

UNIVERSIDADE FEDERAL DE VIÇOSA  
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PATRÍCIA PONTES LOPES

**IMPLEMENTAÇÃO DE UM SIMULADOR *HARDWARE-IN-THE-LOOP*  
EM PLATAFORMA ARDUINO DUE PARA AUXÍLIO NO ENSINO DE  
CONTROLE**

VIÇOSA  
2019

PATRÍCIA PONTES LOPES

**IMPLEMENTAÇÃO DE UM SIMULADOR *HARDWARE-IN-THE-LOOP*  
EM PLATAFORMA ARDUINO DUE PARA AUXÍLIO NO ENSINO DE  
CONTROLE**

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 – Monografia e Seminário – e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Rodolpho Vilela Alves Neves

VIÇOSA  
2019

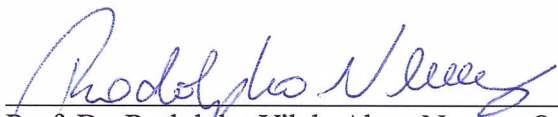
**PATRÍCIA PONTES LOPES**

**IMPLEMENTAÇÃO DE UM SIMULADOR *HARDWARE-IN-THE-LOOP* EM  
PLATAFORMA ARDUINO DUE PARA AUXÍLIO NO ENSINO DE  
CONTROLE**

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 - Monografia e Seminário e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

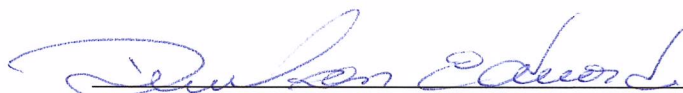
Aprovada em 11 de dezembro de 2019.

**COMISSÃO EXAMINADORA**



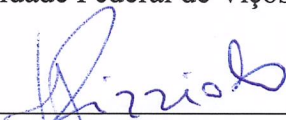
---

Prof. Dr. Rodolpho Vilela Alves Neves - Orientador  
Universidade Federal de Viçosa



---

Prof. Dr. Denílson Eduardo Rodrigues - Membro  
Universidade Federal de Viçosa



---

Prof. Dr. Tarcísio de Assunção Pizziolo - Membro  
Universidade Federal de Viçosa

## *Agradecimentos*

Agradeço ao professor Rodolpho, pela orientação e pela confiança em todo o processo de elaboração desta monografia.

Agradeço aos técnicos do LEE, Cristiano e Lúcio, pela ajuda durante a realização dos experimentos.

Agradeço ao meu amigo Leonardo, pelo apoio e pela confiança durante todo o processo de realização desta monografia, que foi essencial nos momentos de dificuldade. Agradeço ao meu amigo Lucas, pela ajuda a desbravar o mundo do Arduino. Agradeço também aos meus amigos Antônio e Matheus, pela ajuda na preparação da apresentação final.

## ***Resumo***

Este trabalho propõe a implementação e validação de um simulador *Hardware-in-the-loop* para aplicação em aulas práticas de Engenharia de Controle, utilizando como plataforma de desenvolvimento a placa Arduino Due. Para auxiliar o processo de implementação, são utilizados dois modelos para Estudo de Caso – um conversor *boost* e um inversor monofásico – que se assemelham a modelos tradicionalmente aplicados no ensino de Controle. A validação do simulador HIL é feita através da comparação dos seus resultados com simulações realizadas no *software* MATLAB e a sua ferramenta Simulink. O simulador HIL implementado não obteve os resultados esperados, sendo que os seus conversores digital/analógico não apresentaram bom desempenho. Os resultados mostraram que a plataforma Arduino Due não é capaz de processar modelos como forma implementada em simuladores HIL que necessitem de pequenos períodos de amostragem e que, para esses casos, é indicada a utilização de outras plataformas.

Palavras-chave: *Hardware-in-the-loop*, Arduino Due, Ensino, Controle.

## *Abstract*

This work proposes the implementation and validation of a Hardware-in-the-loop simulator for use in practical classes of Control Engineering, using the Arduino Due board as its development platform. To assist the implementation process, two models are used as Case Studies – a boost converter and a single-phase inverter – which are similar to models traditionally applied in Control Education. The validation of the HIL simulator is done by comparing its results to simulations performed in the MATLAB software and its Simulink toolbox. The implemented HIL simulator did not obtain the expected results, as its digital/analogue converters did not performed well. The results show that the Arduino Due platform is not capable of process models implemented as HIL simulators that require short sampling periods of time and, therefore, the use of other platforms is indicated in such cases.

Keywords: Hardware-in-the-loop, Arduino Due, Teaching, Control.

## *Lista de Figuras*

Figura 1 – Exemplo de uma Simulação HIL. ....	11
Figura 2 – Uma placa Arduino Due.....	16
Figura 3 – Gerador de funções ATTEN ATF10B. ....	20
Figura 4 – Osciloscópio Minipa MVB-DSO.....	21
Figura 5 – Pseudocódigo da interrupção do HIL.....	22
Figura 6 – Modelo do conversor <i>boost</i> . ....	24
Figura 7 – Modelo do conversor <i>boost</i> com a chave fechada ( $D=1$ ) ( <i>a</i> ) e chave aberta ( $D=0$ ) ( <i>b</i> ). ....	24
Figura 8 – Modelo do inversor monofásico conectado à rede.....	25
Figura 9 – Modelo simplificado do inversor monofásico conectado à rede.....	25
Figura 10 – Pseudocódigo da leitura serial dos valores de saída do sistema.....	26
Figura 11 – Pseudocódigo da rotina de determinação do tempo de amostragem mínimo. ....	27
Figura 12 – Montagem da simulação HIL para o conversor <i>boost</i> . ....	29
Figura 13 – Montagem da simulação HIL para o inversor monofásico. ....	29
Figura 14 – Resultados das simulações do conversor <i>boost</i> ( $I_L$ ). ....	33
Figura 15 – Resultados das simulações do conversor <i>boost</i> ( $V_c$ ). ....	33
Figura 16 – Resultados das simulações do inversor monofásico ( $i_{ac}$ ). ....	34
Figura 17 – Resultados das simulações do inversor monofásico ( $i_{dc}$ ). ....	34
Figura 18 – Teste dos resultados digitais da simulação HIL. ....	35
Figura 19 – Teste dos resultados do DAC da simulação HIL. ....	36

## *Lista de Tabelas*

Tabela 1 – Principais especificações técnicas da Arduino Due.....	17
Tabela 2 – Atribuição do contador de <i>timer clock</i> .....	23
Tabela 3 – Parâmetros do conversor <i>boost</i> .....	24
Tabela 4 – Parâmetros do inversor monofásico.....	25
Tabela 5 – Valores mínimo e máximo atingidos pelas variáveis de saída. ....	31



# Sumário

1	Introdução.....	10
1.1	<i>Hardware-in-the-loop</i> .....	10
1.1.1	Panorama Histórico.....	12
1.1.2	<i>Digital Real-Time Simulators</i> .....	13
1.2	HIL no Ensino da Engenharia de Controle.....	13
1.3	Objetivos.....	15
2	Materiais e Métodos.....	16
2.1	Materiais.....	16
2.1.1	Arduino Due.....	16
2.1.2	MATLAB e Simulink.....	18
2.1.3	Gerador de Funções.....	19
2.1.4	Osciloscópio.....	20
2.2	Métodos.....	21
2.2.1	Desenvolvimento da Interrupção.....	21
2.2.2	Estudos de Caso.....	23
2.2.3	Conversão dos Valores Digitais para Analógicos.....	26
2.2.4	Definição do Período de Amostragem.....	27
2.2.5	Simulação HIL.....	28
2.2.6	Validação do Simulador HIL.....	30
3	Resultados e Discussão.....	31
4	Conclusões.....	38
	Referências Bibliográficas.....	39
	Apêndice A – Código da Leitura Serial dos Valores de Saída.....	43
	Apêndice B – Código da Leitura do Tempo de Execução.....	44
	Apêndice C – Código da Simulação HIL.....	46

# ***1 Introdução***

Já é amplamente reconhecido que a experimentação em laboratório é uma forma importante de se entender os conceitos abstratos de engenharia e de relacioná-los com os desafios da construção de projetos. Isto é particularmente verdadeiro no ensino da Engenharia de Controle (WELLSTEAD, 1990; GREGA, 1999). A dificuldade no aprendizado e a desmotivação dos alunos muitas vezes provêm da falta de interligação da teoria com exemplos realistas (ONEL, EVANS, *et al.*, 2010). Portanto, a aplicação de exercícios e atividades em laboratório que ilustrem a teoria no domínio de situações práticas é um aspecto importante do ensino de controle de sistemas (ZILOUCHIAN, 2003).

Entretanto, as instalações de plantas físicas podem ser problemáticas em termos de alocação de fundos, infraestrutura, manutenção, segurança e portabilidade (SOBOTA, GOUBEJ, *et al.*, 2019). Uma abordagem que permita que os alunos experimentem a elaboração de todas as etapas do projeto do sistema de controle, eliminando as desvantagens das plantas físicas, é a utilização de simuladores em tempo real com entradas e saídas físicas, cuja tecnologia é denominada “*Hardware-in-the-loop*” (GREGA, 1999).

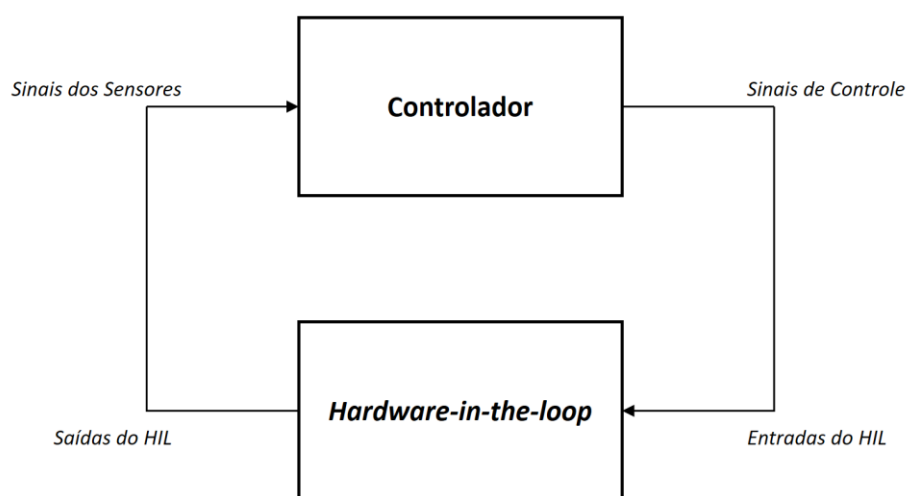
## ***1.1 Hardware-in-the-loop***

*Hardware-in-the-Loop* (HIL) é uma técnica concebida para realizar testes de sistemas embarcados de forma abrangente, econômica e repetível (LEDIN, 1999). No entanto, a técnica evoluiu rapidamente para se tornar uma ferramenta que suporta a modelagem de sistemas e subsistemas, bem como para a síntese de sistemas, devido à combinação sinérgica de prototipagem física e virtual (simulada) (RENAUX, LINHARES e RENAUX, 2017). Portanto, o HIL pode ser definido como uma técnica onde subsistemas reais, réplicas físicas de subsistemas ou subsistemas simulados são combinados para emular um sistema de *loop* fechado (RENAUX, LINHARES e RENAUX, 2017).

Na simulação HIL, ocorre a substituição do sistema físico pela sua representação matemática, sendo que o modelo matemático é simulado em um simulador em tempo real onde as medidas do sistema são convertidas em sinais físicos, estes enviados para o dispositivo de *hardware* a ser testado (CUPELLI, RICCOBONO, *et al.*, 2018). Um ou mais componentes reais são estudados em interação com componentes simulados em tempo real, sendo que em um experimento realista de HIL, o *hardware* sob teste não consegue distinguir o ambiente real do seu equivalente emulado (MONTI, D'ARCO e DESHMUKH, 2008). Por exemplo, partes de um veículo (placas eletrônicas, sensores etc.) podem ser testadas em diferentes condições de operação com um modelo simulado do veículo onde serão instaladas (WAEELTERMANN, MICHALSKY e HELD, 2013). A Figura 1 apresenta um exemplo de sistema de simulação HIL em diagrama de blocos.

