UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”

FACULDADE DE ENGENHARIA

CAMPUS DE ILHA SOLTEIRA

ALEXANDRE ARAUJO AMARAL DE ALMEIDA

DESENVOLVIMENTO DA FERRAMENTA MS²PSoC: TRADUÇÃO DE MODELOS DESCRITOS NO MATLAB/SIMULINK PARA O AMBIENTE PSOC CREATOR

Ilha Solteira

2015

ALEXANDRE ARAUJO AMARAL DE ALMEIDA

DESENVOLVIMENTO DA FERRAMENTA MS²PSoC: TRADUÇÃO DE MODELOS DESCRITOS NO MATLAB/SIMULINK PARA O AMBIENTE PSOC CREATOR

Dissertação apresentada à Faculdade de Engenharia – UNESP – Campus de Ilha Solteira como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica. Área de Conhecimento: Automação.

Prof. Dr. Alexandre César Rodrigues da Silva
Orientador

Ilha Solteira
2015

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Almeida, Alexandre Araujo Amaral de.
A447d Desenvolvimento da ferramenta MS²PSOC: tradução de modelos descritos no matlab/simulink para o ambiente psoc creator / Alexandre Araujo Amaral de Almeida. -- Ilha Solteira: [s.n.], 2015
92 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação

Orientador: Alexandre César Rodrigues da Silva
Inclui bibliografia

1. Circuitos eletrônicos. 2. Matlab/Simulink. 3. Psoc.

unesp **UNIVERSIDADE ESTADUAL PAULISTA**

CAMPUS DE ILHA SOLTEIRA

FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

CERTIFICADO DE APROVAÇÃO

TÍTULO: Desenvolvimento da ferramenta MS2PSoC: Tradução de modelos descritos no Matlab/Simulink para o ambiente PSoC Creator

AUTOR: ALEXANDRE ARAUJO AMARAL DE ALMEIDA

ORIENTADOR: Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovado como parte das exigências para obtenção do Título de Mestre em Engenharia Elétrica , Área: AUTOMAÇÃO, pela Comissão Examinadora:

Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. CARLOS ANTONIO ALVES
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. TÉRCIO ALBERTO DOS SANTOS FILHO
Departamento de Ciências Da Computação / Universidade Federal de Goiás

Data da realização: 02 de setembro de 2015.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Alexandre César Rodrigues da Silva, por ter me concedido a imensa oportunidade de desenvolver este projeto de pesquisa com sua orientação.

À minha esposa, Graziela, pela companhia, compreensão e carinho nesta jornada.

À minha filha, Sofia, pelos momentos de muita alegria.

À minha família pelo eterno apoio, não importando a distância.

Aos meus amigos e parceiros pela ajuda e companhia diária.

Ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo fomento através dos processos de n °309023/2012-2, n °550134/2013-1 e n °166837/2013-0.

À todos que de alguma forma ou de outra tiveram sua parcela de contribuição para a realização deste trabalho, meu muito obrigado.

RESUMO

Usualmente, realiza-se a modelagem de um sistema no Simulink e, posteriormente, no PSoC Creator para implementá-lo no hardware PSoC. Neste trabalho apresenta-se a ferramenta computacional denominada MS^{2}PSoC que analisa um modelo desenvolvido em alto nível de abstração no ambiente Matlab/Simulink e o traduz para o ambiente PSoC Creator. A tradução tem por finalidade a implementação do modelo no sistema embarcado utilizando a tecnologia PSoC. No processo de tradução, a ferramenta MS^{2}PSoC também gera a estrutura de arquivos necessária para que o modelo traduzido seja aberto no ambiente PSoC Creator. A inovação deste trabalho é permitir que o modelo possa ser simulado antes de ser implementado em hardware, haja vista que o ambiente PSoC Creator não dispõe de simulador. Como estudo de casos utilizou-se o código de linha MLT-3, um modelo de conversor D/A Ladder R-2R e um sistema de controle de temperatura veicular. Os sistemas estudados foram implementados em hardware e avaliados, comparando-os com os resultados obtidos nas simulações no Simulink. Com os resultados obtidos concluiu-se que a ferramenta MS^{2}PSoC traduz corretamente os modelos descritos no ambiente Matlab/Simulink para o ambiente PSoC Creator, automatizando a implementação de modelos no sistema embarcado PSoC.

Palavras-chave: Ferramenta computacional. Matlab/Simulink. PSoC.

ABSTRACT

Usually, a system is modeled in Simulink and later in PSoC Creator to deploy it in hardware PSoC. We present in this work a computational tool called MS^{2}PSoC which analyses a model developed at a high level of abstraction in Matlab/Simulink and translates it to the PSoC Creator software. The goal of this translation is to implement the model in embedded system with PSoC technology. In the translation process, the MS^{2}PSoC tool also generates the file structure needed for the translated model to be open in PSoC Creator. The innovation of this work is to allow the model to be simulated before it is implemented in hardware, given that the PSoC Creator software has no simulator. As a case study was used a MLT-3 line code, a D/A converter Ladder R-2R and a temperature control system for vehicles. The systems were implemented in PSoC hardware and evaluated by comparing them with the results obtained in the Simulink simulations. With the results it was concluded that the tool MS^{2}PSoC correctly translates the models described in Matlab/Simulink environment for PSoC Creator, automating the deployment models in embedded system PSoC.

Keywords: Computational tool. Matlab/Simulink. PSoC.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analógico para Digital
API	*Application Programming Interface*
D/A	Digital para Analógico
CLP	Controlador Lógico Programável
CORDIC	*Coordinate Rotation Digital Computer*
CPU	*Central Processing Unit*
CSP	*Communicating Sequential Processes*
DC	*Direct Current*
DMA	*Direct Memory Access*
ECC	*Error Checking & Correction*
FDDI	*Fiber Distributed Data Interface*
FIFO	*First In First Out*
FPGA	*Field Programmable Gate Array*
FSM	*Finite State Machine*
HDL	*Hardware Description Language*
HVAC	*Heating, Ventilating, and Air Conditioning*
IDE	*Integrated Design Environment*
LSB	*Least Significant Bit*
LUT	*Lookup Table*
MILP	*Mixed-Integer Linear Programming*
MLT-3	*Multi Level Transmission 3*
MSB	*Most Significant Bit*
MPSoC	*Multiprocessor Systems on Chip*
NVIC	*Nested Vectored Interrupt Controller*

PLD	*Programmable Logic Device*
PSoC	*Programmable System-on-Chip*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
RTL	*Register Transfer Level*
TiDE	*Timeless Design Environment*
UDB	*Universal Digital Blocks*
ULA	Unidade Lógica Aritmética
UML	*Unified Modeling Language*
VANT	Veículo Aéreo Não Tripulado
VHDL	*VHSIC Hardware Description Language*
VHDL-AMS	*VHSIC Hardware Description Language - Analog and Mixed Signal*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Com o aumento da complexidade dos sistemas embarcados, geralmente envolvendo hardware e software, o processo de desenvolvimento destes sistemas tem exigido o emprego de diferentes ferramentas de modelagem e simulação, tais como ferramentas de síntese e tradução.

Síntese lógica é um processo que consiste na conversão de um sistema descrito em alto nível de abstração para o nível RTL (*Register Transfer Level*) sem que haja alteração no comportamento do sistema. Logo, uma ferramenta de síntese é um programa computacional que realiza esta conversão automática entre diferentes níveis de abstrações visando a implementação do sistema. Neste contexto, define-se tradução como sendo a conversão de um sistema entre dois ambientes computacionais distintos mantendo a mesma funcionalidade.

Um dos ambientes utilizados para a modelagem dos sistemas em alto nível de abstração é o Simulink®. O Simulink é uma ferramenta presente no software MATLAB® na forma de *toolbox*, no qual é possível realizar a análise e simulação dos modelos nele descritos (MATHWORKS, 2013).

Atualmente, o Simulink disponibiliza o suporte à hardware para aplicações em tempo real, ou seja, o modelo descrito é implementado em hardware diretamente do Simulink. Porém, este recurso abrange somente determinados sistemas embarcados, como por exemplo, a FPGA (*Field Programmable Gate Array*).

Um sistema embarcado pode ser definido como um conjunto de dispositivos eletrônicos configurados para realizar tarefas específicas. Denomina-se embarcado, pois os dispositivos presentes se encontram em uma mesma placa. Entre estes dispositivos se encontram microprocessadores, memórias, portas de entrada e saída, etc. Alguns exemplos de sistemas embarcados são celulares, eletrodomésticos, impressoras, etc.

Alguns sistemas embarcados, como FPGA, disponibilizam somente componentes digitais, sendo necessária a utilização de componentes externos para a realização de tarefas

que exijam componentes analógicos. Entretanto, em outros sistemas embarcados não se torna necessário o uso destes componentes externos, pois alguns componentes analógicos se encontram embarcados, como é o caso do PSoC® (*Programmable System-on-Chip*).

O PSoC tem como principal característica a integração de componentes analógicos e digitais em um mesmo hardware, possibilitando a realização de projetos com sinais mistos sem a necessidade de componentes externos. O ambiente utilizado para a descrição dos modelos a serem implementados no hardware PSoC denomina-se PSoC Creator™.

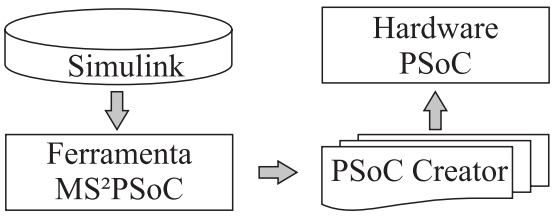
Um ponto relevante com relação ao ambiente PSoC Creator é a falta de um simulador, ou seja, os modelos descritos devem ter seu funcionamento analisado diretamente no hardware. Por isso, pesquisas estão sendo realizadas para se utilizar ambientes de simulação como *front-end* para o desenvolvimento de projetos.

Algumas das ferramentas desenvolvidas são a MS²SV (Matlab/Simulink para SystemVision), apresentada no trabalho de Silva e Grout (2011), que traduz um modelo descrito no Simulink para o ambiente SystemVision da Mentor Graphics e a SF²HDL (Stateflow para linguagem HDL), apresentada no trabalho de Almeida (2009), que traduz uma máquina de estados finita descrita no Stateflow (*toolbox* do Simulink) para as linguagens VHDL (*VHSIC Hardware Description Language*) ou Verilog HDL (*Hardware Description Language*).

Seguindo a linha de pesquisa de desenvolvimento de ferramentas de síntese este trabalho teve por objetivo o desenvolvimento de uma ferramenta computacional denominada MS²PSoC (Matlab/Simulink para PSoC Creator) capaz de realizar a interação entre os ambientes Matlab/Simulink e PSoC Creator.

Esta interação ocorre com a tradução de um modelo descrito no Simulink para o ambiente PSoC Creator. A finalidade desta tradução é a implementação do modelo descrito no Simulink em hardware PSoC. Além da tradução, a ferramenta gera a estrutura de arquivos necessária para que o modelo traduzido seja executado no PSoC Creator. O objetivo do trabalho é apresentado na Figura 1.

Figura 1 – Objetivo do trabalho de pesquisa.

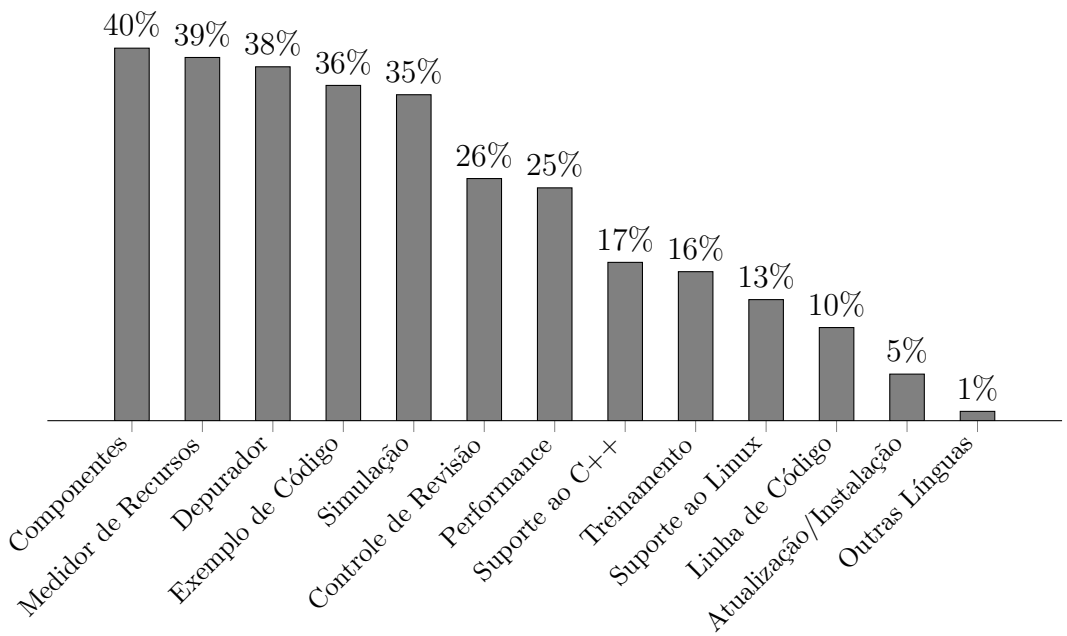


Fonte: Elaborada pelo autor.

Justifica-se o desenvolvimento desta ferramenta computacional, principalmente, em dois fatores. Não é possível realizar a simulação de modelos descritos no PSoC Creator e o sistema embarcado PSoC não é suportado pelo Simulink quanto ao recurso de interação entre software e hardware. Além disto, a motivação para esta pesquisa é a de que, até o momento em que este texto foi escrito, não foi encontrada ferramenta com propósito similar.

É importante ressaltar que em uma pesquisa preliminar realizada pela fabricante do sistema embarcado PSoC, Corporation (2014b), que tinha por objetivo avaliar o PSoC Creator e apontar possíveis melhorias mostrou que 35% dos usuários, que responderam à pesquisa, indicaram a necessidade de um simulador no ambiente computacional. Na Figura 2 apresenta-se o resultado desta pesquisa.

Figura 2 – Pesquisa para melhoria do PSoC Creator.



Fonte: Corporation (2014b).

No Capítulo 2 apresentam-se os conceitos utilizados no desenvolvimento deste trabalho. A metodologia empregada no desenvolvimento da ferramenta MS^{2}PSoC é apresentada no Capítulo 3. No Capítulo 4 apresentam-se os estudos de casos utilizados para avaliar o funcionamento da ferramenta MS^{2}PSoC e os resultados obtidos são apresentados no Capítulo 5. Por fim, apresentam-se as conclusões no Capítulo 6.

1.1 REVISÃO DA LITERATURA

A literatura descreve vários trabalhos em que o ambiente Matlab/Simulink é utilizado como *front-end* para o desenvolvimento de projetos e posterior geração automática de códigos.

No trabalho de Neema et al. (2005) foi apresentada uma ferramenta de modelagem visual denominada *GReAT* que é utilizada para converter modelos desenvolvidos no Stateflow para a linguagem C. Para isso, desenvolveram um meta-modelo UML (*Unified Modeling Language*) para a geração do código a partir do Stateflow. Os autores afirmam que a geração de códigos é eficiente e preserva a estrutura de estados do modelo traduzido. A ferramenta foi validada com vários exemplos desde modelos simples aos complexos.

Ressaltando a falta de suporte do ambiente Matlab/Simulink para o sistema embarcado PSoC, Chindris e Muresan (2006) apresentaram uma metodologia para implementar modelos do Simulink no sistema embarcado PSoC. Desenvolveram componentes do PSoC no ambiente Simulink e os validaram com simulação. Enfatizaram a ideia de desenvolver modelos com estes componentes e gerar automaticamente o código do modelo desenvolvido para serem implementados no PSoC.

O Matlab possui uma ferramenta comercial denominada Link que realiza a interface com o ambiente ModelSim. No trabalho de Gestner e Anderson (2007) foi apresentada uma ferramenta com propósito similar que apresenta algumas melhorias com relação à ferramenta Link. Segundo os autores, na ferramenta Link era necessário realizar alguns passos manualmente, mas que foram automatizados com a ferramenta desenvolvida.

O Simulink disponibiliza a simulação dos circuitos desenvolvidos, porém não leva em consideração as características do hardware. No trabalho de Moon et al. (2007) foi apresentado um bloco híbrido capaz de realizar esta simulação do hardware e posteriormente gerar o código na linguagem C do modelo desenvolvido. Os autores afirmaram que o bloco híbrido proposto reduz o custo, o tempo e o esforço no desenvolvimento de projetos.

A ferramenta de conversão que o Simulink disponibiliza realiza a otimização do circuito, porém em algumas situações essa otimização não é necessária como no trabalho de Tranchero e Reyneri (2007) em que o objetivo era o de simular circuitos assíncronos e traduzi-los para a linguagem VHDL. Foi gerado um bloco de *Data-path* e um bloco com o protocolo necessário para simular e posteriormente traduzir um circuito assíncrono para a linguagem VHDL. Segundo os autores, a geração de código foi realizada com sucesso.

No trabalho de Atat e Zergainoh (2008) foi apresentada uma metodologia para a rápida prototipagem de MPSoC (*Multiprocessor Systems on Chip*) partindo de uma especificação no Matlab/Simulink. A abordagem apresentada pelos autores é a de gerar o código do hardware para uma linguagem de descrição de hardware e o software para a linguagem C. Esta geração de código é realizada em vários níveis de abstração. Os autores afirmaram que, com os resultados encontrados, a metodologia proposta foi eficiente.

O Matlab/Simulink atualmente oferece suporte para CLPs (Controlador Lógico Programável), mas quando o trabalho de Schwarz et al. (2008) foi publicado ainda não era disponibilizado esse suporte. Os autores apresentaram a implementação de modelos Simulink em CLPs para sistemas relacionados à segurança e ressaltaram a ideia de simular exatamente as condições de uma aplicação real.

Seguindo a linha de simulação de circuitos assíncronos, no trabalho de Tranchero et al. (2009) os autores apresentaram uma nova metodologia para a tradução deste tipo de circuito. Foi utilizado uma combinação da ferramenta CodeSimulink com a TiDE (*Timeless Design Environment*) para a geração de um código na linguagem *Haste*. Esta linguagem é utilizada para mapear descrição de hardware no ambiente TiDE. Segundo os autores, a tradução ocorreu com sucesso, porém necessita de melhorias.

No trabalho de Farkas, Neumann e Hinnerichs (2009) foi apresentada uma metodologia para desenvolvimento de sistemas utilizando-se do UML e do Simulink. Esta pesquisa foi voltada para o processo de engenharia de software automotivo. A partir da especificação do modelo em UML e Simulink é realizada a tradução para a linguagem C e aplicada em sistema embarcado. Os autores utilizaram um controle automático da porta de um veículo para validar a metodologia desenvolvida.

Com o intuito de preservar a semântica na geração automática de código a partir do Matlab/Simulink, Wang et al. (2009) propuseram uma metodologia utilizando o *Target Language Compiler* e o *Real-Time Workshop* (ambas ferramentas disponíveis no Matlab). Para isso, desenvolveram blocos customizados para a geração de código na linguagem C que preserva a semântica do modelo desenvolvido. A aplicação é focada em sistemas de comunicação reativos síncronos. Os autores validaram o código gerado emulando-o no microcontrolador PIC18F452.

Seguindo a linha de pesquisa apresentada no trabalho de Wang et al. (2009), Natale et al. (2010) realizou uma melhoria na geração de código para sistemas de controle reativo síncrono utilizado na indústria automotiva e aeronáutica. Foi realizada uma otimização na implementação de algoritmos multi-tarefas em dispositivos com um processador de

memória limitada desenvolvidos no Simulink. O algoritmo desenvolvido é baseado no *framework* de otimização MILP (*Mixed-Integer Linear Programming*) e apresentou a adição mínima de atraso e memória mesmo com altos valores de utilização do processador.

No trabalho de Silva e Grout (2011) foi apresentada uma ferramenta denominada MS^{2}SV que realiza a tradução automática de um modelo de sinal misto (analógico/digital) descrito em alto nível de abstração no ambiente Simulink para a linguagem VHDL-AMS (*VHSIC Hardware Description Language - Analog and Mixed Signal*). Para realizar esta tradução, a ferramenta realiza a leitura do modelo descrito no ambiente Simulink e gera o correspondente código VDHL-AMS estrutural. A ferramenta ainda gera a estrutura de projeto necessária para a simulação do modelo traduzido em ambiente SystemVision da Mentor Graphics. Para validar a ferramenta foram utilizados alguns modelos de conversores de dados D/A.

Continuando o trabalho de Silva e Grout (2011) sobre o desenvolvimento da ferramenta computacional MS^{2}SV, Almeida, Silva e Sampaio (2012b) apresentaram uma melhoria na qual é disponibilizado ao usuário a liberdade de inserir uma biblioteca própria para ser utilizada na tradução. Com este recurso a ferramenta MS^{2}SV passa a ser mais flexível, podendo ser utilizada para o desenvolvimento de sistemas em diversas áreas. Para validar a melhoria realizada, um sistema de controle de um VANT (Veículo Aéreo Não Tripulado) foi desenvolvido e traduzido para o SystemVision.

Combinando as ferramentas do Simulink com a possibilidade de se efetuar cálculos matemáticos com matrizes no Matlab, o trabalho de Valters (2011) apresentou uma interface gráfica para automatizar a implementação do Jacobiano em FPGA. A interface gráfica foi desenvolvida no Matlab e foi utilizado o Simulink HDL Coder para gerar o código VHDL das expressões especificadas. O sistema desenvolvido também calcula os recursos que serão necessários na implementação.

No trabalho de Almeida, Silva e Sampaio (2012a) foi apresentada uma ferramenta computacional denominada BD^{2}XML que traduz um sistema do Matlab/Simulink para a linguagem XML (*eXtensible Markup Language*). O funcionamento da ferramenta se dá pela leitura dos blocos utilizados no desenvolvido do sistema e realiza a tradução bloco a bloco para a linguagem XML. Justifica-se a tradução para a linguagem XML pela sua flexibilidade e facilidade de leitura. Como estudo de caso foi utilizado o conversor AD7528 e os autores afirmam que a tradução ocorreu com precisão.

Lambersky (2012) apresentou em seu trabalho uma metodologia para a geração automática de código para o microcontrolador TMS570. O autor gera as funções de

hardware no HalCoGen e implementa estas funções no Code Composer Studio. O código C gerado no ambiente Simulink do modelo desenvolvido é utilizado no Code Composer Studio junto com as funções de hardware previamente geradas. Ao implementar o código C junto com as funções, é gerado o código específico para o hardware. O autor ressalta que a ferramenta é útil para casos complexos e que nos casos de microcontroladores não suportados só é viável a realização do suporte somente se for utilizado em vários projetos.

Para validar a implementação do filtro de Kalman em um sistema embarcado, Laia e Cruvinel (2012) utilizaram a geração automática de código do ambiente Simulink. Para gerar o código foram utilizadas as ferramentas HDL Worflow Advisor que gera o código HDL automaticamente e a Signal Compiler que compila o HDL gerado e o implementa em uma FPGA. No caso de funções que não são suportadas pelo gerador de código HDL foi utilizado o CORDIC (*Coordinate Rotation Digital Computer*). Os autores validaram o código gerado pelo Simulink com a implementação do filtro de Kalman em uma FPGA.

Com o objetivo de verificar o código traduzido no trabalho de Li e Kumar (2011), Li e Kumar (2012) desenvolveram uma ferramenta para testar automaticamente o código traduzido a partir do Simulink para uma máquina de estados finita. Foram implementadas duas técnicas para a verificação do código e posteriormente comparadas. As duas técnicas utilizadas foram *model-checking* e *constraint solving*. Como estudo de caso para validar e comparar a ferramenta desenvolvida foi utilizado um contador. Os resultados apresentaram que ambos os métodos geraram os resultados esperados e que a técnica de *constraint solving* é mais rápida.

Segundo os autores Majumdar et al. (2013), a verificação do código gerado pelo Simulink é necessária para sistemas de controle críticos industriais. Portanto, os autores desenvolveram uma ferramenta que realiza a verificação do código gerado automaticamente pelo Simulink denominada CSEC (*Compositional Symbolic Equivalence Checker*). O CSEC utiliza a metodologia *bottom-up* para verificar a equivalência de cada bloco do sistema e sua respectiva função. Os autores afirmam que ao se trabalhar com sistemas complexos em escala industrial a ferramenta resulta em alguns falsos positivos.

