço de Memória.
216     Wire.endTransmission();               // Finalizando Transmissão de Dados.
217     Wire.requestFrom(EEPROM_ADDR, len);   // Requisitando Dado(s) à memória EEPROM.
218
219     if(Wire.available()) {                // Se houver dado(s) a serem recebido(s)
220         while(Wire.available() && len--)   // Enquanto houver dados E espaço
221             *(buffer++) = (char)Wire.read(); // Grava o dado recebido no buffer
222     } else return READ_ERROR; // Senão, informa erro de leitura na memória EEPROM.
223
224     return NO_ERRORS; // Informa que não houve erros durante a execução da função.
225 }
226

```

```

227
228
229
230 int8_t at24_erase(){
231     /* Esta função preenche com zeros toda a memória EEPROM externa. */
232
233     char *buffer;
234
235     // Alocando Espaço para Armazenar o Buffer de 'EEPROM_BUF_MAX' bytes
236     if (!(buffer=(char*)malloc(sizeof(char)*EEPROM_BUF_MAX)){ // Em caso de erro
237         Serial.println(F("--[!] Falha grave ao tentar Alocar Espaço na Memória."));
238         return MEM_ERROR; // Retornar o código do erro após imprimir a mensagem.
239     }
240
241     memset(buffer, 0, EEPROM_BUF_MAX); // Preenchendo Buffer com Zeros.
242
243     // Gravando Blocos de Dados preenchidos com zeros na Memória (zerofill).
244     for(uint32_t i=0; i<EEPROM_LEN; i+=EEPROM_BUF_MAX)
245         if(at24_write(i, buffer, MENOR(EEPROM_BUF_MAX, EEPROM_LEN-i)))
246             return WRITE_ERROR; // Retorna código em caso de erro.
247
248     free(buffer); // Liberando o espaço na memória.
249
250     // Se não retornar em nenhum dos pontos anteriores, então:
251     return NO_ERRORS; // Informar que não houve erros durante a execução da função
252 }
253
254 #endif
255
256
257
258
259 #ifdef SDCARD
260 // FUNÇÕES DE LEITURA E ESCRITA NO CARTÃO SD
261
262 int8_t sd_write(uint32_t addr, char* buffer, size_t block_len){
263     /*
264      * A file.write não precisa de um endereço (addr), então essa variável é
265      * aproveitada para o armazenamento da quantidade de bytes escritos pela fun-
266      * ção write. Depois ela é retornada em forma de código de erro:
267      *
268      * Se total de bytes escritos for igual a 0, então houve um WRITE_ERROR.
269      * Senão, deu tudo certo = NO_ERROS.
270      */
271
272     addr = file.write(buffer, block_len); // Grava os dados no Arquivo
273     file.flush(); // Força a gravação desses dados
274     return !addr; // Retorno
275 }
276
277 int8_t sd_read(uint32_t addr, char* buffer, size_t block_len){
278     return !file.read(buffer, block_len);
279 }
280
281 #endif
282
283
284

```

```

285
286
287
288
289
290
291
292
293
294
295 int8_t run(access_func write, uint32_t mem_len, size_t block_len){
296 /* Essa função grava dados alocados na memória SRAM para uma memória externa.
297 Para isso, ela recebe os parâmetros: uma função de escrita, o tamanho da memória
298 e o tamanho dos blocos que serão utilizados no processo. Retorna um não-zero em
299 caso de erros. */
300
301     char *buffer; // Ponteiro para Buffer que será dinamicamente alocado
302     uint32_t t_exec; // Variável auxiliar para salvar Tempo de Execução
303
304     // Alocando Espaço na Memória para Armazenar o Buffer de 'block_len' bytes.
305     if (!(buffer = (char *) malloc(sizeof(char) * block_len))){ // Em caso de erro
306         Serial.println(F("--[!] Falha grave ao tentar Alocar Espaço na Memória."));
307         return MEM_ERROR; // Retornar o código do erro após imprimir a mensagem.
308     }
309
310     for(uint32_t i=0; i<mem_len; i+=block_len){ // Laço de Gravação dos Dados.
311         memset(buffer, 0x00, block_len); // Limpando Buffer.
312
313         // Copiando para o buffer a correta subcadeia do vetor de dados.
314         // O i%DATA_LEN é responsável pelo rotacionamento desse vetor.
315         strncpy(buffer, data + i%DATA_LEN, MENOR(block_len, mem_len-i));
316
317         t_exec = micros(); // Armazenando Tempo Inicial.
318
319         // Gravando na memória externa e checando se houve erros durante o processo.
320         if((*write)(i, buffer, MENOR(block_len, mem_len-i))) return WRITE_ERROR;
321
322         t_exec = micros() - t_exec; // Calculando o Tempo de Execução.
323         Serial.print(t_exec), Serial.print(';'); // Imprimindo o Tempo de Execução.
324         // Obs.: Imprimindo o Tempo de Exec. em formato CSV p/ facilitar importação.
325     }
326
327     free(buffer); // Liberando o espaço na memória.
328
329     // Se não retornar em nenhum dos pontos anteriores, então:
330     return NO_ERRORS; // Informar que não houve erros durante a execução da função
331 }
332
333
334
335
336
337
338
339
340
341
342

```

