UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RAUL SCARMOCIN DE FREITAS

ANÁLISE DO DESEMPENHO DE MICROCONTROLADORES PARA SISTEMAS DE CONTROLE

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2019

RAUL SCARMOCIN DE FREITAS

ANÁLISE DO DESEMPENHO DE MICROCONTROLADORES PARA SISTEMAS DE CONTROLE

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Dr. Diogo Ribeiro Vargas

PATO BRANCO
2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

TERMO DE APROVAÇÃO

Às 08 horas e 20 minutos do dia 10 de dezembro de 2019, na sala V103, da Universidade Tecnológica Federal do Paraná, *Campus* Pato Branco, reuniu-se a banca examinadora composta pelos professores Diogo Ribeiro Vargas (orientador). César Rafael Claure Torrico e Gustavo Weber Denardin para avaliar o trabalho de conclusão de curso com o título **Análise do Desempenho de Microcontroladores para Sistemas de Controle**, do aluno **Raul Scarmocin de Freitas**, matrícula 1636871, do curso de Engenharia de Computação. Após a apresentação o aluno foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Prof. Diogo Ribeiro Vargas
Orientador (UTFPR)

_____
Prof. César Rafael Claure Torrico
(UTFPR)

_____
Prof. Gustavo Weber Denardin
(UTFPR)

_____
Prof. Marco Antonio de Castro Barbosa
Coordenador de TCC

_____
Prof. Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Aos meus pais, pelo incentivo e apoio durante os anos em que estava cursando a graduação.

Ao meu orientador Prof. Dr. Diogo Ribeiro Vargas, pelo apoio, suporte e empenho durante o desenvolvimento deste trabalho, tal como nas disciplinas que ministrou.

À todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional.

À esta universidade, seu corpo docente, direção e administração que favoreceram a janela que hoje vislumbro um horizonte superior, eivando pela acendrada confiança no mérito e ética aqui presentes.

Agradeço a todos os amigos que cursaram esta universidade, companheiros de trabalhos e irmãos na amizade que fizeram parte da minha formação pela acendrada confiança no mérito e ética aqui presentes.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

O ego é dotado de um poder, de uma força criativa, conquista tardia da humanidade, a que chamamos vontade. (JUNG, Carl, 1875).

RESUMO

SCARMOCIN, Raul. Análise do Desempenho de Microcontroladores para Sistemas de Controle. 2019. 66 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2019.

Microcontroladores são circuitos integrados que podem ser programados para aplicações específicas, entre elas, uma aplicação comum é em sistemas de controle. No entanto, na implementação de um sistema de controle, muitas vezes o modelo do microcontrolador é escolhido baseado apenas em seus periféricos ou de outras características como velocidade de *clock*, quando uma alternativa para tal escolha é efetuar o projeto baseado na capacidade de tal microcontrolador controlar o sistema em malha fechada, sem que hajam falhas por via de sua capacidade de processamento para tal aplicação. Tal dado não pode ser simplesmente encontrado na documentação do microcontrolador, pois trata-se de um dado da aplicação em questão, o que muitas vezes gera o seguinte problema: precisa-se de um método para determinar se um determinado microcontrolador vai ou não atender os requisitos de desempenho para a aplicação de controle desejada. Neste trabalho é proposto o uso de uma metodologia para estimar o desempenho de alguns microcontroladores atuando em aplicações de sistema de controle, e até que ponto pode-se obter resultados adequados para a planta sobre determinadas condições. Os microcontroladores escolhidos tem características únicas, a fim de obter-se um comparativo real entre os modelos ESP32D0WDQ6, STM32F407VET6, STM32F103C8T6 e MSP430G2553.

Palavras-chave: Microcontrolador. Medidas de Desempenho. *Benchmark*. Sistemas de Controle. Tempo Discreto.

ABSTRACT

SCARMOCIN, Raul. Microcontroller Performance Analysis for Control Systems. 2019. 66 f. Trabalho de Conclusão de Curso – Computer Engineering Course, Universidade Tecnológica Federal do Paraná. Pato Branco, 2019.

Microcontrolers are integrated circuits that can be programed for a specific application. One common example of these applications is in control systems. However, in the design of a control system, it's common to choose the microcontroler based only on its peripherals or other features like clock frequency. An alternative for such a choice is made it based on the ability of a microcontroller to control the closed-loop system without failures via its processing capability for such an application. Such information can not be simply found in the microcontroller's documentation, because it is given by the application, which often generates the following problem: the need for a method to determine whether or not a microcontroller meets the performance requirements for the desired control application. In this paper, a methodology is proposed to estimate the performance of some microcontrollers that can operate in control system applications, and to what extent one can obtain suitable results on the plant under certain conditions. The choosen models are ESP32D0WDQ6, STM32F407VET6, STM32F103C8T6 e MSP430G2553, due to having diferente features, which allows to obtain a real comparitive between different devices.

Keywords: Microcontroler. Performance Measure. Benchmark. Control System. Discrete-Time.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	Do Inglês, Conversor Analógico para Digital.
ALU	Do Inglês, Unidade Lógica e Aritmética.
ARM	Do Inglês, Máquina *RISC* Avançada.
ART	*Adaptive Real-Time Memory Accelerator*.
BCU	Do Inglês, Unidade Básica de *Clock*.
CI	Circuito Integrado.
CPU	Do Inglês, Unidade Central de Processamento.
DAC	Do Inglês, Conversor Digital para Analógico.
DMA	Do Inglês, Acesso Direto a Memória.
DSP	Do Inglês, Processamento de Sinal Digital.
FPU	Do Inglês, Unidade de Ponto Flutuante.
GPIO	Do Inglês, Entrada e Saída de Propósito Geral.
HAL	Do Inglês, Camada de Abstração de *Hardware*.
I2C	*Inter-Integrated Circuit*.
IoT	Do Inglês, Internet das Coisas.
IRAM	*Instruction* RAM.
JTAG	*Joint Test Access Group*.
MCU	Do Inglês, Unidade de Controle de Memória.
MFLOPS	Do Inglês, Milhões de Operações de Ponto Flutuante Por Segundo.
MIPS	Do Inglês, Milhões de Instruções Por Segundo.
NVS	*Non-Volatile Storage*.
PD	Proporcional-Derivativo
PI	Proporcional-Integral
PID	Proporcional-Integral-Derivativo.

PWM	Do Inglês, Modulação por Largura de Pulso.
RAM	Do Inglês, Memória de Acesso Aleatório.
RISC	Do Inglês, Computador com Conjunto de Instruções Reduzido.
RTOS	Do Inglês, Sistema Operacional de Tempo Real.
SaH	Do Inglês, Amostrador e Retentor.
UART	*Universal Asynchronous Receiver/Transmitter*.
UC	Unidade de Controle.
WCET	Do Inglês, Tempo de Execução no Pior Caso.
ZOH	Do Inglês, Retentor de Ordem Zero.

LISTA DE SÍMBOLOS

$\mathcal{Z}$	Transformada Z
$\mathcal{Z}^{-1}$	Transformada Z Inversa
$\mathcal{L}$	Transformada de Laplace
$\mathcal{L}^{-1}$	Transformada Inversa de Laplace
$\prod$	Produtório Matemático
f_{sa}	Frequência de Amostragem
$f_{sa.min}$	Frequência Mínima de Amostragem
$f_{sa.max}$	Frequência Máxima de Amostragem
$f_{clk.PWM}$	Frequência de *clock* do PWM
f_{PWM}	Frequência do PWM
$n_{bits.ADC}$	Quantidade de *bits* do ADC
$n_{bits.PWM}$	Quantidade de *bits* efetivos do PWM
$res_{ef.PWM\%}$	Resolução efetiva do PWM
t_{adc}	Tempo de conversão do ADC
$t_{cpu.max}$	Tempo de Execução das operações da CPU no Pior Caso
$t_{e.max}$	Tempo de Execução Total no Pior Caso
t_{ss}	Tempo de Assentamento
Δt	Variação Máxima na Medida de Tempo
$T_{sa.max}$	Período Máximo de Amostragem
$T_{ef.PWM}$	Período Efetivo do PWM

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos o uso de sistemas computacionais em aplicações diversas vem aumentando de forma crescente, sendo utilizados desde dispositivos móveis pessoais, como celulares, até sistemas computacionais embarcados usados em aplicações que necessitam de mais segurança ou efetuar algum tipo de controle específico, como os sistemas de uma aeronave, por exemplo.

Muitas dessas aplicações necessitam de sistemas computacionais reduzidos e de baixo custo, tamanho e consumo de energia. Entre esses, é comum o uso de microcontroladores, por sua capacidade de suprir tal necessidade em geral, de forma que pode ser encontrada uma variedade de microcontroladores com modelos e marcas diferentes. Tal diversidade de microcontroladores apresenta uma enorme variedade de configurações e periféricos, podendo-se escolher desde baixo custo e consumo de energia, quantidade de portas de comunicação, velocidade de *clock* principal, número de *bits* dos conversores, vários tipos de memória, periféricos adicionais, como módulo *wireless* e *bluetooth*, entre outras possibilidades.

Com tamanha diversidade entre os modelos, torna-se difícil para o projetista efetuar a escolha certa do microcontrolador, e quais parâmetros do dispositivo devem ser avaliados para tal. Normalmente tal escolha é feita a partir de informações não tão relevantes para o projeto, como o número de portas e pinos do microcontrolador, ou mesmo periféricos que não são realmente utilizados. Em alguns casos extremos, a escolha é feita de forma aleatória, baseando-se nos modelos disponíveis, ou ainda, utiliza-se um sistema legado, em que o projetista está familiarizado com um modelo obsoleto utilizado em projetos anteriores.

Fazer a escolha do microcontrolador sem uma análise prévia dos requisitos da aplicação e sobre as características dos possíveis modelos pode causar diversos problemas ao projeto. Alguns destes são: dificuldade durante o desenvolvimento, déficit de periféricos necessários para o projeto, dificuldade ou impossibilidade de depuração em casos de erro e finalmente a possibilidade de o microcontrolador escolhido não suprir as necessidades de desempenho necessárias para efetuar sua devida função. Outro problema que pode ocorrer é a elevação do custo do projeto por superdimensionamento do microcontrolador.

1.1 OBJETIVOS

Este trabalho tem como objetivo obter resultados experimentais de testes com medidas de desempenho de alguns microcontroladores que serão definidos posteriormente, fazendo uso de métodos já existentes para a estimar a frequência para cada algoritmo em cada microcontrolador, que se dá no pior caso de tempo.

1.1.1 OBJETIVOS ESPECÍFICOS

a) Projetar um código genérico de controle PID (Proporcional-Integral-Derivativo) que será aplicado em cada microcontrolador escolhido, uma vez que a estrutura do código principal não deve sofrer grandes modificações entre os diferentes parâmetros de projeto, como: compilador, arquitetura e outras características dos microcontroladores utilizados.
b) Implementar uma planta genérica para a qual será efetuado o controle, a fim de verificar o funcionamento do algoritmo desenvolvido.
c) Realizar os testes enquanto executa-se uma função com cálculos diferenciados, como divisões, de forma que venham a gastar tempo do microcontrolador, afetando a medida de desempenho.
d) Realizar os testes propostos com diferentes representações numéricas dos parâmetros do código computacional, mostrando a diferença de performance entre eles.
e) Analisar a velocidade de conversão do ADC (*Analógic to Digital Converter*, Conversor Analógico-Digital) e seu impacto sobre a performance e sobre o sistema.
f) Organizar os dados obtidos nos testes de forma gráfica e tabulada, facilitando a análise e a compreensão dos fatores que podem afetar o desempenho dos microcontroladores sob as condições aplicadas e discutir os resultados obtidos.

1.2 ORGANIZAÇÃO DO TRABALHO

Para possibilitar uma melhor compreensão, este trabalho foi organizado seguindo uma estrutura a partir dos objetivos citados na Seção 1.1, de forma que os tópicos podem ser facilmente relacionados para o estudo.

O Capítulo 2 aborda o conhecimento necessário das áreas de estudo que esse trabalho abrange. Partindo dos princípios de sistemas de controle, controle digital, microcontroladores, arquitetura de computadores e finalmente medidas de desempenho, as seções do Capítulo 2 fundamentam os métodos utilizados nesse trabalho.

O Capítulo 3 faz uso da teoria fundamentada no Capítulo 2 para assim apresentar a proposta de trabalho que será efetuada para cumprir os objetivos citados na Seção 1.1. Essa seção contém os detalhes para o desenvolvimento geral do trabalho. Além disso, esse capítulo contém todo o desenvolvimento do trabalho, além da coleta e tabulação de dados.

O Capítulo 4 demonstra os resultados obtidos pelo estudo realizado e pelos dados coletados no Capítulo 3, realizando uma sequencia de análises sobre os aspectos dos microcontroladores de forma gráfica e didática.

O Apêndice A contém bibliotecas que complementam o conteúdo apresentado nos demais capítulos, observando-se o algoritmo desenvolvido na Seção 3.2. O código da biblioteca foi desenvolvido na linguagem de programação C e pode ser usado de forma genérica para efetuar a ação de controle PID que foi usada para os estudos e testes desse trabalho.

O Apêndice B contém bibliotecas desenvolvidas em linguagem de programação C para

manipulação genérica de números em formato Q_n, conforme os algoritmos desenvolvidos na Seção 3.3.

2 REVISÃO DE LITERATURA

Neste capítulo encontra-se o referencial teórico necessário para a compreensão dos detalhes e objetivos do trabalho, além de conter a base teórica em resumo usado durante o desenvolvimento, que segue uma ordem cronológica incremental visando facilitar o entendimento. Partindo dos princípios de sistemas de controle e sua suma importância no campo de aplicação deste trabalho, revisando então os princípios de sistemas microcontrolados e suas arquiteturas, comentando enfim sobre *benchmarks* e métodos para a mensuração de performance de sistemas digitais.

2.1 SISTEMAS DE CONTROLE

Sistemas de controle automático tornaram-se indispensáveis em qualquer área da tecnologia, independente de sua aplicação. Partindo de simples reguladores de tensão para a eletrônica, aplicando-se a automação de máquinas industriais e sistemas robóticos, até o controle de sistemas espaciais, as aplicações de sistemas de controle tornaram-se praticamente indispensáveis para a engenharia.

Segundo Nise e Silva (2002), sistemas de controle são constituídos por subsistemas e processos, que tem o objetivo de obter uma saída desejada, a partir de uma entrada conhecida. Define-se perturbação como um sinal externo aplicado ao sistema, que tende a afetar sua resposta. Define-se também sistema estável como um sistema linear e invariante no tempo, no qual sua resposta natural tende a anular-se conforme o tempo tende a infinito (OGATA; SEVERO, 1998).

Considerando que um sistema é estável, sua resposta pode ser dividida em duas variantes, visando facilitar a análise de controle, sendo elas:

a) Resposta transitória (ou dinâmica), trata-se da análise da resposta do sistema após ser aplicada uma perturbação, na qual o sistema permanece ressonante por um determinado período de tempo, apresentando consideráveis variações em sua saída.
b) Resposta em regime permanente, trata-se da análise da resposta do sistema após um longo período de tempo, na qual o sistema se encontra estático, sem apresentar maiores variações em sua resposta.

Sistemas de controle de malha aberta são aqueles em que sua resposta não afeta a ação de controle do sistema, sendo então vulneráveis às perturbações, que podem eventualmente afetar a resposta do sistema. Esse problema pode ser solucionado fazendo uso de uma realimentação no sistema para fechar a malha, com o objetivo de eliminar o desvio devido a perturbações.

Visto que esse trabalho tem como um de seus objetivos específicos efetuar o controle de forma a eliminar qualquer possível erro na resposta do sistema, buscando otimizar o sinal de saída, não serão abordado sistemas de controle em malha aberta, mas sim em malha fechada, que serão discutidos na seção a seguir.

2.1.1 SISTEMAS DE CONTROLE EM MALHA FECHADA

Sistemas de controle em realimentação são aqueles que tem uma parcela de seu sinal de saída devolvidos em sua entrada, fazendo uma comparação entre tal parcela e o sinal de referência do sistema, com o objetivo de tornar o sistema menos vulnerável a perturbações e erro em regime permanente.

Sistemas de controle em malha fechada são sistemas de controle com realimentação que podem gerar um sinal de erro, denotado por $e(t)$, comparando-se o sinal de saída do sistema com o sinal de referência. O controlador responde ao sinal de erro $e(t)$ com um sinal de controle, denotado por $u(t)$, que é gerado esperando-se que a planta responda a tal sinal de forma a reduzir ou eliminar o sinal de erro.

A Figura 1 exemplifica um sistema de malha fechada, constituído por um controlador para minimizar o erro, um atuador, que recebe um sinal de controle do controlador, e a partir desse, aplica um sinal $v(t)$ que age sobre a planta, e um transdutor, que trata-se de um componente que converte uma grandeza física de uma natureza em uma grandeza física de uma natureza diferente.

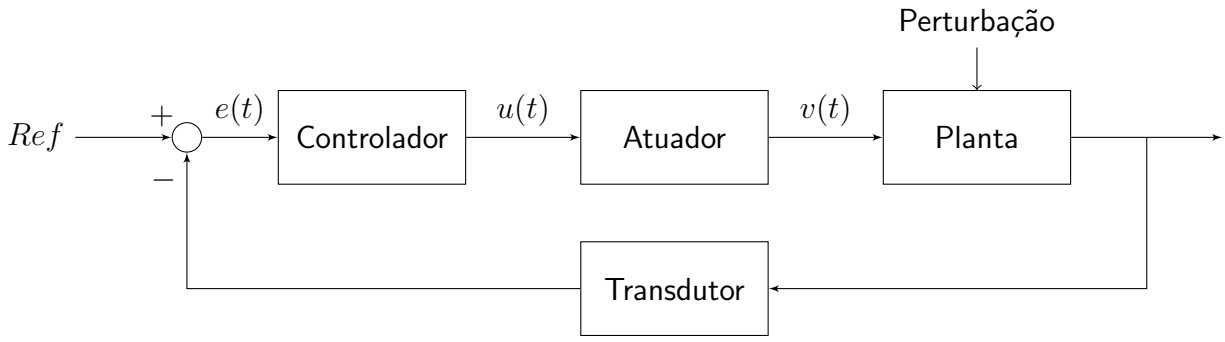


Figura 1 – Exemplo de um Sistema de Controle de Malha Fechada

Fonte: Adaptado de Dorf et al. (2005)

2.1.2 CONTROLADOR PROPORCIONAL - INTEGRAL - DERIVATIVO

Existem incontáveis topologias que podem ser usadas no projeto de um controlador de malha fechada, entre elas, uma das topologias de controladores mais utilizadas é o controlador PID. Essa topologia é amplamente utilizada na indústria e será abordada e implementada para os testes nesse trabalho (DORF et al., 2005).

Um controlador PID tem três termos acompanhados por constantes, as quais devem ser ajustadas pelo projetista para tornar o controle do sistema o mais eficiente possível para a aplicação, sendo:

i) K_p é a constante que acompanha o termo proporcional do controlador, que cresce proporcionalmente com o erro medido na realimentação do sistema.

ii) K_i é a constante que acompanha o termo integral do controlador, o qual acumula uma parcela do erro medido na realimentação do sistema.

iii) K_d é a constante que acompanha o termo derivativo do controlador, que representa variações do erro medido na realimentação do sistema.

Esses termos podem ser compreendidos observando-se a Equação (1) que determina a saída do controlador em relação ao erro no domínio da frequência:

$$G_c(s) \;=\; \frac{U_c(s)}{E_c(s)} \;=\; K_p \;+\; \frac{K_i}{s} \;+\; K_d \cdot s \tag{1}$$

em que $E_c(s)$ representa o erro no domínio da frequência, $U_c(s)$ representa o sinal de controle no domínio da frequência e $G_c(s)$ representa a função de transferência do próprio controlador.

Uma maneira de representar a Equação (1) por diagrama de blocos é observada na Figura 2:

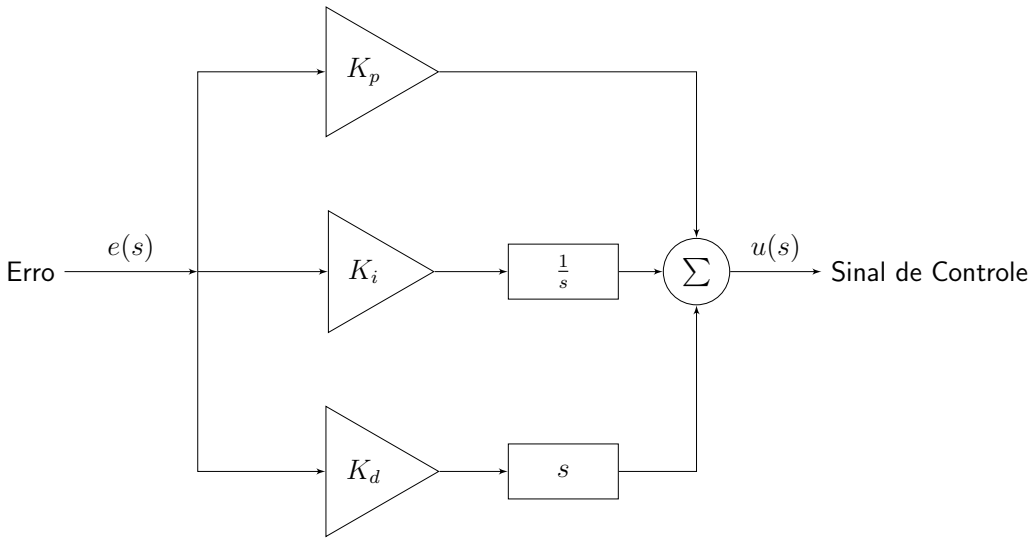


Figura 2 – Diagrama de Blocos de um Controlador PID no Domínio da Frequência

Fonte: Adaptado de Kuo (1980)

Existem algumas variações dessa topologia, como o PI (*Proporcional-Integral*), ou ainda o PD (*Proporcional-Derivativo*), os quais não serão abordados nesse trabalho.