No trabalho de Netland e Skavhaug (2013) foi apresentada uma metodologia denominada SMRT (*Software Module Real-time Target*) que tem por objetivo a inserção do código gerado automaticamente pelo Simulink em um código de sistema embarcado existente. A metodologia SMRT consiste na substituição do arquivo que inicializa e executa o código gerado pelo Simulink. Este arquivo é substituído por uma biblioteca de funções que controla a comunicação do código gerado e o código desenvolvido a mão.

Os autores afirmam que a metodologia SMRT vem sendo usada no desenvolvimento de controle de sistemas embarcados.

Utilizando o Simulink como *front-end* para o desenvolvimento de projetos, Zou et al. (2013) realizaram a tradução de modelos Simulink para *Hybrid* CSP (*Communicating Sequential Processes*). HCSP é uma linguagem utilizada para especificar sistemas híbridos. O objetivo da tradução é analisar o modelo desenvolvido para buscar e identificar erros de projeto. Como estudo de caso foi utilizado um sistema de controle de trens chinesa de nível 3. Os resultados apresentados mostram que na tradução realizada e na simulação do sistema foram encontrados erros de projeto.

Discutindo a importância da verificação do código gerado automaticamente pelo Stateflow para o setor automotivo, Sampath, Rajeev e Ramesh (2014) apresentaram uma ferramenta que utiliza o verificador de código CBMC para validar a tradução realizada pelo ambiente Stateflow para a linguagem C. Os autores ressaltam que a ferramenta tem o objetivo de validar códigos gerados para sistemas complexos que são modelados com máquinas de estados hierárquicas. Os autores ressaltam que a ferramenta tende a ser expandida para trabalhar com o Simulink.

Seguindo a linha de pesquisa de análise dos sistemas descritos em alto nível de abstração, o trabalho de Araiza-Illan, Eder e Richards (2014) aborda a verificação das propriedades dos sistemas desenvolvidos no Simulink utilizando uma ferramenta denominada Why3. A metodologia utilizada nesta verificação é traduzir o modelo do Simulink para a ferramenta Why3 e realizar a verificação das propriedades em que ambos ocorrem automaticamente. Para facilitar a tradução foram desenvolvidos blocos no Simulink, em forma de biblioteca, com as funcionalidades necessárias para serem executadas na ferramenta Why3. Como estudo de caso foi desenvolvido um sistema de primeira ordem para ser verificado a estabilidade de Lyapunov. Os autores afirmam que os resultados obtidos em simulação conferem com os da metodologia proposta.

Krizan et al. (2014) apresentou um estudo com relação à geração automática do código C pelo ambiente Matlab/Simulink voltada a aplicações críticas. Como estudo de caso os autores desenvolveram um sistema de controle para um motor DC (*Direct Current*) sem escovas no Simulink. Foram utilizados apenas componentes básicos (soma, ganho, operadores lógicos, etc) para evitar códigos complexos. Após a análise e simulação, gerou-se o código C do sistema desenvolvido para ser implementando em um kit de desenvolvimento da *Texas Instruments* com o microcontrolador TMS320. Os resultados viabilizam a geração do código C com componentes básicos para o dispositivo utilizado.

Dando continuidade no trabalho de Li e Kumar (2011) em que se traduzia diagramas do Stateflow apenas com blocos disparados por *clock*, Li e Kumar (2014) apresentaram uma metodologia para traduzir um diagrama do Stateflow para um modelo formal de autômato finito que também traduz blocos disparados por eventos. Esta metodologia trata cada estado do diagrama desenvolvido como um módulo atômico e recursivamente aplica regras em cada bloco conforme a característica do mesmo. Como estudo de caso desenvolveram um controle de velocidade de um servo motor e aplicaram o diagrama na ferramenta desenvolvida. A tradução foi implementada na ferramenta SS2EFA (ferramenta de tradução de modelos autômatos) e validada. Os autores afirmam que o comportamento discreto do modelo desenvolvido foi mantido na tradução realizada.

No trabalho de Wang et al. (2014) foi apresentado um protótipo de uma ferramenta dividida em duas partes que funciona em sequência. Esta ferramenta tem por objetivo a tradução automática de um modelo desenvolvido no Simulink e posterior verificação do código traduzido. A primeira parte da ferramenta realiza a tradução do modelo Simulink para a linguagem C juntamente com o controle da semântica do sistema. A segunda parte verifica se a semântica está de acordo com o código traduzido. Como estudo de caso a ferramenta foi aplicada em um sistema de motor de jato. A tradução do modelo Simulink do motor de jato foi implementada em uma bancada de teste virtual. Os resultados apresentados validaram o protótipo da ferramenta desenvolvida

Foram revisados trabalhos a partir de 2005 e não encontrou-se ferramenta computacional com o mesmo objetivo da desenvolvida nesta pesquisa. No trabalho de Chindris e Muresan (2006) o autor cita a possível implementação de códigos traduzidos a partir do Simulink no sistema embarcado PSoC, porém não encontrou-se trabalhos do autor com esta implementação.

Com várias ferramentas de tradução utilizando o Simulink como ambiente de desenvolvimento de projetos e a não utilização destes códigos no ambiente PSoC Creator motivou o desenvolvimento da ferramenta MS^{2}PSoC. Salienta-se que além de traduzir o modelo desenvolvido no Simulink para as linguagens aceitas no PSoC, a ferramenta gera a estrutura de arquivos necessária para o projeto ser aberto no ambiente PSoC Creator.

2 AMBIENTES COMPUTACIONAIS E A LINGUAGEM DE DESCRIÇÃO DE HARDWARE VERILOG

Neste capítulo apresentam-se os conceitos utilizados para o desenvolvimento da ferramenta computacional MS^{2}PSoC. Na Seção 2.1 apresentam-se as características do ambiente MATLAB/Simulink. Na Seção 2.2 apresenta-se a arquitetura do sistema embarcado PSoC, suas características e o ambiente PSoC Creator. Uma das características do ambiente PSoC Creator é a possibilidade de descrever componentes digitais utilizando-se a linguagem de descrição de hardware Verilog. Portanto, na Seção 2.3 apresentam-se os recursos da linguagem aceitos pelo compilador do PSoC Creator, denominado *warp*.

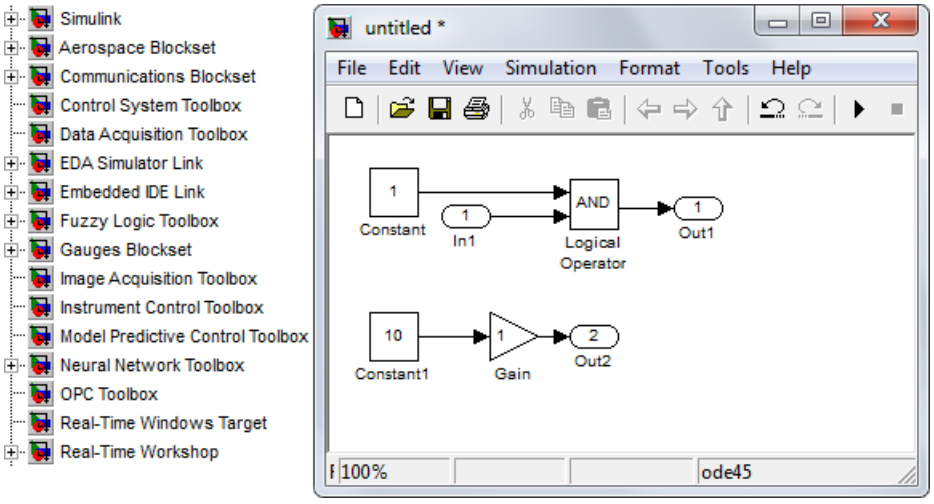
2.1 MATLAB/SIMULINK

O Matlab é uma ferramenta computacional que emprega uma linguagem em alto nível de abstração e disponibiliza um ambiente interativo para computação numérica, desenvolvimento de projetos de sinais mistos e programação. Com o Matlab é possível analisar dados, desenvolver algoritmos, criar e simular projetos com uso em diversas aplicações (MATHWORKS, 2013).

O Simulink é uma ferramenta disponível no Matlab que tem por função o desenvolvimento e simulação de sistemas dinâmicos, elétricos, eletrônicos, processamento de sinais, etc. Possui uma interface gráfica de diagramas de blocos que facilita o desenvolvimento de projetos. Uma característica do Simulink é a possibilidade de criar uma biblioteca de componentes, utilizando-se dos componentes existentes ou com componentes próprios (MATHWORKS, 2013).

Os projetos desenvolvidos são hierárquicos, ou seja, é possível transformar um sistema em componente para que este esteja disponível no Simulink. Na Figura 3 apresenta-se a janela de desenvolvimento de projetos do Simulink e algumas das bibliotecas disponíveis para uso.

Figura 3 – Ambiente computacional Simulink e algumas bibliotecas.

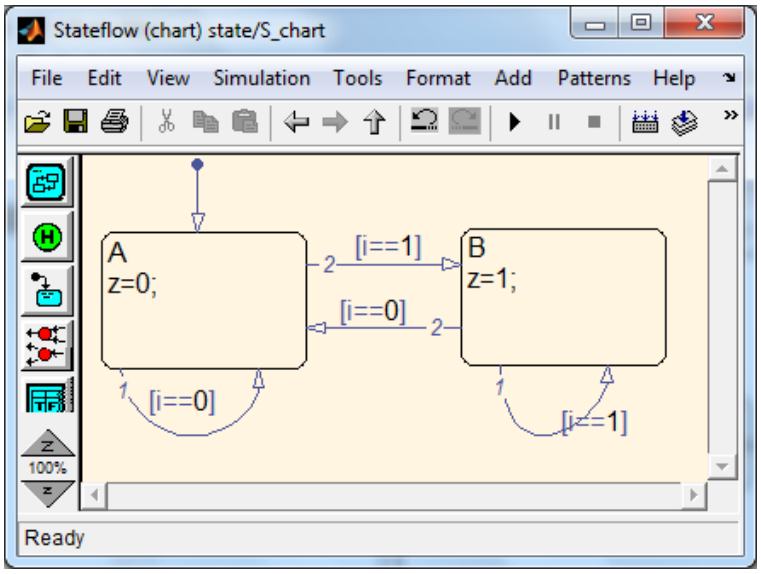


Fonte: Elaborada pelo autor.

O Matlab/Simulink foi escolhido para ser utilizado como *front-end* para o desenvolvimento de projetos, pois se trata de uma ferramenta com recursos para análise e simulação de sistemas digitais e analógicos desenvolvidos em alto nível de abstração. Um destes recursos é o desenvolvimento e análise de máquinas de estados finitos utilizando-se a ferramenta denominada Stateflow.

Segundo MathWorks (2013), o Stateflow é uma biblioteca disponível no ambiente Simulink em que é possível modelar máquinas de estados finitos. Devido à esta funcionalidade escolheu-se, também, utilizar o Stateflow para o desenvolvimento e análise de projetos. Na Figura 4 apresenta-se o ambiente Stateflow.

Figura 4 – Ambiente computacional Stateflow.



Fonte: Elaborada pelo autor.

2.2 PROGRAMMABLE SYSTEM-ON-CHIP (PSOC)

O *Programmable System-on-Chip* (PSoC) é um sistema embarcado que tem por principal característica a integração de componentes analógicos e digitais em um mesmo *chip*, dando suporte ao desenvolvimento de projetos de sinais mistos. O sistema embarcado PSoC é composto de um microcontrolador, memória, componentes analógicos e digitais e pinos de entrada e saída (CURRIE; ESS, 2010).

Outra característica do PSoC é o consumo de energia reduzido devido à sua arquitetura. Os componentes permanecem em modo *stand-by* e são ativados (energizados) via software quando necessário.

O PSoC é produzido pela empresa Cypress Semiconductor e atualmente consta de quatro gerações. A principal diferença entre as gerações se encontra no microcontrolador que é utilizado em cada uma. A primeira geração, o PSoC 1, utiliza o micontrolador de 8-bit M8C. A geração PSoC 3 que utiliza o microcontrolador de 8-bit 8051 foi a utilizada para o desenvolvimento deste trabalho. As duas outras gerações são o PSoC 4 que conta com o microcontrolador 32-bit ARM® Cortex™ - M0 e o PSoC 5 que utiliza o microcontrolador 32-bit ARM® Cortex™ - M3.

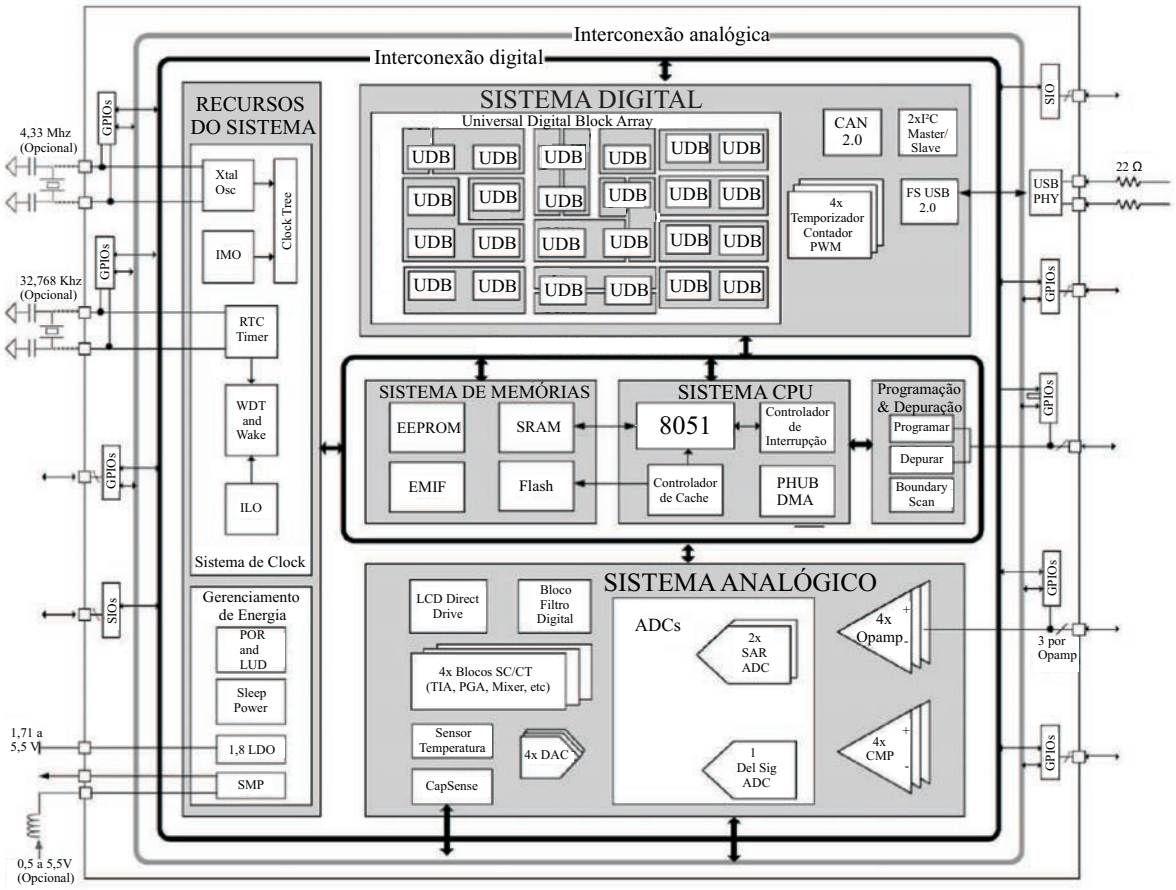
Como o PSoC 3 foi utilizado neste trabalho, apresentam-se características mais detalhadas sobre esta geração. O PSoC 3 integra o microcontrolador 8051, componentes disponíveis para o sistema (ex.: *clock*) e os sistemas analógicos e digitais programáveis à um sistema de entradas e saídas configuráveis (CURRIE; ESS, 2010). A arquitetura do PSoC 3 é apresentada na Figura 5.

Uma característica relevante do ambiente PSoC Creator é a possibilidade de descrever componentes e sistemas utilizando-se da linguagem de descrição de hardware Verilog. A implementação do sistema digital do PSoC Creator é realizada nas UDBs (*Universal Digital Blocks*). Na Figura 6 apresenta-se o diagrama de blocos de uma UDB.

O PSoC 3 contém uma estrutura com 24 UDBs. Em cada UDB contém dois PLDs (*Programmable Logic Device*) que são utilizados para implementar máquinas de estados, realizar entrada e saída de dados condicionais e criar *lookup tables* (LUTs) (CORPORATION, 2014a).

O bloco *Datapath* contém uma ULA (Unidade Lógica Aritmética) dinamicamente programável, quatro registradores, dois FIFOs (*First In First Out*), comparadores e geradores de condição. O *Status* e Controle controlam o roteamento interno da UDB e o estado dos registradores. O bloco Clock e Controle do Reset fornecem a seleção de

Figura 5 – Arquitetura do sistema embarcado PSoC 3.



Fonte: Corporation (2014a).

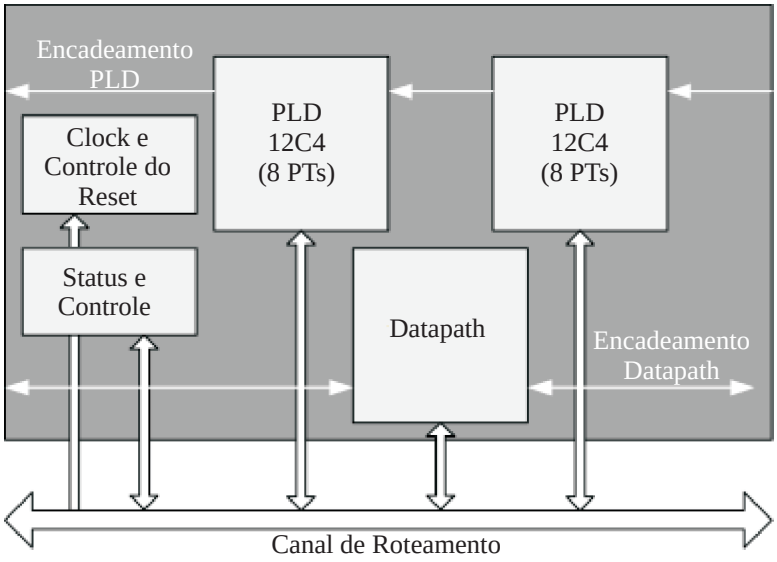
clock e reset para os outros blocos da UDB (CORPORATION, 2014a).

Além destes blocos, a UDB possui sinais que interconectam os blocos para realizar funções com maior precisão, um canal de roteamento para se interligar com outras UDBs e um barramento do sistema que está conectado a todos registrados e memórias RAM (*Random Access Memory*) acessíveis pela CPU (*Central Processing Unit*) e DMA (*Direct Memory Access*) (CORPORATION, 2014a).

Os componentes analógicos do PSoC 3 estão implementados no *chip* do sistema embarcado e são ativados via software. O sistema analógico possui um barramento que interliga os componentes com os portos de entrada e saída. Os componentes disponíveis[1] são amplificador operacional, amplificador operacional com ganho programável, amplificador de transimpedância, modulador de sinal, amplificador *Sample and Hold*, amplificador *Track and Hold*, comparadores, conversores digital - analógico (D/A) com resolução de 8 bits, conversor analógico - digital (A/D) com resolução de 8 a 20 bits e multiplexador.

[1]Ressalta-se que o software utilizado é o PSoC Creator 2.2 Component Pack 6.

Figura 6 – Diagrama de blocos do UDB.



Fonte: Corporation (2014a).

Segundo Corporation (2014a), os sinais analógicos podem ser roteados utilizando o barramento analógico interno, permitindo ao dispositivo trabalhar com até 62 sinais analógicos. O roteamento pode ser feito manualmente ou o ambiente PSoC Creator otimiza o roteamento automaticamente. Na Figura 7 apresenta-se um exemplo deste roteamento utilizando dois amplificadores operacionais.

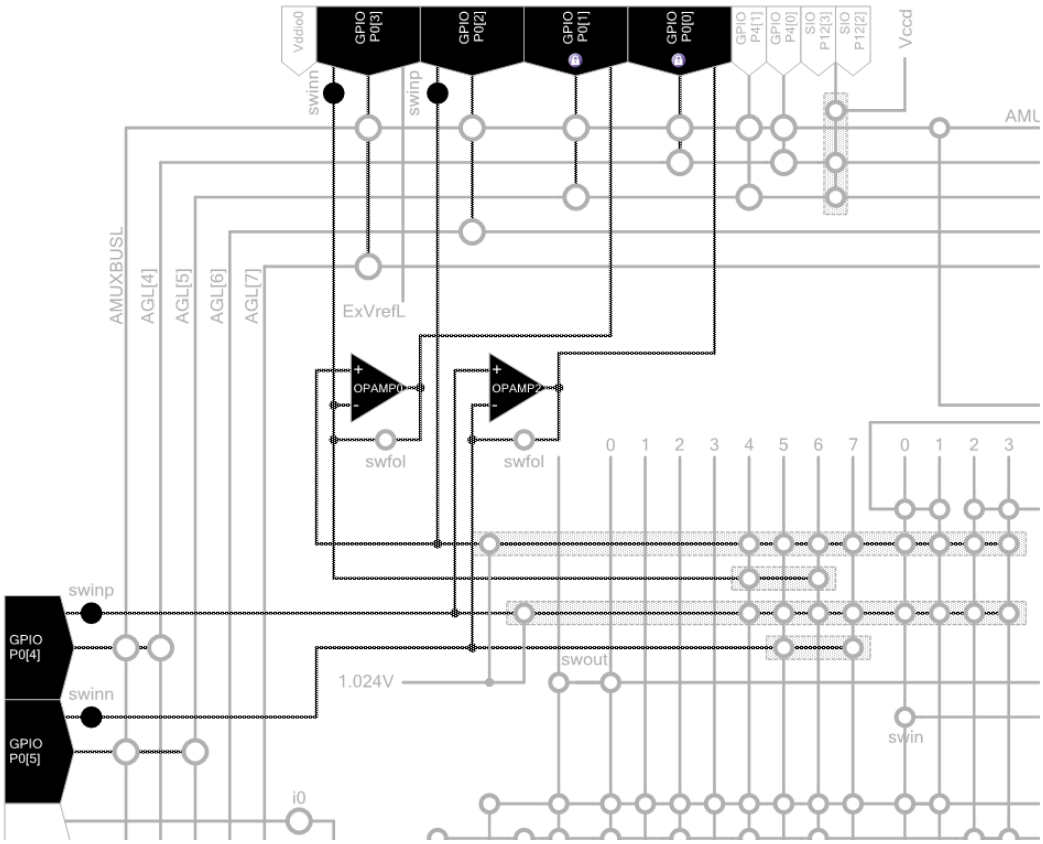
Nota-se, na Figura 7, que os portos P0[2] e P0[3] estão conectados, respectivamente, às entradas positiva e negativa do amplificador operacional 0 (OPAMP0). As entradas positiva e negativa do amplificador 2 (OPAMP2) estão conectadas aos portos P0[4] e P0[5], respectivamente. A saída do OPAMP0 esta conectada ao porto P0[1] e a saída do OPAMP2 esta conectada ao porto P0[0].

Outra característica do PSoC é a possibilidade de programar com Assembly ou com a linguagem de programação C. Com a linguagem C é possível ativar componentes analógicos, ler e escrever em portos de entrada e saída, realizar operações matemáticas, etc. O software, no PSoC 3, é implementado no microcontrolador 8051.

As instruções do 8051, no PSoC 3, são compatíveis com o original MCS-51 e executadas em um único ciclo de máquina em *pipeline*, chegando a 67 MHz. Um ciclo da CPU do 8051 é executado dez vezes mais rápido que o 8051 padrão (CORPORATION, 2014a).

O subsistema da CPU inclui *Nested Vectored Interrupt Controller*[1] (NVIC) programável, controle de DMA, memória flash cache ECC (*Error Checking & Correction*) e RAM. O controle de DMA permite a comunicação dos periféricos sem envolver a CPU

Figura 7 – Exemplo de roteamento analógico do PSoC 3.



Fonte: Elaborada pelo autor.

no processo, fazendo com que a CPU trabalhe mais devagar economizando energia ou utilizando o ciclo para melhorar a resposta de algoritmos (CORPORATION, 2014a).