```

343 int8_t check(access_func read, uint32_t mem_len, size_t block_len){
344 /* Esta função verifica os dados gravados na unidade de armazenamento com os da-
345 dos originais alocados na memória SRAM do Arduino. Além de imprimir a quantidade
346 total dos dados corrompidos, a função também apresenta de forma gráfica a dispo-
347 sição deles no monitor serial. */
348
349     char *buffer;           // Ponteiro para Buffer que será dinamicamente alocado
350     uint32_t i;             // Variável para controlar o laço de leitura de blocos
351     uint16_t byte_idx;      // Variável para controlar o laço de leitura de bytes
352     uint32_t bad_bytes = 0; // Variável auxiliar para contar os bytes corrompidos
353
354     // Alocando Espaço na Memória para Armazenar o Buffer de 'block_len' bytes.
355     if (!(buffer = (char *) malloc(sizeof(char) * block_len))){ // Em caso de erro
356         Serial.println(F("--[!] Falha grave ao tentar Alocar Espaço na Memória."));
357         return MEM_ERROR; // Retornar o código do erro após imprimir a mensagem.
358     }
359
360     for(i=0; i<mem_len; i+=block_len){ // Laço de Leitura.
361         memset(buffer, 0, block_len); // Limpando Buffer.
362
363         // Lendo 'block_len' bytes e checando se houve erros durante o processo.
364         if((*read)(i, buffer, MENOR(block_len, mem_len-i))) return READ_ERROR;
365
366         for(byte_idx=0; byte_idx<block_len; byte_idx++){ // Checando Bytes do Bloco.
367             // Verificar antes se o byte lido não é um sinalizador de END_OF_DATA
368             if(buffer[byte_idx] == END_OF_DATA) break; // Se sim, então sair do laço.
369
370             // Aparecendo algum byte divergente, contabilizar esse byte corrompido.
371             if(buffer[byte_idx] != data[(i+byte_idx) % DATA_LEN]) bad_bytes++;
372         }
373
374         // Checando se estamos no fim dos registros.
375         if(buffer[byte_idx] == END_OF_DATA) break; // Se sim, então sair do laço.
376         else byte_idx=0; // Senão, zerar o byte_idx - para cálculo de posicionamento
377     }
378
379     #if DEBUG == false
380     // Apresentação do Resultado no Monitor Serial.
381     Serial.print(F("\n\n--[i] Resultado: "));
382     Serial.println(bad_bytes ? F("[FALHOU]") : F("[OK]"));
383     Serial.print(F("--[i] Bytes Verificados: "));
384     Serial.print(i+byte_idx), Serial.println(F("B"));
385     if(bad_bytes){ // Se houver dados corrompidos...
386         digitalWrite(LED_RED, HIGH); // Acender LED vermelho.
387         Serial.print(F("--[i] Bytes Corrompidos: ")); // Mostrar a contagem.
388         Serial.print(bad_bytes), Serial.println(F("B"));
389     }
390     #else
391     // Obs.: Imprimindo dados em formato CSV para facilitar importação no Excel.
392     Serial.print(bad_bytes), Serial.print(F("/"));
393     Serial.print(i+byte_idx), Serial.print(F(";"));
394     #endif
395
396     free(buffer); // Liberando o espaço na memória.
397
398     // Se não retornar em nenhum dos pontos anteriores, então:
399     return NO_ERRORS; // Informar que não houve erros durante a execução da função
400 }