Tipicamente, as simulações consistem em um conjunto de entradas, alguns algoritmos e rotinas para modelar o comportamento do sistema e um conjunto de saídas. Quando o HIL é empregado como parte de um sistema de simulação maior, a simulação pode consistir em mais do que apenas uma plataforma de computação *host* para executar o código de simulação. Os componentes de *hardware* que dão forma à parte da arquitetura da simulação podem servir a uma variedade das funções dentro do sistema de simulação maior (BURBANK, KASCH e WARD, 2011).

Figura 1 – Exemplo de uma Simulação HIL.



Fonte: LEDIN, 1999.

### 1.1.1 Panorama Histórico

Isermann, Schaffnitt e Sinsel (1999, p. 645) afirmam que os primórdios da técnica de simulação em HIL foram provavelmente relacionados aos simuladores de voo, datados em 1936. Bacic (2005, p. 3194) complementa que a partir da década de 1960, a simulação HIL começou a ser amplamente desenvolvida e utilizada no ensaio de sistemas de orientação de mísseis, sendo que a NASA foi um dos responsáveis tanto pela aplicação de HIL em sistemas de orientação de mísseis quanto no desenvolvimento de tecnologia de aeronaves altamente manobráveis.

Mais recentemente, nos últimos 30 anos, a simulação de HIL ganhou popularidade na indústria automotiva. A simulação de *hardware-in-the-loop* é usada para o projeto de sistemas de frenagem antibloqueio (ABS), sistemas de controle de tração (TCS) e sistemas de suspensão (WAEFTERMANN, MICHALSKY e HELD, 2013). Outras duas áreas importantes da indústria em que as simulações HIL são amplamente utilizadas e são consideradas metodologias eficazes para testar sistemas são as aplicações aeroespaciais e de controle de sistemas de potência (BACIC, 2005). Além dos campos clássicos já mencionados, a técnica HIL tem auxiliado o desenvolvimento em diversos campos da ciência, sendo alguns deles: manufatura (HARRISON, TILBURY e YUAN, 2012), *smart grids* (EBE, IDLBI, *et al.*, 2018), veículos autônomos (CHEN, CHEN, *et al.*, 2018), Indústria 4.0 (LIMA, COSTA, *et al.*, 2019), mobilidade urbana (NEMTANU, COSTEA e OBREJA, 2017), construção civil (HUANG, WANG, *et al.*, 2018), medicina (QUESADA, ROJAS, *et al.*, 2019) e educação (SOBOTA, GOUBEJ, *et al.*, 2019).

As simulações HIL vêm sendo aplicadas nos domínios de pesquisa e desenvolvimento de produtos, e foi demonstrado que esta técnica pode reduzir drasticamente o tempo e os custos no processo de concepção e ensaio, assim como traz segurança no processo de testes (MONTI, D'ARCO e DESHMUKH, 2008).

### 1.1.2 *Digital Real-Time Simulators*

As simulações HIL requerem a utilização de máquinas simuladoras em tempo real, que ofereçam um poder computacional suficientemente rápido para fornecer as características comportamentais exatas quando comparado com o sistema físico original. Para atingir a precisão, os modelos devem ser simulados um passo de cada vez, sincronizados com o *real-world clock* (MENGHAL e JAYA LAXMI, 2012). Essas máquinas, utilizadas na técnica HIL, são denominadas *Digital Real-Time Simulators (DRTS)*.

Nos últimos 40 anos, o desenvolvimento de DRTS no mercado se ampliou. Isso se deve tanto ao aumento demanda das indústrias e centros de pesquisa por máquinas simuladoras quanto à ocorrência de uma revolução na informática (NOUREEN, SHAMIM, *et al.*, 2017). Visto que as simulações em tempo real são baseadas em passos de tempo discreto, em que o simulador resolve as equações do modelo de forma sucessiva, a revolução na informática veio a possibilitar passos de tempo discreto cada vez menores, melhorando a *performance* dos DRTS (MENGHAL e JAYA LAXMI, 2012).

*Digital Real-time Simulators* começaram a ser produzidos em escala comercial desde o a década de 1990. Apesar de terem decaído ao longo do tempo, os preços de tais simuladores ainda são bastante elevados para clientes que não sejam grandes indústrias e grande centros de pesquisa. No passado, os custos típicos de *Digital Real-Time Simulators* eram na faixa de 100 mil a 1 milhão de euros, e atualmente já é possível encontrar no mercado alguns modelos na faixa de 10 mil a 20 mil euros (RENAUX, LINHARES e RENAUX, 2017). De acordo com Noreen *et al.* (2017, p. 266), os expoentes do comércio de *Digital Real-Time Simulators* são as empresas: RTDS Technologies Inc., OPAL-RT Technologies, dSPACE e Typhoon HIL Inc.

## 1.2 *HIL no Ensino da Engenharia de Controle*

Sala e Bondia (2006, p. 128) concluíram em seu trabalho que a aplicação de simulações HIL em cursos de Engenharia de Controle apresenta resultados bastante satisfatórios, que assim é alcançada uma experiência mais realista do estudante, sem a necessidade de múltiplas peças caras de equipamentos de processo reais. Segundo os autores, a simulação HIL é um passo intermediário entre a simulação de controlador puro (por exemplo, no ambiente Simulink) e o

controle de uma planta real com PID's industriais, PLC's e/ou *software* de controle em tempo real, pois a simulação HIL oferece a mesma interface entrada-saída que uma planta real e, portanto, os alunos devem projetar controladores e não a simulação deles.

Sobota *et al.* (2019, p. 68) destacam que, da perspectiva do educador, as simulações HIL seriam a abordagem ideal para a educação de controle, pois a manutenção e os riscos são reduzidos e a repetibilidade e a sustentabilidade são garantidas, enquanto o funcionamento dos modelos físicos não só coloca problemas com a segurança dos alunos durante os laboratórios, mas também requer a manutenção dos modelos e seus reparos (por exemplo, as bombas ficam presas, os rolamentos desgastam-se, os motores queimam-se etc.).

A substituição de plantas físicas por seus modelos simulados em HIL também apresenta as seguintes vantagens no ensino da Engenharia de Controle: qualquer sistema pode ser simulado, não dependendo da existência de um circuito RLC equivalente no laboratório; os parâmetros do modelo estudado podem ser customizados, de forma que cada estudante tenha que desenvolver o controle para um modelo único, evitando assim problemas de plágio (SOBOTA, GOUBEJ, *et al.*, 2019).

Mesmo existindo a comercialização de plataformas HIL com enfoque ao ensino, a faixa de preço de milhares de euros ainda é elevada para a maioria das instituições de ensino e pesquisa universitárias (RENAUX, LINHARES e RENAUX, 2017). Para uso educacional, que normalmente requer uma unidade de equipamento para cada grupo de 2 ou 3 alunos em um laboratório de ensino, a aquisição de apenas uma unidade de simulador, portanto, não se faz suficiente para a demanda (SALA e BONDIA, 2006).

Além disso, os simuladores comerciais são limitados quanto à portabilidade, devido ao elevado custo e ao tamanho da infraestrutura, impossibilitando que os estudantes ampliem a sua experiência com o conteúdo ao realizar experimentos extraclasse (LEE, PARK, *et al.*, 2015). Uma boa maneira de fazer uma plataforma de *hardware* com baixo custo e portabilidade é dar-lhe a interface de comunicação USB e código aberto. E nesta vertente, várias abordagens para a educação e pesquisa ao redor do mundo estão atualmente utilizando plataformas de *hardware* de código aberto em suas atividades (LEE, PARK, *et al.*, 2015).

Com a disponibilidade de muitos kits de desenvolvimento programáveis de baixo custo no mercado, os projetos envolvendo simuladores em tempo real agora podem fazer parte da

realidade de muitos laboratórios de ensino. Shi e Gan (2016, p. 6280-6283) agruparam os principais *hardwares* embarcados de código aberto que podem ser utilizados para o fim de simulação em tempo real, comparando-os em termos de custo, arquitetura de *hardware*, metodologia de desenvolvimento e recursos de *software*. Os *hardwares* mais populares incluídos foram os das famílias Arduino, Raspberry Pi, Texas Instruments e National Instruments (SHI e GAN, 2016).

### 1.3 Objetivos

Este trabalho tem por objetivo geral implementar e validar um simulador *Hardware-in-the-loop* para a aplicação no ensino de Controle em aulas de laboratório, baseado em materiais de baixo custo, considerável portabilidade e código aberto.

Para alcançar o objetivo geral, foram estipulados os seguintes objetivos específicos:

- Selecionar a placa de desenvolvimento que será utilizada para implementar a simulação HIL;
- Desenvolver o código das rotinas de interrupção que irão amostrar as entradas, simular os modelos e oferecer as saídas;
- Selecionar os modelos do Estudo de Caso para teste e validação da plataforma HIL;
- Implementar os modelos discretizados na linguagem de programação relacionada à placa;
- Desenvolver uma rotina para definir a taxa de amostragem adequada para cada modelo utilizado;
- Simular os modelos nos ambientes MATLAB/Simulink;
- Simular os modelos no HIL;
- Comparar os dados das simulações virtuais e do HIL.

## 2 *Materiais e Métodos*

### 2.1 *Materiais*

Nesta seção, são explicitados os materiais utilizados para a elaboração do projeto de implementação de um simulador HIL, baseado em materiais já existentes no inventário do Laboratório de Engenharia Elétrica da UFV (LEE).

#### 2.1.1 **Arduino Due**

Para este trabalho, foi escolhida a placa Arduino Due (Figura 2), que é uma plataforma de *hardware* de código aberto lançada no mercado em 2012 e desenvolvida por uma organização bastante popular mundialmente (DEUTSCH, 2012). A placa Arduino Due possui o microcontrolador Atmel AT91SAM3X8E, com a arquitetura de *core* ARM Cortex-M3.

Figura 2 – Uma placa Arduino Due.



Fonte: ARDUINO, 2019.

Essa é a primeira placa da linha Arduino baseada em microcontrolador com *core* de 32-*bits*, o que significa que os registradores são de 32 *bits* e que operações de 4 *bytes* são realizadas em um ciclo de *clock*. Isso representa um avanço em relação aos demais componentes da linha



Arduino, que são construídos com microcontroladores ATMEGA cuja arquitetura interna possui apenas 8 *bits* (LIMA, 2014).

Equipada com periféricos mais rápidos que as demais placas da família Arduino, a Due possui um clock principal de 84 MHz e uma memória *flash* de 512 KB. Além disso, ela conta com 54 pinos de entrada/saída – sendo que 12 destes podem ser utilizados como saídas PWM –, 12 entradas analógicas (ADC), 2 saídas analógicas (DAC), com resolução de 12 *bits* nas entradas e saídas, 2 portas de comunicação USB (programável e nativa) (ARDUINO, 2019). Demais especificações técnicas da placa estão dispostas na Tabela 1.

Tabela 1 – Principais especificações técnicas da Arduino Due.

Microcontrolador	AT91SAM3X8E
Tensão de operação	3,3 V
Tensão de entrada (recomendada)	7-12 V
Tensão de entrada (limites)	6-16 V
Pinos de E/S digital	54 (dos quais 12 fornecem saída PWM)
Pinos de entrada analógica	12
Pinos de saída analógica	2 (DAC)
Corrente de saída DC total	130 mA
Corrente CC para pino de 3.3V	800 mA
Corrente CC para pino de 5V	800 mA
Memória <i>flash</i>	512 KB
SRAM	96 KB (dois bancos: 64 KB e 32 KB)
Velocidade do <i>clock</i>	84 MHz
Comprimento	101,52 mm
Largura	53,3 mm
Peso	36 g

Fonte: ARDUINO, 2019.

As razões pelas quais a placa Arduino Due foi escolhida para a implementação do HIL proposto estão explicitadas a seguir:

- A plataforma Arduino é uma bastante conhecida e possui uma grande base de usuários, existindo diversos fóruns de discussão e suporte *on-line* pela comunidade;
- Arduinos são frequentemente usados no ensino de outras disciplinas em Engenharia Elétrica, e isso significa que é provável que os alunos já estejam familiarizados com a plataforma;

- A programação é tipicamente realizada em programação de alto nível, com linguagem C ou C++, em um IDE (ambiente de desenvolvimento integrado) aberto oferecido para *download* gratuito pela própria organização Arduino (a versão atual do IDE é a 1.8.10). E o *upload* do código é realizado rapidamente via USB na própria interface do IDE;
- A frequência de *clock* de 84 MHz proporciona uma amostragem e um processamento consideravelmente mais rápidos do que os realizados pelas demais placas da linha Arduino, que contam com *clock* máximo de 16 MHz (DEUTSCH, 2012);
- A Due é o primeiro Arduino a apresentar conversores digital para analógico (DAC), que são essenciais para a realização do projeto proposto;
- O custo da placa Arduino Due é relativamente baixo em comparação aos demais *hardwares* com características semelhantes (JALDÉN, MORENO e SKOG, 2018). No mercado nacional, é possível adquiri-la por valores na faixa de R\$100,00; e,
- Por último, o Laboratório de Engenharia Elétrica (LEE) da UFV já possui algumas unidades da Arduino Due, fato decisivo na escolha desta placa para o projeto, que assim se torna viável.