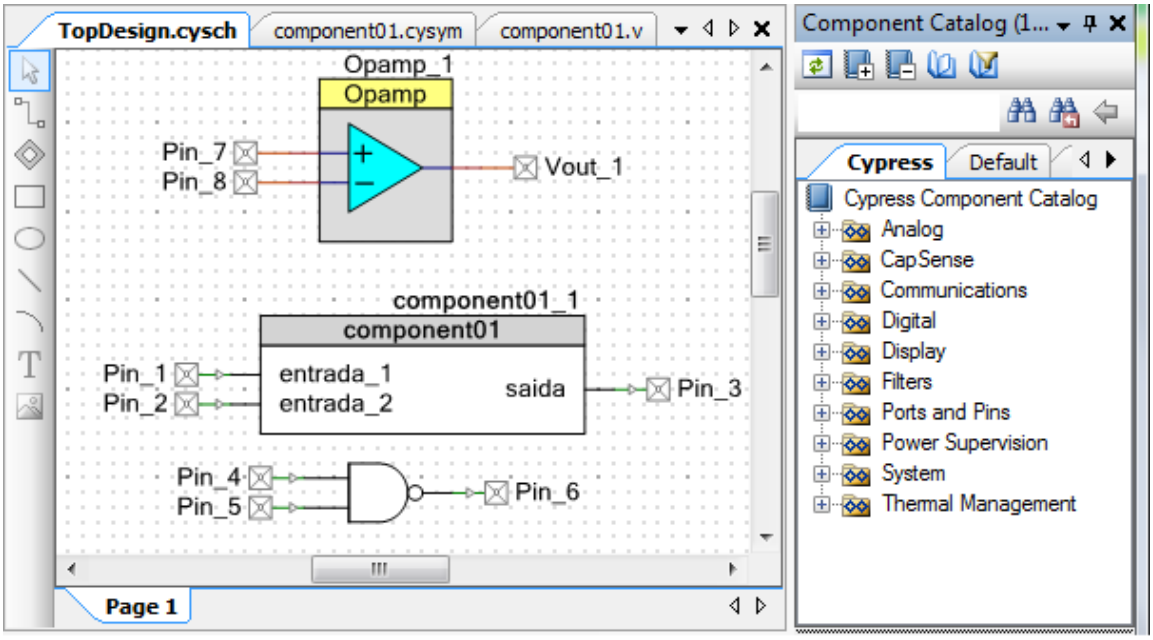
A IDE (*Integrated Design Environment*) utilizada para trabalhar com a geração do PSoC 1 é o PSoC Designer™, enquanto que para as outras gerações trabalha-se com o PSoC Creator, alvo deste trabalho. Na Figura 8 apresenta-se o ambiente gráfico PSoC Creator.

O PSoC Creator é um ambiente gráfico onde componentes são instanciados em forma de blocos com as respectivas funções previamente definidas. Com o ambiente PSoC Creator é possível, graficamente, configurar e conectar os blocos necessários para o desenvolvimento de projeto.

Para configurar os blocos analógicos pode-se utilizar a linguagem C, e a Verilog HDL para implementar componentes digitais. No caso do PSoC 3, pode-se ainda utilizar as instruções do microcontrolador 8051.

[1]Suporte a redefinição de prioridades dinâmica, permitindo que uma interrupção seja executada antes que outras.

Figura 8 – Ambiente PSoC Creator.



Fonte: Elaborada pelo autor.

O ambiente PSoC Creator não disponibiliza um simulador, sendo este um fator importante que contribuiu para o desenvolvimento da ferramenta de tradução desenvolvida neste trabalho. Com o ambiente Matlab/Simulink pode-se analisar e simular o modelo desenvolvido, garantindo o funcionamento para posterior tradução e implementação no PSoC.

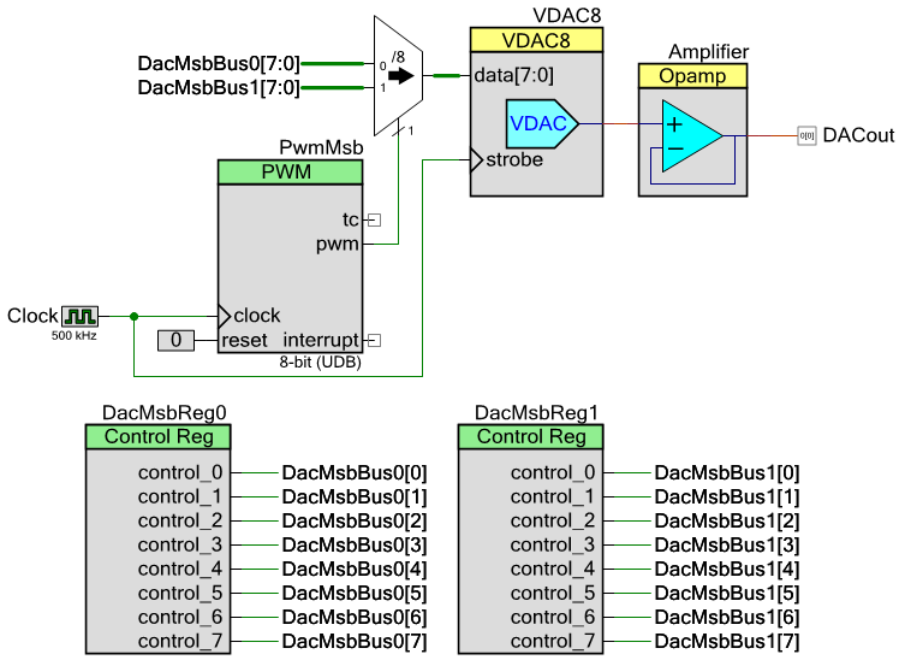
2.2.1 Conversor D/A dithering

O sistema embarcado PSoC possui um conversor D/A de 8 bits que pode operar na faixa de 0 - 1,020 V (4 mV/bit) ou de 0 - 4,080 V (16 mV/bit). Com o objetivo de obter uma maior resolução, desenvolveu-se o circuito apresentado na Figura 9 utilizando-se o conversor D/A operando na faixa de 0 - 4,080 V.

Para desenvolver este circuito utilizou-se a técnica de *dithering* que consiste na inserção proposital de ruído no sistema com o objetivo de se aumentar sua resolução. Este sistema é utilizado juntamente com a ferramenta de tradução MS^{2}PSoC para gerar a tensão de saída dos sistemas traduzidos.

O funcionamento do sistema se dá pelo controle da saída do conversor D/A com um PWM. Por exemplo, o conversor D/A possui uma resolução de 16 mV/bit em uma escala de 256 valores, ou seja, cada número inteiro entre 0 e 255 representa 16 mV na saída do conversor.

Figura 9 – Conversor D/A dithering



Fonte: Elaborada pelo autor.

Em um conversor com esta resolução não é possível ajustar a saída para 40 mV. Para tanto utiliza-se a técnica de *dithering* em que ajusta o PWM para que alterne a saída do conversor entre 48 mV (50% do ciclo) e 32 mV (50% do ciclo). Deste modo, obtém-se a média na saída de 40 mV.

Com este conceito aumentou-se a resolução do conversor para 1 mV/bit utilizando-se 12 bits. A Listagem 1 apresenta o código utilizado para configura o conversor desenvolvido.

Listagem 1 – Código utilizado para configurar o conversor D/A dithering.

```
#include <device.h>
#include <stdio.h>

void main()
{
    uint16 DacValue;
    VDAC8_Start();
    PwmMsb_Start();
    Amplifier_Start();
    DacValue = 350;
    DacMsbReg0_Write(DacValue >> 4);
    DacMsbReg1_Write((DacValue >> 4) + 1);
    PwmMsb_WriteCompare((uint8)(DacValue & 0X0F));
}
```

Para ajustar um valor na saída do conversor utiliza-se a variável *DacValue* que expressa a tensão de saída do conversor desenvolvido em milivolts. Este conversor foi utilizado juntamente com as implementações dos estudos de casos apresentadas no Capítulo 5.

2.3 LINGUAGEM DE DESCRIÇÃO DE HARDWARE VERILOG

O PSoC Creator é um ambiente que permite ao projetista descrever componentes utilizando a linguagem Verilog HDL. Realizou-se um estudo sobre a linguagem e documentou-se os principais aspectos visando possíveis trabalhos futuros.

Como o compilador Verilog do ambiente PSoC Creator, denominado *warp*, não suporta todas as funcionalidades disponíveis na linguagem Verilog, focou-se o estudo somente nas funções suportadas pelo compilador. O PSoC Creator trabalha com a linguagem Verilog HDL padrão IEEE 1364-2005.

A linguagem Verilog é uma linguagem que permite a especificação de um sistema digital em vários níveis de abstrações, sendo que estes níveis podem estar especificados em um mesmo modelo. A linguagem possui construção hierárquica, permitindo um controle na complexidade da descrição (THOMAS; MOORBY, 2002).

2.3.1 Módulos

As unidades básicas do Verilog são os módulos (*module*). Nele se encontram todas as declarações de entradas e saídas, podendo ser descritos em vários níveis de abstração (comportamental, RTL e estrutural). Os módulos são executados de forma concorrente, ou seja, todos os processos contidos nos módulos ocorrem simultaneamente. Para que um código seja executado de forma sequencial, é necessário que esteja inserido em um bloco *begin/end*.

Um projeto em Verilog consiste em um ou mais módulos conectados por portos que realizam a conexão de vários elementos de hardware. Cada porto possui um nome e um tipo associado, podendo ser uma entrada (*input*), uma saída (*output*) ou bidirecional, ou seja, entrada e saída (*inout*). Na Listagem 2 apresenta-se um exemplo de como é realizada a descrição de um módulo.

Listagem 2 – Exemplo da descrição de um módulo em Verilog HDL.

```
module component (
        input In1,
        input In2,
        output Out1,
        output Out2
        );
endmodule
```

A descrição de um componente inicia-se com a palavra *module*, seguido do nome que será dado ao módulo e entre os parênteses declaram-se os portos. Os portos declarados no módulo por definição são do tipo *wire*, mas se a saída que está sendo declarada for do tipo *register*, altera-se a declaração ficando a descrição da seguinte maneira:

output reg out1

A hierarquia nos projetos em Verilog ocorre instanciando-se um ou mais módulos em um módulo *top level*, que pode ser definido como um módulo que não pode ser instanciado. É possível instanciar um mesmo módulo várias vezes. Na Listagem 3 apresenta-se um modelo com um módulo sendo instanciado. Como pode ser visto, o módulo *my_dff* é instanciado quatro vezes.

Listagem 3 – Exemplo de instanciação de um módulo em Verilog HDL.

```
module shift_reg (
        input d,
        input clk,
        output q
        );
        wire q0, q1, q2;
        my_dff i0 (clk, d, q0);
        my_dff i1 (clk, q0, q1);
        my_dff i2 (clk, q1, q2);
        my_dff i3 (clk, q2, q);
endmodule
```

A conexão entre os módulos instanciados é realizada utilizando-se de sinais do tipo *wire*. Esta conexão pode ser feita por associação ordenada ou a associação por nome dos portos. Na Listagem 4 apresenta-se um exemplo com a diferença entre as associações.

Listagem 4 – Exemplo de conexão de módulos Verilog HDL.

```
my_dff i0 (clk, d, q0);      //associação por ordem
my_dff i1 (.d(d), .q(q0), .clk(clk)); //associação por nome
```

Nota-se que a principal diferença entre as duas formas de conexão é o fato de que na associação por ordem, deve-se seguir, obrigatoriamente, a mesma ordem que os portos foram declarados no módulo que está sendo instanciado. Entretanto, na associação por nome, é obrigatório explicitar a conexão que está sendo realizada.

Os comentários feitos na linguagem Verilog são realizados após o símbolo "//", para comentar a linha inteira. Para comentar um bloco de texto utiliza-se os símbolos "/* */" e todo o conteúdo que estiver entre estes dois símbolos será interpretado como comentário.

2.3.2 Constantes

Valores constantes podem ser declarados em decimal, hexadecimal, octal ou binário, podendo começar com + ou -, opcionalmente. Para o Verilog utilizado no PSoC Creator, ao não se especificar o tamanho da constante, o software adota um tamanho de 32 bits (CORPORATION, 2013).

A declaração de tamanho não definido consta de um número decimal usando dígitos de 0 a 9. A declaração de tamanho definido é composta pelo tamanho da constante, a base em que está declarada e uma sequência de dígitos representando o seu valor. Na Listagem 5 apresentam-se os tipos de declaração de constantes. O sinal "_", no último exemplo, é utilizado apenas para tornar o número mais legível.

Listagem 5 – Exemplos de declarações de constantes.

```
10   //Número decimal
−10   //Número decimal com sinal
2'b1  //Número binário de dois bits (01)
'ha7f  //Número hexadecimal sem tamanho definido
9'o17  //Número octal de nove bits (000001111)
'b0101_1110 //Número binário igual a 01011110
```

2.3.3 Tipos de dados

O Verilog possui três tipos de dados: *net*, *reg* e *parameter*.

2.3.3.1 Nets

O tipo de dado denominado *net* é utilizado para representar uma conexão física entre diferentes blocos de hardware, sendo que este tipo de dado não é capaz de armazenar

nenhum valor e pode ser dos seguintes tipos:

- *wire/tri*: são usados para conectar diferentes elementos de hardware. Ambos são idênticos, tendo nomes diferentes apenas para distinguir o propósito da ligação e tornar o projeto mais legível. O *wire* é usado para *net* de porta simples ou atribuição contínua. O *tri* é usado quando se tem múltiplos *drivers*. Define-se *driver* como sendo uma fonte de sinal;
- *supply0/supply1*: são usados para modelar fontes de alimentação. O *supply0* representa nível lógico 0 (*ground*) e o *supply1* representa nível lógico 1 (*vcc*).

Os dados do tipo *net* podem ser escalares ou vetoriais, sendo diferenciadas na sua declaração. Os escalares representam sinais individuais e os vetoriais representam barramentos.

2.3.3.2 Registers

Os registradores são usados como variáveis e são capazes de armazenar seu valor até que outra atribuição atualize o seu valor. São declarados com a palavra *reg* e só podem ser utilizados nos blocos *always*, *function* e *task*. Um outro tipo de dado de registrador é o *integer*, possuindo uma quantidade fixa de 32 *bits*.

Os blocos *function* e *task* não foram abordados, pois não são suportados pelo compilador *warp*.

2.3.3.3 Parameters

Os dados do tipo *parameter* são constantes e não podem ter o seu valor alterado durante o tempo de execução. Entretanto *parameter* declarados dentro de módulos instanciados podem ser alterados durante a instanciação. Caso se encontrem dentro de um módulo, o seu valor pode ser redefinido utilizando-se o recurso *defparam*:

defparam *modulo_sendo_instanciado.variavel* = *valor*;

2.3.4 Bloco *always*

O bloco *always* é utilizado quando determinado código deve ser executado sempre que algumas condições forem satisfeitas. Estas condições estabelecidas é conhecida como

lista de sensibilidade, sendo obrigatório a sua presença. Na Tabela 1 apresentam-se três possibilidades de construção, sendo uma assíncrona, uma síncrona com a borda de subida e síncrona com a borda de descida do sinal.

Tabela 1 – Exemplo de construções do bloco *always*.

//1.Disparo assíncrono	//2.Disparo síncrono	//3.Disparo síncrono
always @(x **or** y)	**always** @(**posedge** clk)	**always** @(**negedge** clk)
begin	**begin**	**begin**
//ocorre quando há mudança	//ocorre na borda de subida	//ocorre na borda de descida
//de estado em x ou y.	//do sinal de clk.	//do sinal de clk.
end	**end**	**end**

Fonte: Elaborada pelo autor.

No disparo assíncrono a lista de sensibilidade tem como parâmetros as variáveis "x" e "y", fazendo com que o bloco seja executado sempre que ocorrer uma transição de estado em uma das duas variáveis. No disparo síncrono a palavra *posedge* faz com que o bloco seja executado sempre que houver a transição na borda de subida da variável "clock". Analogamente, para a borda de descida usa-se a palavra *negedge*. Ambas as possibilidades (síncrono e assíncrono) não podem ocorrer em um mesmo bloco *always*.

2.3.5 Atributos

As atribuições no Verilog podem ocorrer de duas maneiras, contínua ou processual. A atribuição contínua, como o próprio nome diz, ocorre continuamente e deve ser usada "fora" do bloco *always*, sendo utilizada com o tipo de dado *wire* com a palavra *assign*. O "lado" direito da atribuição pode conter dados do tipo *wire*, *reg*, constante ou uma combinação dos três. Na Listagem 6 apresenta-se um exemplo de como é realizada esta atribuição.

Listagem 6 – Exemplo de atribuições contínuas.

```
//z e p são do tipo wire
assign z = x & y
assign p = 1'b1;
```

A atribuição processual ocorre dentro do bloco *always* e pode ser de dois tipos, *blocking* e *non-blocking*. Na atribuição *blocking* utiliza-se o operador "=", sendo executada de forma sequencial, ou seja, uma atribuição após a outra. Na atribuição *non-blocking* utiliza-se o operador "<=", sendo executada de forma paralela, ou seja, todas as atribuições ocorrem concorrentemente. Na Tabela 2 apresenta-se um exemplo destacando a diferença entre as

duas formas de atribuição.

Tabela 2 – Exemplo de atribuições *blocking* e *non-blocking*.

Valores iniciais: a=1, b=2, c=3 clk -> clock do sistema	
//Atribuição blocking	//Atribuição non-blocking
always @(**posedge** clk)	**always** @(**posedge** clk)
begin	**begin**
a = b;	a <= b;
c = a;	c <= a;
end //final: a = b = c = 2	**end** //final: a = b = 2, c = 1

Fonte: Elaborada pelo autor.

No exemplo apresentado na Tabela 2 considera-se que todas as variáveis são registradores e que os valores iniciais são iguais a **a=1**, **b=2**, **c=3**, e que clk é utilizado como *clock* do sistema. Na atribuição *blocking*, após um pulso de clk, as variáveis "a", "b" e "c" assumem o valor 2, pois trata-se de um código sequencial. Na atribuição *non-blocking*, após o pulso de clk, as variáveis "a" e "b" assumem o valor 2 e o "c" assume 1, pois as duas atribuições são executadas concorrentemente, ou seja, o valor que "c" recebe é o valor de "a" anterior ao sinal de clk ser disparado.

Existem algumas regras quanto ao uso das atribuições (CORPORATION, 2013):

- Em lógica sequencial como máquinas de estado finitas, usa-se atribuições *non-blocking*;
- Em lógica combinacional dentro de um bloco *always*, usa-se atribuição *blocking*;
- Em lógica sequencial e combinacional no mesmo bloco *always*, usa-se atribuição *non-blocking*;
- Não misturar atribuições *blocking* e *non-blocking* no mesmo bloco *always*;
- Não realizar atribuições a uma mesma variável em mais de um bloco *always*.

2.3.6 Diretivas do compilador

As diretivas aceitas pelo compilador *warp* são (IEEE, 2006):

- **`define**: cria uma macro para substituição de texto. Pode ser usado "dentro" ou "fora" de uma definição de módulo. Exemplo: `define selA 4'b1;

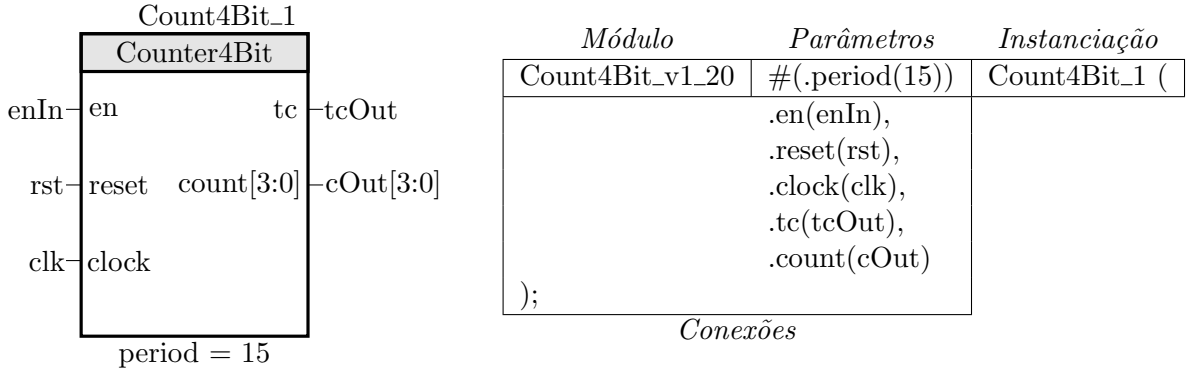
- **`undef**: serve para cancelar a definição feita com o `define. Exemplo: `undef selA;
- **`include**: permite inserir todo o conteúdo de outro arquivo durante a compilação. Exemplo: `include “cypress.v”.

2.3.7 Instanciação de um componente no PSoC Creator

No ambiente PSoC Creator é possível realizar a instanciação de módulos existentes na biblioteca do software ou de módulos criados pelo usuário. A instanciação é realizada de maneira semelhante ao esquemático, ou seja, arrasta-se um componente para o projeto gráfico, o que equivale a instanciá-lo utilizando o código Verilog.

Como exemplo, instanciou-se um contador de 4 bits que se encontra disponível na biblioteca do PSoC Creator. O bloco do contador é apresentado na Figura 10, bem como, o padrão estabelecido que deve ser seguido para instanciá-lo. Este padrão consta do nome do módulo a ser instanciado, seguido dos parâmetros definidos, o nome dado a instanciação e suas devidas conexões.

Figura 10 – Exemplo de instanciação de componente.



Fonte: Corporation (2013).

Definido o modo como o bloco deve ser instanciado, apresenta-se o código Verilog completo de uma instanciação. Neste exemplo o objetivo foi o de criar um contador de 4 bits instanciando o componente já existente em biblioteca. Na Listagem 7 apresenta-se o código Verilog do contador de 4 bits. O caminho do arquivo a ser instanciado deve ser declarado no código.

Listagem 7 – Exemplo de código Verilog utilizando a instanciação de componente.

```
//`#start header` -- edit after this line, do not edit this line
//====================================
//
//Instanciação de um componente existente
//no PSoC Creator para implementação de
//um contador de 4 bits.
//
//====================================

`include "cypress.v"
`include "<FullPath>_Count4Bit_v1_20.v"
//`#end` -- edit above this line, do not edit this line
//Component: contador4bits

module contador4bits (
        input clk,
        input rst,
        input enable,
        output [3:0] countOut,
        output tcOut
);

//`#start body` -- edit after this line, do not edit this line

count4Bit_v1_20 #(.period(15)) contador4bits_1 (
                .clock(clk),
                .reset(rst),
                .en(enable),
                .tc(tcOut),
                .count(countOut)
                );

        //`#end` -- edit above this line, do not edit this line
endmodule
        //`#start footer` -- edit after this line, do not edit this line
        //`#end` -- edit above this line, do not edit this line
```

No Capítulo 3 apresenta-se a metodologia utilizada no desenvolvimento da ferramenta de tradução MS^{2}PSoC.

3 FERRAMENTA MS^{2}PSoC

A ferramenta computacional desenvolvida denominada MS^{2}PSoC (Matlab/Simulink para PSoC Creator) traduz um modelo descrito em alto nível de abstração no ambiente Matlab/Simulink para especificações que podem ser sintetizadas em componentes PSoC no ambiente PSoC Creator. A metodologia utilizada na tradução é apresentada na Seção 3.1. Na Seção 3.2 apresenta-se a interface da ferramenta MS^{2}PSoC criada com o framework Qt.

