

Sistema de Instrumentação e Controle com Interface Gráfica em Python baseado no TM4C123GH6PM

Renan Castro de Barros¹, Eliane Silva Custódio², Maria Aline Gonçalves²

¹Estudante de Engenharia Da Computação, Centro Universitário Internacional
UNINTER, Belém, PA, Brasil (renan081239@gmail.com)

²Docente do Centro Universitário Internacional UNINTER, Curitiba, PR, Brasil

Resumo: Este trabalho apresenta um sistema de controle e monitoramento baseado no microcontrolador TM4C123GH6PM, com interface feita em Python. O firmware gera sinais PWM (1Hz-10kHz, razão cíclica 0-100%), adquire quatro canais analógicos e comunica via UART. A interface (PyQt6) exibe as tensões adquiridas, controla os parâmetros de operação e exporta o histórico de medições em formato CSV. O ADC revelou erro máximo de 50 mV e o PWM operou com erro de razão cíclica inferior a 1,5% (1 Hz a 10 kHz).

Palavras-chave: Sistema embarcado; PWM; Aquisição de dados; Interface Gráfica.

INTRODUÇÃO

O crescente desenvolvimento de sistemas eletrônicos embarcados tem impulsionado a criação de soluções cada vez mais integradas para o controle e monitoramento de processos elétricos e eletrônicos. Os microcontroladores representam um avanço significativo na miniaturização de sistemas computacionais, reunindo em um único circuito integrado unidade de processamento, memórias, conversores analógico-digitais e portas de comunicação de entrada e saída. Diferentemente dos microprocessadores, que dependem de componentes externos para funcionar, os microcontroladores são concebidos para operar de forma autônoma na execução de tarefas específicas, o que os torna especialmente adequados para aplicações embarcadas de controle e aquisição de dados (Da Silva et al., 2019). Essa característica, aliada à crescente capacidade de processamento e à diversidade de periféricos integrados disponíveis nas famílias modernas, como o TM4C123GH6PM da Texas Instruments, baseado na arquitetura ARM Cortex-M4F, viabiliza a implementação de sistemas de instrumentação compactos, precisos e de baixo custo.

A modulação por largura de pulso (PWM) é uma técnica amplamente empregada em sistemas de controle eletrônico, na qual um sinal digital de frequência fixa tem sua largura de pulso variada proporcionalmente a um sinal de referência, permitindo o controle preciso da energia entregue a uma carga. Entre suas principais vantagens frente a técnicas analógicas de controle estão a baixa dissipação de potência, a imunidade a variações de

temperatura e envelhecimento dos componentes e a compatibilidade direta com processadores digitais modernos (Yu et al., 1997). No contexto de sistemas embarcados, o PWM é tipicamente gerado por módulos dedicados do microcontrolador, que permitem configurar independentemente a frequência de chaveamento e a razão cíclica, parâmetros que determinam, respectivamente, a velocidade de comutação e o valor médio da tensão de saída. Essa flexibilidade de configuração em tempo de execução torna o PWM gerado por firmware especialmente adequado para aplicações de monitoramento e controle interativo.

A comunicação entre o microcontrolador e o PC é realizada por meio do protocolo UART (Universal Asynchronous Receiver/Transmitter), um dos protocolos de comunicação serial mais amplamente utilizados em sistemas embarcados. O UART opera de forma assíncrona, a sincronização é garantida pela configuração idêntica do baud rate em ambas as extremidades da comunicação, que define a taxa máxima de transferência de bits por segundo. Os dados são transmitidos em pacotes seriais compostos por um bit de início, os bits de dados e um bit de parada, permitindo que o receptor identifique com precisão o início e o fim de cada byte transmitido (Peña; Legaspi, 2020). Essas características tornam o UART especialmente adequado para a comunicação bidirecional entre o firmware embarcado e a interface gráfica em PC, contexto no qual simplicidade, baixo overhead e compatibilidade nativa com microcontroladores são requisitos fundamentais.