2.2 SISTEMAS DE CONTROLE DIGITAL

Com a rápida evolução da tecnologia e dos computadores, torna-se cada vez mais comum o uso de computadores digitais em sistemas de controle, uma vez que esses trazem

diversas vantagens para as aplicações, tais como a redução do custo, maior facilidade de desenvolvimento eletrônico em comparação com controladores eletrônicos analógicos, maior liberdade para mudanças na topologia do controlador em caso de alterações do projeto. Controladores digitais também tornam possível a implementação de vários controladores em um único computador digital (NISE; SILVA, 2002).

Um controlador digital pode ser dividido em três unidades com funções distintas, para facilitar sua análise:

i) Aquisição do valor de entrada no transdutor através do ADC para obtenção do erro.
ii) Processamento dos dados de entrada para efetivar a ação de controle.
iii) Escrita digital do valor de saída para o atuador através do DAC (*Digital to Analógic Converter*, Conversor Digital-Analógico).

O sistema de aquisição de dados e resposta do controlador digital tem uma composição mais específica, que pode acabar por afetar o tempo de resposta do controlador para o sistema de controle, portanto, deve-se leva-los em consideração durante o estudo do desempenho do controlador digital. Antes de prosseguir, deve-se definir brevemente os conceitos de quantização, amostrador e retentor.

Quantização em amplitude é o processo que permite representar um sinal contínuo por uma quantidade finita de estados discretos. A saída de um quantizador é um código numérico, gerado por um processo de codificação. Esse processo gera um número codificado, que pode ser representado por números binários. Porém, o processo de codificação não é instantâneo, ou seja, ele necessita de um numero pulsos de *clock* para gerar o número binário (OGATA, 1995).

Circuitos de amostragem, ou *sample*, tornam-se necessários quando for feita uma leitura no ADC, pois não podem haver variações durante o processo de aquisição de sinal. O circuito de *sample* funciona como uma chave, que se abre no momento da coleta do sinal. O circuito de *hold* nada mais é que um circuito para segurar o nível de tensão no momento em que a chave se abre. Como na prática normalmente são usados juntos são chamados de circuito SaH (*Sampler and Holder*, Segurador e Retentor), mas matematicamente são considerados como dois sistemas separados (TOCCI; WIDMER; MOSS, 2003).

Segundo Kuo (1980), um quantizador Q com n *bits* tem uma resolução que pode ser calculado pela Equação (2).

$$q \;=\; 2^{-n} \cdot FS \tag{2}$$

em que FS representa o fundo de escala do quantizador.

Portanto, o maior erro que o quantizador pode apresentar pode ser calculado pela Equação (3).

$$\frac{q}{2} \;=\; 2^{-n-1} \cdot FS \tag{3}$$

2.2.1 AQUISIÇÃO E CONVERSÃO DO SINAL ANALÓGICO

Segundo Dorf et al. (2005), o sistema de aquisição de dados do controlador digital pode ser observado como várias etapas para a transformação da grandeza física, que será o sinal analógico obtido pelo transdutor até a entrada efetivamente do computador digital que é responsável pelo processamento da informação e por enviar o sinal da ação de controle correspondente para o atuador.

As etapas para a aquisição do sinal são:

i) Obtenção do sinal analógico pelo transdutor, convertendo uma grandeza física de alguma natureza em tensão elétrica, não necessariamente uma grandeza diferente do sinal de entrada.
ii) Amplificação do sinal, para garantir que o sinal obtido não será maior que a tensão teto que o quantizador suporta, também como não será pequeno suficiente para que sua escala de resolução não consiga detectar.
iii) Remoção de ruídos de alta frequência do sinal amplificado através de um filtro passa-baixa, para resultar em menor erro propagado durante o processamento da ação de controle.
iv) Multiplexação do sinal obtido em um multiplexador analógico, pois mais de um sinal pode estar sendo obtido pelo ADC em canais diferentes.
v) Amostragem do sinal pelo circuito de SaH.
vi) Aquisição do sinal pelo ADC.

A Figura 3 ilustra as etapas de aquisição de dados descritas acima (NISE; SILVA, 2002).

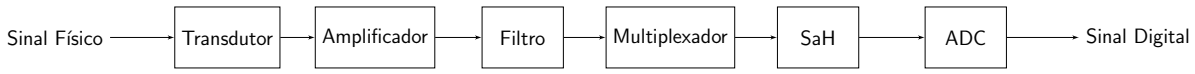


Figura 3 – Aquisição e Conversão do Sinal Analógico para Digital

Fonte: Adaptado de Nise e Silva (2002)

Antes de definir como ocorre a conversão da resposta digital para analógica, deve-se definir o conceito de ZOH (*Zero Order Holder*, Retentor de Ordem Zero), que é utilizado por ter a característica de não causar atraso em sua resposta, como ocorre com extrapoladores de ordem superior (KUO, 1980).

Segundo Dorf et al. (2005), o circuito ADC pode ser representado como um circuito ZOH. Um circuito ZOH pode ser representado por um sistema recebe o valor de entrada e mantém esse valor em sua saída por um intervalo constante de tempo, independente das variações do sinal de entrada nesse período. A função de transferência circuito ZOH é definida na Equação (4):

$$\frac{1 - e^{-sT}}{s} \tag{4}$$

2.2.2 CONVERSÃO DA RESPOSTA DIGITAL PARA ANALÓGICA

Para que o sinal digital de controle possa ser efetivado pelo atuador, ele deve antes ser convertido novamente para um sinal analógico. Isso pode ser feito por um DAC, que normalmente usam um dos seguintes métodos de conversão:

a) Resistores ponderados.
b) Rede R-2R.
c) PWM (*Pulse Width Modulation*, Modulação em Largura de Pulso).

A conversão do sinal digital para analógica normalmente segue as seguintes etapas:

i) Carregamento do valor da resposta digital nos registradores.
ii) Demultiplexação do valor.
iii) Conversão do sinal digital para analógico pelo DAC.
iv) Mantimento do nível de tensão do sinal analógico obtido através do circuito de *Hold*.

A Figura 4 ilustra as etapas conversão do sinal digital do controlador para um sinal analógico que que comanda o atuador (NISE; SILVA, 2002).

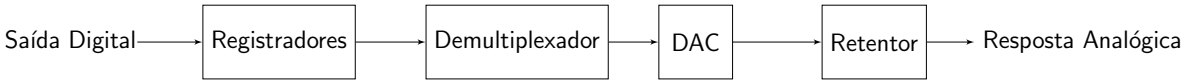


Figura 4 – Conversão da Resposta Digital do Controlador para Analógica

Fonte: Adaptado de Nise e Silva (2002)

2.2.3 CONTROLADOR PID DIGITAL

Como especificado no início da Seção 2.2, um controlador digital pode ser dividido em três unidades distintas. Tendo definido como é realizada a aquisição do sinal analógico e sua conversão para digital, e a conversão da resposta digital do controlador para analógica, e lembrando os conceitos de controlador de malha fechada definidos na Subseção 2.1.1, finalmente observa-se na Figura 5 um exemplo de sistema de controle digital.

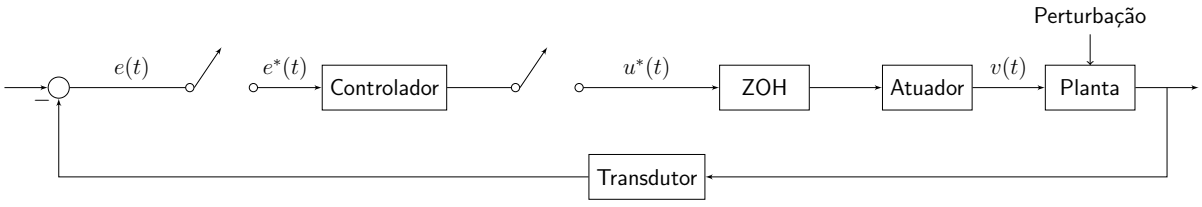


Figura 5 – Exemplo de um Sistema de Controle com Controlador Digital em Cascata

Fonte: Adaptado de Kuo (1980)

Para se aplicar a transformada Z em um sistema no domínio da frequência, deve-se utilizar uma aproximação numérica, pois o sinal depende da frequência de amostragem em

relação ao sinal contínuo. Segundo Starr (2006) as três aproximações mais utilizadas são o método de *forward*, método de *backward* e método trapezoidal, também conhecido como aproximação de *Tustin*.

No Quadro 1 é feita uma comparação entre as equações das aproximações citadas em relação ao domínio da frequência.

Método	Aproximação Numérica	Mapeamento Inverso
Forward	$s \approx \frac{z-1}{T}$	$z = 1 + s \cdot T$
Backward	$s \approx \frac{z-1}{s \cdot T}$	$z = \frac{1}{1 - s \cdot T}$
Tustin	$s \approx \frac{2}{T} \cdot \frac{z-1}{z+1}$	$z = \frac{1 + s \cdot T/2}{1 - s \cdot T/2}$

Quadro 1 – Comparação Entre Aproximações Numéricas.

Fonte: Starr (2006)

Segundo Kuo (1980), aplicando a transformada Z usando a aproximação de *Tustin* na função de transferência do controlador PID definida na Equação (5), é obtida a seguinte função de transferência do controlador PID de tempo discreto:

$$G_t(z) = \frac{(2K_pT + K_iT^2 + 2K_d)z^2 - (K_pT + 2K_d)z + K_d}{2T \cdot z^2 - 2T \cdot z} \tag{5}$$

Manipulando a Equação (5) a fim de obter-se apenas termos com atraso, tem-se a Equação (6):

$$G_t(z) = \frac{U_t(z)}{E_t(z)} = \frac{K_a - K_b \cdot z^{-1} + K_c \cdot z^{-2}}{1 - z^{-1}} \tag{6}$$

Em que as constantes K_1, K_2 e K_3 representam respectivamente:

$$K_a = \frac{K_i \cdot T}{2} + K_p + \frac{K_d}{T} \tag{7}$$

$$K_b = \frac{K_i \cdot T}{2} - K_p - \frac{2 \cdot K_d}{T} \tag{8}$$

$$K_c = \frac{K_d}{T} \tag{9}$$

Aplicando a transformada inversa na Equação (6) para o tempo discreto, obtém-se por inspeção a Equação (11).

$$\mathcal{Z}^{-1}\{U_t(z) \cdot (1 - z^{-1})\} = \mathcal{Z}^{-1}\{E_t(z) \cdot (K_1 + K_2 \cdot z^{-1} + K_3 \cdot z^{-2})\} \tag{10}$$

$$u[n] - u[n-1] = K_a \cdot e[n] + K_b \cdot e[n-1] + K_c \cdot e[n-2] \tag{11}$$

2.2.4 IMPLEMENTAÇÃO DO CONTROLADOR PID DIGITAL

Segundo Vargas (2017), uma forma de implementar o controlador PID digital é dividindo o programa em dois módulos: o programa principal e o código de interrupção do *timer*.

O programa principal normalmente contém as configurações usadas nos periféricos, declarações de variáveis e a execução de um *loop* infinito, que executa a operação desejada ciclicamente por tempo indeterminado, como observa-se no Algoritmo 1.

Algoritmo 1: Algoritmo do Programa Principal

define K_1, K_2, K_3
$Erro_n \leftarrow 0$, $Erro_{n-1} \leftarrow 0$, $Erro_{n-2} \leftarrow 0$
Configura os periféricos
Ativa Interrupções do Timer
while *true* **do**
| Executa outras tarefas / Ou não executa nenhuma ação
end

Fonte: Adaptado de Vargas (2017)

O código de interrupção do *timer* é configurado com o objetivo de executar uma ação em intervalos periódicos de tempo, observado no Algoritmo 2.

Algoritmo 2: Algoritmo de Controle PID na Interrupção do *Timer*

EscritaPWM($Output$**)**
$Input \leftarrow$ **LeituraADC()**
$Erro_{n-2} \leftarrow Erro_{n-1}$
$Erro_{n-1} \leftarrow Erro_n$
$Erro_n \leftarrow Ref - Input$
$Output \leftarrow Output + Ka \cdot (Erro_n) + Kb \cdot (Erro_{n-1}) + Kc \cdot (Erro_{n-2})$
if $Output > Max$ **then**
| $Output \leftarrow Max$
end
else if $Output < Min$ **then**
| $Output \leftarrow Min$
end

Fonte: Adaptado de Vargas (2017)

Um problema que pode ocorrer em um sistema de controle digital de malha fechada é conhecido como ciclo limite, ou oscilações em regime permanente. O problema se origina quando o conversor AD efetua a aquisição do dado analógico do sistema em regime permanente e calcula um valor de erro $e[k]$ e uma ação de controle $u[k]$. Porém, quando o módulo PWM efetuar a ação de controle $u[k]$, pode acabar gerando um erro com a magnitude de $e[k]$ e sinal oposto.

Segundo Kuo (1980), a amplitude da oscilação no ciclo limite normalmente é um múltiplo do nível de quantização q, que pode ser descrito pela Equação (2). Assim, considera-se que a adição de quantizadores no sistema de controle afetam a estabilidade do sistema.

2.3 SISTEMAS MICROCONTROLADOS

Com o rápido desenvolvimento da tecnologia, o uso de microcontroladores tornou-se cada vez mais popular nas ultimas décadas, sendo comumente utilizado em aplicações diversas, desde sistemas de automação industrial até veículos espaciais. A utilização em larga escala de microcontroladores oferece diversas vantagens como redução de custos, maior intolerância a falhas e flexibilidade no projeto.

Microcontroladores são CI (Circuitos Integrados) programáveis que contém alguns dos módulos mais utilizados para o projeto de sistemas, visando facilitar o uso pelo desenvolvedor dos vários componentes que seriam necessários, integrando todos em um único CI.

Microcontroladores normalmente contem os seguintes módulos (GRIDLING; WEISS, 2007):

a) CPU (*Central Processing Unit*), que contém a ALU (*Arithimetic and Logic Unit*, Unidade Lógica e Aritmética) responsável por realizar operações lógicas aritméticas. Contém também a UC (*Unidade de Controle*) e vários registradores essenciais para o funcionamento do processador.
b) BCU (*Basic Clock Unit*, Unidade Básica de *Clock*), responsável por controlar as múltiplas fontes de *clock* do sistema. Normalmente contém osciladores internos de alta frequência, e em alguns casos, pode incluir também cristais oscilatórios de maior precisão como fonte externa.
c) Memória, necessária para armazenar dados, podendo ser dividida entre memória de dados, ou memória RAM (*Random Access Memory*), e memória do programa, também denominada memória *Flash*. Diferente da memória *Flash*, a memória RAM é considerada volátil, o que impede que dados sejam armazenados caso a alimentação seja interrompida. Porém, é possível armazenar e executar instruções de programa na memória RAM durante o tempo de execução.
d) Controlador de interrupções, unidade que pode desviar o fluxo de código quando um determinado evento ocorre;
e) *Timer*, módulo que pode desempenhar inúmeras funcionalidades para o sistema, sendo algumas delas: geração de PWM, aquisição do tempo decorrido e contagem através do modo comparação e captura, entre outros.
f) GPIO (*General Purpose Input/Output*, Portas de Entrada e Saída de Propósito Geral);
g) Módulo de aquisição analógico, citado na Subseção 2.2.2.
h) Interfaces de comunicação serial, tal como UART (*Universal Asynchronous Receiver/-Transmitter*, Transmissor/Receptor Assíncrono Universal), I2C (*Inter-Integrated Circuit*, Circuito Inter-Integrado), entre outras interfaces com o objetivo geral de comunicação

entre dispositivos.

i) Unidade de depuração, que pode ser encontrada em alguns microcontroladores na forma de um *hardware* que permite depurar o sistema em tempo de execução.

2.3.1 ARQUITETURA DE MICROCONTROLADORES

Antes de aprofundar o conteúdo sobre as arquiteturas de computadores mais utilizadas, deve-se lembrar que após o surgimento do primeiro computador eletrônico de proposito geral em meados dos anos 70, houve um enorme avanço na tecnologia computacional. Tal avanço, combinado com o desenvolvimento e produção massiva de microprocessadores, abriram as portas do mercado para novas arquiteturas de processadores.

Segundo Hennessy e Patterson (2011), o desenvolvedor computacional tem a tarefa principal de determinar que aspectos devem ser levados em consideração durante o desenvolvimento do processador, desde o conjunto de instruções que será utilizado, organização, custo de energia implementação e a aplicação-alvo, e a partir disso, como a arquitetura do microprocessador vai controlar seus recursos da maneira mais otimizada possível. A Figura 6 ilustra a arquitetura básica de um processador.

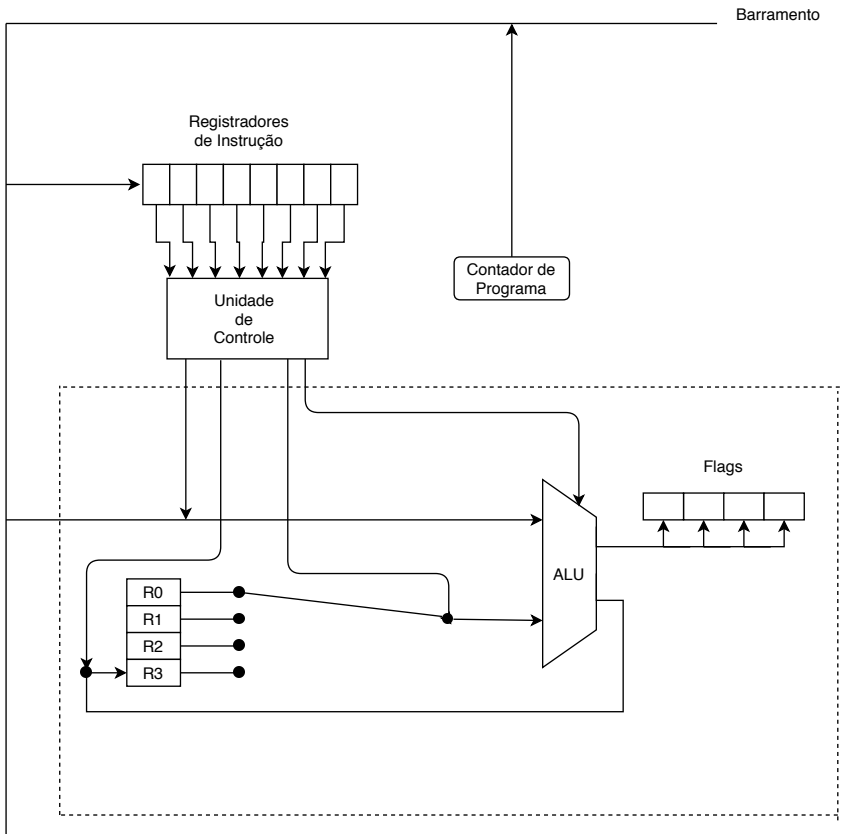


Figura 6 – Arquitetura Básica de um Processador

Fonte: Adaptado de Gridling e Weiss (2007)

Com o surgimento de arquiteturas baseadas em RISC (*Reducted Instruction Set Computer*, Computadores com Conjunto de Instruções Reduzidas), o uso de microprocessadores

com menos instruções tornou-se cada vez maior em sistemas embarcados em geral, quando surgiu a arquitetura ARM (*Advanced RISC Machine*, Máquina RISC Avançada), que tornou-se dominante.

As principais características da arquitetura ARM são seu baixo custo e consumo, simplicidade e modularidade, desenvolvidas para ser facilmente adaptadas à aplicação que o projetista deseja solucionar de forma otimizada, que tem sido utilizada em larga escala em microcontroladores de grandes empresas como Texas Instruments, STMicroeletronics, Atmel, entre muitas outras.

A arquitetura ARM utiliza instruções RISC, que é um conjunto de instruções reduzido, muito utilizado em sistemas pequenos e embarcados. Isso permitiu o desenvolvimento de processadores simples e eficientes, viabilizando o uso de um decodificador de instruções *hardwired*, que possuí muitas vantagens em relação a um decodificador baseado em microcódigo (PEREIRA, 2007).

Existem também outras arquiteturas de microprocessadores, como a Xtensa®, arquitetura com conjunto de instruções desenvolvidos pela Tensilica. A arquitetura foi desenvolvida com o intuito de facilitar o projeto e a modificação de microprocessadores com aplicações mais genéricas que os demais, dando liberdade para o projetista na configuração do microprocessador, para que possa criar soluções eficientes em suas aplicações (TENSILICA, INC., 2010).

2.3.2 OPERAÇÕES COM PONTO FLUTUANTE

A notação binária de ponto fixo normalmente utilizada em sistemas computacionais pode ser usada para representar uma faixa de números inteiros ou mesmo fracionários, escolhendo uma posição do número binário representado e fixando-se um ponto nesse. Porém, tal representação pode ficar limitada para um intervalo numérico menor ou frações com menos precisão.

Para solucionar tal problema, representa-se o número desejado da mesma maneira em que é representado usando a notação cientifica, fazendo uso de uma mantissa M, uma base B e um expoente E, como pode ser observado na Equação (12) (STALLINGS, 2010):

$$\pm M \ + \ B^{\pm E} \tag{12}$$

Números fracionários representados por 32-*bits* são denominados *single-precision floating-point*, ou *float*, sendo 1-*bit* sinalizador, 8-*bits* para o expoente e 23-*bits* fracionários. Já números com precisão dupla, *double-precision floating-point* (*double*), são representados utilizando 64-*bits*, 1-*bit* sinalizador, 11-*bits* para o expoente e 52-*bits* fracionários (754-2008, 2008).