3.1 METODOLOGIA

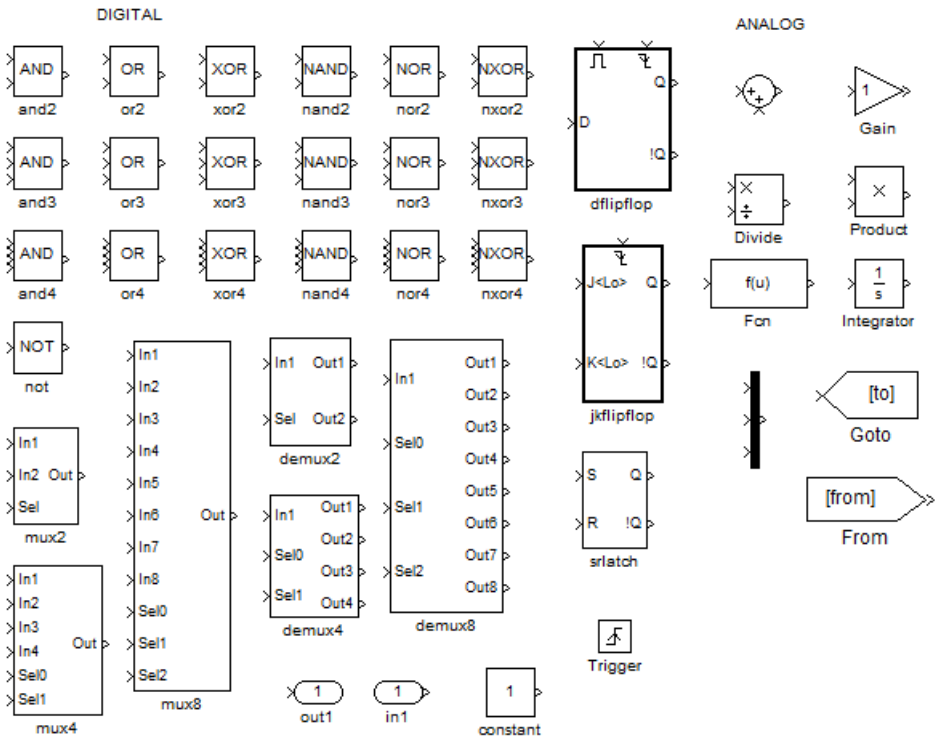
A ferramenta MS^{2}PSoC foi desenvolvida utilizando a linguagem de programação C ANSI. O objetivo desta ferramenta é o de auxiliar a implementação de modelos desenvolvidos no ambiente Simulink em hardware PSoC. Como o PSoC Creator não oferece suporte para a simulação, optou-se por utilizar o Simulink para esta finalidade.

A abordagem utilizada pela ferramenta MS^{2}PSoC é a de traduzir o sistema modelado no Simulink para a linguagem Verilog ou para a linguagem de programação C, sendo que alguns componentes de hardware podem ser inseridos diretamente no ambiente PSoC Creator.

A ferramenta possui uma biblioteca própria denominada *ms2psoc_lib* contendo componentes analógicos e digitais que devem ser utilizados para o desenvolvimento dos modelos no Simulink. Cada componente digital da biblioteca *ms2psoc_lib* tem o seu código Verilog que é utilizado na instanciação, sendo este código aceito pelo compilador *warp*. Os componentes analógicos terão suas funções implementadas em linguagem C.

Para o correto funcionamento da ferramenta, o modelo descrito no Simulink deve ser desenvolvido somente com os componentes da biblioteca *ms2psoc_lib*. Caso seja utilizado algum componente que não esteja presente nesta biblioteca, ocorrerá um erro e a conversão do projeto será finalizada. Na Figura 11 apresentam-se os componentes da biblioteca *ms2psoc_lib*.

Figura 11 – Biblioteca *ms2psoc_lib*.



Fonte: Elaborada pelo autor.

Um ponto relevante para a descrição do modelo no Simulink é que a ferramenta MS^{2}PSoC funciona com a propriedade *SubSystem*, ou seja, é possível criar hierarquia de componentes para gerar um novo componente. Por exemplo, pode-se utilizar portas lógicas para gerar um flip-flop.

Outro aspecto importante com relação à ferramenta é o funcionamento com o *toolbox* Stateflow. Ao se gerar um componente com o Stateflow, é necessário que o nome do componente se inicie com "S_" ou com "F_" no caso do modelo Simulink conter função juntamente com a descrição da máquina de estados finita. Para o sistema digital, é necessário que esteja na forma de subsistema e o nome do componente gerado se inicie com "D_". Desta forma, a ferramenta identifica o sistema digital e gera a respectiva descrição para ser implementada no ambiente PSoC Creator.

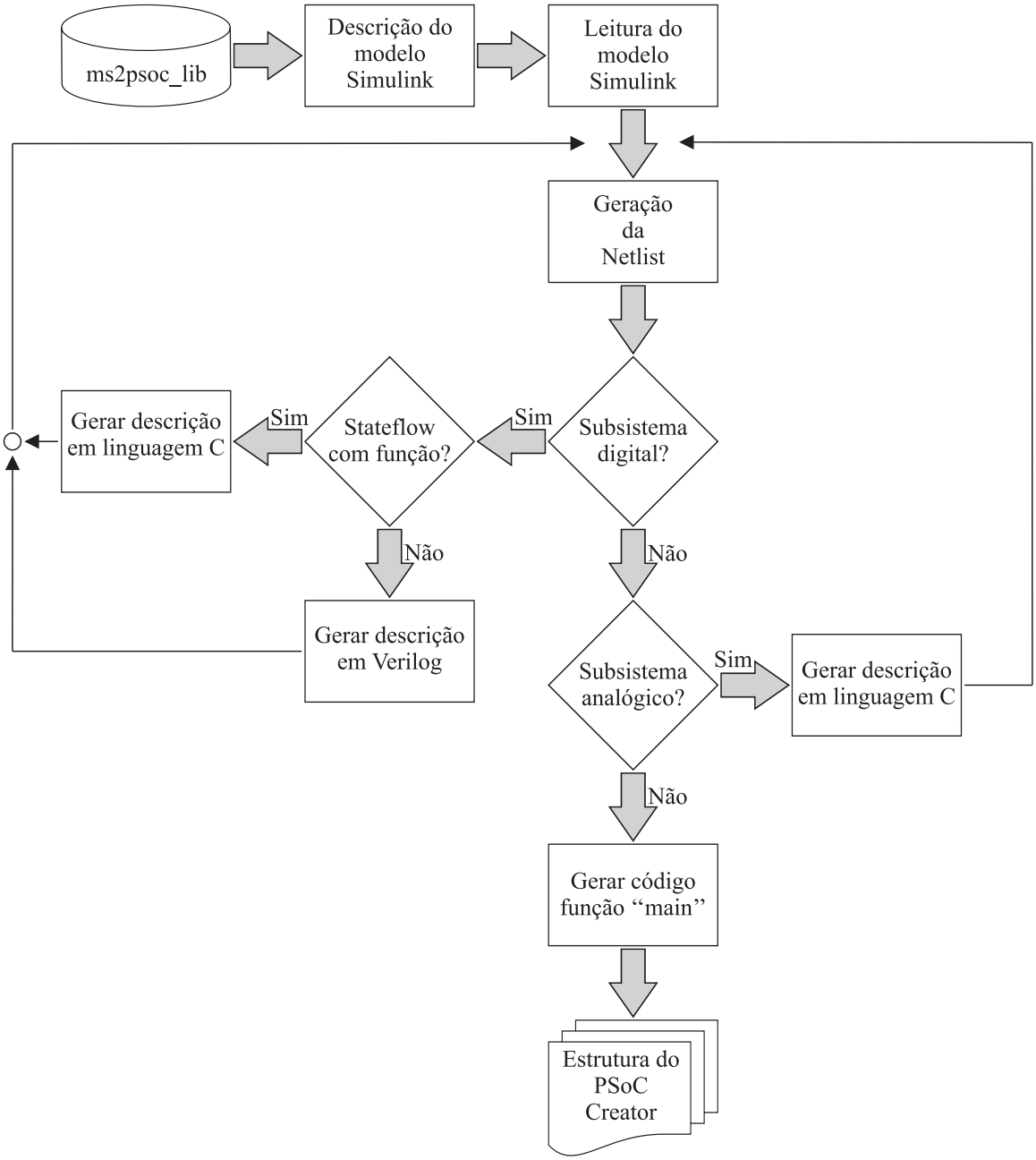
As diferenças da ferramenta MS^{2}PSoC com relação às ferramentas de conversão convencionais é a geração do código aceito pelo compilador *warp*, presente no PSoC Creator e a geração da estrutura de arquivos necessária para que o modelo seja aberto no ambiente PSoC Creator.

A metodologia empregada no desenvolvimento da ferramenta consiste na conversão de um modelo desenvolvido e simulado no Simulink para o ambiente PSoC Creator. Após

a conversão, o modelo é configurado e implementado em hardware PSoC.

O sistema digital é traduzido para a linguagem Verilog HDL ou para a linguagem C, no caso de diagramas de transições de estados descritos no Stateflow contenham funções. Um sistema analógico é traduzido para linguagem C, enquanto que os componentes de hardware são instanciados no ambiente PSoC Creator. Na Figura 12 apresenta-se o fluxograma com a metodologia utilizada na ferramenta MS^{2}PSoC.

Figura 12 – Fluxograma da ferramenta MS^{2}PSoC.



Fonte: Elaborada pelo autor.

O funcionamento da ferramenta se inicia pelo desenvolvimento de um modelo em alto nível de abstração no Matlab/Simulink utilizando somente componentes presentes na

biblioteca *ms2psoc_lib*.

Com o modelo desenvolvido e simulado, inicia-se a execução da ferramenta MS^{2}PSoC. O arquivo gerado pelo Simulink (extensão .mdl) é utilizado como entrada na ferramenta para que as informações do modelo desenvolvido sejam obtidas.

As informações que a ferramenta busca são os componentes utilizados na descrição do modelo e as respectivas conexões realizadas. As informações dos componentes estão organizadas por blocos, identificados pela palavra "*Block*" e as conexões pela palavra "*Line*". Na Listagem 8 apresenta-se um exemplo de descrição de um componente e uma conexão contida no arquivo Simulink.

Listagem 8 – Exemplo de descrição de um bloco e uma ligação no arquivo Simulink.

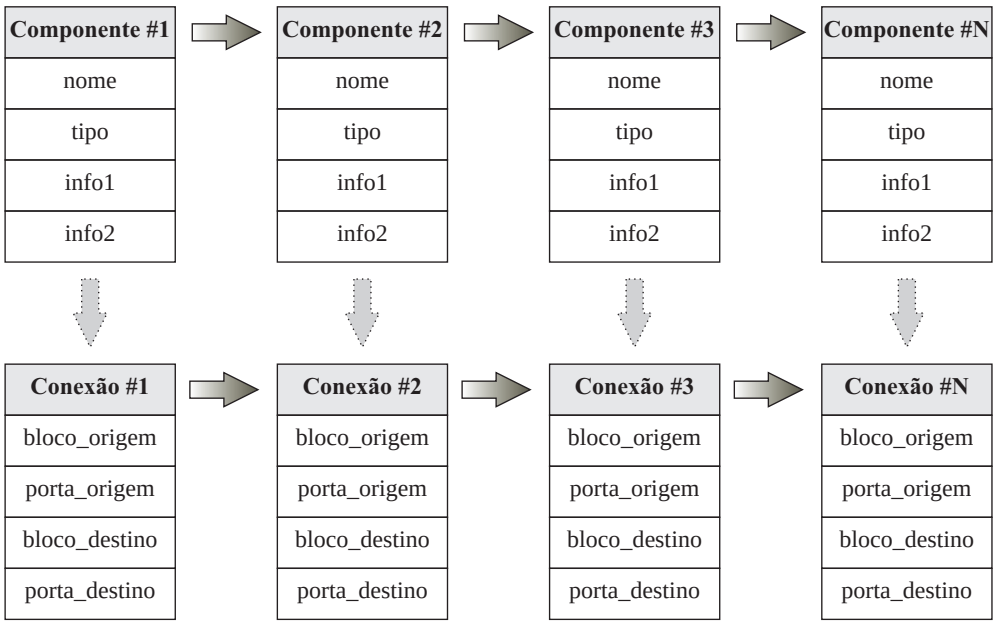
```
Block {
  BlockType  Logic
  Name   "and2"
  SID   "20"
  Ports   [2, 1]
  Position  [295, 102, 325, 133]
  AllPortsSameDT off
  OutDataTypeStr "boolean"
}
Line {
  SrcBlock  "constante"
  SrcPort  1
  DstBlock  "jkflipflop"
  DstPort  2
}
```

A ferramenta ao realizar a leitura identifica se trata de um componente ou de uma conexão e armazena as informações em memória na forma de estrutura dinâmica, conforme apresentada na Figura 13.

No caso de um componente (*Block*) são armazenados o nome do componente, o tipo de componente e outras informações dependendo do tipo de componente, por exemplo, o valor no caso de constantes. Para as conexões, são armazenados as informações sobre os componentes que estão sendo conectados e suas respectivas portas utilizadas nesta conexão. Foi utilizada alocação dinâmica para armazenar as informações para se obter uma maior flexibilidade quanto ao número de componentes e conexões.

Após a leitura do componente a ferramenta verifica se o componente é digital ou analógico. Sendo o componente digital, inicia-se a leitura do sistema contido neste

Figura 13 – Ilustração do armazenamento das informações.



Fonte: Elaborada pelo autor.

componente para ser realizada a descrição Verilog. Caso o componente for descrito no Stateflow juntamente com uma função, descreve-se o componente com a linguagem C. Caso contrário, descreve-se com linguagem Verilog.

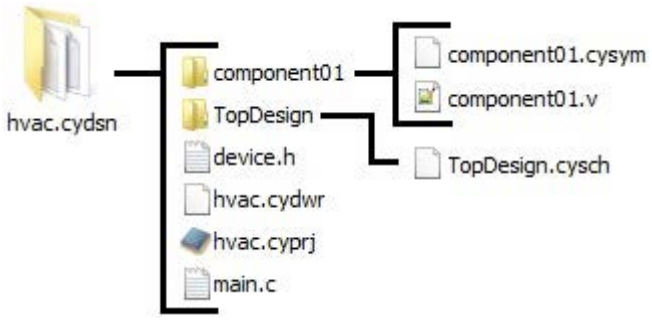
Cada descrição Verilog gerada na conversão do componente digital é implementada em um componente gerado no ambiente PSoC Creator, ou seja, para cada subsistema digital no Simulink, um componente correspondente é gerado no PSoC Creator.

Se o componente é analógico, a ferramenta verifica tratar-se de um subsistema. Se for subsistema, inicia-se a conversão deste subsistema analógico para a linguagem de programação C na forma de função. Não sendo subsistema, trata-se da descrição do sistema em alto nível de abstração, sendo realizada a conversão para a função principal do código C, ou seja, a "*main*".

Verifica-se a cada bloco convertido, se todos os componentes presentes no subsistema estão presentes na biblioteca *ms2psoc_lib*. Se algum componente utilizado não estiver na biblioteca, ocorrerá um erro e o processo de conversão será finalizado.

Após gerar o código da função "*main*", inicia-se a geração da estrutura de arquivos necessária para o projeto traduzido ser aberto no ambiente PSoC Creator. Com a estrutura gerada, os arquivos temporários criados são copiados para as respectivas pastas e depois excluídos. Na Figura 14 apresenta-se os arquivos e pastas geradas pela ferramenta MS^{2}PSoC.

Figura 14 – Arquivos e pastas gerados pela ferramenta MS^{2}PSoC.



Fonte: Elaborada pelo autor.

A pasta que contém a tradução recebe o nome do arquivo Simulink traduzido e possui a extensão *cydsn* que na Figura 14 recebe o nome *hvac*. Esta pasta contém os seguintes arquivos:

- *hvac.cyprj*: arquivo de projeto executável do ambiente PSoC Creator. Este arquivo contém toda a configuração do ambiente e o endereço para os outros arquivos necessários;
- *hvac.cydwr*: arquivo onde é realizada a configuração dos portos do hardware PSoC;
- *main.c*: arquivo utilizado com o sistema analógico para configuração e inicialização de componentes do hardware PSoC;
- *device.h*: arquivo cabeçalho;
- *TopDesign*: esta pasta contém um arquivo com o mesmo nome e extensão *cysch*. Este arquivo contém o esquemático do ambiente PSoC Creator;
- *component01*: esta pasta contém o arquivo Verilog gerado pela ferramenta e outro arquivo com o mesmo nome da pasta e extensão *cysym*. Este arquivo contém o esquemático do componente criado.

A ferramenta MS^{2}PSoC realiza a geração de um componente digital automaticamente no ambiente PSoC Creator. Se houver mais de um subsistema digital descrito no modelo Simulink, é necessário realizar a geração dos demais componentes digitais no PSoC Creator manualmente. Vale ressaltar que isto ocorre devido ao arquivo que gera o componente ser codificado.

Após gerar toda a estrutura de arquivos a ferramenta encerra o seu funcionamento com uma mensagem de sucesso na tradução do modelo.

Para implementar o modelo em hardware é necessário abrir o projeto no ambiente PSoC Creator para configuração e compilação. Esta configuração se refere à nomeação dos portos dos componentes gerados.

Existem algumas informações que devem ser seguidas para o correto funcionamento da ferramenta:

- O nome do componente e do módulo Verilog no ambiente PSoC Creator não devem ser alterados;
- A pasta com o projeto para o ambiente PSoC Creator é criada no mesmo local do arquivo Simulink que foi utilizado na ferramenta. Esta pasta não deve ter seu local alterado, pois os componentes instanciados são acompanhados do endereço da pasta de onde o projeto foi criado;
- Os componentes utilizados para descrição do modelo no Simulink devem estar todos presentes na biblioteca *ms2psoc_lib*;
- Os sistemas digitais devem estar em subsistemas e os nomes iniciados com "D_";
- Os sistemas do Stateflow devem ser iniciados com "S_" ou "F_" em caso de função inserida na máquina de estados finita;
- Os arquivos Simulink a serem traduzidos não devem estar dentro de pastas que possuem caracteres não aceitos pelo idioma inglês, como por exemplo "ç" e "∼".

3.2 INTERFACE

Para tornar o uso da ferramenta MS²PSoC intuitivo foi criada uma interface gráfica utilizando-se do framework Qt. O framework Qt é uma aplicação multi-plataforma para o desenvolvimento de interfaces gráficas. Utilizou-se a linguagem C++ para o desenvolvimento desta interface (BLANCHETTE; SUMMERFIELD, 2008).

A interface principal da ferramenta consta de quatro botões e uma linha de texto não editável utilizada para explicitar o arquivo que foi selecionado. A interface principal é apresentada na Figura 15.

A funcionalidade dos botões são descritas a seguir:

- **Open File**: abre uma caixa de diálogo para a escolha do arquivo Simulink a ser traduzido;

Figura 15 – Interface principal da ferramenta MS^{2}PSoC.

Fonte: Elaborada pelo autor.

- **Run**: inicia a tradução do arquivo selecionado;
- **Close**: encerra o uso da ferramenta;
- **About**: informações sobre o desenvolvimento da ferramenta.

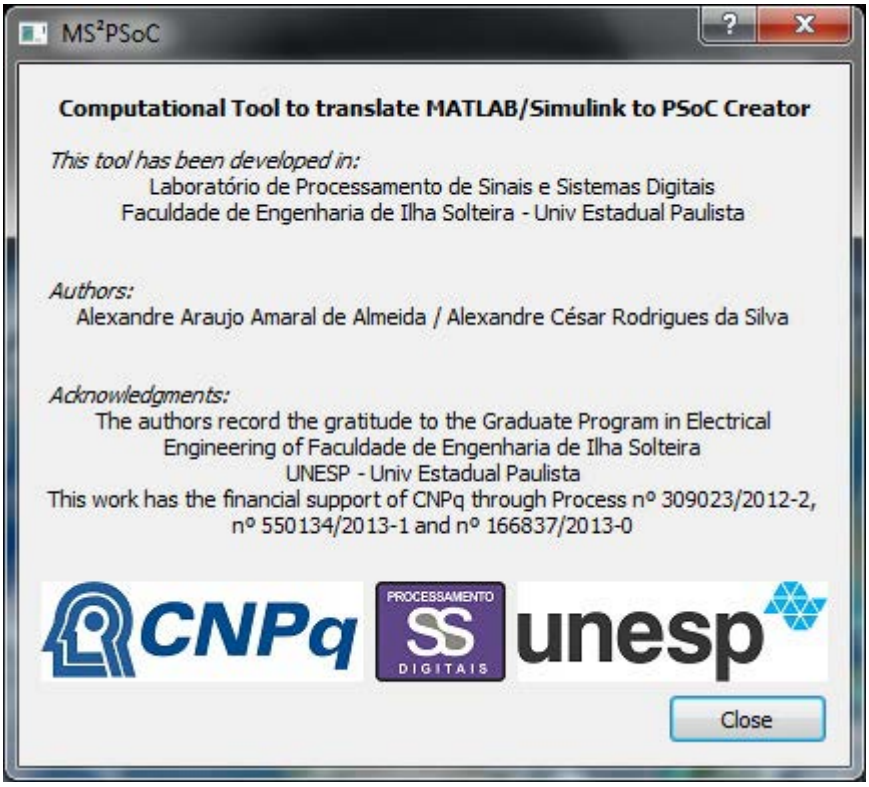
Após selecionar o arquivo Simulink a ser traduzido utilizando o botão *Open File*, a informação sobre o endereço do arquivo é explicitada na barra de *status* da ferramenta. Se a tradução for realizada com sucesso uma mensagem é exibida confirmando a criação do projeto para o ambiente PSoC Creator.

Antes de iniciar a tradução do modelo escolhido a ferramenta verifica a existência de um diretório com o mesmo nome no local onde será realizada a tradução. Em caso afirmativo, a ferramenta exibe uma mensagem informando a existência deste diretório.

Se algum componente utilizado na descrição do modelo não estiver presente na biblioteca *ms2psoc_lib*, uma mensagem é exibida informando a falta de um componente.

O botão *About*, presente na interface principal contém algumas informações com relação ao desenvolvimento da ferramenta MS^{2}PSoC e ao fomento disponibilizado pelo CNPq. Na Figura 16 apresenta-se a interface que é exibida ao se clicar neste botão.

Figura 16 – Informações sobre a ferramenta MS^{2}PSoC.



Fonte: Elaborada pelo autor.

Apresenta-se no próximo capítulo alguns modelos que foram descritos e simulados no Simulink para posterior implementação em hardware empregando a ferramenta MS^{2}PSoC.

4 ESTUDO DE CASOS

Neste capítulo serão apresentados os estudos de casos utilizados para avaliar a ferramenta MS^{2}PSoC. Na Seção 4.1 apresenta-se o código de linha MLT-3 utilizado para avaliar a tradução de um sistema digital. Na Seção 4.2 um conversor digital-analógico denominado Ladder R-2R foi desenvolvido para avaliar a tradução de um sistema analógico. Na Seção 4.3 apresenta-se o funcionamento de um sistema de controle de temperatura veicular. Este sistema foi utilizado para avaliar a tradução de sinais mistos realizada pela ferramenta MS^{2}PSoC.

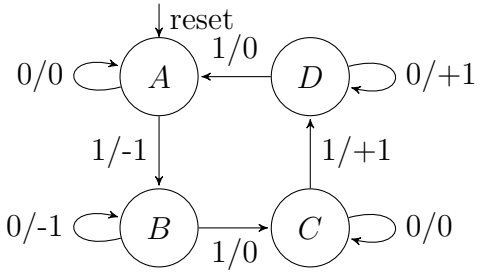
4.1 CÓDIGO DE LINHA MLT-3

A codificação MLT-3 (*Multi Level Transmission 3*) é um código de linha utilizado para melhorar a eficiência da rede de transmissão FDDI (*Fiber Distributed Data Interface*) usando cabo par trançado padrão de 10 Mb/s (EVERITT et al., 1998).

O código de linha MLT-3 possui uma codificação com três níveis de tensão (-1, 0, +1) em que uma entrada de dados com nível lógico alto realiza a transição entre os três níveis. Uma entrada com sinal lógico baixo faz com que a saída permaneça no nível de tensão atual.

O diagrama de transição de estados do código de linha MLT-3 é apresentado no diagrama de transição de estados da Figura 17.

Figura 17 – Diagrama de transição de estados do código de linha MLT-3.



Fonte: Elaborada pelo autor.

A transição entre os níveis de tensão ocorre, obrigatoriamente, passando por zero, ou seja, não é possível uma transição direta de nível -1 para o +1 ou vice-versa.

Para desenvolver o circuito do código de linha MLT-3 optou-se por utilizar como elemento de memória o flip-flop JK. A tabela de excitação do flip-flop JK é apresentada na Tabela 3.

Tabela 3 – Tabela de excitação do flip-flop JK.

Q_{atual}	$Q_{próx}$	JK
0	0	0X
0	1	1X
1	0	X1
1	1	X0

Fonte: elaborada pelo autor.

Na alocação de estados utiliza-se o estado A como “00”, o B como “11”, o C como “10” e o D como “01”. Na alocação da saída utilizou-se o nível 0 como “00”, o nível +1 como “01” e o nível -1 como “11”. Com a alocação realizada, seguindo o diagrama de transição de estados e a tabela de excitação do flip-flop JK, montou-se a Tabela 4.