No desenvolvimento da interface gráfica, a linguagem Python se destaca como uma plataforma versátil e acessível para a construção de aplicações de monitoramento e visualização de dados. Por meio da biblioteca PyQt6, é possível desenvolver interfaces gráficas completas, desde aplicações simples até sistemas de médio e grande porte utilizados por cientistas e engenheiros em contextos profissionais (Summerfield, 2007). A combinação do Python com bibliotecas especializadas como pyqtgraph, voltada para a renderização de gráficos em tempo real com alta performance, e PySerial, para comunicação com portas seriais, resulta em um ambiente de desenvolvimento capaz de substituir ferramentas proprietárias como o LabVIEW em aplicações de instrumentação.

Trabalhos recentes demonstram a crescente aplicabilidade dos sistemas embarcados em contextos que exigem monitoramento contínuo e confiável. Queiroz et al. (2024) realizaram uma revisão sistemática da literatura evidenciando que tecnologias embarcadas, combinadas com RFID e IoT, mostraram-se eficazes em aplicações de monitoramento remoto de variáveis físicas, controle de acesso e aquisição de dados em tempo real, inclusive em cenários críticos como o da pandemia de COVID-19. Essa versatilidade reforça o potencial dos sistemas embarcados como plataformas de instrumentação de baixo custo e alta integração, motivando o desenvolvimento de soluções cada vez mais completas, como a proposta neste trabalho, que combina aquisição analógica multicanal, geração de PWM e interface gráfica em tempo real em uma única plataforma.

Diante desse contexto, este trabalho apresenta o desenvolvimento de um sistema integrado de controle e monitoramento baseado no microcontrolador TM4C123GH6PM da Texas Instruments, com interface gráfica em tempo real desenvolvida em Python. O sistema é capaz de gerar sinais PWM com frequência e razão cíclica ajustáveis, adquirir sinais analógicos em quatro canais simultâneos e transmiti-los via UART para uma interface gráfica que exibe as variáveis monitoradas em tempo real, permite o controle interativo dos parâmetros de operação e exporta o histórico de medições para análise posterior.

Os principais objetivos do trabalho são: implementar um firmware em linguagem C capaz de gerar um sinal PWM ajustável e realizar aquisição analógica multicanal com transmissão serial dos dados, desenvolver uma interface gráfica em Python com visualização em tempo real, controle interativo via

UART e exportação de dados em CSV e validar o funcionamento integrado do sistema, demonstrando sua aplicabilidade como plataforma de monitoramento.

MATERIAL E MÉTODOS

O sistema é composto por duas camadas funcionais: o firmware responsável pela geração do sinal PWM, aquisição dos sinais analógicos e comunicação serial, implementado no microcontrolador TM4C123GH6PM e a interface gráfica em Python, responsável pela visualização em tempo real e pelo envio dos comandos de controle.

A. Firmware

O firmware foi desenvolvido em linguagem C utilizando a biblioteca TivaWare, com o microcontrolador TM4C123GH6PM operando a 80MHz.

O microcontrolador foi escolhido por reunir, em um único dispositivo, os periféricos necessários para a aplicação desenvolvida. Baseado na arquitetura ARM Cortex-M4F com unidade de ponto flutuante, opera a até 80 MHz, e possui conversor analógico-digital de 12 bits com sequenciador multicanal e interface UART com suporte a interrupções, características que viabilizam a geração de sinais de chaveamento precisos e a aquisição analógica simultânea em múltiplos canais sem necessidade de hardware externo adicional. Em comparação com plataformas como o Arduino Uno, baseado no ATmega328P, o TM4C123 oferece clock significativamente superior (80 MHz contra 16 MHz), ADC de maior resolução (12 bits contra 10 bits) e módulo PWM independente do timer, o que evita conflitos de recursos em aplicações que exigem aquisição e geração de sinal simultâneas. As configurações de cada componente são descritas a seguir.