Alguns microprocessadores possuem módulos que podem efetuar algoritmos de multiplicação e divisão com ponto flutuante via *hardware*, reduzindo o custo da operação em para poucos ciclos de *clock*, ou até mesmo um único ciclo. Tais módulos são conhecidos como FPU (*Floating-Point Unit*).

Existem ainda muitos outros algoritmos para efetuar divisões e multiplicações via *hardware*, como o algoritmo de multiplicação de Wallace, que pode efetuar uma multiplicação em alguns ciclos de *clock*, e o algoritmo de Booth, que usa deslocamento de *bits* para efetuar as operações, o que é mais rápido em comparação com uma soma (BOOTH, 1951; WALLACE, 1964).

Os algoritmos citados acima podem ser implementados no *hardware*, utilizando circuitos micro-eletrônicos, com o objetivo de optimizar o desempenho durante uma determinada operação de multiplicação ou divisão. O mesmo pode ser feito para vários outros algoritmos e métodos criados com o mesmo objetivo.

Microcontroladores mais simples não incluem tais sofisticações de *hardware*, tornando a operação desejada por vezes mais lenta e custosa. Isso acontece pois o compilador deve incluir um algoritmo para efetuar a operação por meio de operações de *software* em sequência, o que requer múltiplos ciclos de *clock*.

2.3.3 OPERAÇÕES COM PONTO FIXO

Labrosse (2000) diz que a aritmética de ponto fixo sinalizada pode representar computacionalmente um número inteiro x em um intervalo $-2^n < x < 2^n - 1$, em que $n + 1$ representa a quantidade de *bits* utilizada, considerando o *bit* sinalizador. A Figura 7 ilustra a representação binária de um número inteiro em ponto fixo.

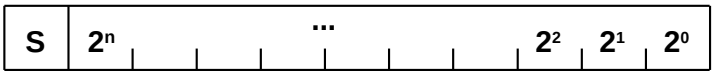


Figura 7 – Representação de um Número em Ponto Fixo

Fonte: Adaptado de Labrosse (2000).

Dessa forma, pode-se representar um número de ponto flutuante usando ponto fixo, deslocando a virgula imaginária, nesse caso também conhecida como mantissa, reduzindo também o alcance do intervalo em que o número pode ser representado. Essa definição pode ser representada também como I_mQ_n, na qual m refere-se ao número de *bits* utilizado para representar a parte inteira e n representa a parte fracionária.

A Figura 8 exemplifica como isso é feito para um caso especifico, representando um número com precisão de 3 *bits*.

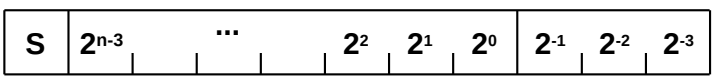


Figura 8 – Representação de um Número Fracionário em Ponto Fixo com Precisão de 3 *bits*

Fonte: Adaptado de Labrosse (2000)

Segundo Stevenson (2002), pode-se fazer com que um número de ponto flutuante seja representado por uma variável de ponto fixo fazendo uso de uma normalização em formato Q_n. Assim, pode-se representar um número fracionário x no formato Q_n, ou $Q(1, n)$ como observado na Equação (13):

$$\pm x \ = \ \sum_{i=0}^{n} b_i 2^{-i} \tag{13}$$

em que b é um valor binário 0 ou 1.

Dessa forma, pode-se representar um número fracionário x no intervalo $-1 \ \leq \ x \ < 1$ utilizando todos os *bits* como mantissa. A Figura 9 ilustra como um número fracionário pode ser representado em formato Q_n:

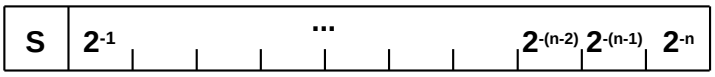


Figura 9 – Representação de um Número Fracionário em Formato Q_n

em que a resolução da representação numérica é dada por 2^{-n}.

Retomando os conceitos de quantização descritos na Seção 2.2, deve-se considerar que quando os ganhos de realimentação são implementados computacionalmente em um sistema de controle digital, esses ganhos estarão sujeitos a efeitos de quantização. Portanto, a alocação de polos do sistema com malha fechada não será efetuada com precisão absoluta, afetando a estabilidade do sistema, podendo gerar uma resposta divergente da desejada (KUO, 1980).

Assim, ao fazer uso de operações com ponto fixo, considera-se também o efeito de quantização que o formato numérico utilizado tem sobre o sistema.

2.4 MEDIDAS DE DESEMPENHO

Esta seção é destinada a revisar o estudo sobre métricas de performance de computadores, que é a principal ferramenta para cumprir os objetivos desse trabalho. Serão também fundamentados trabalhos relacionados a *benchmarks*, medidas de desempenho de sistemas computacionais e sistemas de controle, entre outros tópicos que vem à complementar o entendimento sobre o assunto desse trabalho.

2.4.1 ASPECTOS DE DESEMPENHO

Antes de compreender e estudar os aspectos de performance de um sistema computacional, deve-se determinar quais parâmetros de medições são úteis para o estudo. Algumas das características que comumente tem-se interesse em mensurar são: uma contagem de quantas vezes um evento ocorre, a duração temporal de algum evento ou de um intervalo de eventos, ou ainda o tamanho de algum parâmetro (LILJA, 2005).

Segundo Hennessy e Patterson (2011), a comparação de desempenho entre dois sistemas computacionais pode ter vários aspectos de avaliações, como diminuir o tempo de execução de uma ação, ou executar mais ações em um período de tempo. Assim, pode-se definir a taxa de *troughput* como sendo a quantidade de ações realizadas em um determinado intervalo de tempo. Assim, pode-se dizer que um sistema computacional *X* é mais rápido que um sistema computacional *Y*, levando em consideração uma determinada tarefa, se o tempo de execução de *X* é menor que o tempo de execução da mesma tarefa em *Y*. Ou seja, o sistema computacional *X* pode executar um número maior de tarefas em um determinado período de tempo do que o sistema computacional *Y*, inferindo que a taxa de *troughput* de *X* é maior que a que a taxa de *troughput* de *Y*. Diz-se então que o sistema computacional *X* é n vezes mais rápido que o sistema computacional *Y*:

$$n = \frac{T_{Exec_Y}}{T_{Exec_X}} \tag{14}$$

em que T_{Exec_X} e T_{Exec_Y} são respectivamente o tempo de execução computacional de *X* e *Y*. A Equação (14) também é conhecida como *speed-up* (LILJA, 2005).

Outra técnica utilizada para expressar a performance de um sistema *Y* em relação a um sistema *X* é utilizando percentual de mudança relativa:

$$n = \frac{T_{Exec_Y} - T_{Exec_X}}{T_{Exec_X}} \cdot 100\% \tag{15}$$

Portanto, justifica-se o uso de métricas, baseadas na quantidade de operações computacionais por unidade de tempo, para medir tal performance, como MIPS (*Millions of Instructions Per Second*, Milhões de Instruções por Segundo), e MFLOPS (*Millions of Float Point Operations Per Second*, Milhões de Operações de Ponto Flutuante por Segundo) (LILJA, 2005):

$$MIPS = \frac{n_i}{10^6 \cdot t_i} \tag{16}$$

$$MFLOPS = \frac{n_f}{10^6 \cdot t_f} \tag{17}$$

em que t_i é o tempo necessário para executar n_i instruções e t_f é o tempo necessário para executar n_f operações de ponto flutuante.

Lilja (2005) destaca também uma metodologia padronizada para mensurar a performance de um sistema computacional, chamada *System Performance Evaluation Cooperative* (SPEC). Tal método consiste em:

i) Medir o tempo de execução de cada programa no conjunto de testes para o sistema a ser testado;
ii) Dividir o tempo medido em cada programa no passo anterior pelo tempo de execução em uma máquina-base escolhida, visando normalizar os tempos de execução;

iii) Calcular a média de todos os tempos de execução normalizados usando uma média geométrica para obter uma métrica de performance.

A definição da média geométrica pode ser observada da Equação (18) (HENNESSY; PATTERSON, 2011):

$$\bar{T_m} = \sqrt[n]{\prod_{i=1}^{n} t_i} \tag{18}$$

em que t_i representa cada uma das n amostras e T_m a média geométrica dessas.

Segundo Jain (1990), a média geométrica é mais utilizada em *benchmarks* do que a média aritmética por sua característica de ser menos afetada por valores discrepantes no conjunto de dados amostrados.

2.4.2 BENCHMARKS

Segundo Lilja (2005), *benchmarks* são métodos de executar um programa computacional de forma que permita-se medir a eficiência do mesmo em um determinado *hardware*. Porém, existem dificuldades durante a medição de tal desempenho. Primeiramente, deve-se esperar que apenas um programa esteja sendo executado no sistema durante o teste, ou que os demais programas não afetem consideravelmente o desempenho. A partir de tal suposição, pode-se dizer o quão bem a maquina que está sendo avaliada pode executar tal aplicação.

Define-se alguns tipos clássicos de *benchmarks* simples (LILJA, 2005; HENNESSY; PATTERSON, 2011):

a) *Benchmarks Kernel:* São códigos pequenos, projetados para testar o desempenho de um bloco unitário do sistema, ou seja, uma parte pequena do sistema completo que desempenha uma função especifica.
b) *Microbenchmarks:* Algoritmos clássicos de até 100 linhas, como o algoritmo de ordenação *quicksort*, que tem uma complexidade computacional conhecida. *Microbenchmars* tornaram-se populares por permitir que um programa computacional seja inteiramente executado em um processador simulado (LILJA, 2005).
c) *Benchmarks sintéticos:* São programas de aplicações artificiais que simulam a aplicação real, porém não executam o trabalho que e efetivamente executado pela aplicação real.
d) *Benchmarks de Aplicação:* É o efetivamente o programa que efetuará a ação real do sistema. Segundo Hennessy e Patterson (2011) são a melhor maneira de testar o desempenho do sistema.

É dito que o melhor *benchmark* para um determinado sistema e a própria aplicação. Portanto, neste trabalho será utilizada uma metodologia que torne possível analisar separadamente os fatores que determinam o desempenho de um microcontrolador para a aplicação em questão (LILJA, 2005).

Weicker (1990) faz um estudo detalhado sobre os *benchmarks* mais comuns, suas características e quando utiliza-los, propondo que a principal justificativa do uso de *benchmarks* em computadores é o aumento de performance ao desempenhar as tarefas do sistema. Entre os *benchmarks* escolhidos, são citados:

a) *Wheatstone: Benchmarks* de *Wheatstone* tem um grande número de dados e operações usando ponto flutuante, assim, grande parte do desempenho do sistema é utilizado por funções de bibliotecas matemáticas e suas operações.
b) *Linpack:* Desenvolvido originalmente como um pacote de bibliotecas para efetuar operações relacionadas a álgebra linear. Por tornar-se um *benchmark* numérico, é esperado que *Linpack* realize uma grande quantidade de operações com ponto flutuante.
c) *DhryStrone:* É um *benchmark* sintético, que foi originalmente desenvolvido para linguagens de programação com características predominantemente não-numéricas. Sendo assim, contém menos *loops* e operações computacionais mais simples, em contrapartida, contém mais *if* operações e chamadas de função.

Hennessy e Patterson (2011) sugere ainda que uma forma de melhorar o desempenho de um *benchmark* é fazer uso de *flags* do compilador, que em muitos *benchmarks* podem reduzir sua performance, ou não podem ser desempenhadas em outros. Assim, deve-se usar o mesmo compilador, a mesma linguagem de programação e as mesmas *flags* de compilação, e evitar modificações no código, para testar o desempenho de um sistema usando um *benchmark*.

2.4.3 TRABALHOS RELACIONADOS À MENSURAÇÃO DE DESEMPENHO

Existem vários métodos que podem ser utilizados para fazer a medição do tempo de atuação do microcontrolador em um sistema de controle.

Lilja (2005) diz que a forma mais simples de mensurar o tempo de um sistema computacional é análogo a usar um cronômetro para medir o tempo de algum evento qualquer, com a diferença que um cronômetro inicia com tempo inicial $t_0 = 0$, enquanto um sistema computacional tem um *clock* interno, que faz a contagem de *ticks* de *clock* que ocorreram desde que o sistema foi iniciado. O tempo total necessário para executar tal evento é dado pelo número de *ticks* contados multiplicado pelo período dos *ticks*.

Em sua tese, Vargas (2017) faz uso de um método similar para obter os resultados desejados para os modelos de microcontroladores TivaC TM4C123G, MSP30G2553 e Freescale MCF51JE256, que podem ser observados na Tabela 1.

A frequência máxima de amostragem observada na Tabela 1 pode ser calculada a partir da Equação (19).

Como citado anteriormente na Seção 2.2, a ação de controle pode ser dividida em três unidades, sendo elas: Aquisição do valor de entrada no transdutor através do ADC para obtenção do erro, processamento dos dados de entrada para efetivar a ação de controle e Devolução do valor de saída para o atuador através do DAC. Deve-se ainda esperar um pequeno *overhead* entre as chamadas de função e entradas de interrupção.

Tabela 1 – Resultados do Teste Efetuado por Vargas (2017).

	TM4C123G1	MCF51JE256	MSP430G2553
Tempo de Execução PID (μs)	4,874	8,960	36,930
Frequência de amostragem máxima (kHz)	205,170	111,607	27,078
Tempo de assentamento possível (μs)	48,740	89,600	369,300

Fonte: Vargas (2017)

A metodologia empregada consiste em utilizar um pino de uma porta GPIO para saber quando efetivamente o microcontrolador está efetuando as operações de cálculo da ação de controle. Ou seja, o pino da porta GPIO escolhido tem sua saída escrita em nível alto imediatamente antes de iniciar-se a conversão pelo módulo ADC, e escrito novamente em nível baixo logo após o sinal de controle ser escrito no módulo PWM.

Assim, enquanto o microcontrolador estiver efetuando a operação de cálculo da ação de controle, o pino escolhido estará em nível alto, podendo ser amostrado por um osciloscópio digital ou similar, para obter o tempo necessário da operação completa.

Após a obtenção e normalização dos resultados em relação a diferentes *clocks* dos quais os dados dependem, é feita uma análise em que foram obtidas, para cada microcontrolador, informações como: frequência máxima de operação, resolução máxima do ADC para evitar ciclo limite, e resolução efetiva módulo de PWM.

Para obter tais informações, (VARGAS, 2017) define também a frequência máxima de operação $f_{sa.max}$ como o inverso do WCET (*Worst-Case Execution Time*, Tempo no Pior Caso de Execução), como observa-se na Equação (19):

$$f_{sa} \;=\; \frac{1}{WCET} \tag{19}$$

Além disso, define-se uma faixa de valores em que é possível efetuar a amostragem, sendo $f_{sa.min} \;\leq f_{sa} \;\leq\; f_{sa.max}$, na qual a frequência miníma de amostragem pode ser obtida a partir do tempo de assentamento do sistema, como observa-se na Equação (20):

$$T_{sa.max} \;\leq\; \frac{t_{ss}}{10}, \qquad\qquad f_{sa.min} \;\geq\; \frac{10}{t_{ss}} \tag{20}$$

em que t_{ss} é o tempo de assentamento do sistema.

Além dos critérios de desempenho, são definidos o período, número de *bits* e a resolução efetivos do módulo PWM, como mostrado na Equação (21), Equação (22) e Equação (23):

$$T_{ef.PWM} \;=\; trunc\Big(\frac{f_{clk.PWM}}{f_{PWM}}\Big) \tag{21}$$

$$n_{bits.PWM} \;=\; \log_2\left(T_{ef.PWM}\right) \;=\; \log_2\left(trunc\Big(\frac{f_{clk.PWM}}{f_{PWM}}\Big)\right) \tag{22}$$

$$res_{ef.PWM\%} = \frac{1}{T_{ef.PWM}} \cdot 100\% \quad (23)$$

em que T_{ef} é o período efetivo do módulo PWM, f_{cPWM} é a frequência que alimenta o módulo PWM, f_{PWM} frequência modulada pelo módulo DPWM, f_{efPWM} é a frequência efetiva do módulo PWM e $res_{efPWM\%}$ é a resolução efetiva do módulo PWM.

Kramer, Stolze e Banse (2009) faz um estudo de caso sobre o uso de diferentes módulos de *benchmarks* para fazer a melhor escolha do microcontrolador para desempenhar as funções do projeto. Entre os módulos de teste usados, são escolhidos 8 aspectos de desempenho implementados pelos códigos de *benchmarks*:

i) Algoritmos matemáticos com operações de ponto fixo.
ii) Algoritmos matemáticos com operações de ponto flutuante.
iii) Operações lógicas.
iv) Operações de controle digital.
v) Operações com *loops* e desvio do fluxo de código.
vi) Operações com cálculos polinomiais.
vii) Operações com transformadas rápidas de *Fourier*.

Os testes foram realizados com microcontroladores Zilog Z8Encore Z8F6423, Texas Instruments MSP430F449, Zilog ZNeo Z16F2811, Infineon XC167ci, NXP LPC2138, Infineon Tricore TC1796, normalizando o tempo de amostragem em relação a frequência de operação de cada microcontrolador, como citado anteriormente, para obter-se uma medida de desempenho dos microcontroladores, e não do tempo de operação.

3 DESENVOLVIMENTO DOS TESTES

Este capítulo descreve os métodos utilizados nesse trabalho para a coleta e análise de dados, bem como o desenvolvimento dos algoritmos de controle escolhido e a planta em que serão aplicados os testes. Também serão descritos e justificados os materiais utilizados nos testes.

Dando continuidade à metodologia de Vargas (2017) apresentada na Subseção 2.4.3, este capítulo tem o objetivo utilizar o mesmo princípio de análise para comparar o desempenho, expandindo o estudo para mais microcontroladores, que serão propostos na Seção 3.1, verificando a viabilidade de seu uso em aplicações de sistemas de controle.

Para tal, deve-se:

i) Especificar os microcontroladores e os materiais utilizados.
ii) Desenvolver uma planta de controle para o controlador, visando tornar o teste mais próximo de uma aplicação real.
iii) Desenvolver um algoritmo de controle genérico, de forma que respeite os requisitos citados por Hennessy e Patterson (2011) na Subseção 2.4.2, ou seja, evitar o uso de versões diferentes do compilador, linguagens de programação e modificações no código, visando melhorar a qualidade da comparação do desempenho de *hardware* dos sistemas.
iv) Desenvolver algoritmos genéricos para efetuar operações numéricas no formato Q_n, fundamentadas na Subseção 2.3.3.
v) Descrever a metodologia utilizada para aquisição e representação de dados.
vi) Efetuar a coleta e tabulação dos dados.

3.1 MATERIAIS

Tendo especificado o procedimento básico usado na metodologia, foram escolhidos os materiais necessários que serão utilizados durante o processo de pesquisa do trabalho, entre eles:

a) Osciloscópio Digital.
b) Microcontrolador STM32F103C8T6.
c) Microcontrolador STM32F407VET6.
d) ST-Link v2 - para microcontroladores ST.
e) Gerador de código STM32CubeMX para STM32.
f) IDE Eclipse com compilador GCC ARM-32 para STM32.
g) Microcontrolador ESP32CP2102.
h) Conector Micro-USB.
i) Compilador Xtensa GCC Versão 5.2.0 para ESP32.
j) Microcontrolador MSP430G2553.

k) IDE Code Composer Studio para microcontroladores Texas.
l) Conector Mini-USB.
m) Compilador GCC Versão 5.2.0 para MSP430.

3.1.1 ESPECIFICAÇÃO DOS MICROCONTROLADORES

Com a rápida incorporação da arquitetura ARM no mercado, microcontroladores com núcleos baseados na arquitetura 32-bit tornaram-se cada vez mais presentes na engenharia. A linha de microcontroladores STM32, tem sido utilizada em larga escala nos últimos tempos graças a sua relativamente alta capacidade e baixo custo, além de facilitar o desenvolvimento de projetos em seu ambiente com sua HAL (*Hardware Abstraction Layer*, Camada de Abstração de *Hardware*) consideravelmente intuitiva.

Tendo tais vantagens em vista, um dos microcontroladores escolhidos nesse trabalho é da linha STM32F103, que possuí divisão via *hardware* e multiplicação em um único ciclo de *clock*, porém não possui FPU, como citado na Subseção 2.3.2.

Foi escolhido também um microcontrolador da mesma família, STM32F407VET6, por possuir características superiores com relação a linha F1, MCU (*Memory Control Unit*), *timers* de 32-*bits*, DAC, entre outros periféricos. Esse microcontrolador possuí um processador ARM® Cortex®-M4 com FPU e Acelerador ART (*Adaptive Real-Time Memory Accelerator*), que segundo STMicroelectronics (2016), pode aumentar sua performance atingindo 210 DMIPS. Baseado no *benchmark* CoreMark, esse desempenho equivale a um *software* executando na Flash com *0-wait-state*, estado no qual o processador não precisa esperar para acessar a memória. Vale ressaltar que esse microcontrolador suporta instruções DSP (*Digital Signal Processor*) em sua arquitetura, porém tal análise encontra-se fora do escopo desse trabalho. No caso de uma análise utilizando um algoritmo mais complexo, como um filtro digital ou similar, onde as instruções DSP são realmente úteis, tal análise não deve ser desconsiderada.

Outra linha de microcontrolador que está sendo cada vez mais utilizada no mercado é a ESP32D0WD, da *Espressif Systems*, sucessor da linha ESP8266, que já vem sendo utilizada em escala considerável em pesquisas que envolvem IoT (*Internet of Thinks*, Internet das Coisas). Com seus módulos *Wireless* e *Bluetooth* de 2,4GHz, possuindo ainda dois núcleos munidos com FPU, o ESP32 demonstra potencial para desenvolver novas aplicações. Em contrapartida, o microcontrolador ESP32 conta com a interface de desenvolvimento ESP-IDF (*Espressif IoT Development Framework*), que se trata de um *framework* completo para desenvolvimento de aplicações IoT, no qual o uso de um sistema de RTOS (*Real Time Operating Systems*) é intrinsecamente implementado, e os mecanismos utilizados pela interface ESP-IDF podem afetar o desempenho do microcontrolador.