### 2.1.2 MATLAB e Simulink

Com o nome derivado da frase “laboratório matricial”, o MATLAB foi lançado comercialmente em 1984, fornecendo uma ferramenta interativa de programação em C para desenvolvimento de problemas científicos e de engenharia e, mais geralmente, para desenvolvimento de áreas onde cálculos numéricos significativos devem ser realizados (MOLER, 2018).

Este é provavelmente o *software* de análise numérica comercial mais bem-sucedido do mundo, que permite manipulação de estruturas matriciais, um vasto número de poderosas rotinas embutidas que estão em constante crescimento e desenvolvimento, poderosos recursos gráficos bidimensionais e tridimensionais, um sistema de *scripting* que permite aos utilizadores desenvolver e modificar o *software* para as suas próprias necessidades, coleções de funções,

chamadas *toolboxes*, que podem ser adicionadas às instalações do núcleo MATLAB e interface com programas escritos em outras linguagens (LINDFIELD e PENNY, 2019).

Simulink é uma *toolbox* do MATLAB que representa um ambiente de programação gráfica em blocos, utilizada para modelar, simular e analisar sistemas dinâmicos. Com essa ferramenta, é possível construir modelos a partir do zero ou modificar modelos existentes para atender às necessidades do usuário. O Simulink suporta sistemas lineares e não lineares, modelados em tempo contínuo, tempo amostrado ou um híbrido de ambos. Os sistemas também podem ter várias taxas, com diferentes partes que são amostradas ou atualizadas em diferentes taxas (HAHN e VALENTINE, 2019).

O MATLAB e a sua *toolbox* Simulink foram escolhidos para este trabalho por serem *softwares* capazes de simular virtualmente os modelos que também serão simulados na plataforma HIL a ser implementada, de forma a posteriormente processar os dados adquiridos e compará-los com seus próprios resultados. Além disso, o Laboratório de Engenharia Elétrica (LEE) da UFV possui licenças do MATLAB e Simulink, o que contribui para a viabilidade do projeto.

### 2.1.3 Gerador de Funções

Os geradores de funções são dispositivo que podem produzir vários padrões de tensão em uma variedade de frequências e amplitudes. Eles são fontes de sinais que fornecem uma tensão especificada aplicada durante um determinado período de tempo, como um sinal de onda senoidal, onda triangular, onda quadrada etc. O seu principal uso é fornecer tensões de entrada para sistemas (PENGRAS, 2007).

Neste trabalho, foi utilizado o gerador de funções disponível no LEE da UFV, cujo modelo é o ATF10B da ATTEN Electronics. A Figura 3 apresenta a imagem do modelo da mesma família, o ATF20B, cujo diferencial é a extensão do espectro de frequências geradas: 40 mHz – 20 MHz para o “20B” contra 40 mHz – 10 MHz do “10B”.

Figura 3 – Gerador de funções ATTEN ATF10B.



Fonte: ATTEN ELECTRONICS CO.

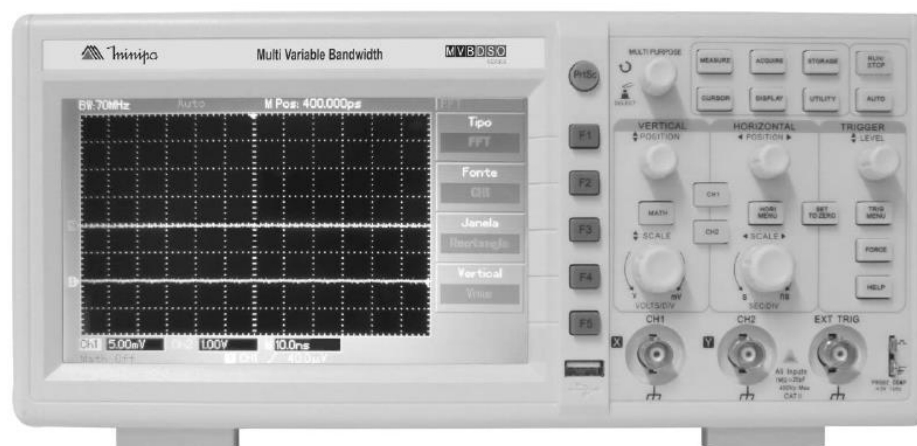
### 2.1.4 Osciloscópio

Osciloscópios são um tipo de analisador de sinais, eles mostram ao experimentador uma imagem do sinal, geralmente na forma de um gráfico de tensão *versus* tempo. O usuário pode então estudar esta imagem para aprender sobre a sua amplitude, frequência e forma geral do sinal, que pode depender da física que está sendo explorada no experimento (PENGRÁ, 2007).

Osciloscópios exibem uma representação bidimensional de uma ou mais diferenças potenciais, normalmente de tensão no eixo y em relação ao tempo no eixo x. O gráfico é desenhado na tela por um ponto que responde a uma tensão variável no tempo, movendo-se na direção vertical enquanto escaneia repetidamente da esquerda para a direita, tornando o osciloscópio útil para exibir sinais periódicos. O padrão resultante é chamado de “traço” ou “forma de onda”.

Para este trabalho, foi utilizado o osciloscópio digital da marca Minipa, modelo MVB-DSO (Figura 4), disponível no LEE da UFV. Dentre os osciloscópios disponíveis no LEE, este foi o modelo escolhido porque ele apresenta a melhor resolução de amostragem de pontos e possui a opção de exportação dos dados para um dispositivo USB, de forma a permitir a sua análise no *software* MATLAB.

Figura 4 – Osciloscópio Minipa MVB-DSO.



Fonte: MINIPA DO BRASIL LTDA., 2013.

## 2.2 Métodos

Esta seção apresenta o detalhamento da metodologia de desenvolvimento do projeto de implementação e validação do simulador HIL proposto.

### 2.2.1 Desenvolvimento da Interrupção

Um simulador HIL, assim como qualquer sistema que opera em tempo real, deve ser capaz de fornecer os resultados corretos dentro do intervalo de amostragem exigido. Assim, é necessária a implementação de uma interrupção a cada período de amostragem, de forma a garantir que a atividade do microcontrolador será periodicamente dedicada à execução do código inserido dentro da rotina de interrupção (TOULSON e WILMSHURST, 2017).

A implementação da interrupção do simulador HIL foi realizada no IDE da Arduino, versão 1.8.10, através da manipulação dos registradores de acesso aos periféricos de *clock*, de contadores e de interrupções. A lógica está disposta em pseudocódigo na Figura 5.

Figura 5 – Pseudocódigo da interrupção do HIL.

```

1 /*--> Declaração das seguintes constantes:
2     Período de amostragem;
3     Parâmetros do sistema simulado;
4 --> Declaração das seguintes variáveis:
5     Status altobaixo do PWM;
6     Variáveis auxiliares nos cálculos;
7     Saídas do sistema; */
8
9 void setup()
10 { /*
11 --> Configuração da resolução de 12bits (0-4095) da saída analógica DAC;
12 --> Configuração de quais pinos são entrada e saída;
13 --> Habilitação do timer clock no controlador de gestão de energia (PMC):
14     Dasabilitar a proteção contra escrita nos registradores do PMC;
15     Habilitar o clock periférico: TC0;
16 --> Configuração do clock periférico do TC:
17     Selecionar o Timer Counter: TC0;
18     Selecionar o canal: 0;
19     Selecionar a fonte de clock interno: TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC | TC_CMR_TCCLKS_TIMER_CLOCK1;
20 --> Configuração do número de ticks do TIMER0 para gatilho da interrupção: TC_SetRC(TC0, 0, xxxx);
21 --> Habilitação do TC0;
22 --> Habilitar as interrupções de tempo no TC0: TC0->TC_CHANNEL[0].TC_IER=TC_IER_CPCS;
23 --> Adição da interrupção no Nested Vectored Interrupt Controller;
24 */ }
25
26 void loop()
27 {
28     /* vazio */
29 }
30
31 void TC0_Handler() // Função chamada a cada interrupção:
32 { /*
33 --> Manipulação do registrador TC e limpeza do seu status para que a interrupção dispare novamente;
34 --> Leitura da entrada do PWM e a aloca em uma variável alta (1) ou baixa (0);
35 --> Implementação das equações de diferença do modelo simulado;
36 --> Conversão dos valores das saídas digitais em bits de 0 a 4095;
37     Escrita dos bits das saídas digitais nos pinos do conversor DA.
38 */ }

```

Fonte: Próprio autor.

É importante explicitar o processo de seleção do *clock* interno para o disparo da interrupção, pois alguns registradores relacionados deverão ser alterados para alcançar as exigências de período de amostragem de cada modelo em simulação. De acordo com as informações do *datasheet* do microcontrolador (ATMEL CORPORATION, 2015), uma forma de se controlar o *clock* do contador é através do modo *waveform* (*TC\_CMR\_WAVE*), em que o canal do *Time Counter* (TC) gera sinais de pulsos a serem comparados com o sinal do oscilador RC (*TC\_CMR\_WAVSEL\_UP\_RC*).

Para configurar o tempo entre os gatilhos da interrupção, é necessário definir o número de ciclos de máquina para que ocorra o estouro do contador selecionado. Então, seleciona-se a escala do *clock* interno para o contador (conforme valores da Tabela 2, em que MCK corresponde ao *master clock* de 84 MHz e SLCK ao *slow clock* de 32 kHz), de modo a ter a frequência do contador, que será multiplicada pelo intervalo de tempo da interrupção desejada,

obtendo-se o número de ciclos necessários para o contador estourar. Na linha 20 do pseudocódigo (Figura 5), o termo “xxxx” de será substituído pelo número de ciclos demandado por cada modelo estudado.

Tabela 2 – Atribuição do contador de *timer clock*.

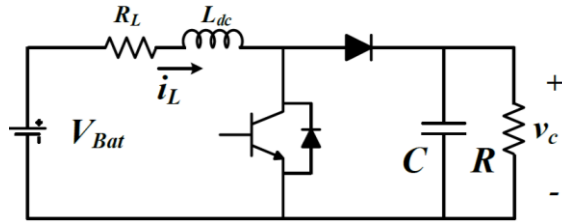
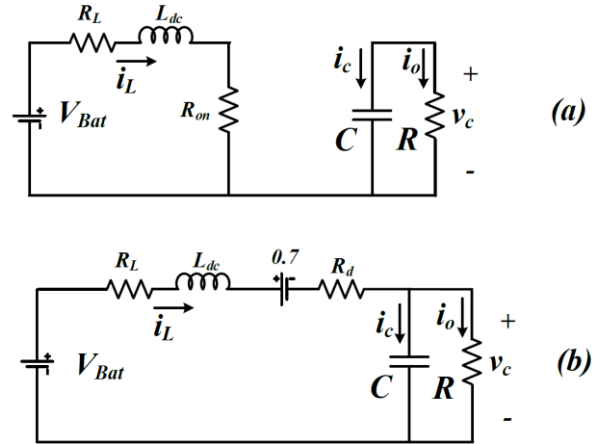
Nome	Definição
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5	SLCK

Fonte: ATMEL CORPORATION, 2015.

### 2.2.2 Estudos de Caso

A fim de auxiliar a implementação e verificação do simulador HIL, fazem-se necessários Estudos de Caso de sistemas reais, que ofereçam parâmetros numéricos de entrada para a simulação. Neste trabalho, foram estudados dois sistemas comumente aplicados em aulas práticas de Engenharia de Controle, que são um conversor *boost* e um inversor de frequência monofásico. Visto que a modelagem e discretização de sistemas não fazem parte do escopo deste trabalho, foram utilizados os parâmetros e as equações de diferença desenvolvidos por Bastos *et al.* (2019).