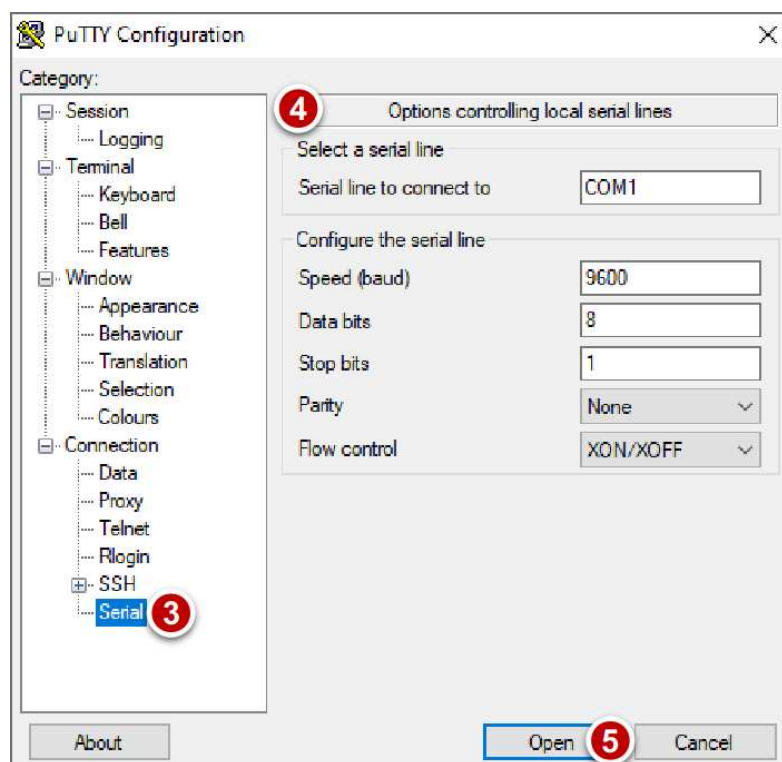
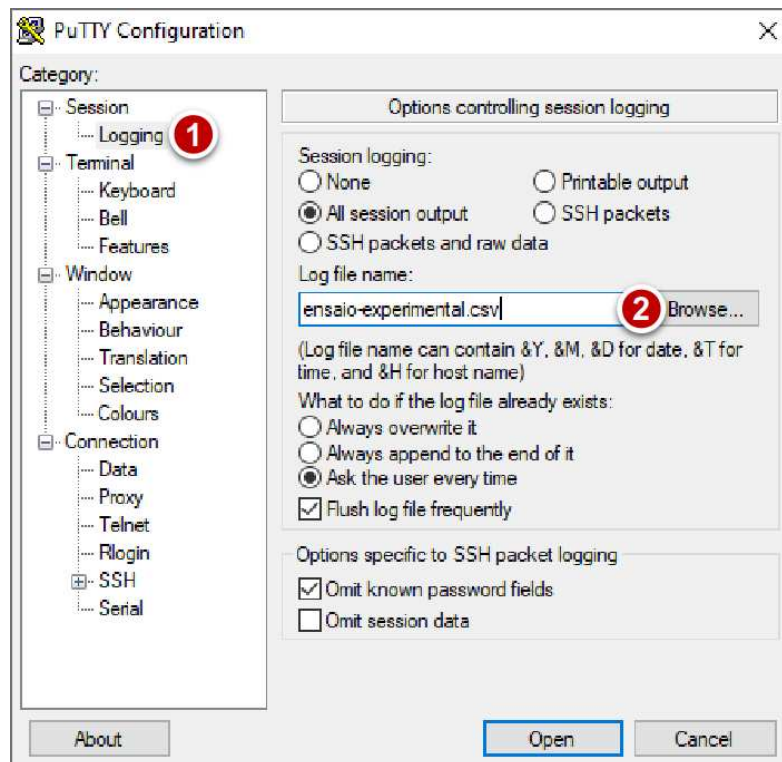
```

```

401 void setup() {
402     /* Função Principal do Programa. */
403
404     // Preparativos antes dos testes...
405     DDRD  |= B11100000;           // Configurando os Pinos dos LEDs para OUTPUT.
406     PORTD &= B00011111;           // Inicializando esses Pinos com LOW.
407     Serial.begin(SERIAL_BAUDRATE); // Iniciando a Comunicação Serial.
408     flash.begin(FLASH_LEN);        // Iniciando a Comunicação com a Flash SPI.
409     delay(1000);                   // Delay de 1s p/ inicialização de componentes
410
411     // Sinalização Luminosa de Inicialização do Programa
412     digitalWrite(LEDs(HIGH);        // Acende todos os LEDs
413     delay(500);                     // Delay de meio segundo
414     digitalWrite(LEDs(LOW);         // Apaga todos os LEDs novamente
415     delay(500);                     // Delay de meio segundo
416
417     // Verificando se o programa finalizou corretamente na sua última execução.
418     if(EEPROM.read(LAST_STATE) == FINISHED){
419         EEPROM.write(LAST_STATE, CLEARING); // Alterando o Estado para CLEARING
420         if(!flash.eraseChip()) return;       // Limpando unidade de armazenamento
421
422         EEPROM.write(LAST_STATE, WRITING);    // Alterando o Estado para WRITING
423         digitalWrite(LED_YELLOW, HIGH);       // Sinalização Luminosa do Estado
424         if(run(w25_write, FLASH_LEN, 32)) return; // Executando o Teste de Desempenho
425         digitalWrite(LED_YELLOW, LOW);        // Sinalização Luminosa fim do Estado
426     } // Senão, ele foi interrompido e precisa verificar a integridade dos dados
427
428     // Checando os dados gravados na memória
429     digitalWrite(LED_GREEN, HIGH);           // Sinalização Luminosa do Estado
430     EEPROM.write(LAST_STATE, CHECKING);       // Alterando o Estado para CHECKING
431     if(check(w25_read, FLASH_LEN, 32)) return; // Executando o Teste de Integridade
432     digitalWrite(LED_GREEN, LOW);             // Sinalização Luminosa fim do Estado
433     EEPROM.write(LAST_STATE, FINISHED);       // Alterando o Estado para FINISHED
434
435     if(digitalRead(LED_RED)){ // Avisar ao usuário sobre falha no teste de integridade
436         Serial.println(F("\n\n--[!] Falha no Teste de Integridade dos Dados."));
437         while(!Serial.available()); // Aguarda interação do usuário pelo monitor serial
438     } else {
439         #if DEBUG == true
440             // Aguarda interação do usuário pelo monitor serial
441             Serial.print(F("\n\n--[i] Envie um caractere para reexecutar teste."));
442             while(!Serial.available());
443         #else
444             // Aguardando 5s para reiniciar o programa automaticamente
445             for(uint8_t i=0; i<5; i++) blink_green(1000); // Macro p/ piscar LED verde
446         #endif
447     }
448
449     asm volatile ("jmp 0"); // Reinicia o Programa
450 }
451
452 void loop() {
453     /* Avisar ao usuário da ocorrência de uma falha grave na execução do programa. */
454     Serial.print(F("--[!] Falha grave durante a execução do programa."));
455     for(uint8_t i=0; i<5; i++) blink_red(250); // Macro para piscar LED Vermelho
456     asm volatile ("jmp 0"); // Reinicia o Programa
457     while(true);
458 }

```

APÊNDICE B – Extração de Dados via Putty



APÊNDICE C – Extração de Dados via Python