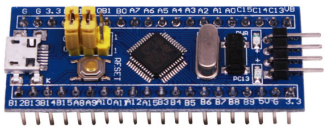
Por último, foi também escolhido o microcontrolador MSP430G2553, que foi projetado para aplicações com foco em baixo consumo de energia, tendo assim características inferiores em relação a outros microcontroladores. Algumas características que impactam na análise desse microcontrolador são: arquitetura do processador de 16-*bits*; Resolução do ADC de 10-*bits*;

Resolução máxima do *timer* de 16-*bits*; Frequência máxima de *clock* da CPU de 16MHz.

Os microcontroladores utilizados nesse trabalho podem ser observados na Figura 10. Na Tabela 2 é feito um comparativo entre as características dos microcontroladores.

(a) ESP32CP2102

(b) MSP430G2553



(c) STM32F103C8T6

(d) STM32F407VET6

Figura 10 – Microcontroladores Utilizados

3.2 DESENVOLVIMENTO DO CÓDIGO-FONTE PARA O CONTROLE PID

Para efetuar os testes propostos no controlador digital, torna-se necessário o desenvolvimento do controlador. Utilizando os conceitos descritos na Subseção 2.2.4, assim como o Algoritmo 1 e o Algoritmo 2, foi projetado um código-fonte genérico em linguagem de programação C. O fluxograma da Figura 11 ilustra o funcionamento do algoritmo. Observa-se que a metodologia consiste em utilizar duas saídas digitais para mensurar tanto o tempo de execução total quanto o tempo de execução das operações aritméticas da CPU, como será discutido na Seção 3.5.

O código-fonte foi desenvolvido na forma de uma biblioteca, ou seja, um par de arquivos *pidlib.h* e *pidlib.c*, os quais contém as declarações das estruturas de dados necessárias e os protótipos das funções utilizadas, e as definições das funções, respectivamente.

Tabela 2 – Características dos Microcontroladores Utilizados

Microcontrolador	STM32F103C8T6	ESP32D0WDQ6
Fabricante	STMicroelectronics	Espressif Systems
Arquitetura da CPU	ARM® Cortex®-M3 32-bits	Xtensa® LX6 Dual-Core 32-bits
Clock da CPU (Máximo)	72 MHz	240 MHz
Cristal Oscilador	4 a 16 MHz	2 MHz a 60 MHz 32 kHz (RTC *Clock*)
Resolução do ADC	12-bits	12-bits
Taxa de Amostragem do ADC	1 Msps	2 Msps
Resolução do PWM	16-bits	16-bits
Unidade de Ponto Flutuante	Não	Sim
IDE Utilizada	SW4STM32 (Eclipse)	Code Composer Studio (V. 8.0)
Compilador Utilizado	arm-none-eabi-gcc (V. 7.2.1)	xtensa-esp32-elf-gcc (V. 5.2.0)
Nível de Otimização	Desativado (-O0)	Otimização de Depuração (-Og)
Microcontrolador	**STM32F407VET6**	**MSP430G2553**
Fabricante	STMicroelectronics	Texas Instruments
Arquitetura da CPU	ARM® Cortex®-M4 32-bits	MSP430 RISC Core 16-bits
Clock da CPU (Máximo)	168 MHz	16 MHz
Cristal Oscilador	8 MHz 32 kHz (RTC)	32 kHz
Resolução máxima do ADC	12-bits	10-bits
Taxa de Amostragem do ADC	7.2 Msps	200 ksps
Resolução do PWM	32-bits	16-bits
Unidade de Ponto Flutuante	Sim	Não
IDE Utilizada	SW4STM32 (Eclipse)	Code Composer Studio (V. 8.0)
Compilador Utilizado	arm-none-eabi-gcc (V. 7.2.1)	msp430-gcc (V. 4.7.2)
Nível de Otimização	Desativado (-O0)	Desativado (-O0)

Fonte: STMicroelectronics (2015), STMicroelectronics (2016), Espressif Systems (2018), Texas Instruments (2010)

No desenvolvimento do código-fonte do arquivo de cabeçalho da biblioteca, *pid.h*, encontra-se a definição de uma estrutura genérica para manipulação do controlador, utilizando a nomenclatura *pid_const_t*. Define-se também um *alias* para o tipo de dado usado nas constantes K_a, K_b e K_c, e nas variáveis que serão operadas, definidos na Subseção 2.2.3.

O arquivo de cabeçalho *pid.h* declara também protótipos de função, *pidLoopRoutine()* e *pidInterruptRoutine()*, utilizados para efetuar a ação de controle periodicamente, utilizando uma interrupção de *timer*.

O arquivo *pid.c* encontram-se a definição das funções descritas acima. Ambas as funções foram definidas com o modificador *inline*, para que as instruções executadas na função sejam armazenados junto da função, eliminando o *overhead* durante a chamada da mesma. Opcionalmente, pode-se adicionar um modificador caso seja necessário executar as funções na memória RAM.

Para o funcionamento da biblioteca, é necessário definir alguns macros de configuração, que podem variar entre diferentes microcontroladores e bibliotecas utilizadas. Os macros

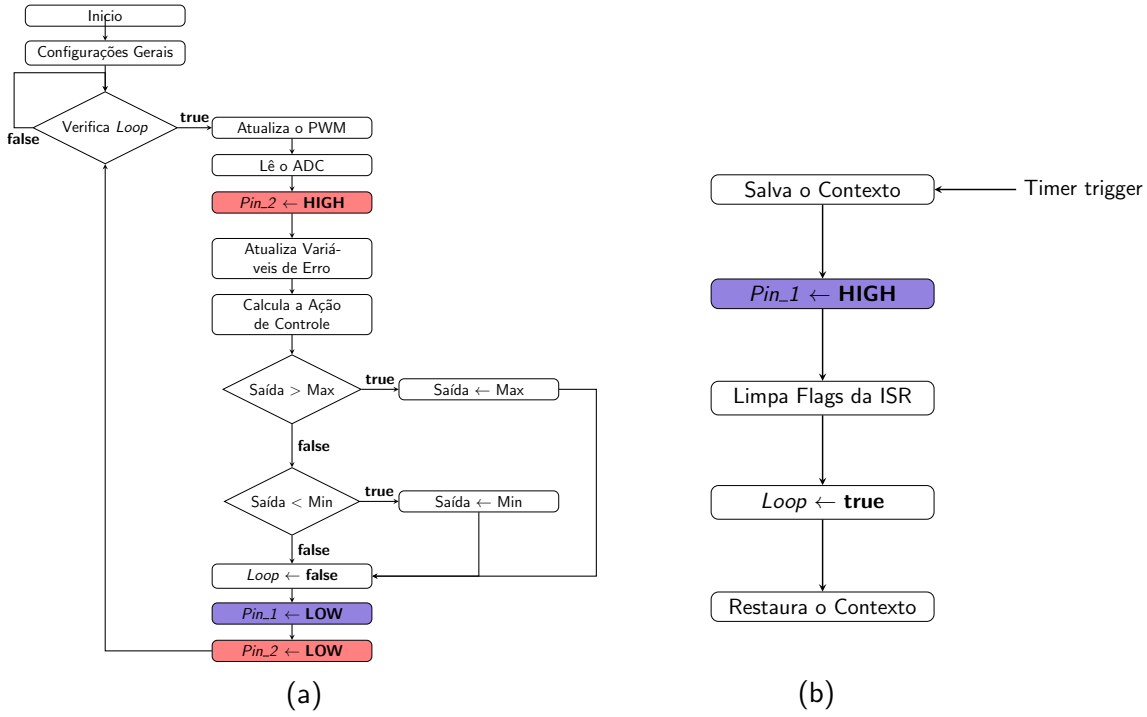


Figura 11 – Fluxograma de Execução do algoritmo de Controle PID

definidos estão descritos no Quadro 2.

Macro	**Descrição**
PID_ADC_READ( )	Efetua a ação de leitura do ADC.
PID_PWM_WRITE( )	Efetua a ação de escrita no módulo PWM.
SET_TEST_PIN	Liga o pino 1, utilizado para efetuar os testes.
RESET_TEST_PIN	Desliga o pino 1, utilizado para efetuar os testes.
SET_TEST_PIN_CALC	Liga o pino 2, utilizado para efetuar os testes.
RESET_TEST_PIN_CALC	Desliga o pino 2, utilizado para efetuar os testes.
CLEAR_TIMER_FLAGS	Limpa as *flags* de interrupção do *timer*, caso necessário.

Quadro 2 – Macros Utilizados na Biblioteca

O Quadro 3 descreve macros opcionais, utilizados para modificar o funcionamento do algoritmo, permitindo efetuar testes diferenciados.

Isso foi feito para facilitar a alteração do código no caso de haver interesse em modificar a operação que será utilizada nos cálculos, por exemplo alterar o tipo de dado de *double* para *float*. Tal feito foi desenvolvido com o intuito de facilitar testes de desempenho futuros utilizando múltiplos tipos de dados nas operações. O Apêndice A demonstra a implementação do código-fonte genérico desenvolvido.

Macro	**Descrição**
Q7_OP	Define *pid_const_t* como *q7_t* para efetuar as operações algébricas.
Q15_OP	Define *pid_const_t* como *q15_t* para efetuar as operações algébricas.
FLOAT_OP	Define *pid_const_t* como *float* para efetuar as operações algébricas.
DOUBLE_OP	Define *pid_const_t* como *double* para efetuar as operações algébricas.
RAM_ATTRIBUTE	Define o atributo necessário para executar os procedimentos na memória RAM.
DIV_ACT	Executa uma quantidade definida de divisões antes de efetuar as operações algébricas.
EXCLUDE_PERIPH_OP	Inibe as operações efetuadas pelos periféricos, a fim de executar somente as operações algébricas.
EXCLUDE_CALC_OP	Inibe as operações algébricas, a fim de executar somente as operações efetuadas pelos periféricos.

Quadro 3 – Macros Opcionais Utilizados para Modificar as Operações

3.3 DESENVOLVIMENTO DO CÓDIGO-FONTE PARA MANIPULAÇÃO DE NÚMEROS EM FORMATO Qn

O código desenvolvido na Seção 3.2 pode também operar utilizando os conceitos de números em formato Q_n comentados na Subseção 2.3.3, podendo optimizar a performance de operação do processador do microcontrolador durante as operações de cálculo do controlador. Para tal, foram desenvolvidos códigos genéricos para operar números fracionários em formato Q_7 e Q_{15}.

Um número h representado em ponto flutuante pode ser convertido para o formato Q_n multiplicando-o por 2^n, e um número representado em formato Q_n pode ser convertido para um número representado em ponto flutuante multiplicando-o por 2^{-n}. Computacionalmente, isso pode ser facilmente implementado fazendo uso de deslocamento de *bits*.

Algoritmo 3: Conversão de um Número em Formato de Ponto Flutuante para o Formato Q_n

Input: Número de bits n, Número em Ponto Flutuante F
Output: Número em Formato Q_n F
$Q \leftarrow F \cdot 2^n$

Algoritmo 4: Conversão de Número no Formato Q_n para Ponto Flutuante

Input: Número de bits n, Número Q
Output: Número em Ponto Flutuante F
$F \leftarrow Q \cdot 2^{-n}$

Para efetuar operações com números em formato Q_n torna-se necessário tomar algumas precauções. No caso da soma, é possível a ocorrência de o resultado ser maior que o valor em ponto flutuante máximo, representado em formato Q_n por $2^n - 1$. Como isso não seria aceitável, saturamos o resultado da operação como sendo seu valor máximo. O mesmo ocorre para valores negativos, saturando o resultado em seu valor mínimo. O Algoritmo 5 representa

como efetuar a soma de dois números em formato Q_n. O algoritmo de subtração não foi implementado, uma vez que esse é um caso particular do algoritmo de soma.

Algoritmo 5: Soma de Dois Números em Formato Q_n

Input: Número de bits n, Número $Q1$, Número $Q2$

Output: Número em Formato Q_n QR

$Q_{2n}(A) \leftarrow Q1 + Q2$

if $A \geq 2^n - 1$ **then**

| $A \leftarrow 2^n - 1$

end

else if $A \leq 2^n$ **then**

| $A \leftarrow 2^n$

end

$QR \leftarrow Q_n(A)$

No caso da operação de multiplicação de dois números em formato Q_n, o resultado será um valor em formato $Q_{2(n+1)-1}$, o que também não é aceitável. Para tanto, pode-se descartar os n *bits* menos significativos, efetuando uma divisão por 2^n fazendo uso de deslocamento de *bits*, como comentado anteriormente. Deve-se perceber que quando isso ocorre, considerando uma arquitetura com barramento de dados de m *bits*, o formato Q_n não deve ultrapassar $\frac{n}{2} - 1$. Se isso ocorrer, não haverá *bits* suficiente nos registradores que efetuam a operação de multiplicação, levando a um resultado incorreto na operação. O Algoritmo 6 ilustra como a multiplicação de dois números em formato Q_n pode ser efetuada.

Algoritmo 6: Multiplicação de Dois Números em Formato Q_n

Input: Número de bits n, Número $Q1$, Número $Q2$

Output: Número em Formato Q_n QR

$Q_{2n}(A) \leftarrow Q1 * Q2$

$A \leftarrow A \cdot 2^{-n}$

$QR \leftarrow Q_n(A)$

Durante a operação de divisão, ocorre o oposto do problema anterior: a resolução é perdida durante a operação, e o resultado estaria formatado em uma base Q_n com menos *bits*. Para contornar esse problema, pode-se também fazer o oposto da solução anterior: multiplicar o numerador por 2^n, fazendo novamente uso de deslocamento de *bits*, e finalmente efetuar a divisão pelo denominador. O Algoritmo 7 representa como dividir dois números em formato Q_n. Os algoritmos foram desenvolvidos em linguagem de programação C e encontram-se no Apêndice B.

Algoritmo 7: Divisão de Dois Números em Formato Q_n

Input: Número de bits n, Número $Q1$, Número $Q2$

Output: Número em Formato Q_n QR

$Q_{2n}(A) \leftarrow Q1 \cdot 2^n$

$QR \leftarrow \frac{Q1}{Q2}$

3.4 EFEITOS DE QUANTIZAÇÃO NUMÉRICA

Como fundamentado nas Seções 2.3.2 e 2.3.3, o formato numérico utilizado para representar os dados quantizados e as constantes referentes ao controlador tem um impacto na resposta do sistema. Esta seção tem como objetivo analisar tal impacto, tanto em termos de desempenho quanto em termos de resolução dos formatos numéricos.

Como definido na Subseção 2.3.2, um número em formato *float* é definido pelo padrão IEEE-754 contendo 23-*bits* para números fracionários, enquanto um número em formato *double* contém 52-*bits*. Portanto, o menor número decimal em formato *float* e em formato *double* podem representar é $119 \cdot 10^{-9}$ e $222 \cdot 10^{-18}$, respectivamente.

Para números definidos em formato de ponto fixo Q_7 e Q_{15} implementados no Apêndice B, a resolução máxima que pode ser utilizada é de $781 \cdot 10^{-5}$ e $305 \cdot 10^{-7}$, respectivamente.

Além disso, para possibilitar o uso dos periféricos sem perda de resolução, deve-se considerar a quantidade de *bits* efetivos do ADC e do modulador PWM. Portanto, para que os cálculos sejam efetuados sem que haja perda de resolução da leitura do ADC, o numero de *bits* desse deve ser igual ou menor ao número de *bits* que representa um passo efetivo na resolução.

$$2^{n_{bits.ADC}} \leq 2^n, \qquad n_{bits.ADC} \leq n \tag{24}$$

O mesmo ocorre considerando as Equações 21 e 22, nas quais define-se o período efetivo do módulo PWM. Para que seja possível aproveitar a resolução total definida no módulo PWM, é necessário que o número de *bits* efetivos não ultrapasse o número de *bits* que representa a resolução.

$$T_{ef.PWM} \leq 2^n, \qquad n_{bits.PWM} \leq n \tag{25}$$

Porém, ainda é possível utilizar um artifício matemático para contornar o problema da Equação (24). Utilizando deslocamento de *bits* para a direita, pode-se reduzir a resolução do ADC, obtendo-se n uma resolução de *bits*, assim como o formato numérico utilizado. O mesmo pode ser implementado para a Equação (25), utilizando descolamento de *bits* para a esquerda, fazendo com que a ação de controle tenha a mesma quantidade de *bits* efetivos do módulo PWM. Deve-se considerar que para evitar oscilações no ciclo limite, descritas na Subseção 2.2.4, a resolução do ADC deve ser menor em relação a resolução do modulo PWM, ou seja:

$$n_{bits.ADC} < n_{bits.PWM} \tag{26}$$

Assim, a máxima resolução que pode ser admitida no ADC para cada microcontrolador, utilizando cada formato numérico, pode ser observada na Tabela 3.

Portanto, o maior erro de quantização que pode ocorrer é dado pela equação Equação (3), sendo metade da resolução observada na Tabela 3 para cada caso. Além disso, o erro de quantização de leitura do ADC será propagado para as operações aritméticas, que, como

Tabela 3 – Resolução máxima admitida pelo ADC para cada microcontrolador em relação a cada tipo de dado

		Q_7	Q_{15}, *Float* e *Double*
STM32F103R8T6	(mV)	25,780	0,8100
	(%)	0,781	0,0978
STM32F407VET6	(mV)	25,780	0,8100
	(%)	0,781	0,0244
MSP430G2553	(mV)	25,780	3,2200
	(%)	0,781	0,0980
ESP32D0WDQ6	(mV)	25,780	0,8100
	(%)	0,781	0,0244

comentado na Subseção 2.3.3, os ganhos K_a, K_b e K_c também podem estar sujeitos a efeitos de quantização, propagando novamente o erro para a ação de controle.

3.4.1 DESENVOLVIMENTO DE UMA PLANTA DE CONTROLE GENÉRICA PARA EFETUAR TESTES DE DESEMPENHO

Para efetuar os testes propostos anteriormente, foi projetada uma planta de controle simples capaz de responder à diferentes características do controlador, que são causadas pelas diferenças entre os microcontroladores escolhidos. Assim, define-se a planta como um circuito eletrônico *RLC*, com componentes que evitem o uso de circuitos auxiliares, como *drivers* e fontes externas.

Os valores dos componentes foram escolhidos baseados nos modelos comerciais disponíveis e nas características de resposta do circuito de alimentação, que nesse caso é a corrente fornecida pela saída digital do microcontrolador com menos capacidade de alimentação. Na Tabela 4 pode-se observar que o microcontrolador com menor capacidade de fornecimento de corrente através da saída digital é o *MSP430G2553*. Observa-se que a máxima corrente de saída do microcontrolador ESP32D0WDQ6 refere-se à corrente total da porta, e não à corrente dos pinos individuais (ESPRESSIF SYSTEMS, 2018).

Tabela 4 – Máxima Corrente de Saída das portas IO dos Microcontroladores - Componentes Utilizados na Planta

Microcontrolador	$I_{o\ max}$
MSP430G2553	5,0μA
STM32F103C8T6	25,0μA
STM32F407VET6	25,0μA
ESP32D0WDQ6	1,2A *

	Valor	Unidade
V_{in}	3,3	V
R	974,8	Ω
L	51,6μ	H
C	210μ	F

Fonte: STMicroelectronics (2015), Espressif Systems (2018), STMicroelectronics (2016), Texas Instruments (2010)

Observa-se na Figura 12 o diagrama eletrônico da planta desenvolvida.

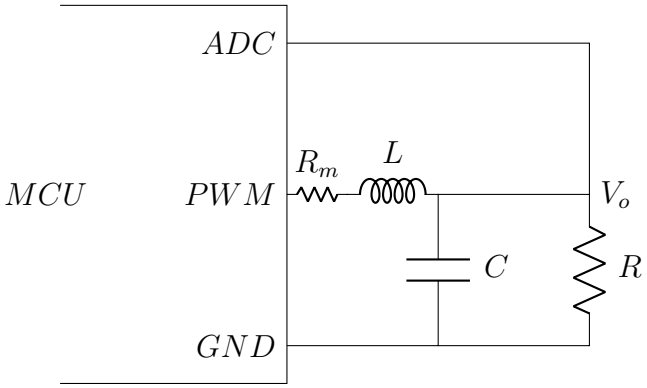
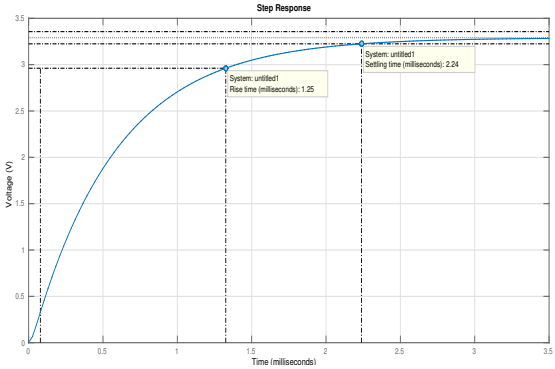


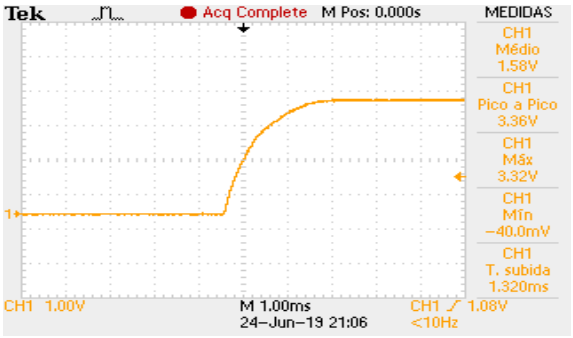
Figura 12 – Circuito RLC

Em que R_m representa um ajuste efetuado experimentalmente, a fim de adequar a resposta com o com valor de máximo de corrente fornecido pela porta IO do microcontrolador durante o regime transitório, equivalendo a aproximadamente 2,8 Ω.