Tabela 4 – Tabela de transições de estados da codificação MLT-3.

	Reset	Data	Q_1 Q_0	$Q_{próx}$	J_1 K_1	J_0 K_0	S_1 S_0
0	0	0	0 0	0 0	0 X	0 X	0 0
1	0	0	0 1	0 0	0 X	X 1	0 0
2	0	0	1 0	0 0	X 1	0 X	0 0
3	0	0	1 1	0 0	X 1	X 1	0 0
4	0	1	0 0	0 0	0 X	0 X	0 0
5	0	1	0 1	0 0	0 X	X 1	0 0
6	0	1	1 0	0 0	X 1	0 X	0 0
7	0	1	1 1	0 0	X 1	X 1	0 0
8	1	0	0 0	0 0	0 X	0 X	0 0
9	1	0	0 1	0 1	0 X	X 0	0 1
10	1	0	1 0	1 0	X 0	0 X	0 0
11	1	0	1 1	1 1	X 0	X 0	1 1
12	1	1	0 0	1 1	1 X	1 X	1 1
13	1	1	0 1	0 0	0 X	X 1	0 0
14	1	1	1 0	0 1	X 1	1 X	0 1
15	1	1	1 1	1 0	X 0	X 1	0 0

Fonte: elaborada pelo autor.

As funções de próximo estado de cada flip-flop e de saída são obtidas a partir da Tabela 4:

- $F_{J_1} = \sum m(12) + d(2, 3, 6, 7, 10, 11, 14, 15)$

- $F_{K_1} = \sum m(2, 3, 6, 7, 14) + d(0, 1, 4, 5, 8, 9, 12, 13)$
- $F_{J_0} = \sum m(12, 14) + d(1, 3, 5, 7, 9, 11, 13, 15)$
- $F_{K_0} = \sum m(1, 3, 5, 7, 13, 15) + d(0, 2, 4, 6, 8, 10, 12, 14)$
- $F_{S_1} = \sum m(11, 12)$
- $F_{S_0} = \sum m(9, 11, 12, 14)$

Com o auxílio do Mapa de Karnaugh realizou-se a minimização das funções booleanas, encontrando-se as funções mínimas:

- F_{J_1} = Reset * Data * $\overline{Q_0}$
- $F_{K_1} = \overline{Reset}$ + (Data * $\overline{Q_0}$)
- F_{J_0} = Reset * Data
- $F_{K_0} = \overline{Reset}$ + Data
- F_{S_1} = (Reset * Data * $\overline{Q_1}$ * $\overline{Q_0}$) + (Reset * $\overline{Data}$ * Q_1 * Q_0)
- F_{S_0} = (Reset * Data * $\overline{Q_0}$) + (Reset * $\overline{Data}$ * Q_0)

O circuito foi descrito no Simulink a partir das funções booleanas minimizadas e é apresentado na Figura 18.

O sistema foi compilado no Simulink e gerou-se o símbolo D_mlt3 para que a ferramenta MS^{2}PSoC o identifique ao realizar a tradução. Vale ressaltar que o nome do subsistema deve iniciar com "D_". Na Figura 19 apresenta-se o subsistema gerado.

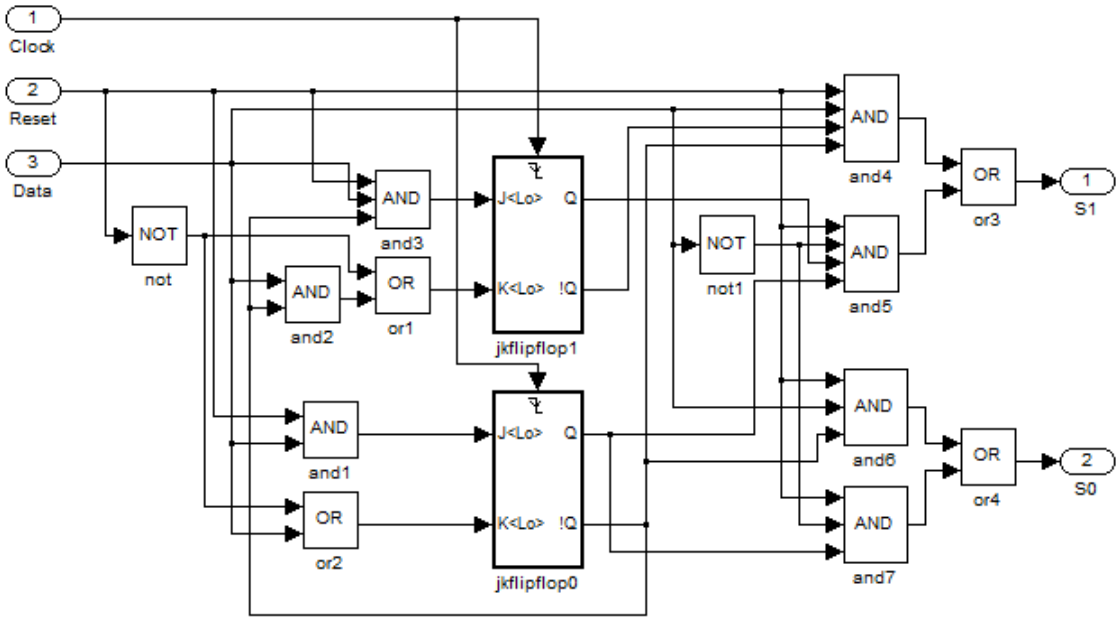
Para realizar a simulação do circuito substitui-se as entradas Clock, Reset e Data pelo componente *Pulse Generator*, disponível na biblioteca Simulink → Sources e as saídas S1 e S0 pelo componente *Scope*, disponível na biblioteca Simulink → Sinks.

Os parâmetros utilizados na simulação foram:

Clock	Reset	Data
período = 1 seg	período = 40 seg	período = 20 seg
amplitude = 1	amplitude = 1	amplitude = 1
largura do pulso = 50%	largura do pulso = 50%	largura do pulso = 50%

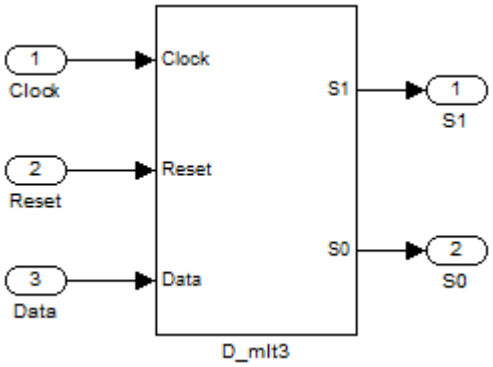
Na Figura 20 apresentam-se os sinais obtidos da simulação do código de linha MLT-3 visualizado pelo componente *Scope*.

Figura 18 – Circuito do código de linha MLT-3 descrito no Simulink.



Fonte: elaborada pelo autor.

Figura 19 – Subsistema do código de linha MLT-3 no Simulink.



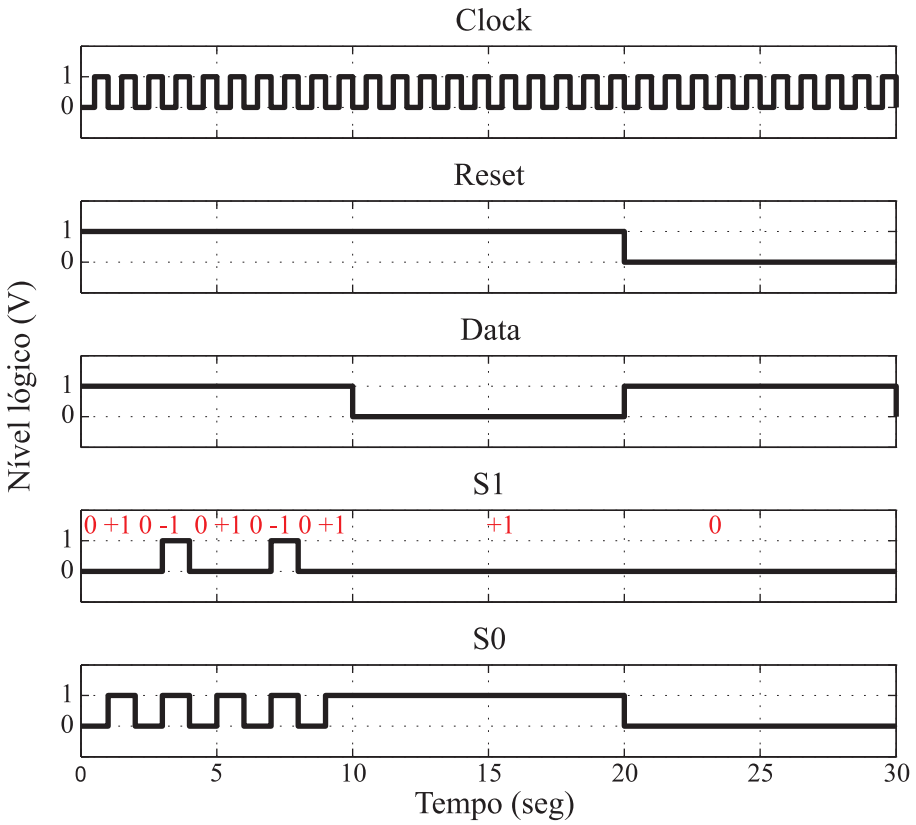
Fonte: elaborada pelo autor.

Foram realizadas três situações na simulação do código de linha MLT-3. No intervalo de tempo de 0 a 10 segundos manteve-se os sinais de entrada Data e Reset em nível lógico alto. Deste modo, ocorrem mudanças de estados nos sinais de saída S1 e S0 conforme apresentado no diagrama de transições de estados na Figura 17.

Nota-se que as mudanças de estados das saídas S1 e S0 repetem-se na seguinte sequência: 0, +1, 0, -1. Devido a alocação, os sinais de saída S1 e S0 são, respectivamente, “00”, “01”, “00”, “11”.

No intervalo de tempo de 10 a 20 segundos manteve-se os sinais de entrada Data e Reset em nível lógico baixo e alto, respectivamente. Deste modo, os sinais de saída S1 e S0 permanecem com seus estados atuais inalterados até que ocorra uma mudança de

Figura 20 – Simulação do código de linha MLT-3 no Simulink.



Fonte: elaborada pelo autor.

estado do sinal de entrada Data para nível lógico alto.

No intervalo de tempo de 20 a 30 segundos manteve-se o sinal Reset em nível lógico baixo. Neste caso, os sinais de saída S1 e S0 se mantém em nível lógico baixo, ou seja, no nível 0 até que ocorra uma mudança de estado no sinal de entrada Reset para nível lógico alto.

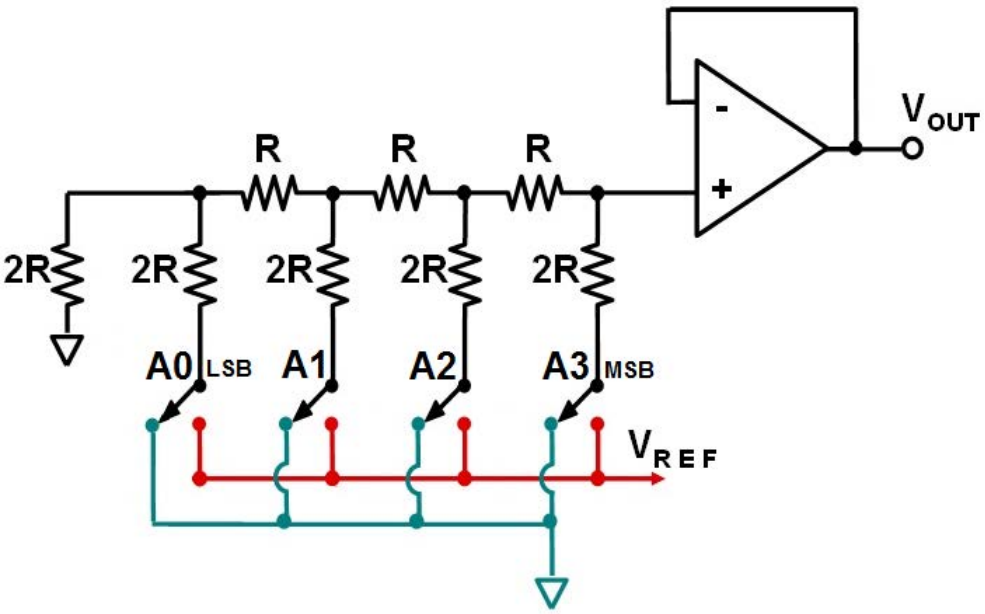
Pela simulação apresentada pode-se comprovar que o modelo do código de linha MLT-3 descrito no Simulink foi desenvolvido corretamente. Com este estudo de caso pretendeu-se avaliar um sistema contendo somente componentes digitais.

4.2 CONVERSOR D/A EMPREGANDO UM MODELO LADDER R-2R DE 4 BITS

O conversor D/A Ladder R-2R consiste de um circuito elétrico composto por resistores em forma de escada que realiza a conversão de um sinal digital para o valor analógico correspondente. A principal característica deste modelo de conversor é a utilização de apenas dois valores de resistência. Na Figura 21 apresenta-se o circuito do conversor D/A

Ladder R-2R de 4 bits.

Figura 21 – Modelo do conversor D/A Ladder R-2R de 4 bits.



Fonte: Elaborada pelo autor.

A precisão deste tipo de conversor está diretamente relacionada com a quantidade de bits de entrada, ou seja, quanto maior o número de bits de entrada, maior a precisão. Cada bit de entrada tem um peso crescente variando, neste caso, do bit A3 - MSB (*Most Significant Bit*) ao bit A0 - LSB (*Least Significant Bit*). O peso de cada bit de entrada segue a ordem apresentada na Tabela 5.

Tabela 5 – Peso de cada bit de entrada para o conversor D/A Ladder R-2R.

Bit	Peso
A3	$2^1 \rightarrow V_{ref}/2$
A2	$2^2 \rightarrow V_{ref}/4$
A1	$2^3 \rightarrow V_{ref}/8$
A0	$2^4 \rightarrow V_{ref}/16$

Fonte: elaborada pelo autor.

Com o valor de cada bit monta-se a tabela com o valor da saída analógica para cada combinação de entrada digital. Na Tabela 6 apresenta-se o valor analógico da saída para cada entrada possível de um conversor Ladder R-2R de 4 bits.

A conversão é realizada com a soma das quedas de tensão nos resistores e a corrente elétrica resultante desta conversão tende a ter um valor baixo. Por isso, utiliza-se um circuito seguidor de tensão implementado com amplificador operacional para elevar a corrente elétrica resultante.

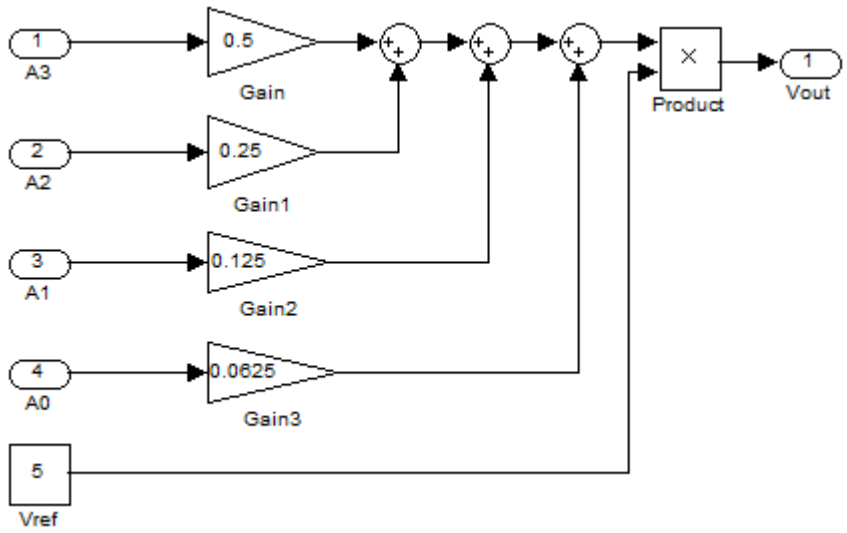
Tabela 6 – Tabela do conversor D/A empregando o modelo Ladder R-2R.

Decimal	A3	A2	A1	A0	*Vout* (Volts)
0	0	0	0	0	0
1	0	0	0	1	V_{ref}*0,0625
2	0	0	1	0	V_{ref}*0,1250
3	0	0	1	1	V_{ref}*0,1875
4	0	1	0	0	V_{ref}*0,2500
5	0	1	0	1	V_{ref}*0,3125
6	0	1	1	0	V_{ref}*0,3750
7	0	1	1	1	V_{ref}*0,4375
8	1	0	0	0	V_{ref}*0,5000
9	1	0	0	1	V_{ref}*0,5625
10	1	0	1	0	V_{ref}*0,6250
11	1	0	1	1	V_{ref}*0,6875
12	1	1	0	0	V_{ref}*0,7500
13	1	1	0	1	V_{ref}*0,8125
14	1	1	1	0	V_{ref}*0,8750
15	1	1	1	1	V_{ref}*0,9375

Fonte: elaborada pelo autor.

Para avaliar o funcionamento do circuito apresentado na Figura 21, desenvolveu-se o modelo do conversor D/A Ladder R-2R no Simulink para posterior implementação em hardware PSoC. O modelo do conversor desenvolvido no ambiente Simulink é apresentado na Figura 22.

Figura 22 – Modelo do conversor D/A Ladder R-2R de 4 bits no Simulink.



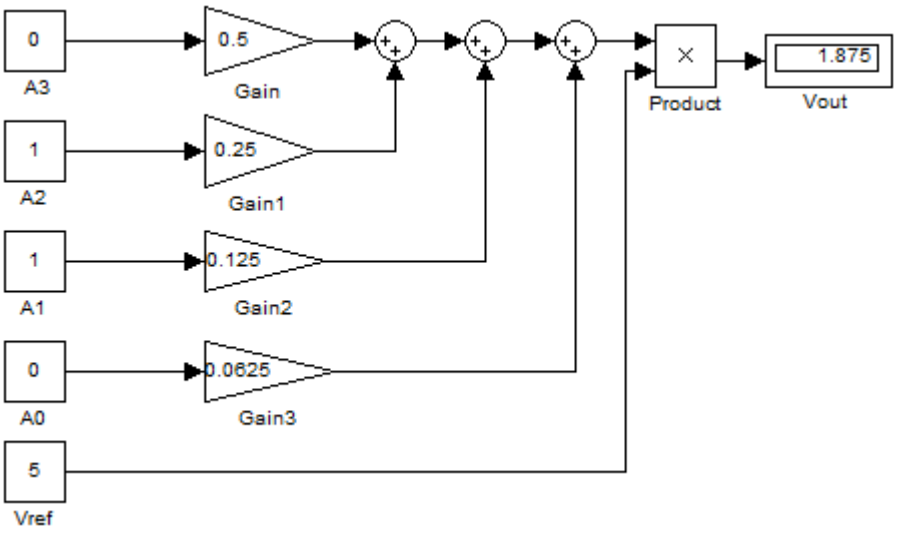
Fonte: Elaborada pelo autor.

Nota-se que foi utilizada uma tensão de referência de 5 Volts e componentes de ganhos que representam o peso de cada bit de entrada conforme apresentado na Tabela 5. A

tensão de referência é multiplicada pela soma dos pesos para se obter a tensão de saída do conversor D/A.

Para simular o conversor D/A Ladder R-2R substitui-se os bits de entradas por constantes e a saída pelo componente *Display* disponível na biblioteca do Simulink → *Sinks*. Como valor de entrada aplicou-se os bits "0110" nas entradas A3, A2, A1 e A0, respectivamente. Obteve-se na saída do conversor D/A o valor de tensão de 1,875 Volts, como apresentado na Figura 23.

Figura 23 – Simulação do conversor D/A Ladder R-2R de 4 bits no Simulink.



Fonte: Elaborada pelo autor.

Comparando o valor obtido na simulação com o apresentado na Tabela 6, constata-se que foi obtido o mesmo valor de saída, ou seja:

$$V_{ref}*0{,}3750 = 5*0{,}3750 = 1{,}875$$

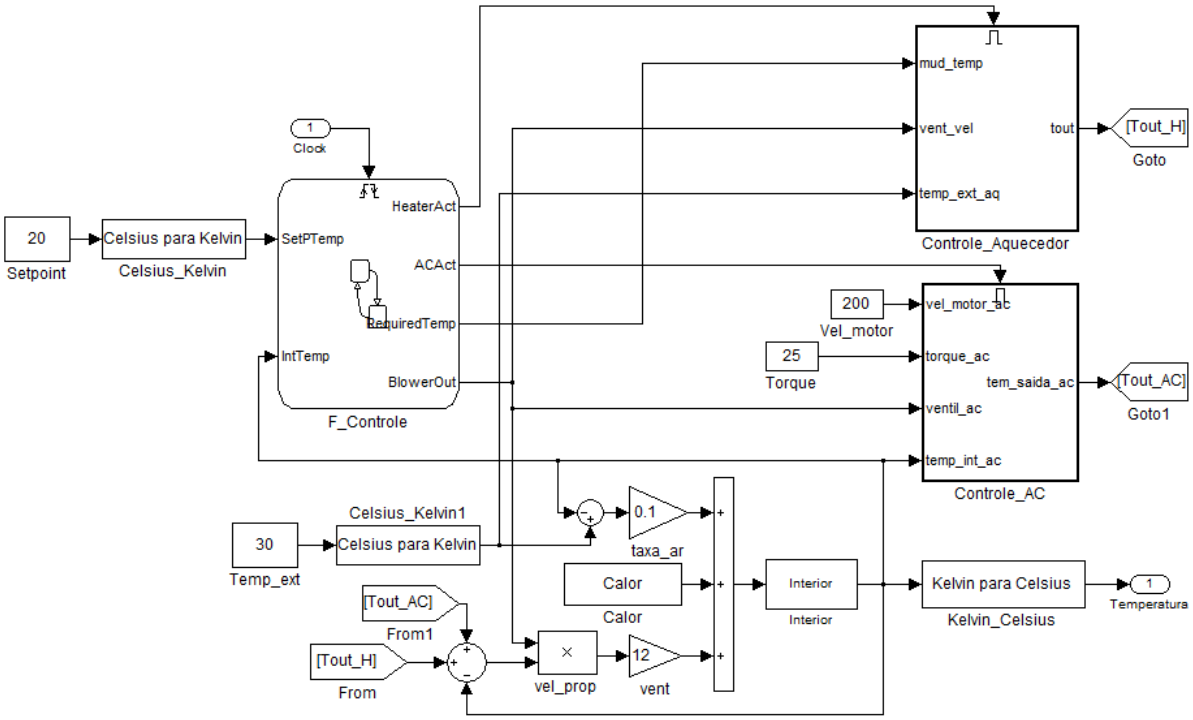
A partir destes resultados, conclui-se que o conversor D/A empregando o modelo Ladder R-2R desenvolvido e simulado no ambiente Simulink funciona conforme a teoria apresentada.

A implementação do conversor no dispositivo PSoC foi realizada utilizando-se da linguagem C para ser executado no microcontrolador 8051. Para efeito de comparação montou-se o circuito apresentado na Figura 21 em matriz de contato conforme o esquemático. Avaliou-se o tempo de resposta das duas implementações, cujos resultados estão apresentados no Capítulo 5.

4.3 SISTEMA DE CONTROLE DE TEMPERATURA VEICULAR

Um sistema de controle de temperatura veicular, HVAC (*Heating, Ventilating, and Air Conditioning*), foi utilizado para avaliar o funcionamento da ferramenta MS^{2}PSoC na síntese de sistema de sinal misto. Este sistema foi adaptado de um exemplo presente no ambiente Simulink, disponível no MathWorks (2013). Na Figura 24 apresenta-se o sistema de controle de temperatura veicular.

Figura 24 – Sistema de controle de temperatura veicular no Simulink.



Fonte: MathWorks (2013).

O sistema de controle de temperatura veicular é composto por um controle geral que supervisiona a temperatura interna do veículo e controla o acionamento do ar condicionado ou do aquecedor conforme a variação da temperatura interna do veículo.

A cada pulso do "Clock", o subsistema "F_Controle" calcula a diferença entre a temperatura interna desejada e a atual. Com essa diferença de temperatura controla-se a velocidade de rotação do ventilador e o acionamento dos subsistemas "Controle_AC" ou "Controle_Aquecedor".