O módulo PWM0 gera o sinal de chaveamento na saída do pino PB6, no modo de contagem decrescente. O período do contador N é calculado conforme a Equação 1, onde f_{clk} é o clock do sistema e f_s é a frequência de chaveamento desejada:

$$N = f_{clk} / f_s \quad (1)$$

A largura de pulso é definida como $D \cdot N$, onde D é a razão cíclica. Ambos os parâmetros são reconfiguráveis em tempo de execução via UART, sem interromper a geração do sinal. A faixa configurável é de 1 Hz a 10 kHz para a frequência e de 0% a 100% para a razão cíclica, com resolução de 1%.

O módulo ADC0 do microcontrolador foi configurado com um sequenciador de quatro canais (CH0 - CH3), mapeados nos pinos PE0 – PE3, com acionamento por trigger de software no laço



principal. Ao final de cada conversão, os valores brutos de 12 bits são convertidos para tensão pela Equação 2:

$$V = ADC \cdot 3,30/4095 \quad (2)$$

onde ADC corresponde ao valor bruto de 12 bits retornado pelo sequenciador e 3,30 V é a tensão de referência nominal do ADC, derivada do pino de tensão de alimentação analógica do microcontrolador.

O intervalo entre cada ciclo de aquisição é configurável via UART e implementado por delay de software proporcional ao clock do sistema. Após cada ciclo, os cinco valores de tensão convertidos são transmitidos pela UART.

Para estimular os quatro canais analógicos durante os ensaios, foram utilizados potenciômetros de 10 kΩ conectados entre o barramento de 3,3 V e o GND do microcontrolador. O cursor de cada potenciômetro foi ligado diretamente ao pino de entrada analógica correspondente (PE0–PE3), formando um divisor de tensão ajustável manualmente. Dessa forma, a rotação do eixo de cada potenciômetro permite variar continuamente a tensão aplicada ao canal entre 0 V e 3,3 V, cobrindo a faixa completa de entrada do ADC. A escolha dos potenciômetros como fonte de sinal de teste se fundamenta em três aspectos. Primeiro, o comportamento elétrico do divisor resistivo é análogo ao de sensores resistivos amplamente utilizados em instrumentação, tornando o ensaio representativo de aplicações reais. Segundo, o controle manual da tensão de entrada permite ao operador impor valores arbitrários de forma rápida e repetível, sem necessidade de equipamentos adicionais como fontes de tensão programáveis. Terceiro, a tensão de saída do cursor é naturalmente limitada ao intervalo 0–3,3 V pela própria alimentação do circuito, garantindo compatibilidade direta com o range de entrada do ADC de 12 bits do microcontrolador e eliminando o risco de sobretensão nos pinos analógicos.

A UART0 foi configurada a 115200 baud, com clock de referência de 16 MHz. Ao final de cada ciclo de aquisição, o firmware transmite uma linha ASCII com os cinco valores separados por espaço, no formato: "V0 V1 V2 V3 V4\r\n". Os quatro primeiros campos (V0–V3) correspondem às tensões adquiridas pelos canais analógicos CH0–CH3. O quinto campo (V4) não representa um canal ADC adicional, mas sim o estado instantâneo do sinal PWM no momento da aquisição, gerado por software: transmite 3,33 V quando o sinal se encontra em nível alto e 0,00 V quando em nível baixo, permitindo que a interface gráfica reconstrua a forma de onda de chaveamento. Os comandos recebidos pelo PC são processados de

forma assíncrona na ISR da UART, conforme descrito na Tabela 1.

Tabela 1. Comandos do protocolo serial

Comando	Parâmetro	Exemplo
i0<val>	Razão cíclica (0-100)	i050
i1<val>	Frequência em Hz (1-10000)	i11000
i2<val>	Intervalo em segundos	i22

B. Interface Gráfica

A interface foi desenvolvida em Python 3 com as bibliotecas PyQt6, pyqtgraph e pandas. A conexão com a porta serial é estabelecida via PySerial, com porta e baud rate configuráveis pelo operador.