A planta foi simulada com o auxílio do *software* MATLAB, obtendo-se a resposta observada na Figura 13a. Observa-se também que o tempo de assentamento da planta é de 119ms, obtendo-se uma grande quantidade de oscilações quando o sistema não é compensado. Na Figura 13b observa-se a resposta da planta implementada em malha aberta não compensada.



(a) Simulação

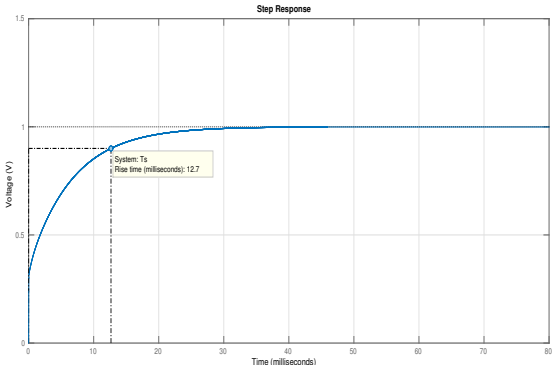


(b) Implementação

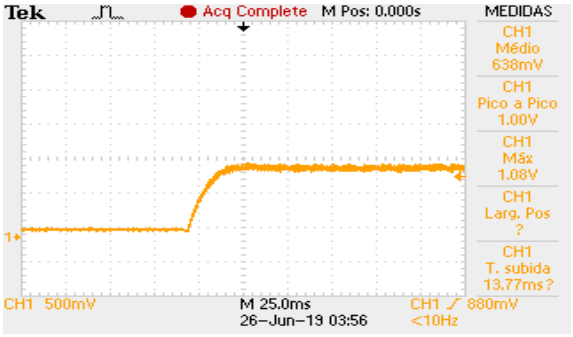
Figura 13 – Resposta da Planta em Malha Aberta sem Compensação

O controlador foi projetado com auxílio da ferramenta *PID-Tunner*, obtendo-se $K_p = 0{,}2008$, $K_i = 68{,}87$ e $K_d = 7{,}85 \cdot 10^{-5}$, respectivamente. Utilizando as equações 7, 8 e 9, obteve-se $K_a = 0{,}6002$, $K_b = -0{,}9789$ e $K_c = 0{,}3925$. O resultado obtido utilizando o microcontrolador ESP32D0WDQ6 podem ser observados na Figura 14b, utilizando os parâmetros $f_{pwm} = 5kHz$ e $f_{sa} = 5kHz$.

Assume-se que a resposta transitória do sistema pode ser afetada por características individuais de cada microcontrolador, tal como a corrente máxima de saída das portas IO, porém esses efeitos não foram analisados na prática. Portanto, durante a realização dos testes, considera-se apenas a estabilidade do sistema, observando a resposta em regime permanente.



(a) Simulação



(b) Implementação - ESP32D0WDQ6

Figura 14 – Resposta da Planta em Malha Fechada

3.5 METODOLOGIA DE TESTES

Como descrito na Seção 2.1, existem incontáveis aplicações de sistemas de controle. Para tornar possível a coleta e analise dos dados neste trabalho, observa-se que um requisito importante para tal é o projeto de uma planta de testes. A planta de aplicação na qual a coleta de dados é efetuada não é de essencial importância para o desenvolvimento desse trabalho, uma vez que um dos objetivos da metodologia utilizada é ser utilizada em diferentes aplicações de controle.

Existem inúmeras topologias de controladores que podem efetuar a ação de controle para o teste escolhido. Entre elas, foi escolhida a topologia de controle PID. A metodologia utilizada no presente trabalho consiste em fazer uso de um osciloscópio para mensurar o tempo necessário pelo sistema computacional efetuar a ação de controle sobre a planta. Na Figura 15 o método de medição é ilustrado.

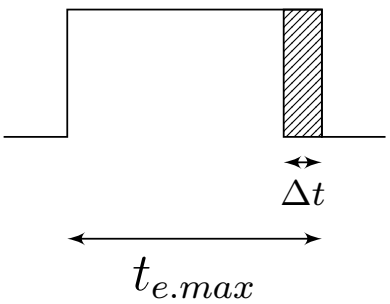


Figura 15 – Representação do Sinal de Tempo de Execução Total do Algoritmo

Assim, pode-se obter a máxima frequência de amostragem do microcontrolador usando a Equação (27), definida neste trabalho como $f_{sa.max}$.

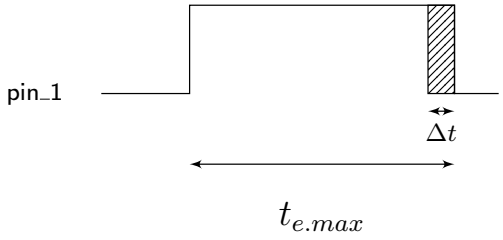
$$f_{sa.max} = \frac{1}{t_{e.max}} \tag{27}$$

em que $t_{e.max}$ representa o tempo total de execução no pior caso, assim como na Equação (19).

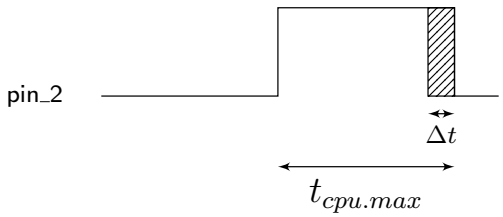
Porém, para efetuar análises mais elaboradas nessa categoria de sistemas, é necessário aprofundar tal metodologia. Para tal, fez-se necessário separar a análise de uso do microcontrolador entre uso do processador e uso dos periféricos, nesse caso, conversor AD, gerador de pulsos e o tempo de saída da interrupção do *timer*.

Para efetuar de fato essas análises separadamente, foram utilizados não um pino das portas GPIO como citado anteriormente, mas sim dois pinos: o primeiro é configurado em nível alto assim que ocorre a interrupção do *timer*; o segundo pino é configurado para nível alto logo após a conversão do ADC ter sido efetuado; após a operação do algoritmo ser concluída, ambos os pinos são configurados para nível baixo.

Na Figura 16 exemplifica-se o método de medição, enquanto a Figura 11 representa o fluxograma do algoritmo apresentado no Algoritmo 1 e Algoritmo 2, adicionando as operações necessárias para efetuar os testes citadas anteriormente.



(a) Tempo de Execução Total



(b) Tempo de Execução da CPU

Figura 16 – Representação do Sinal de Tempo de Execução do Algoritmo

É possível ainda fazer uso de algumas ferramentas disponíveis no osciloscópio digital para facilitar a visualização e aquisição dos dados, tal como:

i) Utilizar o canal matemático para efetuar a soma dos dois canais, resultado em um terceiro formato de onda.
ii) Utilizar a ferramenta cursor para mensurar o tempo máximo com maior precisão.
iii) Utilizar a ferramenta persistência para melhor visualização da variação de tempo.

3.5.1 CONVERSÃO DO ADC

Como fundamentado na Subseção 2.2.1, o ADC dos microcontroladores tem um impacto significativo no desempenho dos microcontroladores para a aplicação de controle PID utilizada nesse trabalho. Para reduzir o impacto desse problema, o algoritmo de aquisição do

ADC foi padronizado, a fim de obter-se o menor tempo de conversão possível. O Algoritmo 8 representa a metodologia descrita.

Algoritmo 8: Algoritmo de Conversão para ADC Utilizado

Output: Valor Convertido
$ADC \leftarrow$ **Start()**
while $ADC \neq$ *End of Conversion* **do**
| **Continue()**
end
Output $\leftarrow Valor ADC$

A conversão do ADC foi implementada em *software*, como pode-se observar no Algoritmo 8, visando tornar o método de medição padrão. Porém, pode-se fazer uso de implementações alternativas, tais como o uso de uma conexão do *trigger* do *timer* para iniciar automaticamente a conversão no ADC e acionar uma *flag* quando a conversão for concluída. Uma alternativa similar é utilizar um DMA (*Direct Memory Access*), reduzindo o *overhead* da troca de contexto da interrupção sobre o sistema. Porém alguns microcontroladores não possuem tais dispositivos em seus periféricos, o que acabaria por restringir os testes efetuados.

3.6 DESENVOLVIMENTO E COLETA DE DADOS

Como citado anteriormente, o método que será utilizado nesse trabalho consiste em fazer a medição do tempo necessário para a ação de controle, tendo como início de contagem tempo imediatamente antes do ADC ter iniciado a conversão, e finalizando imediatamente após o valor de saída ser convertido para o módulo de PWM. A partir desse valor, pode-se estimar a máxima frequência de amostragem na qual o microcontrolador pode atuar em um sistema de controle (VARGAS, 2017).

O método foi aplicado no microcontrolador ESP32D0WDQ6, com as seguintes configurações: o tipo de dados utilizado na operação foi *double*, e o *clock* principal da CPU foi configurado para uma frequência de $240MHz$. Além disso, o *timer* foi configurado para gerar interrupções a cada 200 μs, ou seja, $5kHz$, para efetuar a amostragem dos dados.

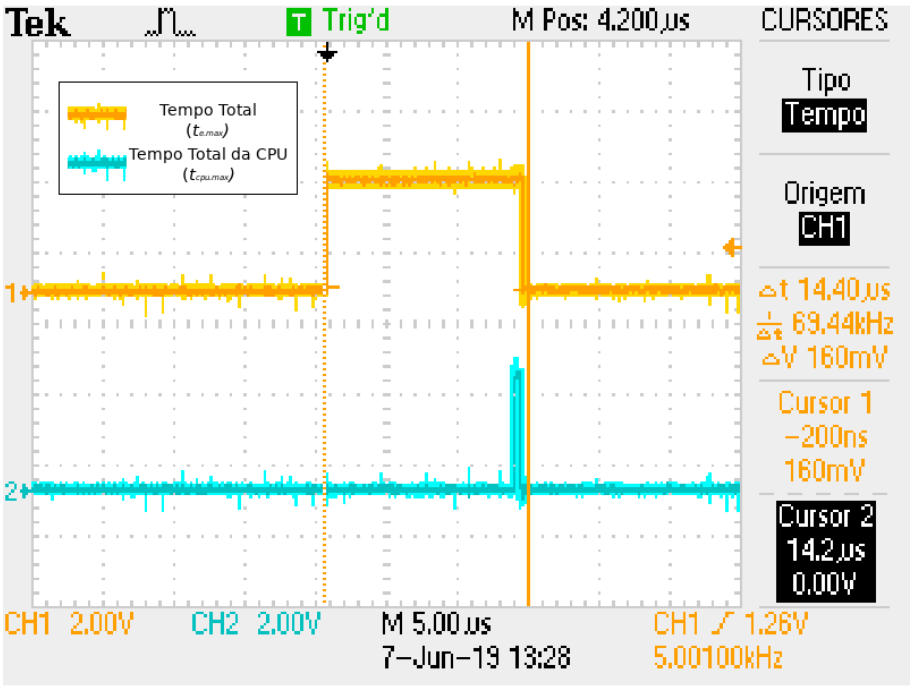


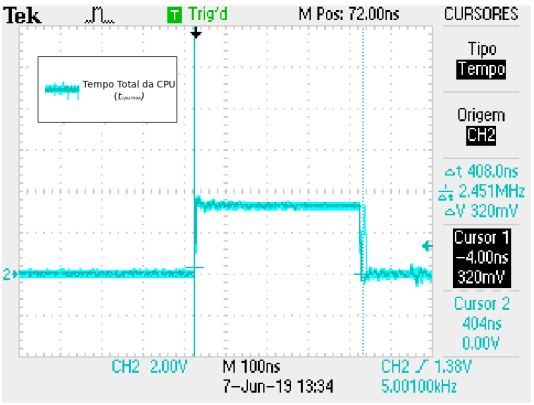
Figura 17 – Tempo total Máximo Usado na Ação de Controle

Utilizou-se o pino *5* da porta *A* para efetuar a medida do tempo gasto pelo microcontrolador durante toda a ação de controle, e o pino *4* para efetuar a medida do tempo gasto nos cálculos da lei de controle executada pelo processador, obtendo-se o resultado observado nas figuras 17 e 18.

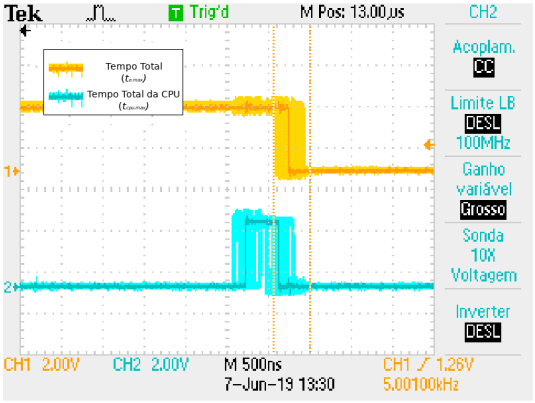
Na Figura 17, observa-se o tempo de execução total máximo do microcontrolador para efetuar a ação de controle, denominado $t_{e.max}$, enquanto a Figura 18a representa o tempo estimado em que o processador efetua os cálculos da lei de controle, $t_{cpu.max}$. Além disso, como fundamentado anteriormente, existe uma latência no tempo total, Δt, que pode ser observada com mais precisão na Figura 18b.

Para obter-se o valor máximo utilizou-se a ferramenta de cursor do osciloscópio digital, que permite fazer uso de duas assintotas verticais para medir a diferença de tempo entre elas. As demais imagens dos testes foram omitidas do trabalho, podendo ser acessadas no repositório *GitHub*, disponível em Scarmocin (2019), juntamente com a implementação do código-fonte desenvolvido na Seção 3.2 para cada microcontrolador utilizado nos testes.

Os testes foram efetuados nos microcontroladores sobre diferentes condições, de forma que a frequência de amostragem escolhida não exceda a frequência máxima $f_{sa.max}$, sendo assim ajustada experimentalmente. Os dados obtidos a partir dos testes podem ser observados na Tabela 5. Além disso, uma série de testes extra foi executada, visando avaliar diferentes variáveis que podem afetar o desempenho do microcontrolador.



(a) Tempo da CPU Máximo Usado na Ação de Controle



(b) Variação do Tempo Total

Figura 18 – Dados Coletados a Partir dos Testes Efetuados no microcontrolador ESP32D0WDQ6

Tabela 5 – Dados Coletados a Partir dos Testes Efetuados nos Microcontroladores - Execução na Memória Flash com Diferentes Formatos Numéricos

Microcontrolador	Tipo de Dado	$t_{e.max}$	$t_{cpu.max}$	Δt
STM32F103R8T6	Float	18,80μs	15,60μs	2,89μs
	Double	23,80μs	21,40μs	3,08μs
	Q_7	8,90μs	6,70μs	0,90μs
	Q_{15}	8,24μs	6,08μs	1,01μs
STM32F407VET6	Float	2,10μs	1,24μs	212ηs
	Double	7,76μs	6,96μs	280ηs
	Q_7	2,52μs	1,68μs	210ηs
	Q_{15}	2,42μs	1,60μs	192ηs
MSP430G2553	Float	160,00μs	147,00μs	7,60μs
	Double	920,80μs	906,00μs	336,00μs
	Q_7	49,6μs	36,80μs	1,68μs
ESP32D0WDQ6	Float	14,40μs	408,00ηs	400,00ηs
	Double	17,50μs	3960,00ηs	400,00ηs
	Q_7	14,40μs	830,00ηs	0,00ηs
	Q_{15}	14,30μs	1000,00ηs	252,00ηs

Os testes genéricos observados na Tabela 5 foram reproduzidos, sendo executados parcialmente na memória RAM, a fim de averiguar diferença no tempo de execução para cada microcontrolador. Os resultados obtidos foram tabulados na Tabela 6.

Tabela 6 – Dados Coletados a Partir dos Testes Efetuados nos Microcontroladores - Execução Parcial na Memória RAM

Microcontrolador	Tipo de Dado	$t_{e.max}$	$t_{cpu.max}$	Δt
STM32F103R8T6	Float	16,20μs	13,10μs	0,00μs
	Double	23,20μs	20,00μs	1,54μs
	Q_7	9,40μs	7,20μs	0,90μs
	Q_{15}	8,90μs	6,90μs	1,14μs
STM32F407VET6	Float	2,26μs	1,32μs	100,00ηs
	Double	10,6μs	9,50μs	820,00ηs
	Q_7	3,16μs	2,24μs	320,00ηs
	Q_{15}	2,92μs	1,92μs	2,40μs
MSP430G2553	Float	165,00μs	154,00μs	15,30μs
	Q_7	50,40μs	36,80μs	1,60μs
ESP32D0WDQ6	Float	14,10μs	500,00ηs	92,00ηs
	Double	17,40μs	4000,00ηs	316,00ηs
	Q_7	14,40μs	1000,00ηs	180,00ηs
	Q_{15}	14,40μs	1000,00ηs	220,00ηs

Os testes também foram executados efetuando algumas divisões durante as operações de cálculo da ação de controle, dentro do *loop* principal, a fim de simular uma função periódica sendo executada juntamente com o processo de controle PID. No decorrer de tais testes, o algoritmo foi executado na memória FLASH. Os resultados podem ser observados na Tabela 7.

Tabela 7 – Dados Coletados - Execução Efetuando Duas Divisões em Série com a Ação de Controle, Operações com Float

Microcontrolador	Quantidade de Divisões	$t_{e.max}$	$t_{cpu.max}$	Δt
STM32F103R8T6	2	18,20μs	16,00μs	2,72μs
	2 (volatile)	29,20μs	26,40μs	2,10μs
	5 (volatile)	64,40μs	60,80μs	20,00μs
STM32F407VET6	2	2,26μs	1,32μs	100ηs
	2 (volatile)	5,90μs	4,72μs	104,00ηs
	5 (volatile)	17,00μs	15,80μs	6,80μs
MSP430G2553	2	192,00μs	181,00μs	5,80μs
	2 (volatile)	340,00μs	330,00μs	28,00μs
	5 (volatile)	1,450,00μs	1,450,00μs	70,00μs
ESP32D0WDQ6	2	14,7μs	520ηs	400ηs
	2 (volatile)	19,80μs	6,28μs	520ηs
	5 (volatile)	29,20μs	15,30μs	1,800μs

A fim de consumir mais tempo da CPU, os testes foram executados utilizando o modificador *volatile* na variável. Tal modificador determina que a região de memória pode ser acessada por meios externos, tal como periféricos, fazendo com que o processador recarregue constantemente o valor no endereço de memória.

3.6.1 PARTICULARIDADES DOS MICROCONTROLADORES

Os testes descritos anteriormente foram também efetuados no microcontrolador MSP430G2553, a fim de obter-se uma métrica de comparação de um microcontrolador de baixo custo, desenvolvido para suportar aplicações mais simples com um baixo consumo de energia, e microcontroladores superiores diversos. Porém, os testes usando o tipo de dado Q_{15} não pode ser executado, uma vez que microcontroladores de 16-*bits* não suportam as operações como implementadas no Apêndice B, como fundamentado na Seção 3.3. Além disso, não foi possível executar as operações a partir da memória RAM utilizando o tipo de dado *Double*, uma vez que essa memória é limitada para 512 *Bytes*. Considerando que a função de controle utiliza cerca de 380 *Bytes*, e as demais variáveis utilizam cerca de 150 *Bytes*, isso excede a capacidade de memória RAM do microcontrolador.

A metodologia foi aplicada também ao microcontrolador ESP32D0WDQ6, porém sofrendo algumas modificações, permitindo efetuar mais testes específicos. O algoritmo de teste também foi executado fazendo uso da memória RAM, porém difere da metodologia implementada para os demais microcontroladores, sendo necessário apenas o uso do atributo *IRAM_ATTR* antes da definição da função. Assim, a função é alocada no devido espaço, localizado na memória IRAM, não sendo necessário copiar as instruções durante a inicialização do código, uma vez que isso é feito automaticamente. Vale ressaltar que, no caso desse microcontrolador, as interrupções devem ser alocadas na IRAM por padrão, portanto, os resultados da Tabela 5 tem a função do Algoritmo 2 sendo executadas na IRAM por padrão.

Foram também realizados testes utilizando divisões em série com as operações, para simular funções periódicas sendo executadas juntamente com o processo de controle PID. Além disso, como observado na Subseção 3.1.1, esse microcontrolador possuí dois núcleos, sendo possível efetuar testes com processos em paralelo. Para tal, foi desenvolvido um *software* que simula uma aplicação real, fazendo leitura e escrita nas variáveis de controle, a fim de validar se existe interferência de cada processo em um núcleo diferente. Os testes específicos efetuados no microcontrolador ESP32D0WDQ6 podem ser observados na Tabela 8.

Tabela 8 – Dados Coletados a Partir dos Testes Efetuados - ESP32D0WDQ6

$t_{cpu.max}$	Memória	Tipo de Dado	Obs
450,00ηs	*Flash*	*Float*	Core-1 task
3.960,00ηs	*Flash*	*Double*	Core-1 task

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Este capitulo tem como objetivo apresentar a análise dos dados obtidos na Seção 3.6, sendo separadas em ação dos periféricos processamento da CPU dos microcontroladores, como discutido anteriormente. Entre as análises efetuadas, serão discutidos os fatores que afetam separadamente o tempo de cada uma das ações dos dados obtidos, tal como: tipo de dado utilizado nas operações aritméticas da CPU; execução do *software* na memória *flash* ou RAM; execução do *software* com tarefas sendo executadas em paralelo no segundo núcleo da CPU, no caso do microcontrolador ESP32D0WDQ6.