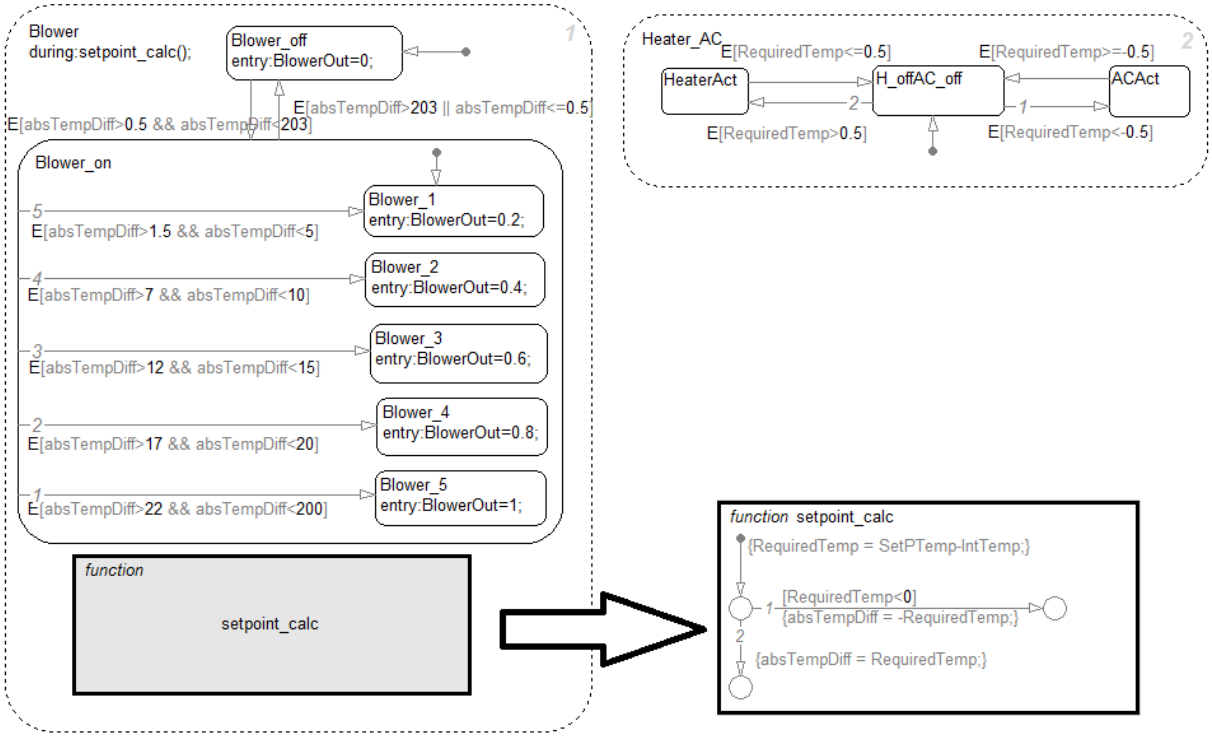
Conforme o valor da temperatura interna vai se alterando, aciona-se o ar condicionado ou o aquecedor para que a temperatura se mantenha no valor desejado. Repete-se este ciclo enquanto o sistema estiver acionado, mantendo a temperatura veicular estável.

Apresenta-se, a seguir, as características de cada subsistema e suas respectivas funções

no sistema de controle de temperatura veicular.

No subsistema "F_Controle" encontra-se o circuito de controle do sistema. Na Figura 25 apresenta-se o sistema presente neste subsistema.

Figura 25 – Subsistema F_Controle para o controle do sistema de temperatura veicular.



Fonte: MathWorks (2013).

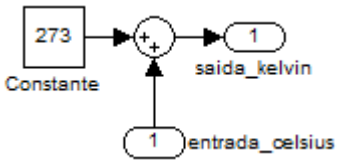
De acordo com a Figura 25 o subsistema "F_Controle" é composto por duas máquinas de estados finitos (FSM) denominadas Blower e Heater_AC. Na FSM Blower define-se a velocidade com que o ventilador girará. Quanto maior a diferença entre a temperatura interna desejada e a atual, mais rápido o ventilador girará.

A FSM denominada Heater_AC controla o acionamento do ar condicionado ou do aquecedor. Para ocorrer o acionamento do aquecedor é necessário que a diferença entre a temperatura interna desejada e a atual seja maior que 0,5°C (Celsius). De maneira análoga, para o acionamento do ar condicionado é necessário que a diferença entre as temperaturas seja menor que 0,5°C. O valor 0,5 na diferença das temperaturas desejada e interna foi estabelecido para evitar o contínuo acionamento do ar condicionado ou do aquecedor.

Os valores de temperatura inseridos pelo usuários estão na escala de grau Celsius, porém no equacionamento do sistema utiliza-se o valor da temperatura em escala Kelvin. Portanto, realiza-se a conversão da temperatura de grau Celsius para grau Kelvin. Na

Figura 26 apresenta-se o circuito inserido nos subsistemas de conversão denominados "Celsius_Kelvin" e "Celsius_Kelvin1". Estes subsistemas podem ser visualizados na Figura 24.

Figura 26 – Subsistema Celsius_Kelvin para conversão de temperatura de Celsius para Kelvin.



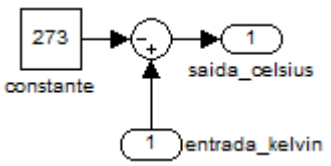
Fonte: MathWorks (2013).

A conversão de grau Celsius para grau Kelvin é realizada somando-se o valor 273 ao valor de temperatura em grau Celsius. Nota-se, na Figura 26 que a variável *saida_kelvin* é a soma da variável *entrada_celsius* e o valor constante 273.

Para fins de simulação, é necessário que o usuário especifique a temperatura externa ao veículo na constante "Temp_ext" e a temperatura interna desejada na constante "Setpoint".

Como a temperatura interna desejada é inserida em grau Celsius, deseja-se que a informação da temperatura interna atual apresentada ao usuário também esteja em grau Celsius. Logo, converte-se a temperatura interna atual de Kelvin para grau Celsius. Esta conversão é realizada no subsistema "Kelvin_Celsius" e é apresentada na Figura 27. O subsistema pode ser visualizado na Figura 24.

Figura 27 – Subsistema Kelvin_Celsius para conversão de temperatura de Kelvin para Celsius.



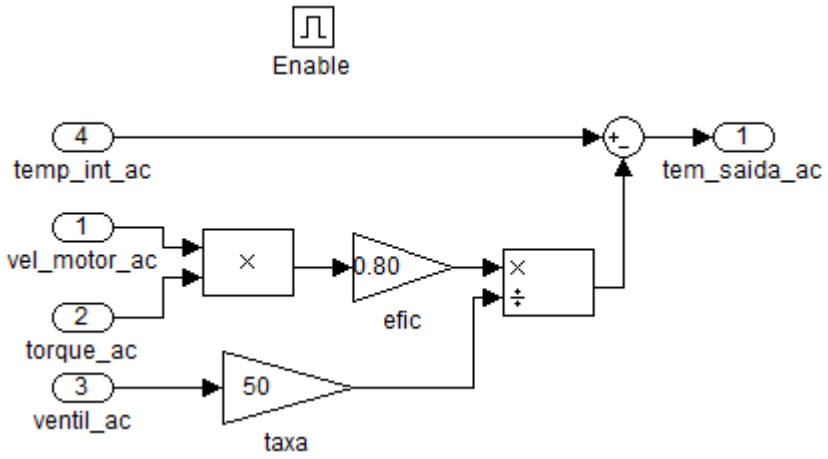
Fonte: MathWorks (2013).

Para realizar a conversão de grau Kelvin para grau Celsius subtrai-se o valor 273 do valor de grau Kelvin. Nota-se que a variável *saida_celsius* é o resultado da subtração entre a variável *entrada_kelvin* e a constante no valor 273.

Caso a diferença entre a temperatura desejada e a atual for menor que 0,5, o controle principal do sistema de temperatura acionará o ar condicionado para resfriar o interior do veículo.

O subsistema “Controle_AC” é o responsável por este resfriamento e o modelo deste subsistema é apresentado na Figura 28.

Figura 28 – Subsistema Controle_AC para o controle do ar condicionado.



Fonte: MathWorks (2013).

Os parâmetros deste subsistema são a velocidade de rotação e o torque do motor do ar condicionado. As outras variáveis são a velocidade do ventilador e a temperatura interna atual do interior do veículo. Para desenvolver o subsistema “Controle_AC” utilizou-se a Equação 1 (MATHWORKS, 2013).

$$y * (w * Tcomp) = m\_dot * (h4 - h1) \tag{1}$$

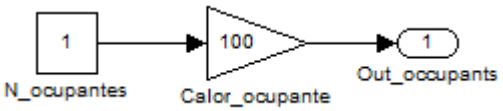
onde:

- y: eficiência;
- m_dot: taxa de fluxo de massa;
- w: velocidade do motor;
- Tcomp: torque do compressor;
- h4, h1: entalpia.

A variável de saída do subsistema “Controle_AC” é denominada “tem_saida_ac” e corresponde a uma atualização da temperatura interna do veículo, sendo seu valor sempre menor que o valor de entrada “temp_int_ac”. Esta variação negativa do valor da temperatura ocorre devido ao funcionamento do ar condicionado e à velocidade de giro do ventilador.

O número de ocupantes no veículo influencia na atuação do sistema de controle de temperatura veicular. Quanto maior o número de ocupantes, maior o calor interno do veículo. Portanto, no subsistema "Calor", apresentado na Figura 24, informa-se o número de ocupantes do veículo para se calcular o calor gerado por pessoa. Na Figura 29 apresenta-se o circuito presente no subsistema "Calor".

Figura 29 – Subsistema Calor para o cálculo do calor gerado por pessoa.



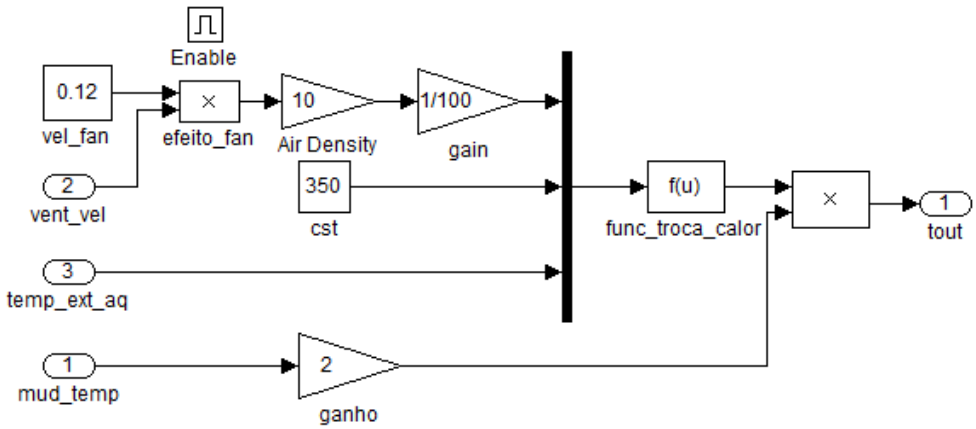
Fonte: MathWorks (2013).

Nota-se que o número de ocupantes é multiplicado por uma constante de valor 100 para calcular o calor gerado por pessoa no interior do veículo. Esta constante é utilizada como uma taxa de calor gerado por cada ocupante presente no interior do veículo (MATHWORKS, 2013).

Para alterar o número de ocupantes do veículo, altera-se o componente *N_ocupantes*. Por padrão, o número de ocupantes é um.

No momento em que a diferença de temperatura desejada e a atual no interior do veículo estiver maior que 0,5 °C, o sistema de aquecimento é acionado. Na Figura 30 apresenta-se o circuito contido no subsistema "Controle_Aquecedor" referente ao aquecedor.

Figura 30 – Subsistema Controle_Aquecedor para o controle do aquecedor.



Fonte: MathWorks (2013).

Para desenvolver este subsistema utilizou-se a Equação 2 (MATHWORKS, 2013).

$$Tout = Ts - (Ts - Tin)e^{[(-pi*D*L*hc)/(m\_dot*Cp)]} \quad (2)$$

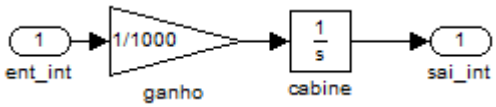
onde:

- Ts: constante de temperatura do radiador;
- D: 0,004 m;
- L: 0,05 m;
- N: 100;
- k: 0,026 W/mK;
- Cp: 1007 J/kgK;
- hc: 3,66 k/D = 23,8 W/m2K.

A variável de saída do subsistema "Controle_Aquecedor" denominada *tout* representa o valor da temperatura de saída do aquecedor e tende a ser maior que a temperatura de entrada. Isto ocorre devido a multiplicação da função de troca de calor com a mudança de temperatura desejada.

No subsistema "Interior", apresentado na Figura 24, calcula-se a dinâmica da temperatura no interior do veículo devido ao calor dos ocupantes e a outros fatores como, por exemplo, a velocidade dos ventiladores. Na Figura 31 apresenta-se o sistema que compõe o subsistema "Interior".

Figura 31 – Subsistema Interior para o cálculo da temperatura veicular.



Fonte: MathWorks (2013).

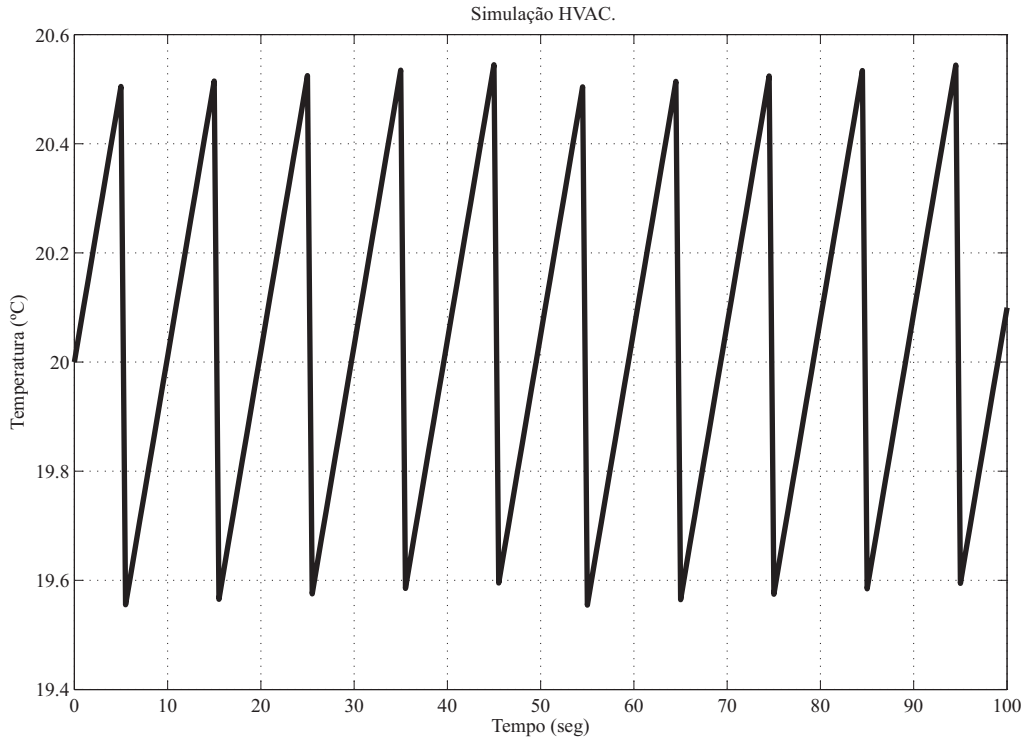
Nesta representação o componente *cabine* é um integrador presente na biblioteca do Simulink. Nota-se que a variável de entrada *ent_int* é multiplicada pelo valor da variável ganho e posteriormente integrada, sendo o resultado obtido na variável *sai_int*.

Analisando o subsistema "Interior", verifica-se que a variável de saída *sai_int* é uma rampa crescente, caso esteja integrando uma constante positiva ou uma rampa negativa, caso esteja integrando uma constante negativa.

Pelo esquema geral do sistema de controle de temperatura veicular verifica-se que a variável de saída *sai_int* assume o valor atual da temperatura interna do veículo, sendo este valor utilizado para realimentar o sistema de controle.

Para simular o controle de temperatura veicular utilizou-se como temperatura desejada o valor de 20 °C, temperatura externa o valor de 30 °C e o Clock com período de 1 segundo. Com o componente *Scope* (possui a mesma funcionalidade que um osciloscópio) obteve-se a forma de onda da temperatura interna do veículo. Na Figura 32 apresenta-se a forma de onda obtida.

Figura 32 – Simulação do sistema de controle da temperatura veicular.



Fonte: Elaborada pelo autor.

Nota-se que a temperatura interna do veículo permanece próxima dos 20 °C com largura de banda de $0,5$ °C para evitar o constante acionamento dos sistemas de ar condicionado ou aquecedor.

Os sistemas apresentados neste capítulo foram implementados no dispositivo PSoC e os resultados obtidos pelos sistemas reais foram comparados com os da simulações. Apresenta-se no próximo capítulo os resultados obtidos e as comparações realizadas.

5 RESULTADOS E DISCUSSÕES

Neste capítulo apresentam-se os resultados obtidos das implementações realizadas em hardware PSoC utilizando-se a ferramenta MS^{2}PSoC para traduzir o modelo descrito no Simulink para a descrição sintetizável no PSoC 3. Na Seção 5.1 apresenta-se o resultado da implementação do código de linha MLT-3. Na Seção 5.2 apresenta-se a implementação do conversor D/A Ladder R-2R e a comparação realizada com hardware. Na Seção 5.3 apresenta-se a implementação do sistema de controle de temperatura veicular.

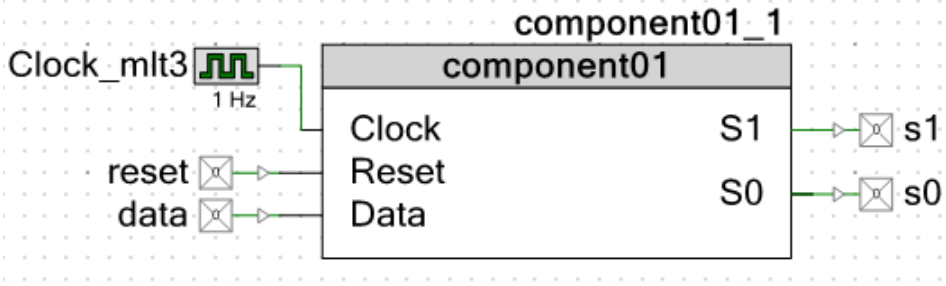
Para a implementação foi utilizado o kit de desenvolvimento PSoC 3 CY8CKIT-030 e para aquisição dos sinais o osciloscópio digital Politerm - modelo POL-15D.

5.1 IMPLEMENTAÇÃO DO CÓDIGO DE LINHA MLT-3

Realizou-se a tradução do modelo do código de linha MLT-3 descrito no Simulink empregando a ferramenta MS^{2}PSoC com o intuito de implementá-lo no sistema embarcado PSoC. Apresenta-se o código Verilog HDL gerado na tradução para este modelo no Apêndice A.

O modelo do código de linha MLT-3 traduzido foi aberto no ambiente PSoC Creator afim de se realizar a configuração dos portos de entrada e saída. Na Figura 33 apresenta-se o componente configurado no PSoC Creator.

Figura 33 – Componente gerado no PSoC Creator para o código de linha MLT-3.



Fonte: Elaborada pelo autor.

Para o *clock* do sistema foi utilizado um componente denominado *Clock*, disponível na biblioteca *System* do ambiente PSoC Creator. Este componente foi conectado ao sinal de entrada Clock do componente gerado e utilizado uma frequência no valor de 1 Hz (Hertz).

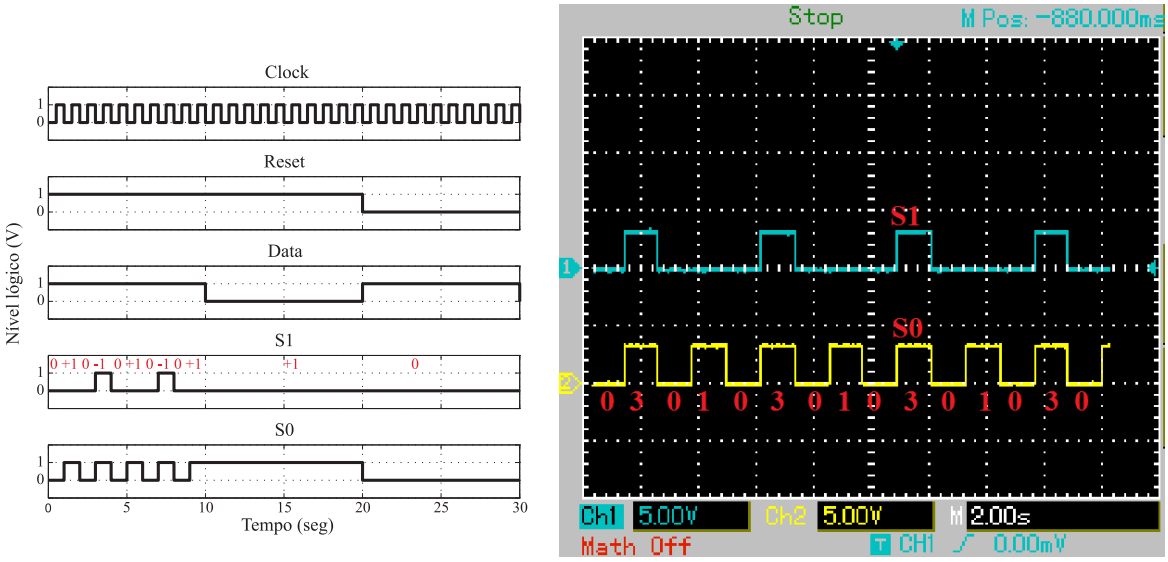
Para os sinais de entrada Reset e Data utilizou-se o componente denominado *Digital Input Pin* disponível na biblioteca *Ports and Pins* do ambiente PSoC Creator. Na mesma biblioteca, utilizou-se o componente denominado *Digital Output Pin* para os sinais de saída S1 e S0. Ressalta-se que a configuração dos portos de entrada e saída é realizada manualmente, dentro do ambiente PSoC Creator.

Os sinais de entrada Reset e Data foram conectados a botões de pressão no sistema embarcado PSoC e para a aquisição dos sinais de saída, S1 e S0 foram conectados aos canais 1 e 2 do osciloscópio, respectivamente.

Com o objetivo de comparar os sinais de saída S1 e S0 do modelo implementado no PSoC com os sinais obtidos na simulação apresentada na Figura 20, reproduziu-se no hardware PSoC as situações apresentadas na simulação.

Na primeira situação (intervalo de tempo de 0 a 10 segundos da simulação) os sinais de entrada Data e Reset são mantidos em nível lógico alto. Na Figura 35(a) apresenta-se a simulação do código de linha MLT-3 no Simulink e os sinais de saída S1 e S0 obtidos no sistema embarcado PSoC implementando a primeira situação na Figura 35(b).

Figura 34 – Sinais de saída S1 e S0 do código de linha MLT-3 implementado no PSoC para os sinais de entrada Data e Reset em nível lógico alto.



(a) Simulação (b) Implementação

Fonte: Elaborada pelo autor.

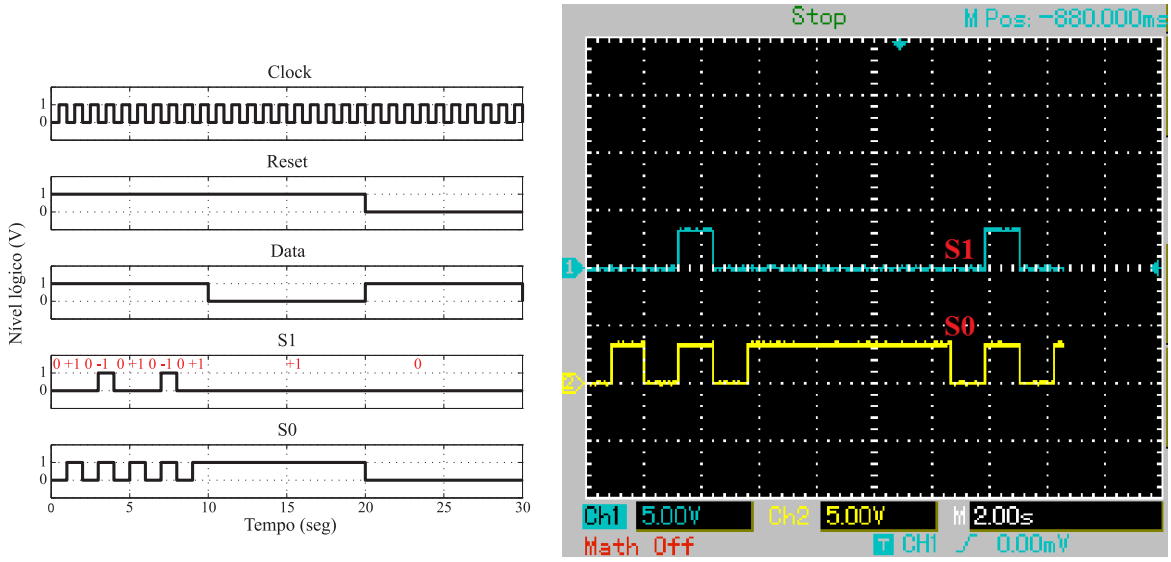
Nota-se na implementação da primeira situação que os sinais de saída S1 e S0 se mantém mudando de estado, visto que os sinais de entrada Data e Reset estão em nível lógico alto.