A biblioteca PyQt6 foi adotada para a construção da interface por oferecer um conjunto completo de componentes gráficos e o mecanismo de sinais e slots do framework Qt, que permite comunicação segura entre threads sem bloqueio da interface. Para a renderização dos gráficos em tempo real, utilizou-se a biblioteca pyqtgraph, que, por ser implementada sobre OpenGL e C++, apresenta desempenho de atualização significativamente superior ao da biblioteca matplotlib em aplicações com alta taxa de chegada de dados, sendo, portanto, mais adequada para visualização contínua de sinais. A comunicação com a porta serial foi realizada via PySerial, biblioteca consolidada para comunicação serial em Python, e o gerenciamento e exportação dos dados foram implementados com pandas, que oferece estruturas de dados tabulares eficientes e exportação nativa para CSV.

A leitura da porta é realizada em uma thread dedicada, que deposita cada linha recebida junto ao seu timestamp em uma fila thread-safe. Um QTimer consome a fila periodicamente no thread principal da GUI, extraindo os valores de tensão, calculado o tempo relativo ao início da sessão e atualizando os buffers de dados. Essa separação entre thread de leitura e thread da GUI garante que a interface permaneça responsiva independentemente da taxa de dados recebidos.

A interface disponibiliza quatro gráficos configuráveis simultaneamente, com eixos X e Y selecionáveis entre os canais A0–A3 e o eixo de tempo. Em cada gráfico, são exibidas a curva de tensão e a curva de corrente estimada do canal selecionado. A corrente exibida em cada canal não é medida diretamente, mas estimada a partir da tensão lida pelo ADC por meio da relação da Equação 3:

$$I = V \cdot 0,008 \quad (3)$$

equivalente a assumir uma resistência de carga de referência de 125 Ω. Essa abordagem dispensa o uso

de shunt resistor ou sensor de corrente dedicado, sendo adequada para fins de monitoramento relativo e visualização de tendências, sem pretensão de medição absoluta de corrente. O número máximo de pontos exibidos nos gráficos é configurável, e a taxa de atualização visual é controlada por um segundo QTimer, independente do timer de leitura. A Figura 1 apresenta a tela da interface.

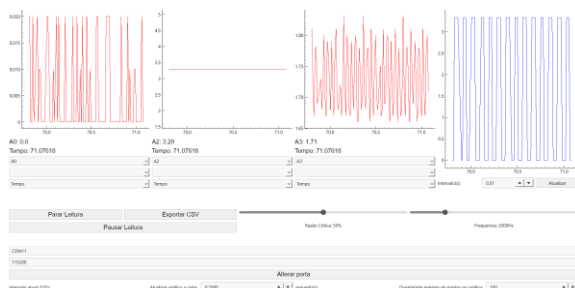


Figura 1. Interface desenvolvida em Python

Dois sliders controlam a razão cíclica (0-100%) e a frequência de chaveamento (1-10000 Hz). Ao ser liberado, cada slider envia o comando correspondente ao firmware via UART. Um terceiro controle ajusta o intervalo de amostragem do firmware. O operador pode exportar o histórico completo de medições para um arquivo CSV com separador ponto-e-vírgula, contendo as colunas de tempo, tensões e correntes estimadas de todos os canais.

A latência de comando foi estimada por meio de um script Python que registrou o instante imediatamente anterior ao envio de cada comando via UART e o instante imediatamente posterior ao retorno de serial.flush(), utilizando time.perf_counter() como base de tempo. Foram enviados quatro comandos distintos de razão cíclica, 25%, 50%, 75% e 100%, e o intervalo medido em cada envio foi registrado. Esse método quantifica o tempo de escrita no buffer serial do host, ao qual se somam analiticamente o tempo de transmissão física serial, calculado a partir do baud rate e do comprimento do comando, e o tempo de execução da ISR da UART no firmware, estimado a partir do clock do sistema.