4.1 VISÃO GERAL

A partir dos dados obtidos na Tabela 5 e da Equação (27), foi possível obter a frequência de amostragem máxima para cada cada microcontrolador operando o algoritmo desenvolvido, sobre condições de execução padrão e com múltiplos tipos de dados em suas operações numéricas, que é o objetivo geral do trabalho.

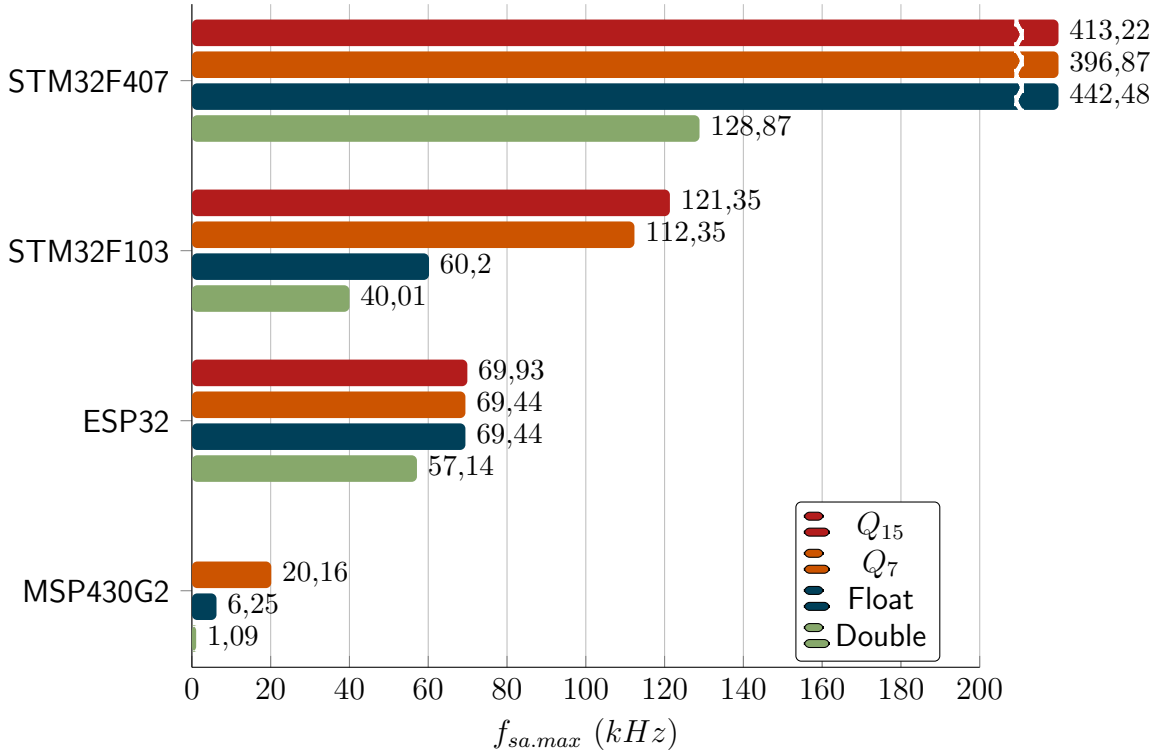


Figura 19 – Comparação Geral da Frequência Máxima de Amostragem e Atuação dos Microcontroladores

Observando o resultado geral no gráfico da Figura 19, percebe-se que o microcontrolador STM32F407 tem capacidade para atuar a uma frequência máxima de 442kHz utilizando *float* nas operações aritméticas, sendo a maior entre os resultados obtidos. Observa-se também

que o menor resultado foi de 1,09kHz, utilizando *double* no microcontrolador MSP430, sendo menor entre os resultados.

4.1.1 ANÁLISE: CPU

Ainda, é possível aplicar as técnicas fundamentadas na Subseção 2.4.1 na Tabela 5 para analisar o desempenho da CPU individualmente, a partir do tempo máximo da das operações para efetuar a lei de controle, $t_{cpu.max}$. A fim de sumarizar os dados da CPU de cada microcontrolador, aplica-se a Equação (18) sobre os diferentes tipos de dado utilizados nas operações, obtendo-se a média geométrica do desempenho para cada CPU. O resultado dessa análise pode ser encontrado na Figura 20.

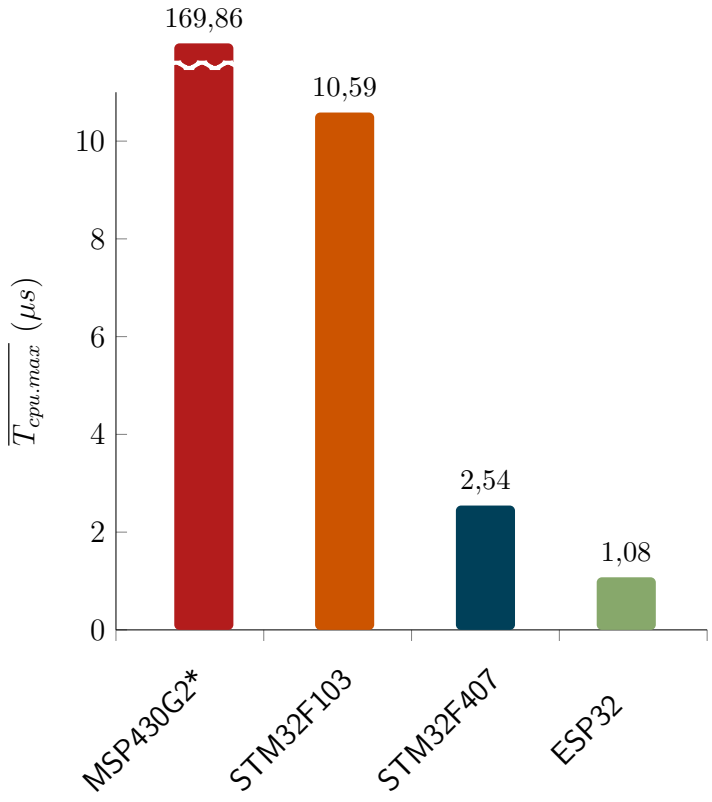


Figura 20 – Comparação Entre as CPUs da Média Geométrica dos Valores de $t_{cpu.max}$ Normalizados Entre os Tipos de Dados

Observando o gráfico da Figura 20, percebe-se que o melhor desempenho médio nas operações da CPU foi de 1,08μs obtido pelo microcontrolador ESP32, enquanto o maior tempo médio foi de 169,86mus, do microcontrolador MSP430G2.

Deve-se lembrar que, como comentado na Seção 3.6, o microcontrolador não possuí operações utilizando o formato numérico Q_{15}, o que afetaria os resultados, mesmo que a média geométrica seja menos afetada por valores distantes da média, como citado na Subseção 2.4.1. Percebe-se também que os resultados da Figura 20 leva em consideração os microcontroladores

STM32F407 e ESP32 utilizando FPU nas operações com tipo de dados *float*, uma vez que isso é uma característica da CPU e é normalmente utilizado em uma aplicação real.

Os dados obtidos na Tabela 7 operando divisões em sequência com a ação de controle podem ser observados mais claramente no gráfico da Figura 21, em que são comparados com os resultados obtidos sem efetuar divisões.

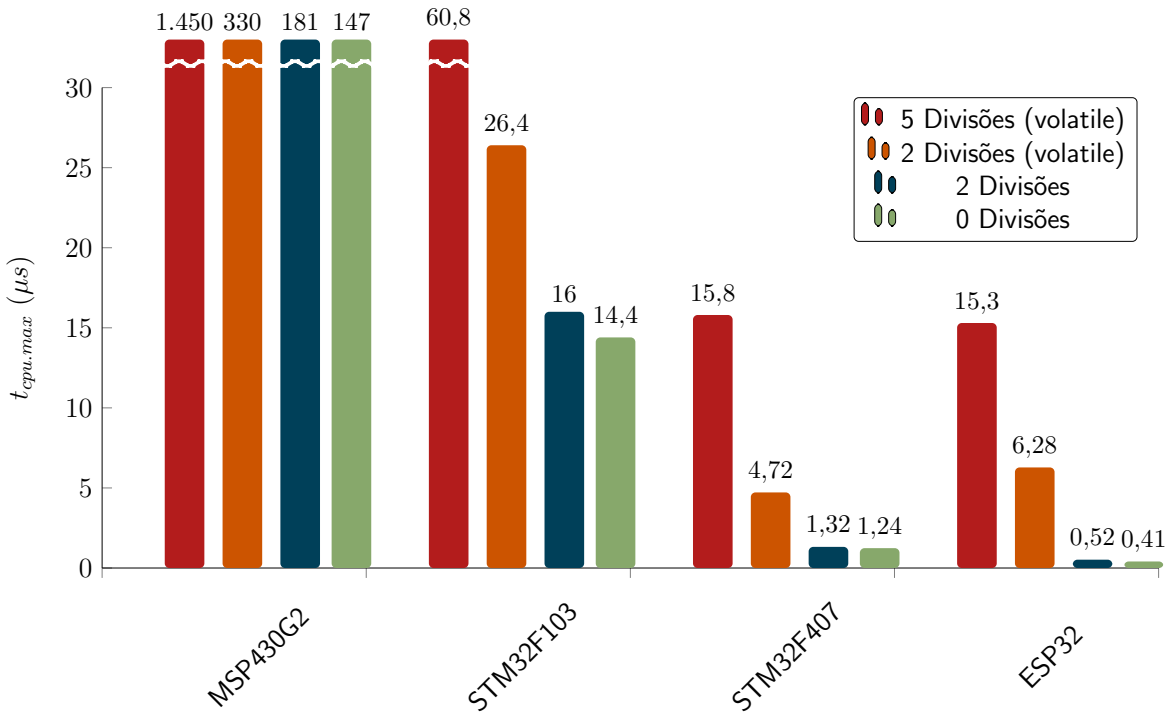


Figura 21 – Tempo Máximo Necessário para que a CPU Efetue a Ação de Controle Operando Divisões em Série

Como comentado na Seção 3.6, as operações de divisão utilizando o modificador *volatile* na variável consomem mais tempo do microcontrolador, obtendo o resultado de 1.450,00μs no microcontrolador MSP430G2.

Utilizando o conceito inverso da Equação (15), pode-se obter o incremento percentual relativo no tempo de execução de cada teste efetuado com divisões em relação ao tempo de execução da CPU sem executar nenhuma divisão.

Pode-se perceber que divisões simples afetam o tempo de processamento em curta escala, não ultrapassando 50% do tempo de execução em nenhum caso, efetuando 2 divisões. porém, operando 2 divisões com modificador *volatile*, o tempo de execução pode aumentar cerca de 15 vezes em relação as operações padrão, no caso do microcontrolador ESP32. É também importante observar que o microcontrolador ESP32 teve o maior aumento relativo no tempo de execução em todos os casos de teste, em relação aos demais microcontroladores, podendo demorar até 37 vezes mais operando 5 divisões utilizando o modificador *volatile*.

Comparando as Figuras 21 e 22, percebe-se que mesmo o microcontrolador MSP430G2 tendo o maior tempo de execução, em termos relativos de processamento, o microcontrolador ESP32 pode tornar-se gradativamente mais lento operando divisões. É também possível perceber

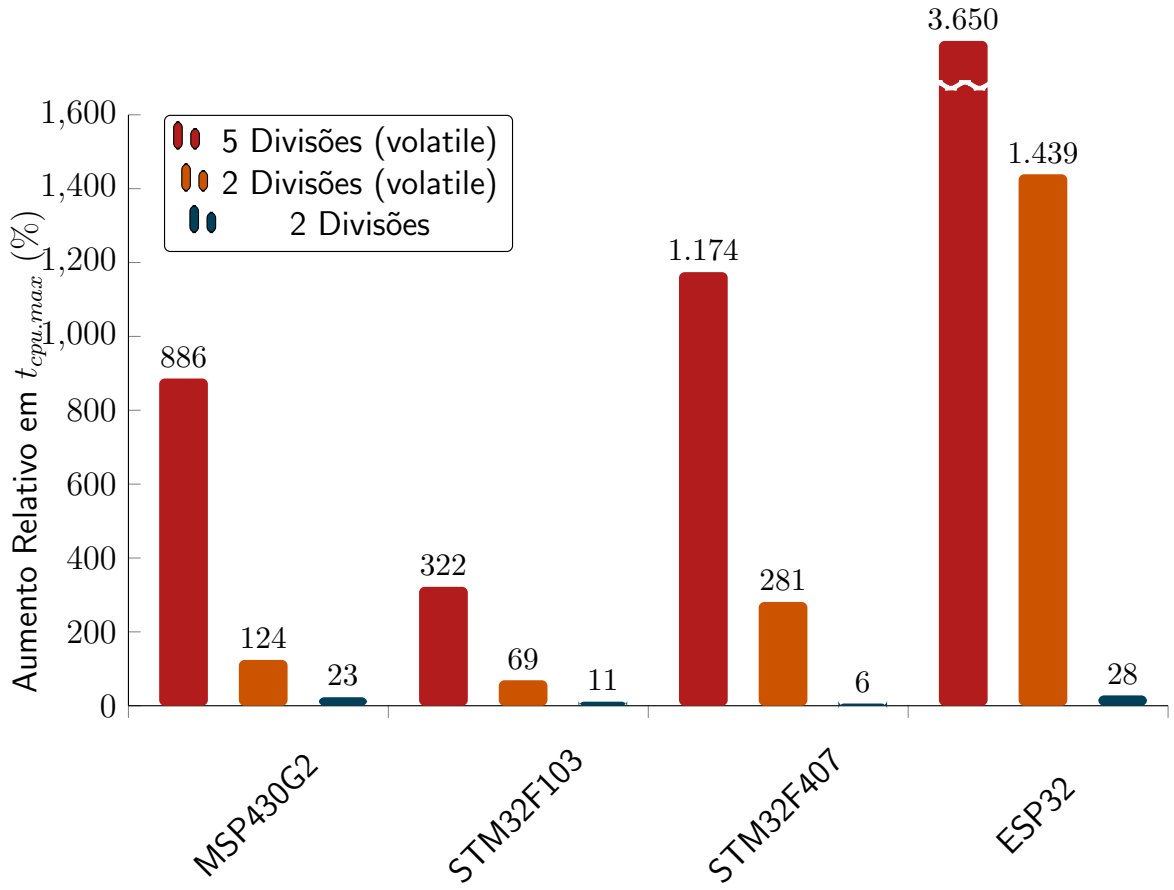


Figura 22 – Tempo Relativo Percentual da CPU Efetuando a Ação de Controle Operando Divisões em Série

que cada microcontrolador tem um aumento relativo diferente no tempo de execução, e portanto, em uma aplicação real, tal informação pode tornar-se mais relevante do que o próprio tempo de execução para cada caso.

4.1.2 ANÁLISE: ADC

Além disso, o ADC de cada microcontrolador foi analisado separadamente, por ser o periférico com maior impacto na análise de tempo de execução, como pode ser observado na Figura 23.

Comparando a Figura 20 com a Figura 23, percebe-se que as características da CPU, como frequência máxima e FPU, não necessariamente são mais impactantes na análise. Tal fator torna-se perceptível no caso do microcontrolador ESP32D0WDQ6, o qual efetuou o processamento das operações com tempo $t_{cpu.max}$ médio mais baixo entre os microcontroladores avaliados. Porém, a velocidade de conversão do ADC obteve o maior tempo t_{adc}, limitando sua frequência máxima $f_{sa.max}$. Além disso, o estudo sobre o ADC do microcontrolador ESP32D0WDQ6 será aprofundado na Seção 4.2.

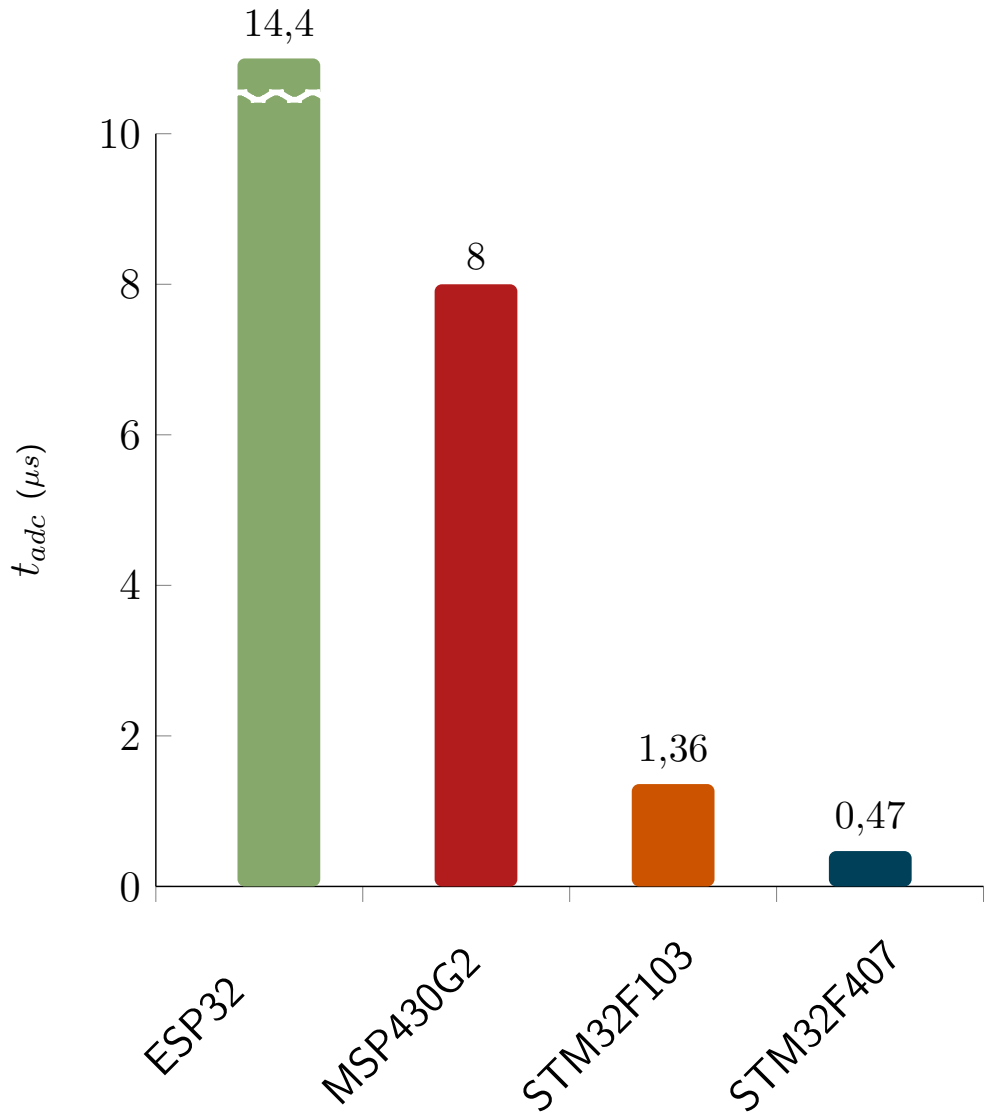
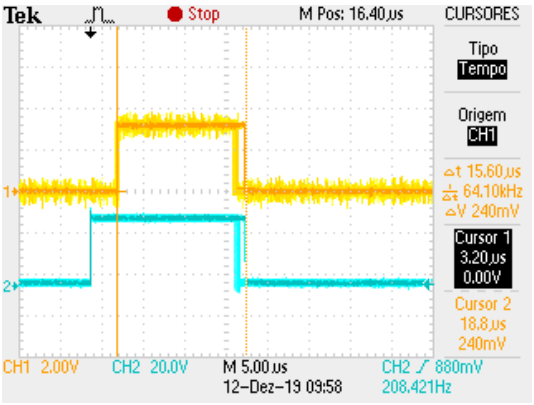


Figura 23 – Comparação do Tempo de Conversão do ADC entre os Microcontroladores

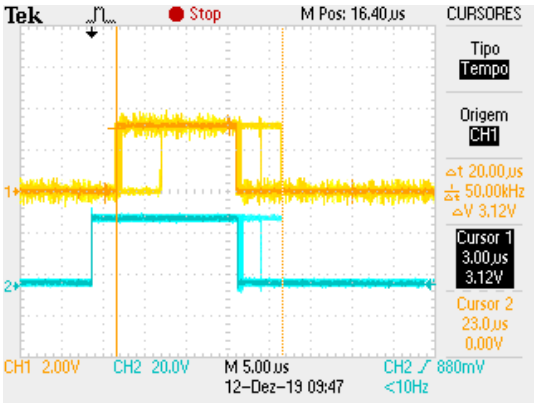
4.1.3 CASO PARTICULAR: EFEITOS DO USO DE UM RTOS

Além dos dados coletados na Seção 3.6, foram obtidos dados experimentais modificando a implementação do código no microcontrolador STM32F103, inserindo o sistema operacional *freertos*, a fim de verificar a interferência desse no algoritmo de controle, tal como a presença de uma latência causada pelo escalonador, denominada *jitter* (LLUESMA et al., 2006).