De acordo com a alocação utilizada no desenvolvimento do modelo apresentada na Seção 4.1, a saída do sistema muda na seguinte sequência: A → B → C → D, repetindo-se enquanto o sinal Data se manter em nível lógico alto.

Comparando os sinais de saída obtidos na implementação do modelo com os da simulação, apresentados na Figura 35(a), verifica-se que o modelo traduzido funcionou corretamente para a primeira situação implementada.

Na segunda situação (intervalo de tempo de 10 a 20 segundos da simulação) mantém-se os sinais de entrada Reset e Data em nível lógico alto e nível lógico baixo, respectivamente. Na Figura 35 apresentam-se os sinais de saída S1 e S0 obtidos com a implementação da segunda situação no PSoC juntamente com a simulação realizada no Simulink.

Figura 35 – Sinais de saída S1 e S0 do código de linha MLT-3 implementado no PSoC para os sinais de entrada Reset em nível lógico alto e Data em nível lógico baixo.



(a) Simulação (b) Implementação

Fonte: Elaborada pelo autor.

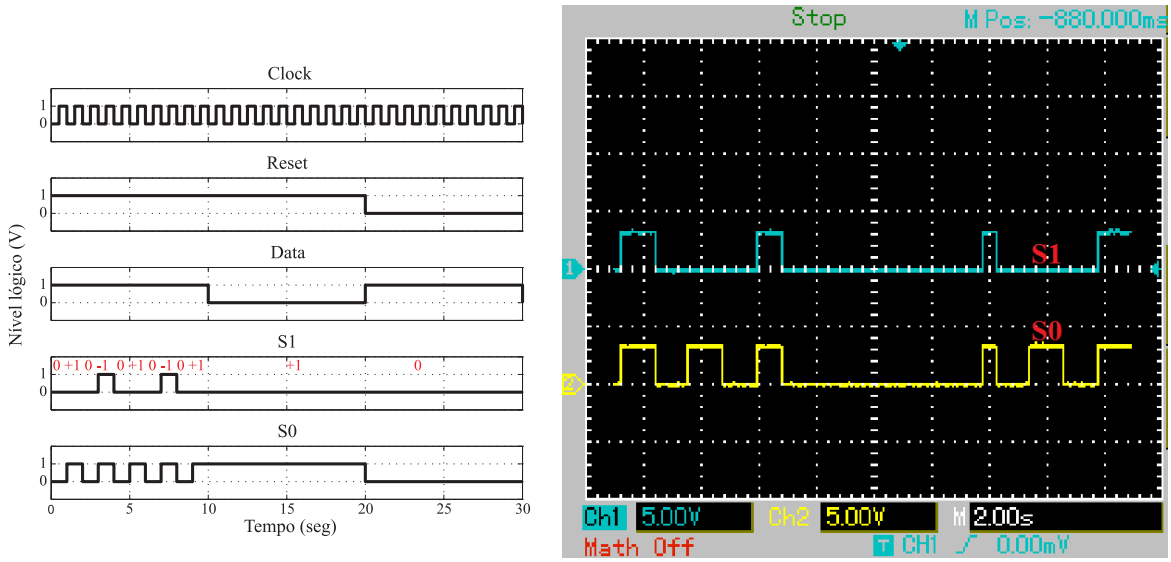
Nota-se que os sinais de saída S1 e S0 não mudam de estado, pois o sinal de entrada Data esta em nível lógico baixo. Logo, o sinal de saída permaneceu no estado atual (estado D) enquanto o sinal Data se manteve em nível lógico baixo.

Comparando a simulação no Simulink com a implementação no PSoC para a segunda situação apresentados na Figura 35, verifica-se que o modelo implementado se comportou

igualmente ao o modelo simulado.

Na terceira situação (intervalo de tempo de 20 a 30 segundos da simulação) o sinal de entrada Reset é mantido em nível lógico baixo. Na Figura 36 apresentam-se os sinais de saída S1 e S0 do modelo traduzido implementando a terceira situação juntamente com a simulação no Simulink.

Figura 36 – Sinais de saída S1 e S0 do código de linha MLT-3 implementado no PSoC para o sinal de entrada Reset em nível lógico baixo.



(a) Simulação (b) Implementação

Fonte: Elaborada pelo autor.

Nota-se que os sinais S1 e S0 se mantém em nível lógico baixo, pois o sinal de entrada Reset se encontra em nível lógico baixo. Como descrito no ambiente Simulink, o sinal de saída transita para o estado A no momento que o sinal Reset transita para nível lógico baixo, permanecendo neste estado até que ocorra uma transição no sinal de Reset para nível lógico alto.

Comparando os sinais de saída S1 e S0 da simulação no Simulink com os sinais de saída S1 e S0 da implementação da terceira situação no PSoC, verifica-se o mesmo comportamento obtido na simulação no modelo implementado apresentado na Figura 36.

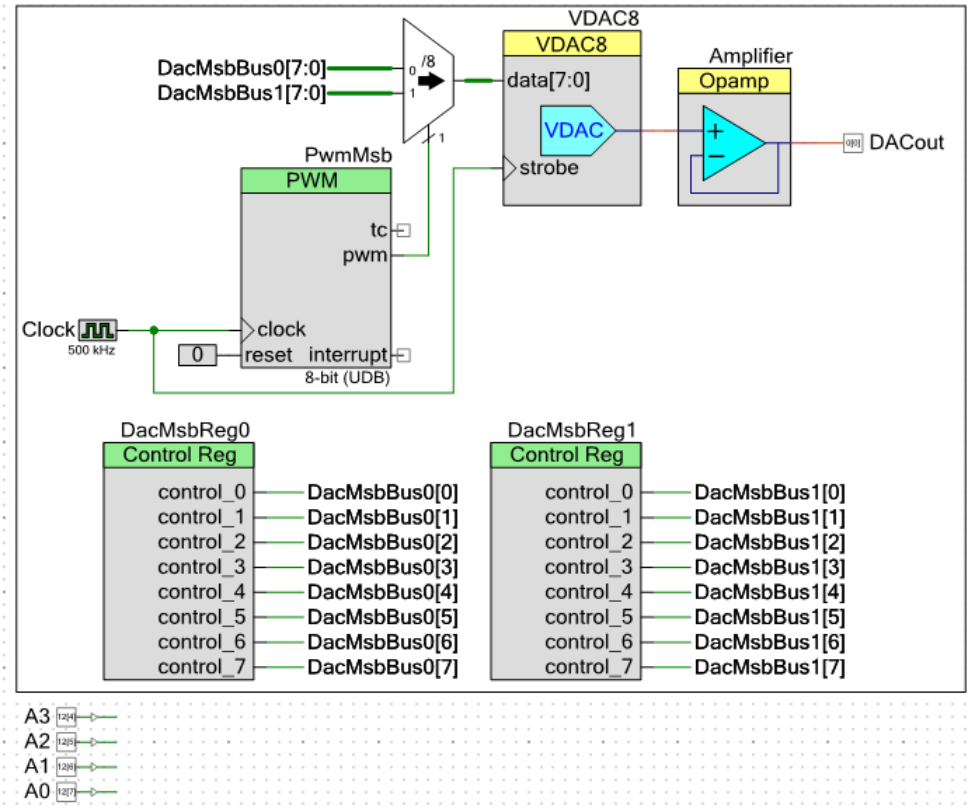
Com os resultados obtidos na implementação do modelo de código de linha MLT-3 conclui-se que a ferramenta MS^{2}PSoC traduziu o modelo descrito no Simulink corretamente, comprovando seu funcionamento com sistemas digitais.

5.2 IMPLEMENTAÇÃO DO CONVERSOR D/A LADDER R-2R

O modelo do conversor D/A Ladder R-2R descrito no Simulink, apresentado na Figura 22, foi traduzido para o ambiente PSoC Creator utilizando-se a ferramenta MS^{2}PSoC. Após traduzido, o projeto gerado foi aberto no PSoC Creator para configuração dos portos de entrada e saída. No Apêndice B apresenta-se o código C gerado para o arquivo *main.c* do modelo conversor D/A Ladder R-2R.

Foi necessário acrescentar as entradas digitais A3, A2, A1 e A0 com a opção *HW Connection* desabilitada para que não ocorresse erro das entradas desconectadas. Na Figura 37 apresenta-se o ambiente PSoC Creator com as configurações realizadas.

Figura 37 – Conversor D/A Ladder R-2R no PSoC Creator.



Fonte: Elaborada pelo autor.

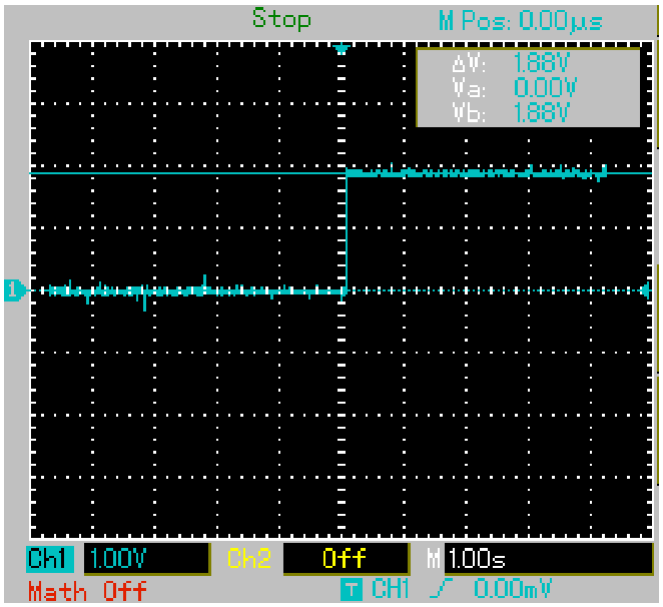
As entradas digitais foram configuradas nos portos digitais e a saída analógica foi configurada no porto analógico. Com os portos configurados, o projeto foi compilado e implementado no sistema embarcado PSoC.

Como entrada para o conversor D/A Ladder R-2R traduzido utilizou-se a mesma combinação de bits aplicada na simulação no Simulink apresentada na Figura 23, ou seja, 0110 para as entradas A3, A2, A1 e A0, respectivamente.

A entrada foi aplicada na forma de onda quadrada com frequência de 0,1 Hz e amplitude de 5 V, gerada utilizando-se um microcontrolador ATMEGA 328P-PU. Portanto, as entradas A3 e A0 foram conectadas ao GND do microcontrolador e as entradas A2 e A1 foram conectadas ao sinal gerado pelo microcontrolador.

O canal 1 do osciloscópio foi utilizado para se obter a forma de onda da saída do conversor D/A Ladder R-2R implementado em hardware PSoC. Na Figura 38 apresenta-se a forma de onda obtida.

Figura 38 – Resultado da implementação do conversor D/A Ladder R-2R no PSoC.



Fonte: Elaborada pelo autor.

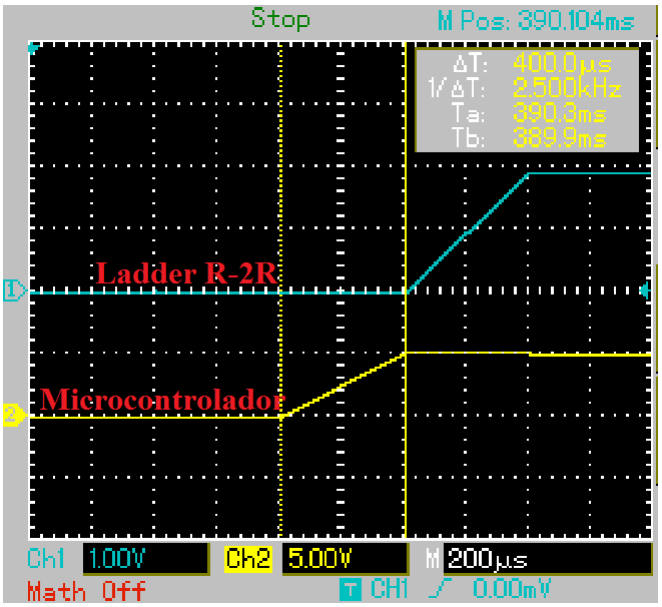
Nota-se que foi obtido o valor de tensão de 1,88 V na saída do conversor D/A Ladder R-2R implementado no PSoC. Comparando este valor com o obtido na simulação apresentada na Figura 23 (1,875 V), pode-se concluir que a ferramenta MS^{2}PSoC traduziu corretamente o modelo do conversor desenvolvido no Simulink.

Utilizou-se o modelo do conversor D/A Ladder R-2R traduzido para verificar o tempo de resposta de um modelo implementado em software no sistema embarcado PSoC. Para isso, conectou-se o canal 2 do osciloscópio na saída do sinal gerado pelo microcontrolador. Desta forma, a diferença de tempo entre o sinal gerado pelo microcontrolador e o sinal de saída do conversor será o tempo de resposta do sistema.

Repetiu-se a implementação apresentada na Figura 37 obtendo-se também o sinal gerado pelo microcontrolador. Na Figura 39 apresenta-se a forma de onda obtida com o osciloscópio.

Obteve-se um tempo de resposta de 400 μs conforme apresentado na Figura 39. Vale

Figura 39 – Resultado da implementação do conversor D/A Ladder R-2R no PSoC para verificar o tempo de resposta do sistema embarcado PSoC.



Fonte: Elaborada pelo autor.

ressaltar que o modelo do conversor D/A Ladder R-2R traduzido executa apenas algumas operações matemáticas para obter o valor de saída do sistema. Logo, o tempo de resposta tende a aumentar para sistemas que exigem maior processamento.

Para se estudar a implementação em hardware e software empregando a ferramenta MS^{2}PSoC realizou-se a comparação do tempo de resposta do modelo do conversor D/A Ladder R-2R implementado no PSoC com um conversor D/A Ladder R-2R implementado em hardware.

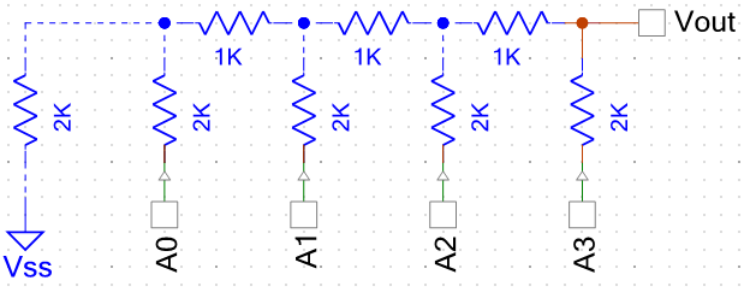
5.2.1 Comparação hardware e software

A ferramenta MS^{2}PSoC gera um código de software que descreve o comportamento do modelo traduzido. Com o objetivo de comparar o tempo de resposta do modelo traduzido com um implementado em hardware, utilizou-se o conversor D/A Ladder R-2R.

Para implementar o circuito do conversor D/A Ladder R-2R de 4 bits em hardware utilizou-se resistores de 1 KΩ. Obteve-se as resistências de 2 KΩ com dois resistores de 1 KΩ em série. Na Figura 40 apresenta-se o esquemático do circuito do conversor D/A Ladder R-2R desenvolvido em hardware.

Como entrada dos sistemas utilizou-se o microcontrolador ATMEGA 328P-PU para gerar um sinal de entrada na forma de onda quadrada de 0,1 Hz com amplitude de 5 V.

Figura 40 – Esquemático do circuito conversor D/A Ladder R-2R implementado em hardware.



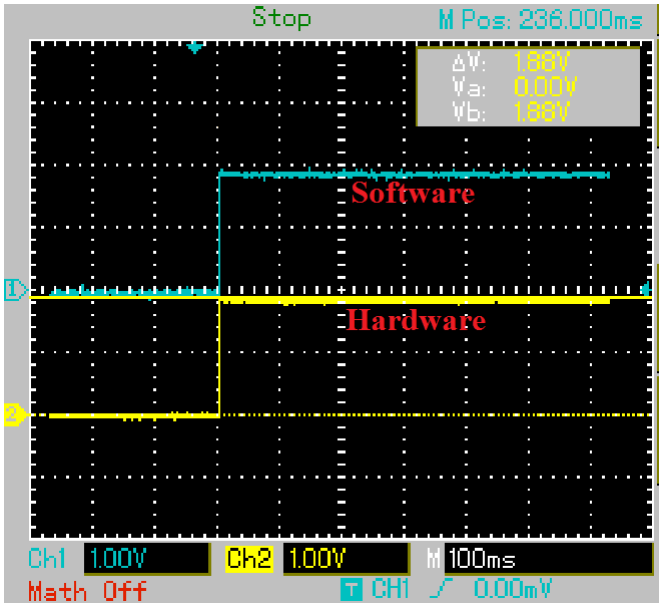
Fonte: Elaborada pelo autor.

O sinal gerado pelo microcontrolador foi utilizado como entrada para ambos os sistemas, hardware e software, simultaneamente e as formas de onda das saídas foram obtidas com o osciloscópio.

As entradas A3 e A0 foram conectadas ao terra (GND) e as entradas A2 e A1 foram conectadas ao sinal de saída do microcontrolador. Vale ressaltar que trata-se das conexões de ambos os sistemas.

A saída do sistema implementado em software foi obtido com o canal 1 do osciloscópio e o implementado em hardware foi obtido com o canal 2. Na Figura 41 apresentam-se as formas de ondas obtidas do conversor D/A Ladder R-2R implementado em software e do implementado em hardware.

Figura 41 – Comparação da saída do conversor D/A Ladder R-2R implementado em software e em hardware.

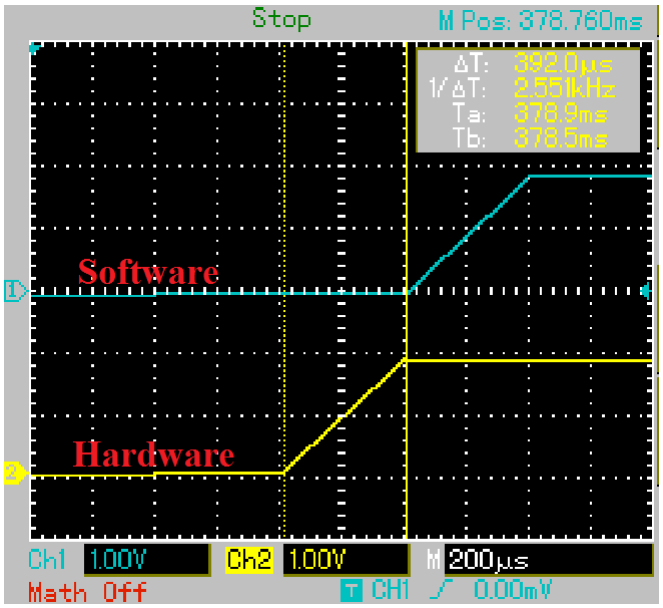


Fonte: Elaborada pelo autor.

Nota-se que ambos os sistemas implementados obtiveram a tensão de saída no valor de 1,88 V confirmando a correta implementação do conversor D/A Ladder R-2R em software quando comparado com o implementado em hardware. No entanto, não foi possível verificar a diferença no tempo de resposta devido à escala de 100 ms utilizada no osciloscópio.

Alterou-se a escala do osciloscópio para 200 μs para ser possível verificar a diferença no tempo de resposta dos sistemas. Na Figura 42 apresenta-se a diferença no tempo de resposta obtida.

Figura 42 – Diferença no tempo de resposta do conversor D/A Ladder R-2R implementado em software e em hardware.



Fonte: Elaborada pelo autor.

Com a resolução de 200 μs no osciloscópio comprovou-se que o modelo do conversor D/A Ladder R-2R implementado em hardware foi 392 μs mais rápido que o sistema implementado em software.

Com esta comparação é possível concluir que a ferramenta desenvolvida, quando realiza as implementações de modelos em software, não é eficiente para sistemas que um atraso de 400 μs comprometerá a saída do sistema. Vale ressaltar que este valor tende a aumentar na medida que a complexidade do modelo desenvolvido aumente.

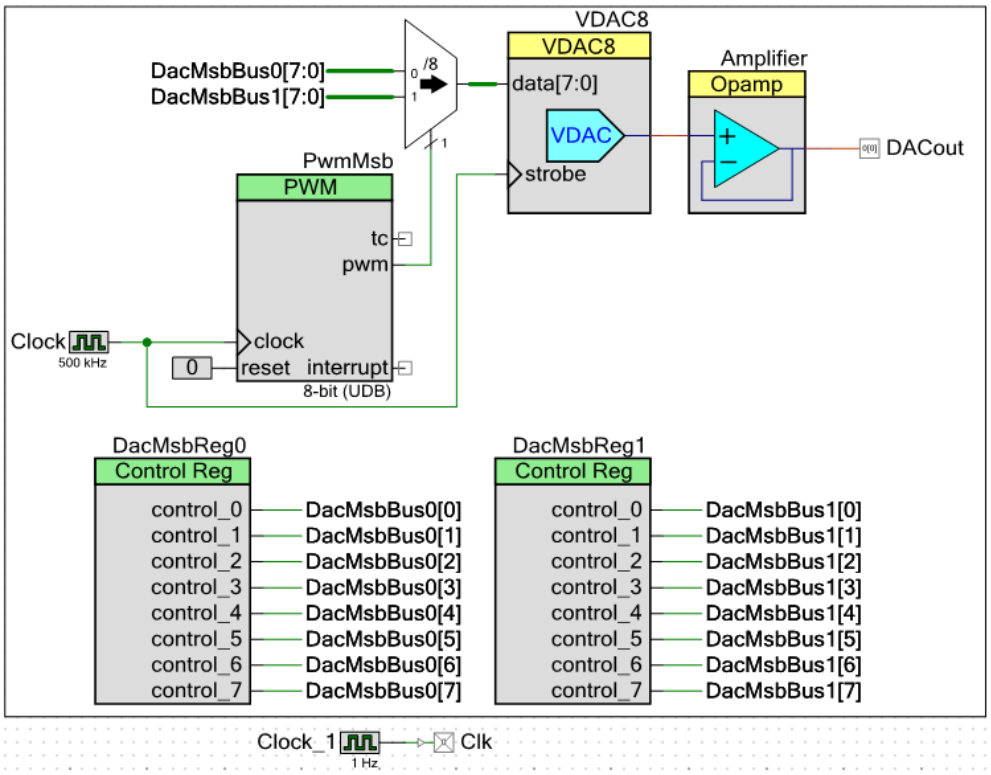
Por exemplo, em um sistema de controle de temperatura veicular um tempo de resposta de microssegundos (μs) não comprometerá a saída do sistema. Porém, em aplicações em que o microssegundo é um tempo significativo para o sinal de saída do sistema, a aplicação em software se torna inadequada.

5.3 IMPLEMENTAÇÃO DO SISTEMA DE CONTROLE DE TEMPERATURA VEICULAR

Para traduzir o modelo de um controle de temperatura veicular utilizou-se o sistema desenvolvido no Simulink, apresentado na Figura 24, como entrada da ferramenta MS^{2}PSoC.

O modelo traduzido foi aberto no ambiente PSoC Creator para a configuração dos portos de entrada e saída. Utilizou-se, como entrada do sistema, um *Clock* de 1 Hz disponível na biblioteca *System* do PSoC Creator. Na Figura 43 apresenta-se o sistema de controle de temperatura veicular descrito no ambiente PSoC Creator.

Figura 43 – Sistema de controle de temperatura veicular implementado no PSoC Creator.



Fonte: Elaborada pelo autor.