A validação do desempenho da comunicação serial foi conduzida por meio de um script Python desenvolvido especificamente para esse fim. O script enviou comandos de configuração de intervalo ao firmware via UART, aguardou 500 ms para estabilização do sistema a cada mudança, limpou o buffer serial e contabilizou os pacotes recebidos durante uma janela de 5 segundos por intervalo testado. Cada pacote foi verificado quanto à integridade estrutural, sendo considerado válido apenas quando continha exatamente cinco valores numéricos separados por espaço, conforme o formato de transmissão definido pelo firmware. Pacotes com

número incorreto de campos ou valores não numéricos foram contabilizados separadamente como corrompidos. Os intervalos testados foram 500 ms, 200 ms, 100 ms, 50 ms, 20 ms e 10 ms, cobrindo a faixa de operação típica do sistema.

O ensaio de duração e integridade foi conduzido simultaneamente durante 30 minutos com intervalo de amostragem de 10 ms. Para verificar a integridade dos pacotes, o firmware foi modificado para incluir um contador incremental no início de cada linha transmitida, alterando o formato de V0 V1 V2 V3 V4\r\n para N V0 V1 V2 V3 V4\r\n. A interface foi atualizada para ler o campo N a cada pacote recebido e comparar com o contador anterior, registrando como pacote perdido qualquer salto na sequência. Os primeiros 5 segundos de cada sessão foram descartados da contagem para eliminar pacotes acumulados no buffer serial durante a inicialização. O consumo de memória do processo foi monitorado em paralelo por um script Python utilizando a biblioteca psutil, registrando o uso de memória RAM a cada 30 segundos durante toda a sessão.

RESULTADOS E DISCUSSÃO

Os resultados são apresentados em: validação do sinal PWM gerado pelo firmware, a precisão da aquisição analógica via ADC e o desempenho da interface gráfica em tempo real. Todos os ensaios foram realizados com o microcontrolador TM4C123GH6PM conectado ao PC via UART a 115200 baud.

A geração do sinal PWM foi avaliada para diferentes combinações de frequência e razão cíclica, com os parâmetros enviados pela interface gráfica em tempo real. A Figura 2 apresenta a forma de onda capturada para razão cíclica de 50% e frequência de 2 kHz.

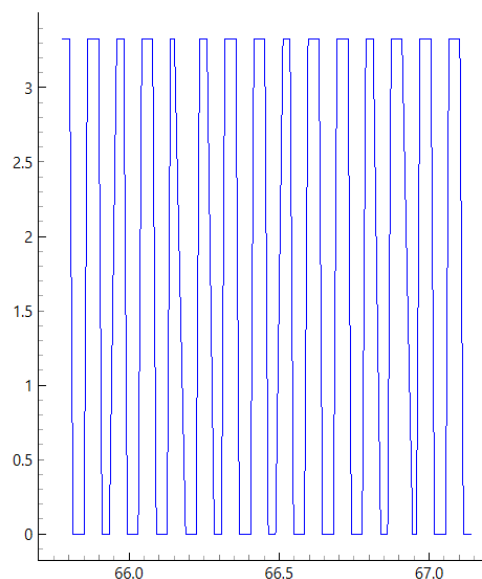


Figura 2. Forma de onda do PWM gerado para D = 50% e $f_s = 2$ kHz

A Tabela 2 apresenta os valores de frequência e razão cíclica configurados pela interface e os valores medidos, evidenciando a precisão do módulo PWM do TM4C123 na faixa de operação testada.

Tabela 2. Comparação entre parâmetros configurados e medidos do PWM

D configurado (%)	D medido (%)
25	24,98
50	49,75
75	76,5
100	100

Os resultados indicam que o módulo PWM0 do TM4C123 operou com erro de razão cíclica inferior a 1,5% em toda a faixa testada. Esse comportamento é esperado dado que o período do contador é calculado diretamente a partir do clock do sistema (80 MHz), sem divisores intermediários variáveis, o que confere alta resolução e estabilidade ao sinal gerado.