A Figura 6 apresenta a topologia básica de um conversor *boost*, que apresenta dois modos de operação: quando a chave está aberta ( $D = 0$ ) (Figura 7.b) e quando a chave está fechada ( $D = 1$ ) (Figura 7.a). Neste modelo,  $L_{dc}$  é a indutância,  $R_L$  representa as perdas indutivas,  $C$  é a capacitância de saída,  $R$  é a carga no terminal de saída,  $R_{on}$  representa a resistência da chave semicondutora e  $R_d$  a resistência do diodo. A grandeza de entrada deste sistema é o valor  $D$  (0 baixo ou 1 alto) que provém do sinal de PWM, já as grandezas de saída são a tensão do capacitor  $v_c$  e a corrente da indutância  $i_L$ . As equações do modelo discretizado, que serão implementadas dentro da função de interrupção na Arduino Due, são definidas por (1) a (4) e os parâmetros do circuito são apresentados na Tabela 3.

Figura 6 – Modelo do conversor *boost*.Fonte: BASTOS, FUZATO, *et al.*, 2019.Figura 7 – Modelo do conversor *boost* com a chave fechada ( $D=1$ ) (a) e chave aberta ( $D=0$ ) (b).Fonte: BASTOS, FUZATO, *et al.*, 2019.

$$\frac{dI_L(k)}{dk} = (V_{bat} - R_L I_L(k-1) - (0.7 + R_d I_L(k-1) + v_c(k-1))(1-D) - D R_{on} I_L(k-1)) \frac{1}{L_{dc}} \quad (1)$$

$$\frac{dv_c(k)}{dk} = \frac{i_L(k-1)}{C} (1-D) - \frac{v_c(k-1)}{RC} \quad (2)$$

$$i_L(k) = i_L(k-1) + \frac{dI_L(k)}{dk} T_s \quad (3)$$

$$v_c(k) = v_c(k-1) + \frac{dv_c(k)}{dk} T_s \quad (4)$$

Tabela 3 – Parâmetros do conversor *boost*.

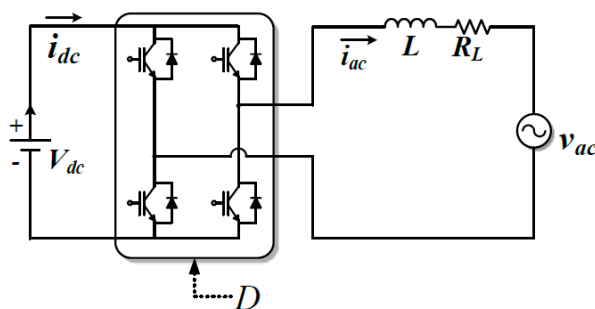
Símbolo	Quantidade
$L_{dc}$	3,8 mH
$R_L$	0,35 $\Omega$
$R_{on}$	0,30 $\Omega$
$R_d$	0,20 $\Omega$
$R$	36 $\Omega$
$C$	940 $\mu$ F
$V_{Bat}$	10 V
$f_{PWM}$	1 kHz

Fonte: BASTOS, FUZATO, *et al.*, 2019 (adaptado).



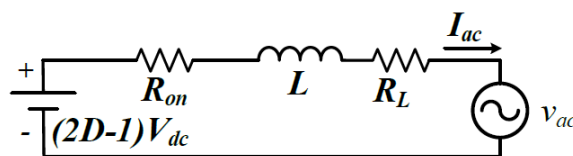
A Figura 8 apresenta a topologia de um inversor monofásico conectado à rede alternada e a Figura 9 apresenta seu modelo equivalente. Em ambas figuras,  $L$  representa o filtro de saída indutivo,  $R_L$  as perdas indutivas e  $v_{ac}$  a tensão monofásica da rede.  $V_{dc}$  é a tensão do *link* DC e  $R_{on}$  são as perdas parasitas dos semicondutores quando em condução. A grandeza de entrada deste sistema é o valor  $D$  (0 baixo ou 1 alto) que provém do sinal de PWM, e as grandezas de saída são a corrente do *link* DC  $i_{dc}$  e a corrente entregue à rede alternada  $i_{ac}$ . Utilizando o modelo simplificado (Figura 9), as equações do modelo discretizado, que serão implementadas dentro da função de interrupção na Arduino Due, são definidas por (5) a (7). E os parâmetros do circuito são apresentados na Tabela 4.

Figura 8 – Modelo do inversor monofásico conectado à rede.



Fonte: BASTOS, FUZATO, et al., 2019.

Figura 9 – Modelo simplificado do inversor monofásico conectado à rede.



Fonte: BASTOS, FUZATO, et al., 2019.

$$\frac{di_{ac}(k)}{dk} = (V_{dc}(2D - 1) - R_L i_{ac}(k - 1) - R_{on} i_{ac}(k - 1) - v_{ac}(k)) \frac{1}{L} \quad (5)$$

$$i_{ac}(k) = i_{ac}(k - 1) + \frac{di_{ac}(k)}{dk} T_s \quad (6)$$

$$i_{dc}(k) = i_{ac}(k)(2D - 1) \quad (7)$$

Tabela 4 – Parâmetros do inversor monofásico.

Símbolo	Quantidade
$L$	3,8 mH
$R_L$	13,5 $\Omega$
$R_{on}$	0,20 $\Omega$
$v_{ac}$	0 V
$V_{dc}$	20 $\Omega$
$f_{ac}$	370 Hz
$f_{PWM}$	2 kHz

Fonte: BASTOS, FUZATO, et al., 2019 (adaptado).

### 2.2.3 Conversão dos Valores Digitais para Analógicos

Outra tarefa necessária para a implementação da simulação HIL na Arduino Due é a conversão das respostas das equações do modelo em *bits*. Isso se deve ao fato de que o conversor digital/analógico (DAC), cuja resolução é de 12 *bits*, trabalha de forma a relacionar valores de 0 a 4095 em valores de tensão dentro de uma faixa aproximada de 1/6 a 5/6 da tensão de operação 3,3 V (ARDUINO SUPPORT FORUM, 2014).

Para tanto, torna-se necessário descobrir quais os valores mínimo e máximo atingidos por cada variável de saída do modelo simulado e, assim, relacioná-los de forma linear na faixa de 0 a 4095 *bits*. O pseudocódigo da lógica para obtenção desses valores está apresentado na Figura 10 e a íntegra do código para o modelo conversor *boost* do Estudo de Caso está disponibilizada no Apêndice A.

Figura 10 – Pseudocódigo da leitura serial dos valores de saída do sistema.

```
1 /*--> Declaração das seguintes constantes:
2     Período de amostragem;
3     Parâmetros do sistema simulado;
4 --> Declaração das seguintes variáveis:
5     Status altobaixo do PWM;
6     Variáveis auxiliares nos cálculos;
7     Saídas do sistema; */
8
9 void setup()
10 { /*
11     --> Habilitação da comunicação serial;
12     --> Todo o setup da interrupção;
13 */ }
14
15 void loop()
16 { /*
17     --> Leitura das saídas do modelo simulado;
18 */ }
19
20 void TCO_Handler() // Função chamada a cada interrupção:
21 { /*
22     --> Limpeza do status da interrupção;
23     --> Leitura da entrada do PWM e a aloca em uma variável alta (1) ou baixa (0);
24     --> Implementação das equações de diferença do modelo simulado;
25 */ }
```

Fonte: Próprio autor.

Empiricamente, foi constatado que a comunicação serial realizou-se de forma correta somente a partir de uma interrupção a partir de 200  $\mu$ s. Portanto, a configuração da rotina de interrupção, já explicada na subseção 2.2.1, deve ser realizada para um período de 200  $\mu$ s.

Após um tempo considerável de execução do programa, que abranja todos os eventos de variação das entradas a serem estudados, os valores de leitura da porta serial são copiados para o *software* MATLAB como vetores e, através das funções *max* e *min*, observam-se os limites atingidos pelas respostas das equações do sistema em questão. Então, esses valores são convertidos, por relação linear, para valores entre 0 e 4095 *bits*. A relação encontrada será adicionada ao código de simulação dos modelos, previamente à função de escrita dos *bits* no conversor digital/analógico.

## 2.2.4 Definição do Período de Amostragem

Para que se utilize o máximo do poder computacional do microcontrolador, com o menor período de amostragem ( $T_s$ ) possível, torna-se necessário implementar uma rotina que descubra o tempo necessário para o microcontrolador simular as equações de diferença do modelo em questão, de forma que  $T_s$  seja nem menor – impedindo a correta execução do modelo – ou maior do que o intervalo gasto para realizar uma iteração da rotina de interrupção. O pseudocódigo da rotina de determinação do tempo de execução do modelo está representado na Figura 11.

Figura 11 – Pseudocódigo da rotina de determinação do tempo de amostragem mínimo.

```

1 /*--> Declaração das seguintes constantes:
2     Período de amostragem;
3     Parâmetros do sistema simulado;
4 --> Declaração das seguintes variáveis:
5     Status alto/baixo do PWM;
6     Variáveis auxiliares nos cálculos;
7     Saídas do sistema;
8     Variáveis auxiliares para medição do tempo; */
9
10 void setup()
11 { /*
12     --> Configuração da resolução de 12bits (0-4095) da saída analógica DAC;
13     --> Configuração de quais pinos são entrada e saída;
14     --> Configuração da comunicação serial;
15 */ }
16
17 void loop()
18 { /*
19     --> Rotina de medição do intervalo de tempo:
20     Ler o tempo (função micros) no início da execução do modelo e guardar em uma variável;
21     Inserir as equações de diferença do modelo;
22     Ler o tempo no fim da execução do modelo e subtrair o valor inicial;
23     Adicionar o tempo de execução de uma iteração em uma variável incremental;
24 --> Leitura do intervalo do tempo de execução do modelo:
25     Calcular a média da variável incremental após um número determinado de iterações e imprimi-la;
26 */ }

```

Fonte: Próprio autor.

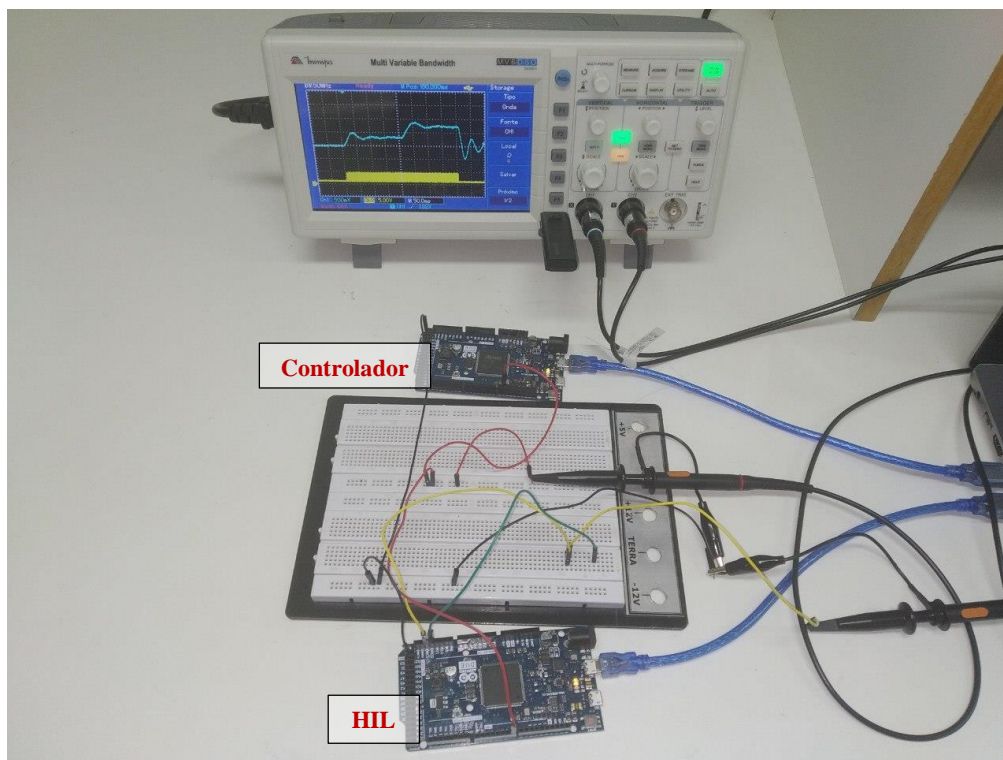
A estratégia de medição do tempo de execução do código foi baseada na utilização da função *micros()* presente na linguagem das placas Arduino. Esta função retorna o número de microssegundos passados desde que o programa atual começou a ser executado, sendo que a sua resolução na placa Due é de 1  $\mu$ s (ARDUINO, 2019). Assim, ao realizar a subtração entre os valores de resposta desta função no final e no início da execução do modelo, define-se o seu tempo de execução. É realizada a leitura da média do tempo de execução de 200 e 400 iterações a fim de suprimir eventuais erros de resolução e instabilidades da comunicação que poderiam ocorrer se a comunicação serial fosse realizada a cada iteração.