```

1  import sys
2  import time
3
4  import serial # Para instalar: pip install pyserial
5  from serial.serialutil import SerialException
6
7
8  def monitor_serial(PORTA:str='/dev/ttyACM0',BAUDRATE:int=115200,ARQUIVO:str=None):
9      #
10     # Bloco de Espera de Conexão
11     #
12     print('[i] Porta de Comunicação Serial:', PORTA)
13     print('[i] Velocidade de Transmissão  :', BAUDRATE, '\n')
14     print('[.] Aguardando Conexão...')
15
16     # Loop de Espera de Conexão
17     while True:
18         try:
19             # Tentando Conectar na Porta Serial
20             arduino = serial.Serial(PORTA, BAUDRATE)
21         except SerialException:
22             # Em caso de Falha... Aguardar 1s e tentar novamente.
23             time.sleep(1)
24         else:
25             # Dispositivo Conectado!
26             print('[!] Conexão Estabelecida.')
27             break
28
29     #
30     # Bloco de Reinicialização Automática do Arduino
31     #
32     print('[!] Reiniciando Dispositivo.')
33     arduino.dtr = False
34     time.sleep(1)
35     arduino.reset_input_buffer()
36     arduino.dtr = True
37     print('[i] Dispositivo Pronto.\n')
38
39     #
40     # Bloco de Coleta de Dados
41     #
42     DADOS = ''
43     print('[i] Coletando Dados...\n')
44     while arduino.is_open:
45         try:
46             # Fazendo a Leitura dos Dados da Porta Serial (Linha por Linha).
47             buffer = str(arduino.readline().decode())
48
49         except SerialException:
50             # Caso haja alguma Falha durante a Leitura dos Dados:
51             print('<!> Falha na Conexão!')
52             arduino.close()

```

```

53         else:
54             # Tratamento de Dados
55             # Se o Buffer não está vazio e nele há mais do que quebras de linhas,
              então:
56             if buffer and len(buffer.replace('\r\n', '')):
57                 # Guardando conteúdo do Buffer sem o \r da função Serial.println()
                    do Arduino.
58                 DADOS += buffer.replace('\r', '')
59
60                 # Imprimir Dados na Tela caso não haja URL do Arquivo de Gravação
61                 not ARQUIVO and print('[>]', buffer, end='')
62             else:
63                 arduino.close()
64
65         #
66         # Bloco de Gravação dos Dados
67         #
68         if ARQUIVO:
69             try:
70                 print(f'[.] Gravando Arquivo de Dados: {ARQUIVO}')
71                 with open(ARQUIVO, 'w') as arquivo:
72                     arquivo.writelines(DADOS)
73             except IOError:
74                 print('<!> Falha na Gravação do Arquivo de Dados.')
75             else:
76                 print(f'[!] Gravação Concluída.')
77
78
79         # Para demonstração do uso...
80         if __name__ == "__main__":
81             monitor_serial(ARQUIVO=sys.argv[1] if len(sys.argv) > 1 else None)

```

APÊNDICE D – Extração de Dados via Dispositivos Móveis