A razão cíclica foi medida indiretamente por meio de um multímetro digital, utilizando a função de medição de tensão contínua. Como o sinal PWM alterna entre 0V e 3,3V (tensão de operação dos GPIOs do TM4C123), a tensão média lida pelo multímetro corresponde ao valor de duty cycle multiplicado por 3,3V.

A precisão do ADC foi avaliada aplicando-se tensões de referência conhecidas nos canais analógicos e comparando os valores lidos pela interface com os valores medidos por um multímetro digital. A validação foi conduzida no canal PE0, cujos resultados são representativos dos demais canais pelo fato de todos compartilharem o mesmo circuito de amostragem e a mesma tensão de referência. A Tabela 3 resume os resultados obtidos.

Tabela 3. Comparação entre tensão de referência e tensão lida pelo ADC.

Tensão aplicada (V)	Tensão lida (V)
0,494	0,49
0,972	0,97
1,675	1,68
2,04	2,01
2,85	2,8
3,32	3,29

O erro absoluto máximo observado foi de 50 mV, correspondendo a um erro relativo de 1,52% em relação ao fundo de escala de 3,30V. Esse resultado é coerente com a resolução teórica do ADC de 12 bits, que corresponde a 1 LSB = $3,30/4095 = 0,806$ mV, e

com os erros de offset e ganho típicos do dispositivo reportados em seu datasheet.

Cabe ressaltar que os valores medidos pelo multímetro digital utilizado como referência estão sujeitos a erros intrínsecos do próprio instrumento, como erros de offset, resolução limitada e deriva térmica. Dessa forma, parte do erro absoluto observado entre a tensão aplicada e a tensão lida pelo ADC pode ser atribuída à incerteza de medição do instrumento de referência, e não exclusivamente ao conversor analógico-digital do microcontrolador.

Embora o protocolo UART tenha se mostrado adequado para a aplicação desenvolvida, ele impõe limitações à taxa máxima de aquisição e transmissão de dados. Configurado a 115200 baud, o canal serial comporta aproximadamente 11520 bytes por segundo. Considerando que cada linha transmitida pelo firmware contém cinco valores de tensão em formato ASCII, com comprimento médio de aproximadamente 25 caracteres incluindo separadores e terminadores, a taxa máxima teórica de transmissão é de cerca de 460 linhas por segundo. Na prática, entretanto, o intervalo de amostragem é limitado também pelo tempo de conversão do ADC e pelo delay de software implementado no laço principal, de modo que taxas de aquisição superiores a algumas centenas de amostras por segundo não são alcançáveis sem reestruturação do firmware.

A arquitetura de leitura mostrou-se eficaz: a interface permaneceu responsiva durante todo o período de monitoramento. A Figura 3 apresenta a tela da interface durante uma sessão de monitoramento, exibindo as curvas de tensão e corrente estimada para os canais ativos.

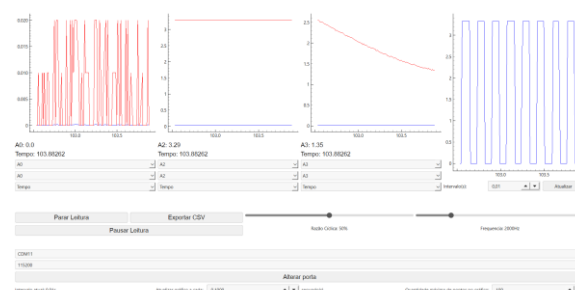


Figura 3. Interface gráfica em operação: curvas de tensão (vermelho) e corrente estimada (azul)

O ajuste de parâmetros pelos sliders mostrou resposta imediata no sinal de saída do microcontrolador. O tempo de escrita no buffer serial apresentou média de 0,323 ms, mínimo de 0,248 ms e máximo de 0,430 ms. Somando-se o tempo de transmissão serial de aproximadamente 0,35 ms e o tempo de execução da ISR da UART, a latência total estimada é inferior a 1 ms.