### 2.2.5 Simulação HIL

Todas as configurações do código de simulação HIL na Arduino Due foram realizadas e, assim, a implementação do sistema já pode ser realizada. Para tanto, são utilizados os parâmetros dos modelos de Estudo de Caso apresentados na Tabela 3 e na Tabela 4.

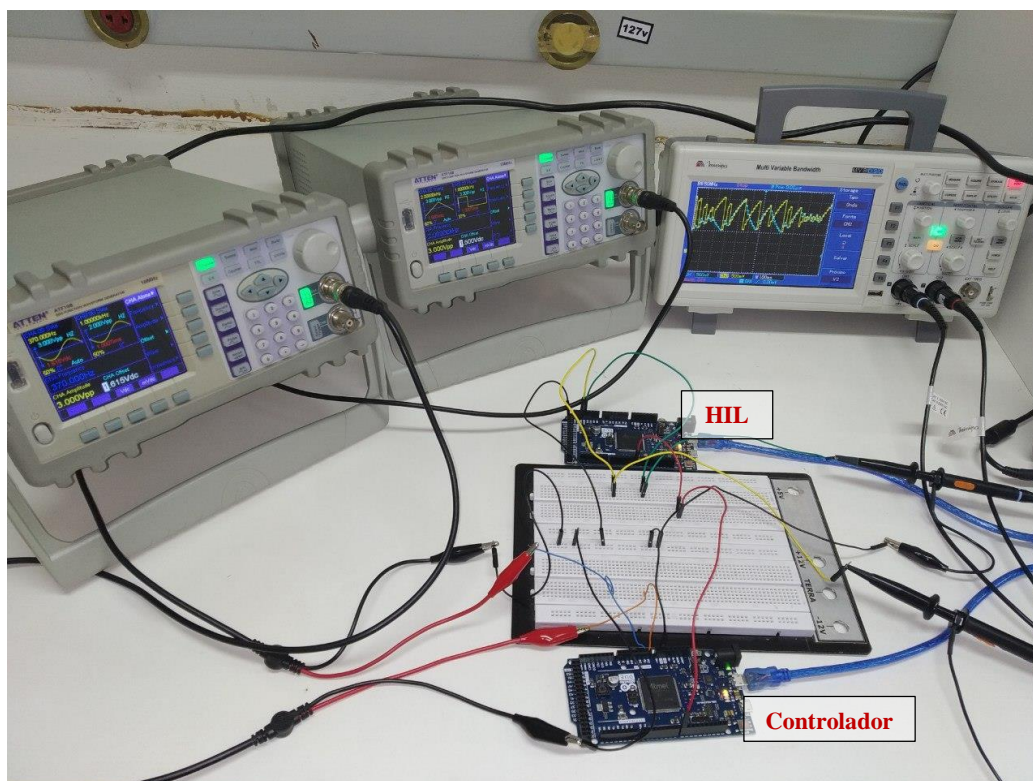
Para a simulação HIL do conversor *boost*, foi montada uma bancada no LEE da UFV com os seguintes materiais: uma Arduino Due atuando como simulador HIL, uma Arduino Due oferecendo o sinal de controle, um osciloscópio digital para observação e aquisição das tensões de saída via USB e uma *protoboard* para interligação dos equipamentos. O sinal de controle consiste em um PWM digital com frequência de 1 kHz, conforme Tabela 3, e variações de *duty-cycle* entre 0% - 30% - 50%. A Figura 12 apresenta a montagem para a simulação HIL deste modelo.

A simulação HIL do inversor monofásico foi montada com os mesmos materiais utilizados para a simulação anterior, com a adição de dois geradores de funções. Conforme os dados da Tabela 4, um gerador de funções foi configurado para a saída de uma senóide (moduladora) com frequência  $f_{ac}$  de 370 Hz e o outro para uma onda triangular (portadora)  $f_{PWM}$  de 2 kHz, sendo que o sinal digital de controle SPWM foi gerado pela comparação entre ambas ondas. Para não danificar a placa Arduino, os geradores de funções devem ser configurados com saídas de tensão de 0 a 3,3 V. A Figura 13 apresenta a montagem para a simulação HIL deste modelo.

Figura 12 – Montagem da simulação HIL para o conversor *boost*.

Fonte: Próprio autor.

Figura 13 – Montagem da simulação HIL para o inversor monofásico.



Fonte: Próprio autor.

### 2.2.6 Validação do Simulador HIL

A validação do simulador HIL implementado é realizada através da comparação gráfica entre os dados de tensão adquiridos pelo osciloscópio e os valores simulados através do *software* MATLAB e da sua *toolbox* Simulink.

No MATLAB, as equações de cada modelo estudado são simuladas numericamente em um *script* em função do vetor tempo e do vetor de estado do PWM importados do osciloscópio, com o mesmo período de amostragem  $T_s$  utilizado no simulador HIL. Visto que os dados adquiridos do osciloscópio são tensões, é necessário convertê-los para a escala apropriada. Os dados do PWM são convertidos para  $D=1$  quando em tensão alta e para  $D=0$  quando em tensão nula. Já a conversão das tensões das saídas de cada modelo para a escala apropriada é realizada através da inversão da relação de *bits* desenvolvida na subseção 2.2.3, em que o valor de 5/6 da tensão de operação da Arduino (3,3 V) corresponde a 4095 *bits* e 1/6 a 0 *bit*.

No Simulink, a simulação é realizada, segundo o mesmo  $T_s$ , com a implementação gráfica da topologia de cada modelo e com a utilização dos blocos *From Workspace* e *To Workspace* a fim de, respectivamente, importar o vetor de estado do PWM e exportar os vetores das variáveis de saída para o ambiente principal do MATLAB.

Então, no ambiente do MATLAB, os gráficos de cada grandeza são gerados e sobrepostos, de forma a comparar os resultados das simulações HIL com a simulação numérica do MATLAB e com a simulação em blocos do Simulink.

### 3 Resultados e Discussão

O primeiro passo para a obtenção dos resultados, conforme explicado na seção de métodos, consiste na realização de um procedimento, descrito na subseção 2.2.3, para determinar as equações de conversão das respostas digitais do simulador HIL em *bits* que serão lidos pelo conversor DAC. Assim, a aplicação da lógica do pseudocódigo para ambos os modelos de Estudo de Caso resultou nos valores mínimo e máximo para cada variável de saída, que estão dispostos na Tabela 5.

Tabela 5 – Valores mínimo e máximo atingidos pelas variáveis de saída.

Conversor <i>Boost</i>				Inversor Monofásico			
$I_L$ (A)		$V_c$ (V)		$i_{ac}$ (A)		$i_{dc}$ (A)	
Mínimo	Máximo	Mínimo	Máximo	Mínimo	Máximo	Mínimo	Máximo
-1,99	3,89	0	20,05	-1,49	1,51	-0,46	1,52

Fonte: Próprio autor.

A partir dos valores dispostos na Tabela 5, foram desenvolvidas as equações (8) a (11) que, respectivamente, convertem linearmente os valores da corrente do indutor  $I_L$ , da tensão no capacitor  $V_c$  do conversor *boost*, da corrente  $i_{ac}$  entregue à rede e da corrente  $i_{dc}$  no *link* DC do inversor monofásico. É importante ressaltar que a relação foi feita deixando uma margem de *bits* a menos que os limites 0 e 4095, a fim de evitar o estouro de *bits* no DAC caso haja pequenas instabilidades no sinal de controle que não ocorreram durante a aquisição dos dados iniciais.

$$i_{L\_bits} = (i_L + 2,4) * 640 \quad (8)$$

$$V_{c\_bits} = V_c * 200 \quad (9)$$

$$i_{ac\_bits} = (i_{ac} + 1,6) * 1300 \quad (10)$$

$$i_{dc\_bits} = (i_{dc} + 1) * 1600 \quad (11)$$

O segundo passo para a obtenção dos resultados consiste na aplicação do método descrito na subseção 2.2.4, de forma a descobrir o tempo de execução de cada modelo e, assim, ser possível configurar a interrupção do simulador HIL de acordo com o menor período de

amostragem possível. O Apêndice B apresenta a íntegra do código utilizado para este processo com o modelo do conversor *boost*, desenvolvido de acordo com o pseudocódigo da Figura 11. O tempo de execução encontrado para o código do conversor *boost* foi de 45  $\mu$ s, enquanto o tempo de execução encontrado para o código do inversor monofásico foi de 35  $\mu$ s, correspondendo ao período de amostragem  $T_s$  a ser aplicado em cada caso. É importante ressaltar que esses valores  $T_s$  já apresentam uma margem em relação ao valor real do tempo de execução, visto que a função *micros()* utilizada para aquisição do tempo possui o seu próprio tempo de execução dentro da linha de código que realiza a subtração dos valores final e inicial.

Então, o próximo passo é a realização da simulação HIL, de acordo com a descrição e as montagens apresentadas na subseção 2.2.5. Para a configuração das interrupções de cada modelo, é necessário o cálculo do número de ciclos do *Timer Counter* TC0 que dispara a interrupção e isso é realizado multiplicando-se o respectivo período de amostragem  $T_s$  pela frequência de *clock* do *Timer Counter* TC0, 42 MHz (MCK/2). Assim, a interrupção para a simulação HIL do conversor *boost* acontece a cada 1890 *ticks* do TC0, e a do inversor monofásico a cada 1470 *ticks* do TC0. O Apêndice C apresenta a íntegra do código da simulação HIL para o conversor *boost*.

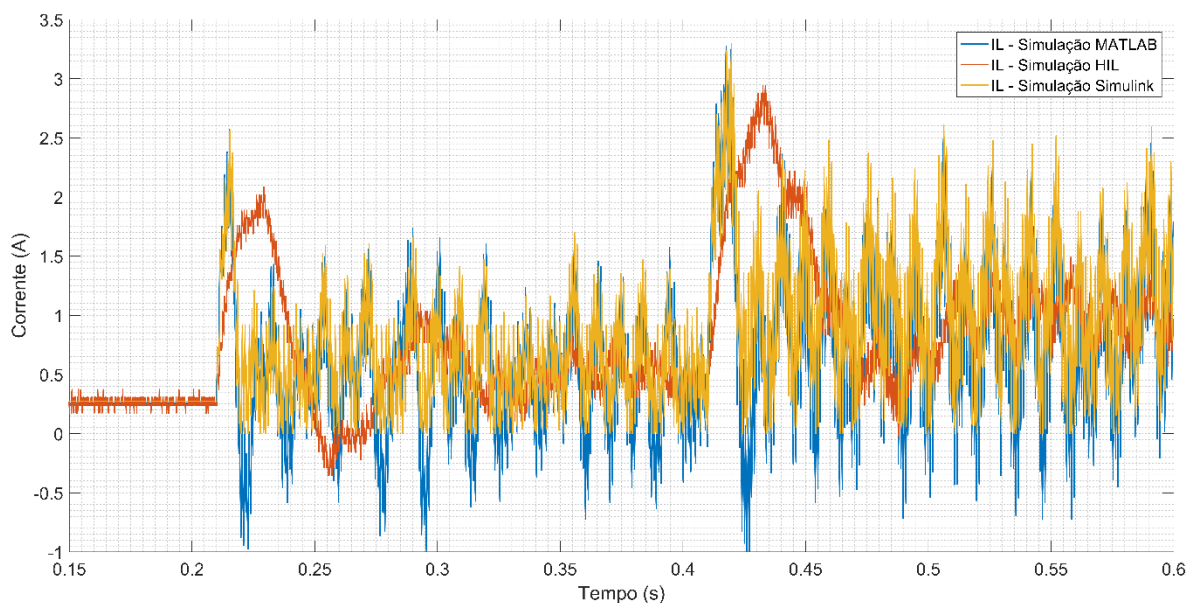
A partir da medição dos valores correspondentes às saídas de 0 *bit* e 4095 *bits* dos DAC's, que são respectivamente 0,62 V e 2,88 V, a equação (12) retorna o número de *bits* correspondente a determinado valor de tensão  $V_{DAC}$ . E, então, são aplicadas as equações (8) a (11) de maneira reversa, de forma a retornar os valores de saída dos modelos na escala compatível às demais simulações. Assim, com a importação para o MATLAB dos dados de tempo, PWM e tensão de saída dos DAC's, e com as conversões de escala necessárias, realiza-se a comparação da simulação HIL com os dados da simulação numérica do MATLAB e da simulação em blocos do Simulink, conforme descrito na subseção 2.2.6.

$$\frac{2,88 - 0,62}{4095} = \frac{2,88 - V_{DAC}}{4095 - bits} \quad \therefore \quad bits = \frac{4095}{2,26} V_{DAC} - \frac{2538,9}{2,26} \quad (12)$$

Os resultados dessas comparações são apresentados nas Figuras 14 a 17, sendo que a Figura 14 e a Figura 15 oferecem a comparação entre as simulações para o modelo do conversor *boost*, enquanto a Figura 16 e a Figura 17 são relativas ao modelo do inversor monofásico.