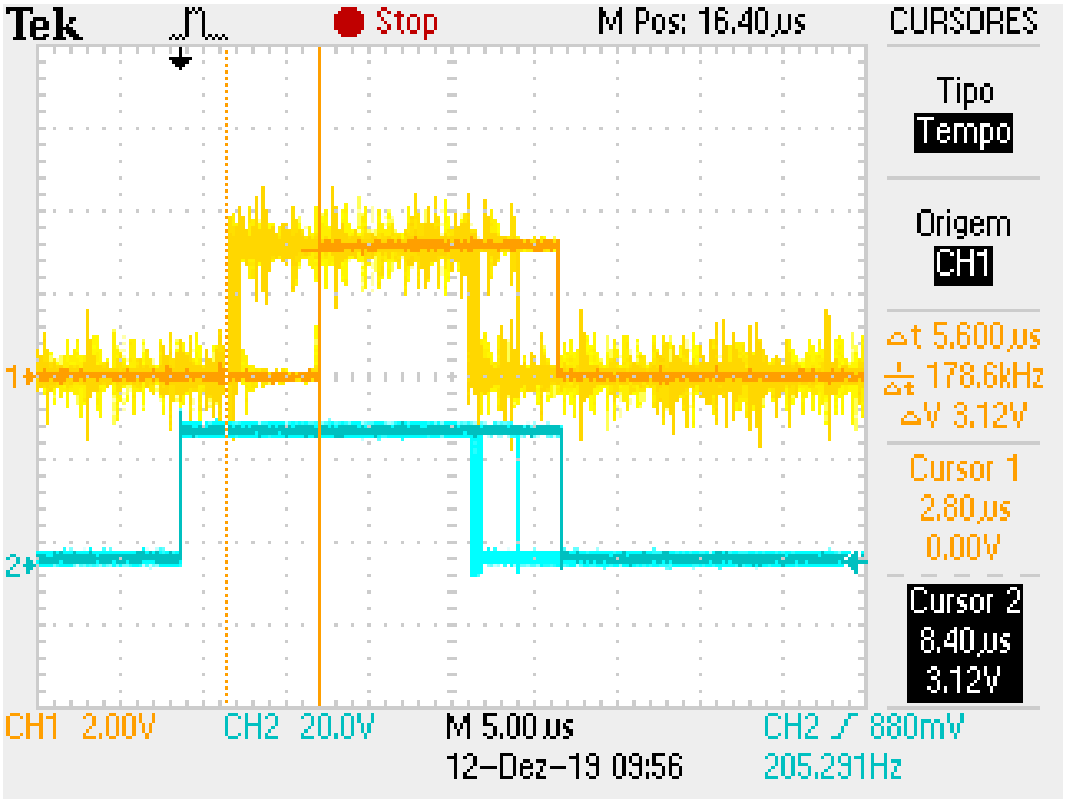
Observa-se a diferença de tempo da CPU quando se faz ou não uso de um RTOS comparando as Figuras 24b e Figura 24a. Tal diferença é causada por uma variação que ocorre arbitrariamente no início da medida de tempo, quando se faz uso do RTOS. Essa variação é denominada *jitter*, como esparecido anteriormente, e pode ser observado na Figura 24c, ocupando um tempo de $5.6\mu s$ da CPU.



(a) Tempo da CPU Máximo



(b) Tempo da CPU Máximo - RTOS



(c) *Jitter*

Figura 24 – Dados Coletados fazendo uso de um RTOS - STM32F103C8T6

4.2 ANÁLISE: ESP32D0WDQ6

Como apresentado na Subseção 3.1.1, a atual implementação da interface ESP-IDF pode afetar consideravelmente o desempenho do microcontrolador, uma vez que seus *drivers* foram desenvolvidos visando atender aplicações específicas. O funcionamento dos *drivers*-IDF para o ADC é feito de forma que os registradores de configuração sejam reconfigurados toda vez que uma conversão é requisitada, fazendo com que o algoritmo consuma mais tempo do

processador.

Utilizando a metodologia apresentada na Seção 3.6 em conjunto com um algoritmo de teste, foi possível analisar separadamente o funcionamento dos conversores AD do microcontrolador ESP32D0WDQ6, obtendo-se os resultados apresentados na Figura 23. Fazendo-se uma reimplementação dos registradores do ADC para atender apenas os requisitos da aplicação de controle utilizada nesse trabalho, obteve-se os resultados observados na Figura 25 em comparação com o que foi obtido utilizando os *drivers* fornecidos pela ESP-IDF.

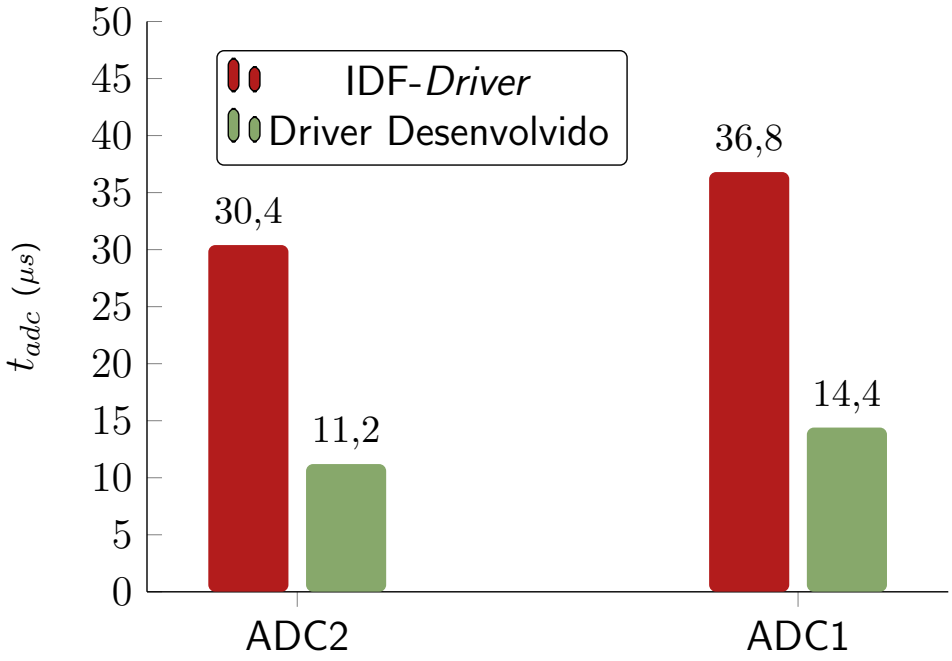


Figura 25 – Comparação do Tempo de Conversão dos ADCs do Microcontrolador

Fazendo uso da Equação (14), percebe-se que utilizando o *driver* desenvolvido no Algoritmo 8, o ADC 2 obteve um *speedup* de 2,71, enquanto o ADC 1 obteve um *speedup* 2,56, para tal aplicação em específico. Isso ocorre devido ao fato de que os *drivers* fornecidos pela ESP-IDF tem uma implementação diferenciada em comparação com *drivers* de outros microcontroladores, como a HAL. Tal funcionamento consiste em reconfigurar o periférico do ADC praticamente por completo, o que acaba por tomar tempo de processamento, que não é necessário no caso dessa aplicação.

Portanto, apenas o uso de um *benchmark* já existente para mensurar o desempenho da CPU, tal como MIPS ou FLOPS citados anteriormente, não é suficiente para avaliar um microcontrolador atuando em determinada aplicação, quando existe o impacto dos demais periféricos utilizados, tal como fundamentado na Subseção 2.4.2.

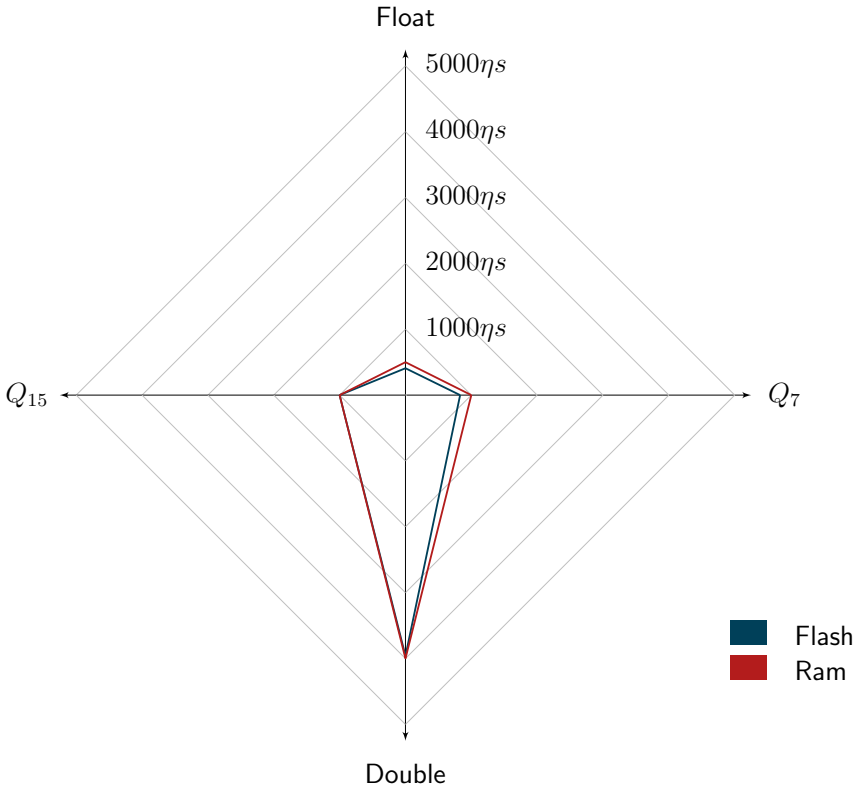


Figura 26 – Comparação do Tempo de Execução da CPU entre Diferentes tipos de Dado e Memórias de Execução - ESP32D0WDQ6 - Xtensa LX6

Compara-se na Figura 26 o desempenho do microcontrolador ESP32 executando o *software* de teste tanto na memória RAM quanto na memória *flash*, também como fazendo uso de diferentes tipos de dado em suas operações numéricas. Deve-se observar que o ADC1 foi utilizado com o Algoritmo 8 para otimizar a ação de controle, porém o ADC2 não foi utilizado por apresentar diferenças em sua estrutura com relação ao ADC1. Além disso, o periférico *Wireless* integrado no microcontrolador ESP32 faz uso intrínseco do ADC 2 quando estiver em uso. Em uma aplicação real na qual ambos podem vir a ser utilizados simultaneamente, isso pode tornar a análise mais complexa (ESPRESSIF SYSTEMS, 2018).

4.3 ANÁLISE: STM32F103C8T6

Analisando os testes efetuados no microcontrolador STM32F103, percebe-se que o desempenho geral da CPU ARM® Cortex®-M3 utilizando tipo de dados de ponto fixo Q_7 e Q_{15} foi maior, o que é esperado para processadores que não possuem FPU. Observando a Figura 27, percebe-se que o desempenho do algoritmo executando parcialmente na memória RAM é superficialmente inferior quando utiliza-se *float* e *double* nas operações.

Isso ocorre devido a ausência de FPU, o que impede o acesso direto à operação de multiplicação, localizada na memória *flash*. Para resolver tal problema, faz-se o uso da *flag -mlong-calls* durante o processo de compilação, além do atributo *long_call* na definição da função executada na memória RAM. Porém, tal método faz com que a CPU efetue um acesso

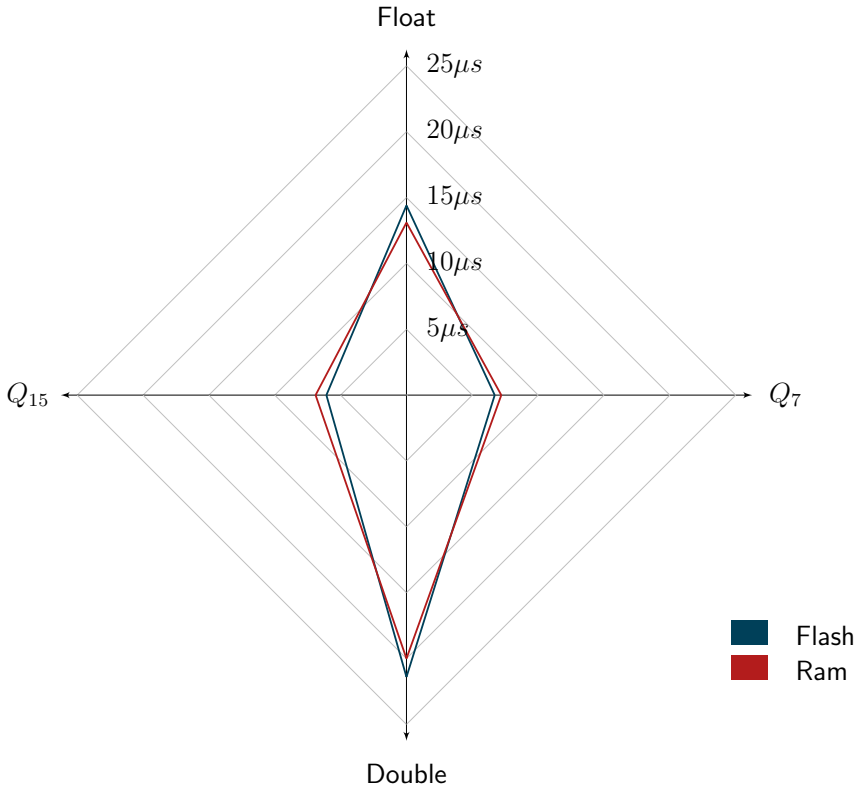


Figura 27 – Comparação do Tempo de Execução da CPU entre Diferentes tipos de Dado e Memórias de Execução - STM32F103C8T6 - ARM Cortex-M3

indireto à operação de multiplicação, armazenando o endereço de memória em um registrador, tornando o acesso mais lento.

4.4 ANÁLISE: STM32F407VET6

Observando a Figura 28, percebe-se que o desempenho da CPU ARM® Cortex®-M4 é maior em comparação com outros microcontroladores, sendo mais eficaz quando se utiliza *Float* em suas operações, uma vez que possui FPU.

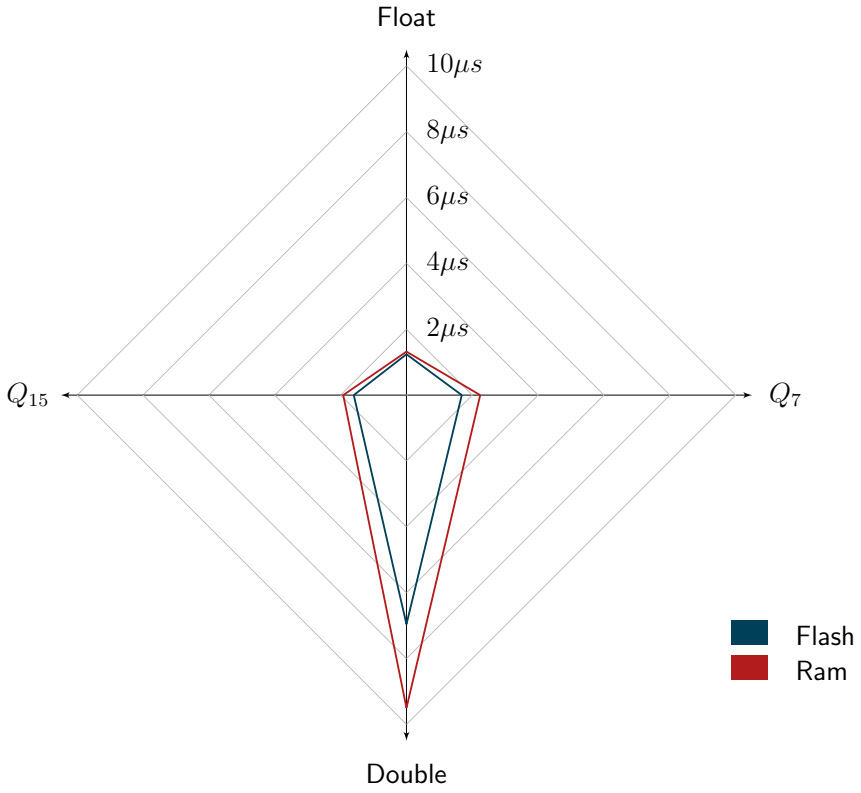


Figura 28 – Comparação do Tempo de Execução da CPU entre Diferentes tipos de Dado e Memórias de Execução - STM32F407VET6 - ARM Cortex-M4

Observa-se que o microcontrolador STM32F407 obteve os melhores resultados em relação com os outros microcontroladores em geral, com frequência de atuação que pode ultrapassar 400kHz.

4.5 ANÁLISE: MSP430G2553

Analisando a Figura 20 e a Figura 23, observa-se que o microcontrolador possuí tanto um ADC não tão superior em relação aos demais, como também sua CPU apresenta menor desempenho, acarretando no referente resultado.

O resultado pode ser observado no gráfico da Figura 29, em que se compara o tempo de execução do algoritmo nas memórias *flash* e RAM. Deve-se observar que não foi possível efetuar os testes utilizando dados em formato *double* executando o algoritmo na memória RAM pelo uso excessivo da mesma, assim como não é possível utilizar dados em formato Q_{15}, como fundamentado na Seção 3.6.

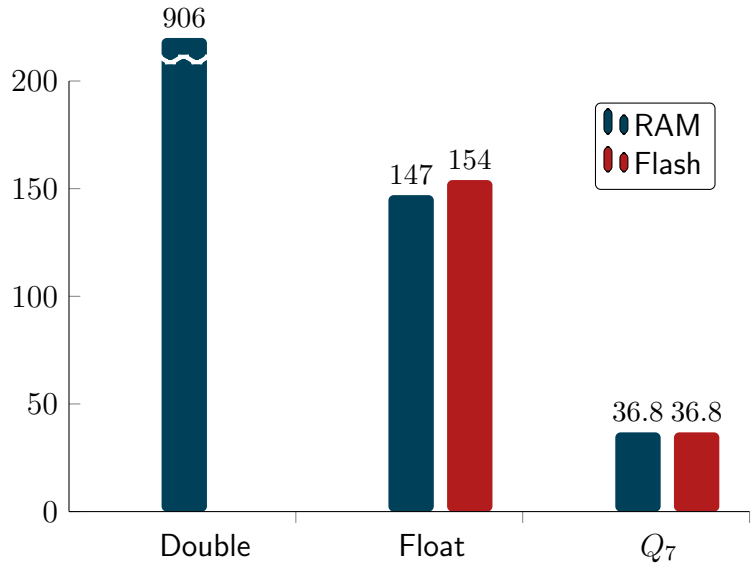


Figura 29 – Comparação do Tempo de Execução da CPU entre Diferentes tipos de Dado e Memórias de Execução - MSP430G2

Analisando a Figura 29, percebe-se que o microcontrolador MSP430 possuí um desempenho similar executando o *software* tanto na memória Flash quanto na memória RAM. observando as tabelas 5 e 6, percebe-se que a variação da medida para operações com formato *float* tem aproximadamente o mesmo valor da diferença entre a medida da CPU observada na Figura 29, comparando-se o algoritmo em execução nas memórias *flash* e RAM.

5 CONCLUSÃO

Conforme apresentado no Capítulo 1, o objetivo proposto para o trabalho no Seção 1.1, de obter resultados experimentais de testes com medidas de desempenho para sistemas de controle foi satisfeito, bem como o desenvolvimento e a obtenção de resultados, tal como estimar a frequência máxima de operação de cada microcontrolador sob condições específicas. A metodologia apresentada na Capítulo 3 provou-se funcional conforme os resultados obtidos na Capítulo 4, e as análises efetuadas servem de base para utilização futura.

Como foi apresentado no Capítulo 3, para analisar o desempenho de um sistema computacional, uma estratégia eficaz é dividir a análise em fragmentos que afetem o resultado separadamente. No caso do referente trabalho, a análise foi fragmentada em análise das operações efetuadas pela CPU e análise dos periféricos, dando ênfase para análise do conversor AD.

Tal método teve seus resultados apresentados no Capítulo 4, no qual foi possível perceber claramente que, na aplicação de controle, um fator discriminante que deve ser levado em consideração são as características do ADC, e não apenas características da CPU.

Tal resultado fica claro na análise da Seção 4.3, em que a CPU Xtensa LX6 obteve o menor tempo de operação em relação aos demais, observado na Figura 20, porém seu desempenho total para a aplicação foi relativamente baixo, como constatado no gráfico da Figura 19.

Analisando o caso em que uso do segundo núcleo para outros propósitos enquanto a ação de controle é efetuada, não houve impacto significativo no resultado. Porém, em uma aplicação na qual os periféricos mais robustos vêm a ser utilizados, ou ainda quando a aplicação implementa configurações utilizando NVS (*Non-Volatile Storage*), não se pode garantir tal resultado, fazendo-se necessário uma análise específica para a aplicação em questão.

Deve-se observar que os resultados obtidos a partir de um código implementado diretamente nos registradores de configuração dos periféricos, a fim de minimizar os efeitos do uso da biblioteca de *drivers* IDF, que impactam no tempo de conversão devido ao fato de que o *driver* reconfigura parte do ADC, como explicado na Seção 4.2. Portanto, tal método utilizado no Algoritmo 8 pode ser implementado de forma similar em outras aplicações onde requisitos de desempenho se fazem presente.

Além disso, os resultados utilizando tipo de dado *double* apresentaram um desempenho proporcionalmente inferior em relação aos demais, para todos os casos analisados, uma vez que não possuem suporte para números com precisão dupla.

Percebe-se também que ao executar os testes a partir da memoria RAM não implica em obter maior desempenho para todos os casos, uma vez que tal resultado depende da arquitetura e das características da CPU. Tal fato pode ser observado na Figura 28, em que o desempenho executando o *software* na memória RAM foi menor, o que também é esperado

nesse caso. Como foi fundamentado na Subseção 3.1.1, o acesso à memória Flash equivale a um *0-wait-sate*.

Observando a Figura 19, percebe-se que o microcontrolador MSP430G2553 possuí menor capacidade em relação aos demais microcontroladores analisados nesse trabalho, uma vez que foi projetado para aplicações que exigem menor custo e baixo consumo de energia. Porém, ainda é possível obter uma frequência aceitável utilizando diferentes formatos numéricos nas operações.

Além do desempenho, observa-se outras características que podem afetar o controle do sistema, tal como efeitos de quantização, em que devido a uma má escolha do formato Q_n a ser utilizado, pode vir a causar perda de resolução.