A taxa de amostragem foi caracterizada para intervalos entre 10 ms e 500 ms, monitorando-se a



integridade dos pacotes recebidos pela interface durante janelas de 5 segundos. Nenhum pacote corrompido foi detectado em nenhum intervalo testado. Para 500 ms, não houve perda. A taxa de perda manteve-se em torno de 10% para intervalos entre 50 ms e 100 ms, elevando-se para 13,6% a 20 ms e 16,6% a 10 ms.

O teste de duração foi conduzido por 30 minutos com intervalo de amostragem de 10 ms, o cenário mais exigente testado. A interface permaneceu responsiva durante todo o período sem travamentos. O consumo de memória cresceu de 143,5 MB para 164,2 MB, incremento de aproximadamente 0,69 MB por minuto, atribuído ao crescimento contínuo dos buffers de histórico completo da sessão. Foram detectados 4718 pacotes perdidos em saltos unitários distribuídos regularmente ao longo da sessão, sem ocorrência de rajadas de perda ou corrupção de dados.

A exportação dos dados para CSV foi validada importando o arquivo gerado no software Excel. O arquivo ilustrado na Figura 4 está com todas as colunas (tempo, tensões e correntes estimadas) corretamente preenchidas, confirmando a integridade do histórico de dados registrado pela interface.

	A	B	C	D	E	F	G	H	I	J
1	Tempo	A0_Tensao	A1_Tensao	A2_Tensao	A3_Tensao	A0_Corrente	A1_Corrente	A2_Corrente	A3_Corrente	Frequencia
2	0	0,26	0,2	0	3,29	0,00208	0,0016	0	0,02632	3,33
3	0,01055	0	0	0,02	3,29	0	0	0,00016	0,02632	3,33
4	0,02032	0	0	0,02	3,29	0	0	0,00016	0,02632	3,33
5	0,03056	0,07	0,04	0	3,29	0,00056	0,00032	0	0,02632	3,33
6	0,0397	0,21	0,19	0,02	3,29	0,00168	0,00152	0,00016	0,02632	3,33
7	0,04954	0,04	0	0	3,29	0,00032	0	0	0,02632	3,33
8	0,06	0,34	0,33	0,02	3,29	0,00272	0,00264	0,00016	0,02632	0
9	0,06956	0,33	0,28	0	3,29	0,00264	0,00224	0	0,02632	0
10	0,07503	0,4	0,36	0	3,29	0,0032	0,00288	0	0,02632	0
11	0,08654	0,1	0,08	0	3,29	0,0008	0,00064	0	0,02632	0
12	0,0987	0,03	0	0	3,29	0,00024	0	0	0,02632	0
13	0,12593	0,33	0,29	0,01	3,29	0,00264	0,00232	8,00E-05	0,02632	0
14	0,13469	0	0	0,02	3,29	0	0	0,00016	0,02632	3,33
15	0,14687	0,1	0,05	0	3,29	0,0008	0,0004	0	0,02632	3,33
16	0,1588	0,41	0,4	0	3,29	0,00328	0,0032	0	0,02632	3,33
17	0,17083	0,16	0,18	0	3,29	0,00128	0,00144	0	0,02632	3,33
18	0,18289	0	0	0	3,29	0	0	0	0,02632	3,33
19	0,19496	0,21	0,15	0	3,29	0,00168	0,0012	0	0,02632	3,33
20	0,20731	0,42	0,34	0	3,29	0,00336	0,00272	0	0,02632	0
21	0,23296	0,01	0	0,02	3,29	8,00E-05	0	0,00016	0,02632	0
22	0,2431	0,32	0,27	0,01	3,29	0,00256	0,00216	8,00E-05	0,02632	0
23	0,2554	0,2	0,11	0	3,29	0,0016	0,00088	0	0,02632	0
24	0,26712	0	0	0	3,29	0	0	0	0,02632	0
25	0,27912	0,2	0,15	0	3,29	0,0016	0,0012	0	0,02632	0
26	0,29136	0,39	0,4	0,03	3,29	0,00312	0,0032	0,00024	0,02632	3,33

Figura 4. Arquivo .csv exportado para o Excel.