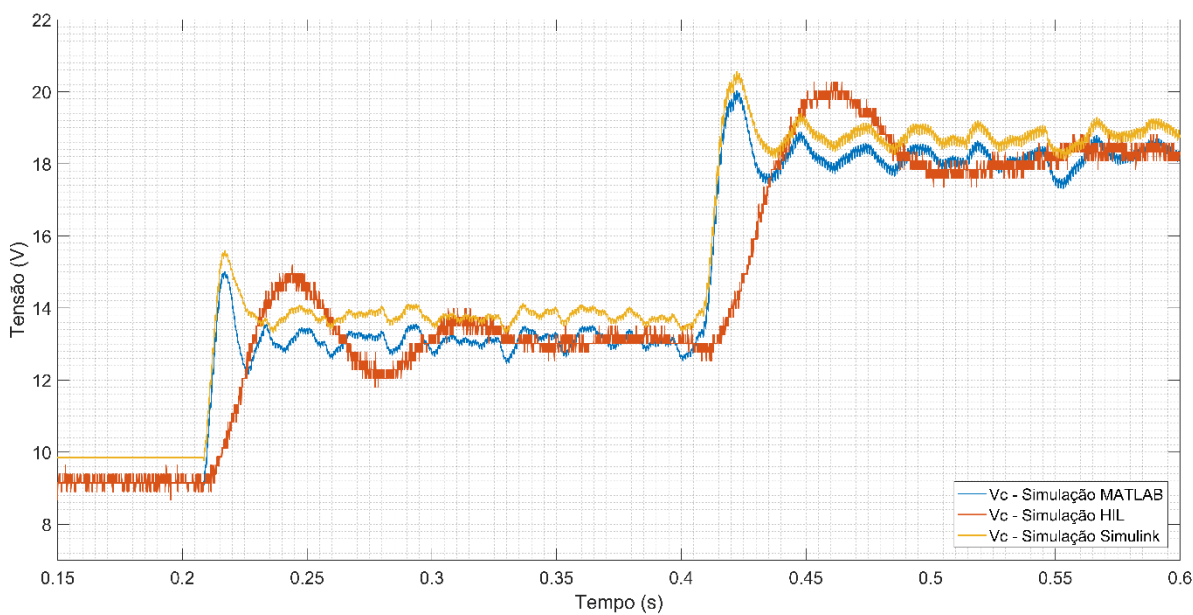


Figura 14 – Resultados das simulações do conversor *boost* ( $I_L$ ).

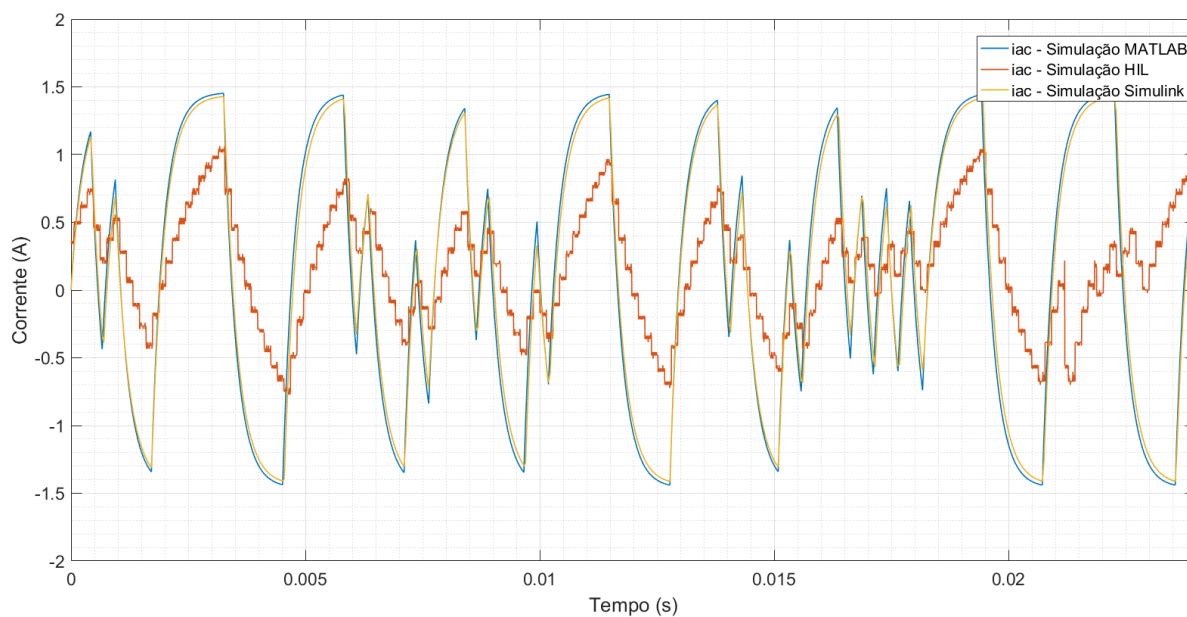


Fonte: Próprio autor.

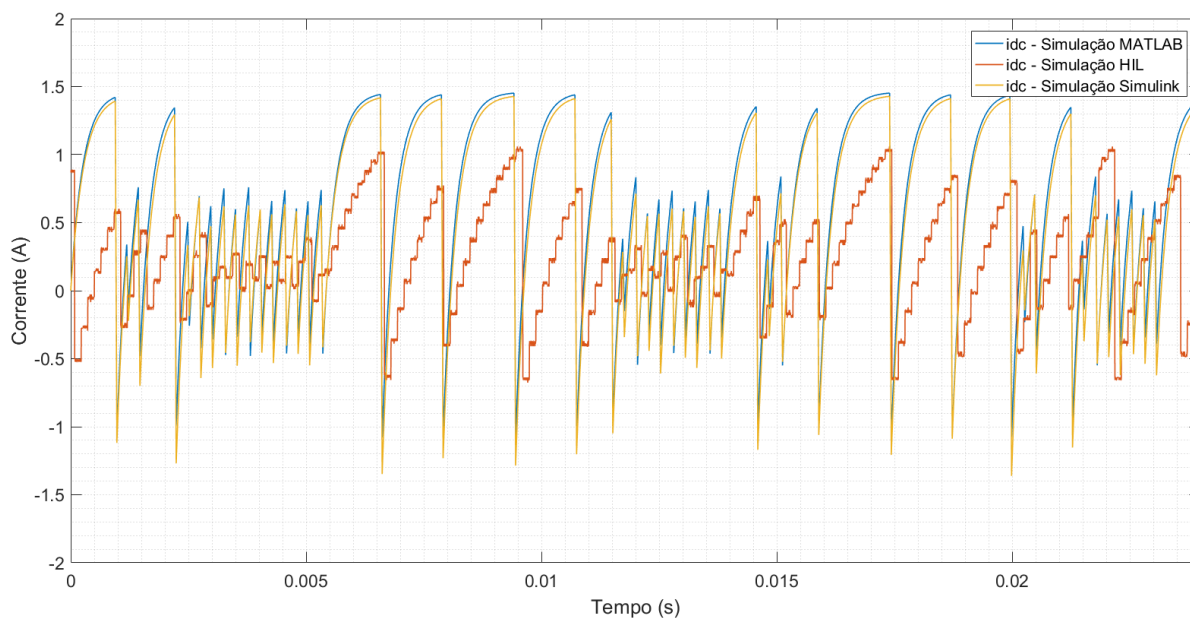
Figura 15 – Resultados das simulações do conversor *boost* ( $V_c$ ).



Fonte: Próprio autor.

Figura 16 – Resultados das simulações do inversor monofásico ( $i_{ac}$ ).

Fonte: Próprio autor.

Figura 17 – Resultados das simulações do inversor monofásico ( $i_{dc}$ ).

Fonte: Próprio autor.

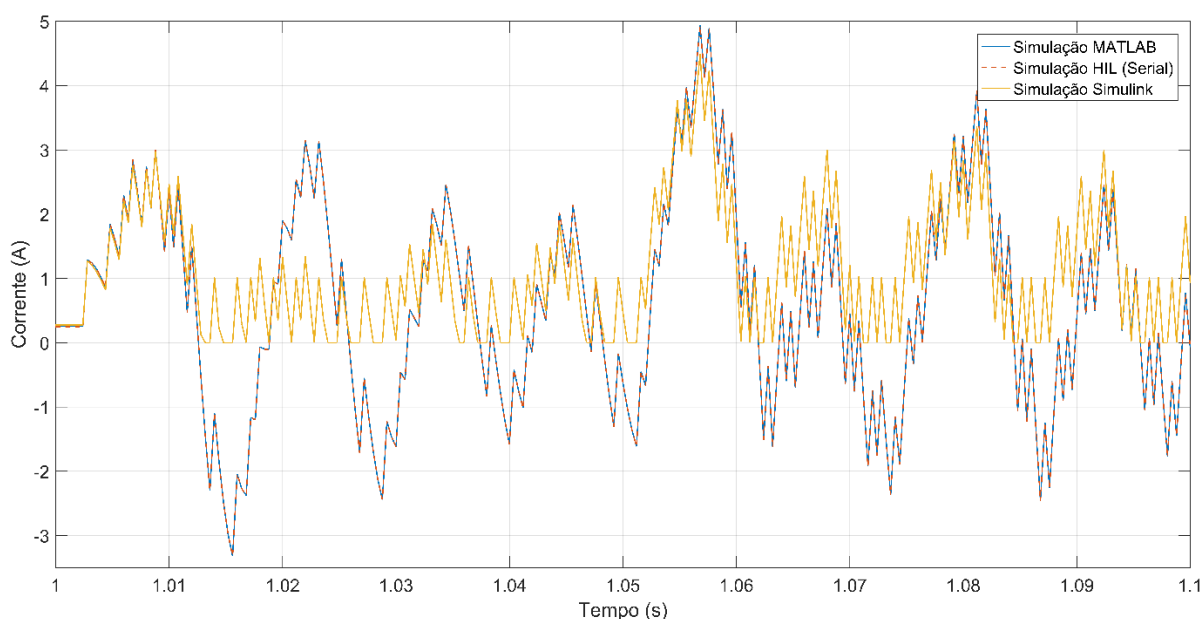
A observação das Figuras 14 a 17 indica que a simulação HIL não está coerente com as simulações em *software*, visto que o comportamento das grandezas de saída da plataforma HIL

não é semelhante ao comportamento das mesmas grandezas provenientes da simulação numérica no MATLAB e da simulação em blocos no Simulink.

A fim de investigar as razões pelas quais a simulação na plataforma HIL implementada ofereceu resultados insatisfatórios, foram realizados experimentos que verificassem em qual a etapa de implementação que o sistema começou a falhar. Então, a comunicação serial se tornou necessária para esse processo de investigação, pois ela permite a leitura de diversas variáveis da placa Arduino. Entretanto, esse tipo de comunicação é relativamente lento e, conseqüentemente, não é possível realizá-la na Arduino Due com programas que trabalham com interrupções de 35 ou 45  $\mu\text{s}$ , que são as interrupções utilizadas nos Estudos de Caso deste trabalho.