O trabalho apresentou a possibilidade de utilizar os microcontroladores avaliados para aplicações de controle, mostrando suas limitações individuais e como é possível lidar com elas em casos específicos. É possível obter um incremento no desempenho configurando manualmente os periféricos, tal como conversor AD, quando *drivers* para aplicações específicas não possuem o desempenho desejado.

5.1 TRABALHOS FUTUROS

Esta seção tem o objetivo de apontar as possibilidades de trabalhos a serem desenvolvidos a partir desse, levando em consideração análises e metodologias que não foram implementadas no mesmo por diversos fatores, além das possibilidades de mudanças nas metodologias apresentadas.

a) Verificar a possibilidade de remoção total do *freeRTOS* e dos *drivers* ESP-IDF no microcontrolador ESP32 para obter-se medidas exatas, comparando ambas as implementações.
b) Efetuar uma comparação nos microcontroladores ESP32 e STM32F4, e opcionalmente outras CPUs com suporte a FPU, a fim de comparar o desempenho das operações de aritméticas com e sem utilização da FPU.
c) Efetuar os testes com aplicações reais executando em série com a ação de controle, tal como um filtro digital, e avaliar o impacto que a mesma tem sobre a CPU.
d) Verificar a possibilidade de implementação dos algoritmos com suas operações aritméticas executando totalmente na memória RAM.
e) Analisar o código de baixo nível gerado pelo compilador, a fim de obter-se um conjunto de instruções mínimo para cada caso, podendo assim efetuar as análises matematicamente, estimando o tempo máximo de cada instrução durante a execução.

Referências

754-2008, I. Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, p. 1–70, Aug 2008. Citado na página 14.

BOOTH, A. D. A signed binary multiplication technique. **The Quarterly Journal of Mechanics and Applied Mathematics**, Oxford University Press, v. 4, n. 2, p. 236–240, 1951. Citado na página 15.

DORF, R. C. et al. **Sistemas de Controle Modernos**. [S.l.]: Pearson Prentice Hall, 2005. Citado 2 vezes nas páginas 5 e 8.

ESPRESSIF SYSTEMS. **ESP32 Series Datasheet"**. [S.l.], 2018. "Rev. v2.5". Citado 3 vezes nas páginas 25, 30 e 46.

GRIDLING, G.; WEISS, B. Introduction to microcontrollers. **Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group**, 2007. Citado 2 vezes nas páginas 12 e 13.

HENNESSY, J. L.; PATTERSON, D. A. **Computer architecture: a quantitative approach**. [S.l.]: Elsevier, 2011. Citado 5 vezes nas páginas 13, 17, 18, 19 e 22.

JAIN, R. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: John Wiley & Sons, 1990. Citado na página 18.

KRAMER, K.; STOLZE, T.; BANSE, T. Benchmarks to find the optimal microcontroller-architecture. In: **2009 WRI World Congress on Computer Science and Information Engineering**. [S.l.: s.n.], 2009. v. 2, p. 102–105. Citado na página 21.

KUO, B. C. Digital control systems, holt, rinehart and winston. **Inc., New York**, v. 1980, 1980. Citado 7 vezes nas páginas 6, 7, 8, 9, 10, 12 e 16.

LABROSSE, J. J. **Embedded Systems Building Blocks**. 2nd. ed. [S.l.]: C M P Books, 2000. ISBN 0879306041. Citado na página 15.

LILJA, D. J. **Measuring computer performance: a practitioner's guide**. [S.l.]: Cambridge university press, 2005. Citado 4 vezes nas páginas 16, 17, 18 e 19.

LLUESMA, M. et al. Jitter evaluation of real-time control systems. In: **12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)**. [S.l.: s.n.], 2006. p. 257–260. ISSN 2325-1271. Citado na página 43.

NISE, N. S.; SILVA, F. R. da. **Engenharia de sistemas de controle**. [S.l.]: LTC, 2002. v. 3. Citado 4 vezes nas páginas 4, 7, 8 e 9.

OGATA, K. **Discrete-time control systems**. [S.l.]: Prentice Hall Englewood Cliffs, NJ, 1995. v. 2. Citado na página 7.

OGATA, K.; SEVERO, B. **Engenharia de controle moderno**. [S.l.]: Prentice Hall do Brasil, 1998. Citado na página 4.

PEREIRA, F. **Tecnologia ARM: Microcontroladores de 32 bits**. [S.l.: s.n.], 2007. Citado na página 14.

SCARMOCIN, R. **Análise do Desempenho de Microcontroladores para Sistemas de Controle**. 2019. Disponível em: <https://github.com/raulscr/analise-de-desempenho-de-microcontroladores-para-sistemas-de-controle.git>. Citado na página 35.

STALLINGS, W. **Arquitetura e Organização de Computadores 8a Edição**. [S.l.]: São Paulo: Pearson Prentice Hall, 2010. Citado na página 14.

STARR, G. P. Introduction to applied digital control. **Lecture Notes in Digital Control**, p. 5–31, 2006. Citado na página 10.

STEVENSON, J. **Q-Values in the Watch Window**. [S.l.], 2002. Citado na página 16.

STMICROELECTRONICS. **STM32F103x8, STM32F103xB Series Datasheet**. [S.l.], 2015. Rev. 17. Citado 2 vezes nas páginas 25 e 30.

STMICROELECTRONICS. **STM32F405xx, STM32F407xx Series Datasheet**. [S.l.], 2016. Rev. 08. Citado 3 vezes nas páginas 23, 25 e 30.

TENSILICA, INC. **Xtensa® Instruction Set Architecture - Reference Manual**. [S.l.], 2010. RC-2010.1 Release. Citado na página 14.

TEXAS INSTRUMENTS. **MSP430G2x52, MSP430G2x12 Series Datasheet**. [S.l.], 2010. REVISED MAY 2013. Citado 2 vezes nas páginas 25 e 30.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas digitais: princípios e aplicações**. [S.l.]: Prentice Hall, 2003. v. 8. Citado na página 7.

VARGAS, D. R. **Co-Projeto de Perspectiva Completa para Conversores CC-CC com Controlador Digital**. Tese (Doutorado) — Universidade Federal de Santa Maria, 2017. Citado 6 vezes nas páginas , 11, 19, 20, 22 e 34.

WALLACE, C. S. A suggestion for a fast multiplier. **IEEE Transactions on Electronic Computers**, EC-13, n. 1, p. 14–17, Feb 1964. ISSN 0367-7508. Citado na página 15.

WEICKER, R. P. An overview of common benchmarks. **Computer**, IEEE, v. 23, n. 12, p. 65–75, 1990. Citado na página 19.

Apêndices

APÊNDICE A – CÓDIGO-FONTE PARA O CONTROLE PID

pidlib.h

```
/**
  ******************************************************************************
  * @file    pidlib.c
  * @author  Raul Scarmocin
  * @version V2.1.2
  * @date    21-Jun-2019
  * @brief   This file provides all the PID-Control functions
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef PID_LIB_PIDLIB_H_
#define PID_LIB_PIDLIB_H_

#include "portMCU.h"

/**
  * @brief  PID-Controller Structure definition
  * @param  Ka: Ka-Constant value, where Ka = Ki / 2 / fs + Kp + Kd * fs
  * @param  Ka: Kb-Constant value, where Kb = Ki / 2 / fs - Kp - 2 * Kd * fs
  * @param  Ka: Kc-Constant value, where Kc = Kd * fs
  * @param  ref: reference value, in range of adc-input
  * @param  min_output: minimum output value, in range of pwm-output
  * @param  max_output: maximum output value, in range of pwm-output
  */
typedef struct {
  pid_const_t Ka, Kb, Kc;
  pid_const_t ref;
  pid_const_t err;
  pid_const_t err_1;
  pid_const_t err_2;
  pid_const_t min_output;
  pid_const_t max_output;
  pid_const_t output;
  pid_const_t input;
  uint8_t loop;

} pid_type;


/**
  * Exported functions
  */
pid_type* new_PID(pid_type *pid, pid_const_t Ka, pid_const_t Kb, pid_const_t Kc,
          int16_t ref, uint16_t min_output, uint16_t max_output);
void pidLoopRoutine(pid_type *pid);
void pidInterruptRoutine(pid_type *pid);

#endif
```

pidlib.c

```
/**
  ******************************************************************************
  * @file    pidlib.c
  * @author  Raul Scarmocin
  * @version V2.1.2
  * @date    21-Jun-2019
  * @brief   This file provides all the PID-Control functions
  ******************************************************************************
  */

#include "pidlib.h"

/**
  * @brief  Try to use ESP32 Dual-core to
  * increase the operations speed
  */

/**
  * @brief  Initializes the PID-Controller
  * @param  pid: pointer to a pid_type structure that will be initialized
  * @param  Ka: Ka-Constant value, where Ka = Ki / 2 / fs + Kp + Kd * fs
  * @param  Ka: Kb-Constant value, where Kb = Ki / 2 / fs - Kp - 2 * Kd * fs
  * @param  Ka: Kc-Constant value, where Kc = Kd * fs
  * @param  ref: reference value, in range of adc-input
  * @param  min_output: minimun output value, in range of pwm-output
  * @param  max_output: maximun output value, in range of pwm-output
  * @retval initialized pid_type pointer
  */
pid_type* new_PID(pid_type *pid, pid_const_t Ka, pid_const_t Kb, pid_const_t Kc,
      int16_t ref, uint16_t min_output, uint16_t max_output){

  pid->Ka = Ka;
  pid->Kb = Kb;
  pid->Kc = Kc;

  pid->ref = ref;
  pid->min_output = min_output;
  pid->max_output = max_output;

  pid->err_2 = 0;
  pid->err_1 = 0;
  pid->err = 0;

  pid->input = 0;
  pid->output = 0;

  pid->loop = 0;

  return pid;
}


/**
  * @brief  Update the pid controller
  * @note   Must be placed in the main-loop
```

```
   * @note   The flag must be set periodically, preferencially by a timer interrupt
   * @param  pid: pointer to a pid_type structure that will be updated
   * @retval none
   */
#ifdef RAM_ATTRIBUTE
RAM_ATTRIBUTE
#endif
inline void pidLoopRoutine(pid_type *pid){
  if(pid->loop){

    /**
      * @brief  Exclude Peripheral operations
      * to measure just the CPU operations
      */
    #ifndef EXCLUDE_PERIPH_OP

    PID_PWM_WRITE(pid->output);
    PID_ADC_READ(pid->input);

    #endif

    SET_TEST_PIN_CALC;
    /**
      * @brief  Exclude CPU operations to measure
      * just the peripheral operations
      */
    #ifndef EXCLUDE_CALC_OP

    /**
      * @brief  Simulate a Volt-to-quantizer
      * conversion cast using divisions
      */
    #ifdef DIV_ACT

    double unused_var;
    int i = DIV_ACT + 1;
    while(i--)
        unused_var = (double)pid->input / (double)i;

    #endif

    /**
      * @brief  PID-Controller specific operations
      */
    pid->err_2 = pid->err_1;
    pid->err_1 = pid->err;
    pid->err = pid->ref - pid->input;

    pid->output = SUM(MULT(pid->Ka, pid->err), (pid_const_t)pid->output);
    pid->output = SUM(MULT(pid->Kb, pid->err_1), (pid_const_t)pid->output);
    pid->output = SUM(MULT(pid->Kc, pid->err_2), (pid_const_t)pid->output);

    if(pid->output > pid->max_output)
        pid->output = pid->max_output;
```

```
      if(pid->output < pid->min_output)
          pid->output = pid->min_output;

      pid->loop = 0;

      #endif
      RESET_TEST_PIN;
      RESET_TEST_PIN_CALC;
    }
  }

  /**
    * @brief  Function placed in the timer
    *  interrupt to periodically set the flag
    * @param  pid: pointer to a pid_type
    *  structure that will be updated
    * @retval none
    */
  #ifdef RAM_ATTRIBUTE
  RAM_ATTRIBUTE
  #endif
  inline void pidInterruptRoutine(pid_type *pid){
    SET_TEST_PIN;
    CLEAR_TIMER_FLAGS;
    pid->loop = 1;
  }
```

portMCU.h

```
/**
*****************************************************************************
* @file     portMCU.h
* @author   Raul Scarmocin
* @version  V2.0
* @date     21-Jun-2019
* @brief    This file provides MCU-peripheral control to the pid-controller
*****************************************************************************
*/

#ifndef PORTMCU_H_
#define PORTMCU_H_
/* Put driver-level includes here
*
*/


/* User-set macros
*
*/

#define FLOAT_OP
//#define DIV_ACT        5
//#define EXCLUDE_PERIPH_OP
//#define EXCLUDE_CALC_OP
//#define RAM_ATTRIBUTE




#define PWM_FREQ     5000
#define COUNTER_PERIOD     8400
#define TIM_FREQ     5000
#define ADC_RANGE    4095




#define SET_TEST_PIN

#define RESET_TEST_PIN

#define SET_TEST_PIN_CALC

#define RESET_TEST_PIN_CALC


/*
* Macro to clear timer flags on ISR
* User must define it
*/
#define CLEAR_TIMER_FLAGS ({})
```

```
/* Macro to read the ADC
 * if Q7_OP defined, a 12-bits must be
 * ranged to 7-bits (loses 5-bits of resolution)
 * User must define it
 */
#define PID_ADC_READ(_INPUT_) ({})

/*
* Macro to write at the PWM
* User must define it
*/
#define PID_PWM_WRITE(_OUTPUT_) ({})

/**
* Data-types definitions according to the user-sets
*/

#if defined(Q7_OP)

#include "qn_lib/q7lib.h"
typedef q7_t    pid_const_t;
#define PARSE_OP(NUM) double_to_q7(NUM)
#define SUM(NUM1, NUM2) q7_sum(NUM1, NUM2)
#define MULT(NUM1, NUM2) (q7_mult(NUM1, NUM2))
#define DIV(NUM1, NUM2) q7_div(NUM1, NUM2)

#elif defined(Q15_OP)

#include "qn_lib/q15lib.h"
typedef q15_t    pid_const_t;
#define PARSE_OP(NUM) double_to_q15(NUM)
#define SUM(NUM1, NUM2) q15_sum(NUM1, NUM2)
#define MULT(NUM1, NUM2) q15_mult(NUM1, NUM2)
#define DIV(NUM1, NUM2) q15_div(NUM1, NUM2)

#elif defined(DOUBLE_OP)

typedef double    pid_const_t;
#define PARSE_OP(NUM) (double)(NUM)
#define SUM(NUM1, NUM2) ((double)NUM1 + (double)NUM2)
#define MULT(NUM1, NUM2) ((double)NUM1 * (double)NUM2)
#define DIV(NUM1, NUM2) ((double)NUM1 / (double)NUM2)

#else

typedef float    pid_const_t;
#define PARSE_OP(NUM) (float)(NUM)
#define SUM(NUM1, NUM2) ((float)NUM1 + (float)NUM2)
#define MULT(NUM1, NUM2) ((float)NUM1 * (float)NUM2)
#define DIV(NUM1, NUM2) ((float)NUM1 / (float)NUM2)

#endif

#endif
```

APÊNDICE B – CÓDIGO-FONTE PARA MANIPULAÇÃO DE NÚMEROS Qn

types_qn.h

```
/**
  ********************************************************************************
  * @file     types_qn.h
  * @author   Raul Scarmocin
  * @version  V1.0.0
  * @date     30-May-2019
  * @brief    This file provides the Qn-types
  ********************************************************************************
  */
#ifndef TYPES_QN_H_
#define TYPES_QN_H_

typedef signed int  q7_t;
typedef signed int  q15_t;
typedef signed int  q31_t;
typedef signed long q63_t;

#endif
```

q7lib.h

```
/**
  ********************************************************************************
  * @file     q7lib.h
  * @author   Raul Scarmocin
  * @version  V1.0.0
  * @date     30-May-2019
  * @brief    This file provides the Q7 addition,
  * multiplication and division functions
  * @note     Subraction is a particular case of addition
  ********************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef Q7LIB_H_
#define Q7LIB_H_

#include "types_qn.h"

q7_t double_to_q7(double n);
double q7_to_double(q7_t n);
q7_t q7_sum(q7_t n1, q7_t n2);
q7_t q7_mult(q7_t n1, q7_t n2);
q7_t q7_div(q7_t n1, q7_t n2);

#endif
```

q15lib.h

```
/**
  ******************************************************************************
  * @file    q15lib.h
  * @author  Raul Scarmocin
  * @version V1.0.0
  * @date    30-May-2019
  * @brief   This file provides the Q15 addition,
  * multiplication and division functions
  * @note    Subraction is a particular case of addition
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef Q15LIB_H_
#define Q15LIB_H_

#include "types_qn.h"

q15_t double_to_q15(double n);
double q15_to_double(q15_t n);
q15_t q15_sum(q15_t n1, q15_t n2);
q15_t q15_mult(q15_t n1, q15_t n2);
q15_t q15_div(q15_t n1, q15_t n2);


#endif
```

q7lib.c

```
/**
  ******************************************************************************
  * @file    q7lib.c
  * @author  Raul Scarmocin
  * @version V2.0.0
  * @date    30-May-2019
  * @brief   This file provides all the Q7 functions
  ******************************************************************************
  */

#include "types_qn.h"
#include "q7lib.h"


/**
  * @brief  Perform the convertion of a double value to Q7
  * @param  n: double value to be converted
  * @retval n converted to Q7-format
  */
q7_t double_to_q7(double n){
    return (q7_t)(n * 0x80);            //  7bit shift (2^7 = 0x80)
}

/**
  * @brief  Perform the convertion of a Q7 to double value
  * @param  n: Q7-formated value
  * @retval n in double format
  */
double q7_to_double(q7_t n){
    return (double)(((double long)n) / 0x80);
}


/**
  * @brief  Perform the addition of two Q7 values
  * @note   Avoid overflow and underflow by saturation
  * (max value = 2^7, min value = - 2^7)
  * @note   Subraction is a particular case of this function
  * @param  n1: Q7-formated first value
  * @param  n2: Q7-formated second value
  * @retval Q7-formated value that represents the
  * addition of n1 and n2
  */
inline q7_t q7_sum(q7_t n1, q7_t n2){
    q15_t   aux;

    aux = (q15_t)n1 + (q15_t)n2;

    if(aux > 0x7F)
        aux = 0x7F;
    else if(aux < -0x7F)
        aux = -0x7F;

    return (q7_t)aux;
}
```

```
/**
  * @brief  Perform the multiplication of two Q7 values
  * @note   The result is a Q15 value, but the 7
  * LSB are discarted
  * @param  n1: Q7-formated first value
  * @param  n2: Q7-formated second value
  * @retval Q7-formated value that represents the
  * multiplication of n1 and n2
  */
inline q7_t q7_mult(q7_t n1, q7_t n2){

    q15_t   aux;
    aux = (q15_t)n1 * (q15_t)n2;

    return aux >> 7;

}

/**
  * @brief  Perform the division of two Q7 values
  * @note   To avoid the lose of the first bits, the first
  * value is bit-shifted 7-bits to left
  * @param  n1: Q7-formated first value
  * @param  n2: Q7-formated second value
  * @retval Q7-formated value that represents the
  * division of n1 and n2
  */
inline q7_t q7_div(q7_t n1, q7_t n2){

    q15_t   aux;
    aux = (q15_t)n1 << 7;

    return aux / (q15_t)n2;

}
```

q15lib.c

```
/**
  ******************************************************************************
  * @file    q15lib.c
  * @author  Raul Scarmocin
  * @version V2.0.0
  * @date    30-May-2019
  * @brief   This file provides all the Q15 functions
  ******************************************************************************
  */

#include "types_qn.h"
#include "q15lib.h"

/**
  * @brief  Perform the convertion of a double value to Q15
  * @param  n: double value to be converted
  * @retval n converted to Q15-format
  */
q15_t double_to_q15(double n){
  return (q15_t)(n * 0x8000);
}

/**
  * @brief  Perform the convertion of a Q15 to double value
  * @param  n: Q15-formated value
  * @retval n in double format
  */
double q15_to_double(q15_t n){
  return (double)(((double long)n) / 0x8000);
}

/**
  * @brief  Perform the addition of two Q15 values
  * @note   Avoid overflow and underflow by saturation
  * (max value = 2^15, min value = - 2^15)
  * @note   Subraction is a particular case of this function
  * @param  n1: Q15-formated first value
  * @param  n2: Q15-formated second value
  * @retval Q15-formated value that represents the addition of n1 and n2
  */
inline q15_t q15_sum(q15_t n1, q15_t n2){
  q31_t aux;

  aux = (q31_t)n1 + (q31_t)n2;

  if(aux > 0x7FFF)
    aux = 0x7FFF;
  else if(aux < -0x7FFF)
    aux = -0x7FFF;

  return (q15_t)aux;
}


/**
```

```
  * @brief  Perform the multiplication of two Q15 values
  * @note   The result is a Q31 value, but the 15 LSB are discarted
  * @param  n1: Q15-formated first value
  * @param  n2: Q15-formated second value
  * @retval Q1-formated value that represents the multiplication of n1 and n2
  */
inline q15_t q15_mult(q15_t n1, q15_t n2){

  q31_t aux;
  aux = (q31_t)n1 * (q31_t)n2;

  return aux >> 15;
}

/**
  * @brief  Perform the division of two Q15 values
  * @note   To avoid the lose of the first bits, the first value is
  * bit-shifted 15-bits to left
  * @param  n1: Q15-formated first value
  * @param  n2: Q15-formated second value
  * @retval Q15-formated value that represents the division of n1 and n2
  */
inline q15_t q15_div(q15_t n1, q15_t n2){

  q31_t aux;
  aux = (q31_t)n1 << 15;

  return aux / (q31_t)n2;
}
```