CONCLUSÃO

Este trabalho apresentou o desenvolvimento de um sistema integrado de controle e monitoramento baseado no microcontrolador TM4C123GH6PM, com interface gráfica em tempo real desenvolvida em Python. O firmware implementado permite a geração de sinal PWM com frequência e razão cíclica ajustáveis, a aquisição simultânea de quatro canais analógicos via ADC de 12 bits e a comunicação serial bidirecional com o PC a 115200 baud. A interface gráfica, por sua vez, viabiliza a visualização em tempo real das tensões e correntes estimadas, o controle interativo dos parâmetros de operação e a exportação do histórico de medições para CSV.

A latência de comando estimada, inferior a 1 ms, demonstra que o tratamento assíncrono dos

comandos na ISR da UART é eficaz para garantir resposta imediata aos ajustes do operador, independentemente do ciclo de aquisição em andamento. O teste de duração de 30 minutos confirmou a estabilidade operacional da interface, que permaneceu responsiva sem travamentos durante todo o período.

Os resultados demonstraram que o sistema opera de forma estável e com precisão compatível com aplicações de instrumentação em laboratório. A arquitetura de software adotada mostrou-se eficaz para garantir a responsividade da interface sem perda de dados durante as sessões de monitoramento. A caracterização da taxa de amostragem indicou que a perda de pacotes observada em intervalos curtos decorre de limitações no timing do delay de software no laço principal do firmware, e não de corrupção ou saturação do canal serial. O sistema opera de forma confiável para intervalos iguais ou superiores a 200 ms, sendo o intervalo de 100 ms o limite prático para aplicações que tolerem perda moderada de amostras. O protocolo de comunicação serial proposto revelou-se simples, robusto e de fácil extensão para novos parâmetros ou periféricos.

O sistema desenvolvido possui potencial de aplicação em diferentes contextos. Na área de eletrônica de potência, pode ser utilizado como plataforma de monitoramento e controle de conversores DC-DC, permitindo o acompanhamento das variáveis elétricas do circuito em operação e o ajuste dos parâmetros de chaveamento sem necessidade de reprogramação do firmware. No contexto educacional, o sistema pode ser empregado em uma bancada didática instrumentada em disciplinas de Eletrônica de Potência, Instrumentação Eletrônica e Sistemas de Controle, oferecendo aos estudantes uma ferramenta prática para observação dos fenômenos envolvidos no chaveamento e na aquisição de sinais.

Como trabalhos futuros, propõe-se a implementação de um controlador em malha fechada, como um controlador PI ou PID, para regulação automática de tensão ou corrente de saída, a expansão da interface para suporte a múltiplos dispositivos conectados simultaneamente a adição de indicadores de eficiência energética calculados em tempo real pela própria interface gráfica.

REFERÊNCIAS

DA SILVA, Rogério Oliveira; ARAUJO, Warley Monteiro; CAVALCANTE, Maxwell Machado. Visão geral sobre microcontroladores e prototipagem com Arduino. **Tecnologias Em Projeção**, v. 10, n. 1, p. 36-46, 2019.

PEÑA, Eric; LEGASPI, Mary Grace. Uart: A hardware communication protocol understanding universal asynchronous



receiver/transmitter. **Visit Analog**, v. 54, n. 4, p. 1-5, 2020.

QUEIROZ, Francisco Alisson et al. Aplicação de sistemas embarcados e suas tecnologias no combate a covid-19: Uma breve revisão da literatura. **RECIMA21-Revista Científica Multidisciplinar-ISSN 2675-6218**, v. 5, n. 8, p. e585603-e585603, 2024.

SUMMERFIELD, Mark. **Rapid GUI programming with Python and Qt: the definitive guide to PyQt programming**. Pearson Education, 2007.

YU, Zhenyu; MOHAMMED, Arefeen; PANAHI, Issa. A review of three PWM techniques. In: **Proceedings of the 1997 American Control Conference (Cat. No. 97CH36041)**. IEEE, 1997. p. 257-261.