Utilizando o programa da simulação HIL para o conversor *boost*, a comunicação serial somente se concretizou a partir de interrupções de 400  $\mu\text{s}$ . Assim, a plataforma HIL implementada foi testada com interrupções e período de amostragem  $T_s$  de 400  $\mu\text{s}$ , a fim de possibilitar a visualização das saídas do sistema em seu formato digital antes da conversão pelo DAC. A Figura 18 apresenta a comparação dos dados provenientes desse teste para a corrente  $I_L$  do conversor *boost*.

Figura 18 – Teste dos resultados digitais da simulação HIL.

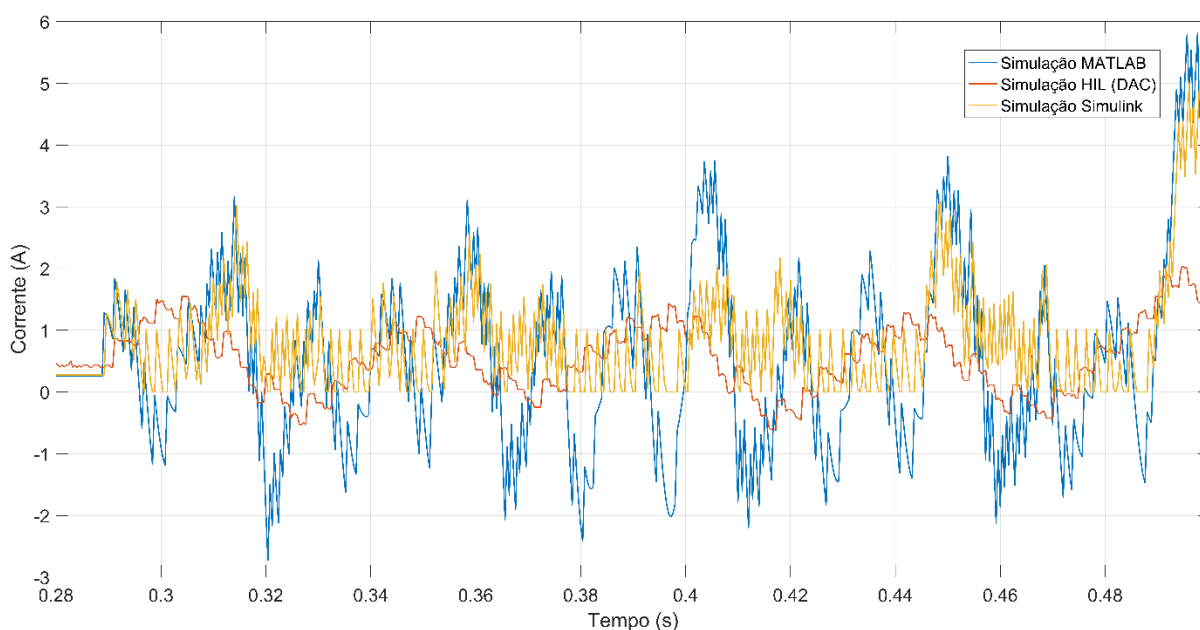


Fonte: Próprio autor.

A análise da Figura 18 permite observar que os valores calculados pelo simulador HIL antes da conversão pelo DAC estão corretos, visto que eles coincidem com os valores simulados em MATLAB. É importante notar que os valores simulados a partir da modelagem em equações de diferença apresentam um comportamento diferente que os do Simulink porque a modelagem não contemplou um diodo em antiparalelo com a chave, que impede a passagem de correntes negativas em  $I_L$ .

Entretanto, os resultados do teste ainda não são suficientes para apontar onde ocorre o erro do simulador HIL implementado, pois ainda há duas hipóteses: a Arduino Due não é capaz de oferecer valores digitais corretos em interrupções mais rápidas do que  $400 \mu\text{s}$ , ou o conversor DAC da Arduino Due não está realizando sua função corretamente. Para verificar tais hipóteses, também foram adquiridos os dados de saída dos DACs na simulação HIL do conversor *boost* com interrupções de  $400 \mu\text{s}$ . Os resultados desse segundo teste estão apresentados na Figura 19.

Figura 19 – Teste dos resultados do DAC da simulação HIL.



Fonte: Próprio autor.

A interpretação da Figura 19, em conjunto com a Figura 18, aponta que a conversão do DAC não ocorreu devidamente e, assim, indica que os resultados do simulador HIL implementado não são satisfatórios devido a erro do DAC. Para confirmar essa hipótese, o teste foi realizado em outra unidade de Arduino Due, de forma a descartar a justificativa de

ocorrência de defeitos num DAC específico, e também foi utilizada a função *map()* para converter os valores digitais em *bits* no lugar das equações (8) a (11), de forma a descartar erros na conversão para *bits*. Ambos os testes confirmaram os resultados insatisfatórios anteriores.

## 4 *Conclusões*

Os resultados deste trabalho apontam que a placa Arduino Due não foi capaz de atender às expectativas do projeto de implementação do simulador HIL, visto que as suas conversões digital/analógico realizadas pelos DACs não foram realizadas corretamente. Portanto, a conclusão deste trabalho é a de que a escolha da plataforma Arduino Due para a realização do projeto de implementação de Simulador HIL proposto não foi adequada.

Nas plataformas de pesquisa de artigos científicos, é possível encontrar projetos de implementação de simuladores HIL com placas Arduino Due que obtiveram sucesso. Entretanto, esses projetos simularam sistemas que demandaram períodos de amostragem bem menores do que os utilizados neste trabalho, pois envolviam processos que ocorrem em escalas de tempo em horas, por exemplo, sistemas refrigeradores (GAMBINO, SIANO, *et al.*, 2014) e o funcionamento de um pâncreas (QUESADA, ROJAS, *et al.*, 2019). Assim, a utilização do Arduino Due não deve ser descartada para a implementação de simuladores HIL, desde que o projeto não necessite uma alta taxa na amostragem dos sinais.

Como continuação deste trabalho, com o objetivo de implementar um simulador *Hardware-in-the-loop* para a aplicação no ensino de Controle em aulas de laboratório, sugere-se a utilização de alguma plataforma de *hardware* embarcado com código aberto com melhor desempenho em altas frequências de amostragem. Exemplos de sugestões são encontrados nos trabalhos de Bastos, Fuzato, *et al.* (2019, p. 3833-3841) e Sobota, Goubej, *et al.* (2019, p. 68-73), que implementaram simuladores HIL utilizando a plataforma LaunchPad C2000 Delfino (Texas Instruments) e a plataforma Raspberry Pi, respectivamente.

## *Referências Bibliográficas*

ARDUINO. Arduino DUE Tech Specs. **Arduino**, 2019. Disponível em: <<https://store.arduino.cc/usa/duel>>. Acesso em: 26 Novembro 2019.

ARDUINO. Arduino Reference: micros(). **Arduino**, 2019. Disponível em: <<https://www.arduino.cc/reference/en/language/functions/time/micros/>>. Acesso em: 26 Novembro 2019.

ARDUINO SUPPORT FORUM. DAC Support. **Arduino Support Forum**, 2014. Disponível em: <<https://forum.arduino.cc/index.php?topic=129765.0>>. Acesso em: 27 Novembro 2019.

ATMEL CORPORATION. **Datasheet: SAM3X / SAM3A Series**. [S.l.], p. 1-1459. 2015.

ATTEN ELECTRONICS CO. **Datasheet: ATFxxB DDS Function Generator**. Nanshan, p. 1-4.

BACIC, M. **On hardware-in-the-loop simulation**. Proceedings of the 44th IEEE Conference on Decision and Control. Seville: IEEE. 2005. p. 3194-3198.

BASTOS, R. F. et al. Model, Design and Implementation of a Low Cost HIL for Power Converter and Microgrid Emulation Using DSP. **IET Power Electronics**, v. 12, n. 14, p. 3833-3841, November 2019.

BURBANK, J.; KASCH, W.; WARD, J. Hardware-in-the-Loop Simulations. In: BURBANK, J.; KASCH, W.; WARD, J. **An Introduction to Network Modeling and Simulation for the Practicing Engineer**. New Jersey: IEEE Press, 2011. p. 114-142.

CHEN, Y. et al. **Autonomous Vehicle Testing and Validation Platform: Integrated Simulation System with Hardware in the Loop**. IEEE Intelligent Vehicles Symposium. Changshu: IEEE. 2018.

CUPELLI, M. et al. Hardware In the Loop Implementation and Challenges. In: CUPELLI, M., et al. **Modern Control of DC-Based Power Systems: A Problem-Based Approach**. Florianópolis: Academic Press, 2018. p. 249-259.

DEUTSCH, S. Microcontroller Maniacs Rejoice: Arduino Finally Releases the 32-Bit Due. **Wired**, 2012. Disponível em: <<https://www.wired.com/2012/10/arduino-due/>>. Acesso em: 26 Novembro 2019.

EBE, F. et al. Comparison of Power Hardware-in-the-Loop Approaches for the Testing of Smart Grid Controls. **Energies**, v. 11, n. 1, p. 3381, 2018.

GAMBINO, G. et al. **A low-cost HIL platform for testing professional refrigerators controllers**. Proceedings of the 19th World Congress The International Federation of Automatic Control. Cape Town: IFAC. 2014. p. 3104-3109.

GREGA, W. **Hardware-in-the-loop simulation and its application in control education**. 29th Annual Frontiers in Education Conference - Designing the Future of Science and Engineering Education. San Juan: IEEE. 1999. p. 12B6-12.

HAHN, B. D.; VALENTINE, D. T. SIMULINK Toolbox. In: HAHN, B. D.; VALENTINE, D. T. **Essential MATLAB for Engineers and Scientists**. 7. ed. Cambridge: Academic Press, v. 1, 2019. Cap. 16, p. 341-357.

HARRISON, W. S.; TILBURY, D. M.; YUAN, C. From Hardware-in-the-Loop to Hybrid Process Simulation: An Ontology for the Implementation Phase of a Manufacturing System. **IEEE Transactions on Automation Science and Engineering**, v. 9, n. 1, p. 96-109, 2012.

HUANG, S. et al. An agent-based hardware-in-the-loop simulation framework for building controls. **Energy & Buildings**, p. 26-37, 2018.

ISERMANN, R.; SCHAFFNITT, J.; SINSEL, S. Hardware-in-the-loop simulation for the design and testing of engine-control systems. **Control Engineering Practice**, v. 7, n. 1, p. 643-653, 1999.

JALDÉN, J.; MORENO, X. C.; SKOG, I. **Using the Arduino Due for Teaching Digital Signal Processing**. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Calgary: IEEE. 2018. p. 6468-6472.

LEDIN, J. A. Hardware-in-the-Loop Simulation. In: LEDIN, J. A. **Embedded Systems Programming**. [S.l.]: [s.n.], v. 12, 1999. p. 42-60.

LEE, Y. S. et al. **Low-cost RCP System for Control Courses using Matlab and the Open-source Hardware**. Proceedings of the World Congress on Mechanical, Chemical, and Material Engineering. Barcelona: MCM. 2015. p. 247-1 - 247-8.

LIMA, J. et al. **Hardware-in-the-loop simulation approach for the Robot at Factory Lite competition proposal**. IEEE International Conference on Autonomous Robot Systems and Competitions. Porto: IEEE. 2019.

LIMA, T. Arduino Due. **Embarcados**, 2014. Disponível em: <<https://www.embarcados.com.br/arduino-due/>>. Acesso em: 26 Novembro 2019.

LINDFIELD, G.; PENNY, J. An Introduction to MATLAB. In: LINDFIELD, G.; PENNY, J. **Numerical Methods Using MATLAB**. 4. ed. London: Academic Press, v. 1, 2019. Cap. 1, p. 1-72.

MENGHAL, P. M.; JAYA LAXMI, A. **Real time simulation**: Recent progress & challenges. International Conference on Power, Signals, Controls and Computation. Thrissur: IEEE. 2012.

MINIPA DO BRASIL LTDA. **Datasheet: Osciloscópio Digital MVB DSO**. 1-78. 2013.



- MOLER, C. A Brief History of MATLAB. **MathWorks**, 2018. Disponível em: <<https://www.mathworks.com/company/newsletters/articles/a-brief-history-of-matlab.html>>. Acesso em: 26 Novembro 2019.
- MONTI, A.; D'ARCO, S.; DESHMUKH, A. **A New Architecture for Low Cost Power Hardware in the Loop Testing of Power Electronics Equipments**. 2008 IEEE International Symposium on Industrial Electronics. Cambridge: IEEE. 2008. p. 2183-2188.
- NEMTANU, F. C.; COSTEA, I. M.; OBREJA, L. G. **Model of intelligent traffic sensors – application in hardware in the loop**. 40th International Spring Seminar on Electronics Technology. Sofia: IEEE. 2017.
- NOUREEN, S. S. et al. Real-Time Digital Simulators: A Comprehensive Study on System Overview, Application, and Importance. **International Journal of Research and Engineering**, p. 266-277, 2017.
- ONEL, O. et al. **Development of a Teaching Laboratory for Undergraduate Control Systems**. UKACC International Conference on Control. Coventry: IET. 2010.
- PENGRÁ, D. **The Oscilloscope and the Function Generator: Some introductory exercise for students in the advanced labs**. University of Washington. Washington, p. 1-19. 2007.
- QUESADA, L. et al. **Open-source low-cost Hardware-in-the-loop simulation platform for testing control strategies for artificial pancreas research**. 12th Symposium on Dynamics and Control of Process Systems, including Biosystems. Florianópolis: IFAC. 2019. p. 275-280.
- RENAUX, P. B.; LINHARES, R. R.; RENAUX, D. P. B. **Hardware-In-The-Loop Simulation Low-Cost Platform**. VII Brazilian Symposium on Computing Systems Engineering. Curitiba: IEEE. 2017. p. 173-180.
- SALA, A.; BONDIA, J. **Teaching Experience with Hardware-in-the-Loop Simulation**. 7th IFAC Symposium on Advances in Control Education. Madrid: IFAC. 2006. p. 123-128.
- SHI, D. Y.; GAN, W.-S. **Comparison of different development kits and its suitability in signal processing education**. IEEE International Conference on Acoustics, Speech and Signal Processing. Shanghai: IEEE. 2016. p. 6280-6284.
- SOBOTA, J. et al. **Raspberry Pi-based HIL simulators for control education**. 12th IFAC Symposium on Advances in Control Education. Philadelphia: IFAC. 2019. p. 68-73.
- TOULSON, R.; WILMSHURST, T. Interrupts, Timers, and Tasks. Em: TOULSON, R.; WILMSHURST, T. **Fast and Effective Embedded Systems Design**. 2. ed. Oxford: Newnes, v. 1, 2017. Cap. 6, p. 199-233.
- WAEALTERMANN, P.; MICHALSKY, T.; HELD, J. Hardware-in-the-Loop Testing in Racing Applications. Em: FEHAN, D. **Design of Racing and High-Performance Engines**. Warrendale: SAE International, 2013. p. 19-32.

WELLSTEAD, P. E. Teaching Control with Laboratory Scale Models. **IEEE Transactions on Education**, p. 285-290, 1990.

ZILOUCHIAN, A. **A Novel Intelligent Control Laboratory for Undergraduate Students** in. Proceedings of the 2003 American Control Conference. Denver: IEEE. 2003. p. 633-638.

## Apêndice A – Código da Leitura Serial dos Valores de Saída

```

// Parâmetros do circuito boost:
float Ldc = 0.0038; // 3.8 mH
float RL = 0.35; // 0.35 Ohm
float Ron = 0.3; // 0.3 Ohm
float Rd = 0.2; // 0.2 Ohm
float R = 36; // 36 Ohm
float C = 0.000940; // 940 uF
float Vbat = 10; // 10 V
float Ts = 0.000200; // período de amostragem

// Variáveis auxiliares nos cálculos:
float D = 0; // estado do PWM
float DIL = 0; // derivada da corrente IL
float DVc = 0; // derivada da tensão Vc

// Saídas do sistema:
float Vc = 0; // tensão no capacitor do conversor boost
float IL = 0; // corrente no indutor do conversor boost

void setup()
{
  Serial.begin(115200); // configuração da comunicação serial com baud rate 115200
  pinMode(7, INPUT); // o pino 7 recebe o sinal digital de PWM

  /* Habilitação do timer clock no controlador de gestão de energia (PMC) */
  pmc_set_writeprotect(false); // desligar a proteção contra escrita nos registradores do PMC
  pmc_enable_periph_clk(ID_TC0); // Habilitar o clock periférico TC0

  /* Configuração do clock periférico do TC */
  TC_Configure(/* clock */TC0, /* canal */0, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
TC_CMR_TCCLKS_TIMER_CLOCK1);
  // TIMER_CLOCK1 = 84MHZ / 2 = 42MHz
  TC_SetRC(TC0, 0, 8400); // Gatilho da interrupção a cada 2520 ticks do TC0
  (2520=42MHz*200us)
  TC_Start(TC0, 0); // Habilitação do TC0

  // Habilitar as interrupções de tempo no TC0
  TC0->TC_CHANNEL[0].TC_IER=TC_IER_CPCS; // IER = habilita o registrador de interrupção
  TC0->TC_CHANNEL[0].TC_IDR=~TC_IER_CPCS; // IDR = desabilita o registrador de interrupção
  (~: negado)

  /* Adição da interrupção no Nested Vectored Interrupt Controller */
  NVIC_EnableIRQ(TC0_IRQn);
}

void loop()
{
  Serial.print(IL); // Leitura serial de IL
  Serial.print("\t");
  Serial.println(Vc); // Leitura serial de Vc
}

void TC0_Handler() // Função chamada a cada interrupção de TC0
{
  TC_GetStatus(TC0, 0); // Limpeza do seu status de TC0

  if (digitalRead(7)==HIGH) D = 1; // Lê a entrada do PWM e designa para valor de D
  alto(1)/baixo(0)
  else D = 0;

  /* Aqui são inseridas as equações do modelo estudado: conversor boost */
  DIL = (Vbat - (RL*IL) - ((0.7+Rd*IL+Vc)*(1-D)) - (D*Ron*IL))/Ldc;
  DVc = (IL/C)*(1-D) - (Vc/(R*C));
  IL = IL + DIL*Ts;
  Vc = Vc + DVc*Ts;
}

```

## ***Apêndice B – Código da Leitura do Tempo de Execução***

```

// Parâmetros do circuito boost:
float Ldc = 0.0038; // 3.8 mH
float RL = 0.35; // 0.35 Ohm
float Ron = 0.3; // 0.3 Ohm
float Rd = 0.2; // 0.2 Ohm
float R = 36; // 36 Ohm
float C = 0.000940; // 940 uF
float Vbat = 10; // 10 V
float Ts = 0.000200; // tempo de amostragem

// Variáveis auxiliares nos cálculos:
float D = 0; // estado do PWM
float DIL = 0; // derivada da corrente IL
float DVc = 0; // derivada da tensão Vc
float IL_t = 0; // IL temporária para conversão em bits
float Vc_t = 0; // Vc temporária para conversão em bits

unsigned long tempol = 0; // variável que recebe o valor do tempo no início do modelo
unsigned long tempo2 = 0; // variável incremental que guarda o tempo de execução de cada
iteração
unsigned long cont = 0; // variável que conta o número de iterações

// Saídas do sistema:
float Vc = 0; // tensão no capacitor do conversor boost
float IL = 0; // corrente no indutor do conversor boost

void setup()
{
  Serial.begin(115200); // Configuração da comunicação serial
  analogWriteResolution(12); // Configuração da resolução de 12bits (0-4095) da saída
analógica DAC
  pinMode(7, INPUT); // O pino 7 recebe o sinal de PWM externo
  pinMode(DAC0, OUTPUT); // Ativa os pinos de saída dos conversores DAC0 e DAC1
  pinMode(DAC1, OUTPUT);
}

void loop()
{
  if (cont==200) Serial.println(tempo2/cont); // leitura serial da média do tempo em 200
iterações
  if (cont==400) Serial.println(tempo2/cont); // leitura serial da média do tempo em 400
iterações

  tempol = micros();

  if (digitalRead(7)==HIGH) D = 1; // Lê a entrada do PWM e designa para valor de D
alto(1)/baixo(0)
  else D = 0;

/* Aqui são inseridas as equações do modelo estudado: conversor boost */
  DIL = (Vbat - (RL*IL) - ((0.7+Rd*IL+Vc)*(1-D)) - (D*Ron*IL))/Ldc;
  DVc = (IL/C)*(1-D) - (Vc/(R*C));
  IL = IL + DIL*Ts;
  Vc = Vc + DVc*Ts;

  IL_t = (IL+2,4)*640; // equação de conversão para bits

```

```
    IL_t = int (IL_t); // arredonda o número de bits para inteiro
    analogWrite(DAC0,IL_t); // o DAC vai converter o número de bits para saída de tensão
analógica

    Vc_t = Vc*200;
    Vc_t = (int) (Vc_t);
    analogWrite(DAC1,Vc_t);

    tempo2 = tempo2 + micros() - tempo1; // adição do tempo de execução à variável de tempo
    cont = cont + 1; // incremento da contagem de iterações
}
```

## Apêndice C – Código da Simulação HIL

```

// Parâmetros do circuito boost:
float Ldc = 0.0038; // 3.8 mH
float RL = 0.35; // 0.35 Ohm
float Ron = 0.3; // 0.3 Ohm
float Rd = 0.2; // 0.2 Ohm
float R = 36; // 36 Ohm
float C = 0.000940; // 940 uF
float Vbat = 10; // 10 V
float Ts = 0.000045; // período de amostragem Ts = 45 us

// Variáveis auxiliares nos cálculos:
float D = 0; // estado do PWM
float DIL = 0; // derivada da corrente IL
float DVc = 0; // derivada da tensão Vc
float IL_t = 0; // IL temporária para conversão em bits
float Vc_t = 0; // Vc temporária para conversão em bits

// Saídas do sistema:
float Vc = 0; // tensão no capacitor do conversor boost
float IL = 0; // corrente no indutor do conversor boost

void setup()
{
    analogWriteResolution(12); // Configuração da resolução de 12bits (0-4095) da saída
    analógica DAC
    pinMode(7, INPUT); // O pino 7 recebe o sinal de PWM externo
    pinMode(DAC0, OUTPUT); // Ativa os pinos de saída dos conversores DAC0 e DAC1
    pinMode(DAC1, OUTPUT);

    /* Habilitação do timer clock no controlador de gestão de energia (PMC) */
    pmc_set_writeprotect(false); // desligar a proteção contra escrita nos registradores do PMC
    pmc_enable_periph_clk(ID_TC0); // Habilitar o clock periférico TC0

    /* Configuração do clock periférico do TC */
    TC_Configure(/* clock */TC0, /* canal */0, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
    TC_CMR_TCCLKS_TIMER_CLOCK1);
    // TIMER_CLOCK1 = 84MHZ / 2 = 42MHz
    TC_SetRC(TC0, 0, 1890); // Gatilho da interrupção a cada 1890 ticks do TC0 (1890=42MHz*45us)
    TC_Start(TC0, 0); // Habilitação do TC0

    // Habilitar as interrupções de tempo no TC0
    TC0->TC_CHANNEL[0].TC_IER=TC_IER_CPCS; // IER = habilita o registrador de interrupção
    TC0->TC_CHANNEL[0].TC_IDR=~TC_IER_CPCS; // IDR = desabilita o registrador de interrupção
    (~: negado)

    /* Adição da interrupção no Nested Vectored Interrupt Controller */
    NVIC_EnableIRQ(TC0_IRQn);
}

void loop()
{
}

void TC0_Handler() // Função chamada a cada interrupção de TC0
{
    TC_GetStatus(TC0, 0); // Limpeza do seu status de TC0
}

```

```

    if (digitalRead(7)==HIGH) D = 1; // Lê a entrada do PWM e designa para valor de D
    alto(1)/baixo(0)
    else D = 0;

/* Aqui são inseridas as equações do modelo estudado: conversor boost */
    DIL = (Vbat - (RL*IL) - ((0.7+Rd*IL+Vc)*(1-D)) - (D*Ron*IL))/Ldc;
    DVc = (IL/C)*(1-D) - (Vc/(R*C));
    IL = IL + DIL*Ts;
    Vc = Vc + DVc*Ts;

    IL_t = (IL+2,4)*640; // equação de conversão para bits
    IL_t = int (IL_t); // arredonda o número de bits para inteiro
    analogWrite(DAC0,IL_t); // o DAC vai converter o número de bits para saída de tensão
analógica

    Vc_t = Vc*200;
    Vc_t = (int) (Vc_t);
    analogWrite(DAC1,Vc_t);
}

```