Foi necessário alterar o nome da variável Clock para Clk, pois trata-se de uma palavra reservada do ambiente PSoC Creator. Outro ajuste necessário foi no código C gerado com relação à leitura da variável Clk para que o programa fosse executado na transição da borda de subida e de descida desta variável. A linha alterada do código é apresentada a seguir:

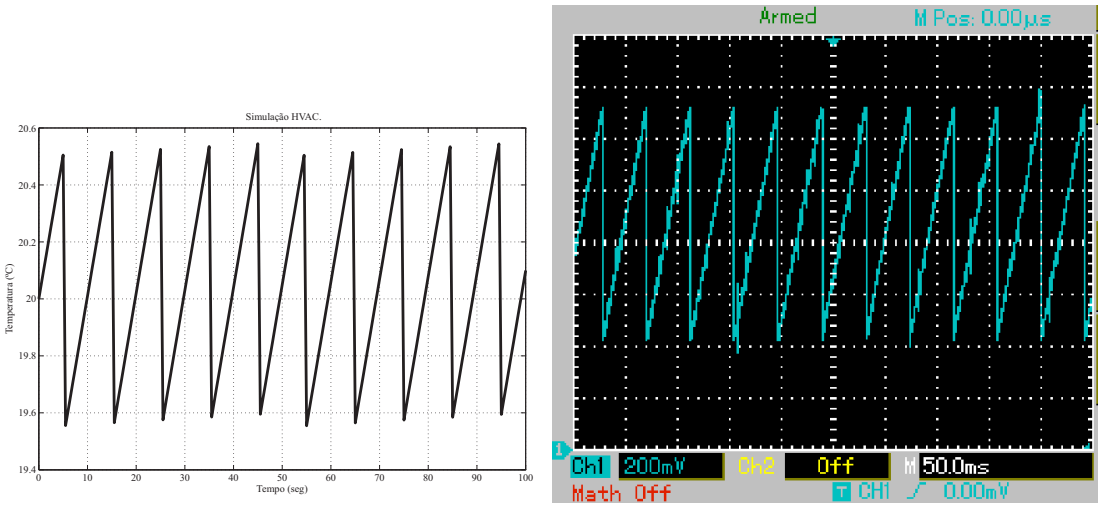
$$\text{if(Clk} == 1 \;||\; \text{Clk} == 0)$$

A condição *if* engloba o código contido na função *main*. Foi necessário, também, adicionar as condições iniciais das variáveis BlowerOut, ACAct e HeaterAct do bloco F_Controle. O código C para o sistema de controle de temperatura é apresentado no Apêndice C.

O acionamento do ar condicionado ou do aquecedor do sistema tem uma largura de banda de $0,5\,^{\circ}C$ do valor utilizado como entrada, ou seja, se configurado o valor $20\,^{\circ}C$ na entrada, a saída varia entre 19,5 a $20,5\,^{\circ}C$, conforme apresentado na Figura 32.

A largura de banda do acionamento no sistema de controle de temperatura veicular traduzido resultaria em uma diferença de 10 mV quando implementado em hardware PSoC. Para facilitar a visualização da saída do sistema implementado multiplicou-se a largura de banda dos acionamentos por um fator 10. Na Figura 44 apresenta-se a forma de onda obtida na saída do sistema de controle de temperatura veicular.

Figura 44 – Sinal do sistema de controle de temperatura veicular implementado no PSoC.



(a) Simulação (b) Implementação

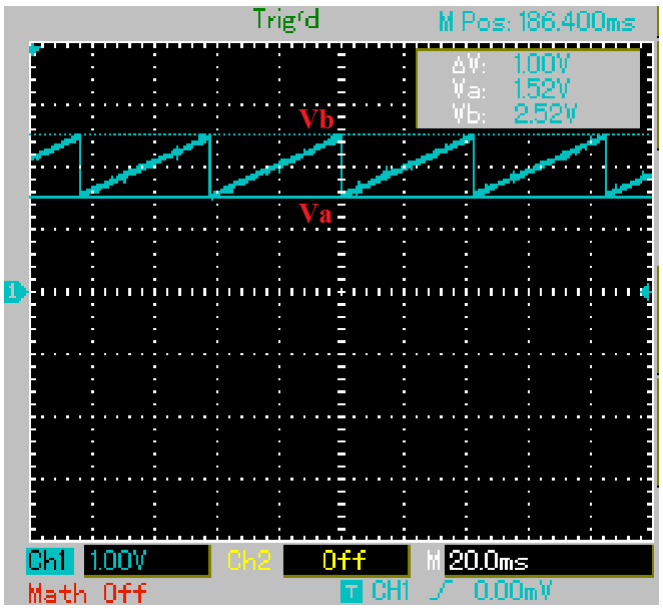
Fonte: Elaborada pelo autor.

Comparando a forma de onda obtida na implementação do sistema no PSoC com o simulado no Simulink pode-se concluir que a ferramenta realizou a tradução do sistema de controle de temperatura veicular corretamente. Vale ressaltar que o valor do Setpoint do sistema foi mantido no valor de $20\,^{\circ}C$, conforme simulação.

Para verificar a mudança realizada na largura de banda dos acionamentos de 0,5 para $5\,^{\circ}C$ do sistema traduzido mediu-se o valor mínimo e o valor máximo da saída afim de se obter o valor da variação da temperatura. Na Figura 45 apresenta-se a forma de onda

com as medições para esta nova situação.

Figura 45 – Variação da temperatura no sistema de controle de temperatura veicular implementado no PSoC.



Fonte: Elaborada pelo autor.

Nota-se que o valor mínimo obtido na saída foi o valor de 1,52 V referente à temperatura de 15 °C e o valor máximo de 2,52 V referente à temperatura de 25 °C. Deste modo, a temperatura de saída do sistema de controle de temperatura veicular variou de 15 °C a 25 °C, ou seja, com uma largura de banda de 5 °C conforme a alteração realizada no sistema.

Com este estudo de caso verificou-se que a ferramenta MS^{2}PSoC opera de forma adequada para projetos contendo sinais mistos desenvolvidos no ambiente Simulink.

No próximo capítulo apresentam-se as conclusões obtidas com o desenvolvimento deste trabalho.

6 CONCLUSÕES

Atualmente tem se observado que o aumento da complexidade dos projetos em sistemas embarcados tem reflexo direto no tempo de desenvolvimento. Esta complexidade é decorrente da natureza dos diferentes componentes envolvidos e da quantidade cada vez maior de funcionalidades que se incorporam em um único circuito integrado.

Para atender a essa demanda e superar as restrições de projeto, várias pesquisas estão sendo desenvolvidas visando o desenvolvimento de metodologias e ferramentas computacionais a serem empregadas nesta classe de sistema eletrônico.

Um recurso que previne identificar erros somente após a implementação de projetos é a simulação. Com a simulação é possível verificar o funcionamento do sistema desenvolvido e, em caso de erros, simplifica-se o processo de análise e correção.

Neste trabalho apresentou-se uma ferramenta computacional denominada MS^{2}PSoC (Matlab/Simulink para PSoC Creator) que traduz um modelo desenvolvido em alto nível de abstração no ambiente Matlab/Simulink para descrições que podem ser sintetizadas no ambiente PSoC Creator.

Esta tradução tem a finalidade de implementar um modelo do Simulink no sistema embarcado PSoC. Vale ressaltar que além da tradução, a ferramenta gera a estrutura de arquivos necessária para que o sistema seja aberto como um projeto no ambiente PSoC Creator.

A motivação para o desenvolvimento desta ferramenta decorre do fato de que o PSoC Creator não disponibiliza o recurso de simulação e não se encontrou na literatura ferramenta com este propósito. Portanto, o Simulink foi utilizado como *front-end* para o desenvolvimento e simulação dos modelos. A inovação proposta neste trabalho é uma ferramenta que automatiza uma parte do processo de implementação de um modelo simulado, no PSoC.

A metodologia empregada é a de realizar a tradução do sistema digital para a linguagem de descrição de hardware Verilog que é implementada em um componente

digital do PSoC Creator ou para linguagem C se houver função na máquina de estados finita. O sistema analógico é traduzido para a linguagem C e implementada como software.

Tendo em vista que o modelo traduzido é implementada em software, realizou-se uma comparação entre um sistema implementado em hardware e em software no PSoC. Esta comparação teve a finalidade de se estudar a eficiência da ferramenta MS^{2}PSoC. Foi utilizado o conversor D/A Ladder R-2R para este fim e o parâmetro de comparação utilizado foi o tempo de resposta dos sistemas para um determinado sinal de entrada.

O resultado obtido da comparação mostrou uma diferença de 392 μs no tempo de resposta entre os dois sistemas, sendo o sistema implementado em hardware mais rápido, como era de se esperar. Portanto, conclui-se que a ferramenta MS^{2}PSoC, quando realiza a implementação de modelos em software, é eficiente para projetos em que um atraso de 400 μs não comprometerá o funcionamento do sistema.

Foram utilizados três estudos de casos para avaliar a ferramenta MS^{2}PSoC. Em todos os estudos desenvolveu-se o sistema no Simulink para análise e simulação e, posteriormente, traduziu-os para o ambiente PSoC Creator utilizando a ferramenta MS^{2}PSoC para serem implementados no PSoC.

Comparando os resultados obtidos no PSoC com os obtidos no Simulink verificou-se que a ferramenta MS^{2}PSoC realizou as traduções dos modelos corretamente. Ressalta-se que as configurações realizadas no PSoC Creator após tradução são necessárias devido aos arquivos que especificam os portos e o ambiente gráfico serem codificados, não permitindo a configuração automática.

Algumas melhorias que podem ser realizadas na ferramenta MS^{2}PSoC com relação a tradução de sistemas é a de uma otimização voltada ao sistema embarcado PSoC antes da tradução. Pode-se, também, permitir que o usuário possa criar outros elementos de biblioteca, ampliando a biblioteca *mspsoc_lib*.

Conclui-se que o objetivo no desenvolvimento de uma ferramenta computacional capaz de traduzir um sistema do ambiente Matlab/Simulink para o ambiente PSoC Creator foi alcançado. Salienta-se que até o momento em que este texto foi escrito não se encontrou nenhum registro de uma ferramenta com objetivo similar.

Como proposta de trabalho futuro pode-se realizar o caminho inverso da ferramenta MS^{2}PSoC, ou seja, gerar um modelo para o Simulink a partir de um sistema descrito diretamente no ambiente PSoC Creator com o objetivo de simular o sistema desenvolvido.

REFERÊNCIAS

ALMEIDA, T. da S.; **Ambiente computacional para projetos de sistemas com tecnologia mista**. 2009. 144 f. Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia, Universidade Estadual Paulista – UNESP, Ilha Solteira, 2009.

ALMEIDA, T. da S.; SILVA, A. C. R. da; SAMPAIO, D. J. B. S. Computational tool to support design of DAC converter model AD7528 with the object code in XML. In: ELETRONICS, ROBOTICS AND AUTOMOTIVE MECHANICS CONFERENCE - CERMA, 9., 2012, Cuernavaca. **Proceedings...** Cuernavaca: IEEE, 2012. p. 327–332.

ALMEIDA, T. da S.; SILVA, A. C. R. da; SAMPAIO, D. J. B. S. Improvement and evaluation of the MS2SV for mixed systems design described in abstraction high level. In: ELECTRONICS, ROBOTICS AND AUTOMOTIVE MECHANICS CONFERENCE - CERMA, 9., 2012, Cuernavaca. **Proceedings...** Cuernavaca: IEEE, 2012. p. 353–358.

ARAIZA-ILLAN, D.; EDER, K.; RICHARDS, A. Formal verification of control systems' properties with theorem proving. In: UKACC INTERNATIONAL CONFERENCE ON CONTROL - CONTROL, 10., 2014, Loughborough. **Proceedings...** Loughborough: IEEE, 2014. p. 244–249.

ATAT, Y.; ZERGAINOH, N. E. Automatic code generation for MPSoC platform starting from Simulink/Matlab: new approach to bridge the gap between algorithm and architecture design. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGIES: FROM THEORY TO APPLICATIONS - ICTTA, 3., 2008, Damascus. **Proceedings...** Damascus: IEEE, 2008. p. 1–6.

CHINDRIS, G.; MURESAN, M. Deploying Simulink models into system-on-chip structures. In: INTERNATIONAL SPRING SEMINAR ON ELECTRONICS TECHNOLOGY - ISSE, 29., 2006, St. Marienthal. **Proceedings...** St. Marienthal: IEEE, 2006. p. 313–317.

CORPORATION, C. S. **PSoC architecture TRM**. California: Cypress Semiconductor Corporation, 2014. 1-455 p.

CORPORATION, C. S. **Just enough verilog for PSoC**. [S.l.: s.n.], 2013. Disponível em: <http://www.cypress.com/>. Acesso em: 10 jun. 2014.

CORPORATION, C. S. **PSoC creator new features survey reminder and partial results**. [S.l.: s.n.], 2014. Disponível em: <http://www.cypress.com>. Acesso em: 11 nov. 2014.

CURRIE, E. H.; ESS, D. V. **PSoC 3/5 reference book**. California: Cypress Semiconductor Corporation, 2010. 534 p.

EVERITT, J.; PARKER, J.; HURST, P.; NACK, D. A 10/100 Mb/s CMOS ethernet transceiver for 10BaseT, 100Basetx, and 100BaseFX. In: SOLID-STATE CIRCUITS CONFERENCE, 1., 1998, San Francisco. **Proceedings...** San Francisco: IEEE, 1998. p. 210–211.

FARKAS, T.; NEUMANN, C.; HINNERICHS, A. An integrative approach for embedded software design with UML and Simulink. In: ANNUAL IEEE INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE - COMPSAC, 33., 2009, Seattle. **Proceedings...** Seattle: IEEE, 2009. v. 2. p. 516–521.

GESTNER, B.; ANDERSON, D. V. Automatic generation of Modelsim-Matlab interface for RTL debugging and verification. In: MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS - MWSCAS, 50., 2007, Montreal. **Proceedings...** Montreal: IEEE, 2007. p. 1497–1500.

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE standard for verilog hardware description language**. New York: IEEE, 2006. p. 1–560.

KRIZAN, J.; ERTL, L.; BRADAC, M.; JASANSKY, M.; ANDREEV, A. Automatic code generation from Matlab/Simulink for critical applications. In: CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING - CCECE, 27., 2014, Toronto. **Proceedings...** Toronto: IEEE, 2014. p. 1–6.

LAIA, M. A. M.; CRUVINEL, P. E. Using Simulink to generate HDL code for validating an embedded kalman filter. In: BRAZILIAN CONFERENCE ON CRITICAL EMBEDDED SYSTEMS - CBSEC, 2., 2012, Campinas. **Proceedings...** Campinas: IEEE, 2012. p. 30–35.

LAMBERSKY, V. Model based design and automated code generation from Simulink targeted for TMS570 MCU. In: EUROPEAN DSP EDUCATION AND RESEARCH CONFERENCE - EDERC, 5., 2012, Amsterdam. **Proceedings...** Amsterdam: IEEE, 2012. p. 225–228.

LI, M.; KUMAR, R. Stateflow to extended finite automata translation. In: ANNUAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE WORKSHOPS - COMPSACW, 35., 2011, Munich. **Proceedings...** Munich: IEEE, 2011. p. 1–6.

LI, M.; KUMAR, R. Model-based automatic test generation for Simulink/Stateflow using extended finite automaton. In: INTERNATIONAL CONFERENCE ON AUTOMATION SCIENCE AND ENGINEERING - CASE, 8., 2012, Seoul. **Proceedings...** Seoul: IEEE, 2012. p. 857–862.

LI, M.; KUMAR, R. Recursive modeling of Stateflow as input/output-extended automaton. **IEEE Transactions on Automation Science and Engineering**, Piscataway, v. 11, n. 4, p. 1229–1239, 2014.

MAJUMDAR, R.; SAHA, I.; UEDA, K.; YAZAREL, H. Compositional equivalence checking for models and code of control systems. In: ANNUAL CONFERENCE ON DECISION AND CONTROL - CDC, 52., 2013, Firenze. **Proceedings...** Firenze: IEEE, 2013. p. 1564–1571.

MATHWORKS. **Documentação**. [S.l.: s.n.], 2013. Disponível em: <http://www.mathworks.com/>. Acesso em: 10 jun. 2014.

MOON, T. M.; SEO, S. H.; KIM, J. H.; HWANG, S. H.; JEON, J. W. Simulation with consideration of hardware characteristics and auto-generated code using Matlab/Simulink. In: INTERNATIONAL CONFERENCE ON CONTROL, AUTOMATION AND SYSTEMS - ICCAS, 7., 2007, Seoul. **Proceedings...** Seoul: IEEE, 2007. p. 1494–1498.

NATALE, M. D.; GUO, L.; ZENG, H.; SANGIOVANNI-VINCENTELLI, A. Synthesis of multitask implementations of Simulink models with minimum delays. **Transactions on Industrial Informatics.** Magarpatta, v. 6, n. 4, p. 637–651, 2010.

NEEMA, S.; KALMAR, Z.; SHI, F.; VIZHANYO, A.; KARSAI, G. A visually-specified code generator for Simulink/Stateflow. In: SYMPOSIUM ON VISUAL LANGUAGES AND HUMAN-CENTRIC COMPUTING, 2., 2005, Dallas. **Proceedings...** Dallas: IEEE, 2005. p. 275–277.

NETLAND, Ø.; SKAVHAUG, A. Software module real-time target: Improving development of embedded control system by including Simulink generated code into existing code. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS - SEAA, 39., 2013, Santander. **Proceedings...** Santander: IEEE, 2013. p. 232–235.

SAMPATH, P.; RAJEEV, A. C.; RAMESH, S. Translation validation for Stateflow to C. In: ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE - DAC, 51., 2014, San Francisco. **Proceedings...** San Francisco: IEEE, 2014. p. 1–6.

SCHWARZ, M.; SHENG, H.; SHELEH, A.; BOERCSOEK, J. Matlab/Simulink generated source code for safety related systems. In: INTERNATIONAL CONFERENCE ON COMPUTER SYSTEMS AND APPLICATIONS - AICCSA, 8., 2008, Doha. **Proceedings...** Doha: IEEE, 2008. p. 1058–1063.

SILVA, A. C. R. da.; GROUT, I. MS2SV: environment for translation of Matlab/Simulink models to VHDL-AMS models. **Latin America Transactions.** Piscataway, v. 9, n. 5, p. 663–672, 2011.

THOMAS, D.; MOORBY, P. **The verilog hardware description language**. 5. ed. Dordrecht: Springer, 2002. 386 p.

TRANCHERO, M.; REYNERI, L. M. Automatic generation of VHDL code for self-timed circuits from Simulink specifications. In: INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS - ICECS, 14., 2007, Marrakech. **Proceedings...** Marrakech: IEEE, 2007. p. 287–290.

TRANCHERO, M.; REYNERI, L.M.; BINK, A.; de WIT, M. An automatic approach to generate haste code from Simulink specifications. In: SYMPOSIUM ON ASYNCHRONOUS CIRCUITS AND SYSTEMS - ASYNC, 15., 2009, Chapel Hill. **Proceedings...** Chapel Hill: IEEE, 2009. p. 175–184.

VALTERS, G. Initial version of Matlab/Simulink based tool for VHDL code generation and FPGA implementation of elementary generalized unitary rotation. In: NORCHIP, 29., 2011, Lund. **Proceedings...** Lund: IEEE, 2011. p. 1–6.

WANG, G.; DI NATALE, M.; MOSTERMAN, P. J.; SANGIOVANNI-VINCENTELLI, A. Automatic code generation for synchronous reactive communication. In: INTERNATIONAL CONFERENCE ON EMBEDDED SOFTWARE AND SYSTEMS - ICESS, 4., 2009, Zhejiang. **Proceedings...** Zhejiang: IEEE, 2009. p. 40–47.

WANG, T.; JOBREDEAUX, R.; PAKMEHR, M.; VIVIES, M.; FERON, E. An application of a prototype credible autocoding and verification tool-chain. In: DIGITAL AVIONICS SYSTEMS CONFERENCE - DASC, 33., 2014, Colorado Springs. **Proceedings...** Colorado Springs: IEEE, 2014. p. 2E4–1–2E4–14.

ZOU, L. et al. Verifying Simulink diagrams via a hybrid hoare logic prover. In: INTERNATIONAL CONFERENCE ON EMBEDDED SOFTWARE - EMSOFT, 8., 2013, Montreal. **Proceedings...** Montreal: IEEE, 2013. p. 1–10.

APÊNDICE A – CÓDIGO GERADO PELA FERRAMENTA MS^{2}PSoC PARA O CÓDIGO DE LINHA MLT-3

```
//`#start header` -- edit after this line, do not edit this line
// ========================================================
//
// UNESP - Universidade Estadual Paulista
// FEIS - Faculdade de Ilha Solteira
// DEE - Departamento de Engenharia Eletrica
// LPSSD - Laboratorio de Sinais e Sistemas Digitais
//
// ========================================================

        `include "cypress.v"
        `include "C:/Users/Alexandre/Desktop/mlt3jk.cydsn\component01\library.v"

//`#end` -- edit above this line, do not edit this line

module component01 (
        input Clock,
        input Reset,
        input Data,
        output S1,
        output S0
        );

//`#start body` -- edit after this line, do not edit this line

        wire and1_o1;
        wire and2_o1;
        wire and3_o1;
        wire and4_o1;
```

```
    wire and5_o1;
    wire and6_o1;
    wire and7_o1;
    wire jkflipflop0_o1;
    wire jkflipflop0_o2;
    wire jkflipflop1_o1;
    wire jkflipflop1_o2;
    wire not_o1;
    wire not1_o1;
    wire or1_o1;
    wire or2_o1;
    wire or3_o1;
    wire or4_o1;


and2 i1 (.a(Reset), .b(Data), .y(and1_o1));
and2 i2 (.a(Data), .b(jkflipflop0_o2), .y(and2_o1));
and3 i3 (.a(Reset), .b(Data), .c(jkflipflop0_o2), .y(and3_o1));
and4 i4 (.a(Reset), .b(Data), .c(jkflipflop1_o2), .d(jkflipflop0_o2), .y(and4_o1));
and4 i5 (.a(Reset), .b(not1_o1), .c(jkflipflop1_o1), .d(jkflipflop0_o1), .y(and5_o1));
and3 i6 (.a(Reset), .b(Data), .c(jkflipflop0_o2), .y(and6_o1));
and3 i7 (.a(Reset), .b(not1_o1), .c(jkflipflop0_o1), .y(and7_o1));
jkflipflop i8 (.j(and1_o1), .k(or2_o1), .clk(Clock), .q(jkflipflop0_o1), .q1(jkflipflop0_o2));
jkflipflop i9 (.j(and3_o1), .k(or1_o1), .clk(Clock), .q(jkflipflop1_o1), .q1(jkflipflop1_o2));
inv i10 (.a(Reset), .y(not_o1));
inv i11 (.a(Data), .y(not1_o1));
or2 i12 (.a(not_o1), .b(and2_o1), .y(or1_o1));
or2 i13 (.a(not_o1), .b(Data), .y(or2_o1));
or2 i14 (.a(and4_o1), .b(and5_o1), .y(or3_o1));
or2 i15 (.a(and6_o1), .b(and7_o1), .y(or4_o1));

assign S1 = or3_o1;
assign S0 = or4_o1;

//`#end` -- edit above this line, do not edit this line

endmodule

//`#start footer` -- edit after this line, do not edit this line
//`#end` -- edit above this line, do not edit this line
```

APÊNDICE B – CÓDIGO GERADO PELA FERRAMENTA MS^{2}PSoC PARA O CONVERSOR D/A LADDER R-2R

```
// ========================================================
//
// UNESP − Universidade Estadual Paulista
// FEIS − Faculdade de Ilha Solteira
// DEE − Departamento de Engenharia Eletrica
// LPSSD − Laboratorio de Sinais e Sistemas Digitais
//
// ========================================================

#include <device.h>
#include <stdio.h>
#include <math.h>

void main()
{
  uint16 DacValue;
  float A3;
  float Gain;
  float A2;
  float Gain1;
  float A1;
  float Gain2;
  float Sum1;
  float Sum2;
  float Sum11;
  float Sum12;
  float Sum21;
  float Sum22;
  float Product1;
  float Vref = 5;
```

```
  float Product2;
  float Vout;
  float A0;
  float Gain3;
  VDAC8_Start();
  PwmMsb_Start();
  Amplifier_Start();

 for(;;)
 {
  A3 = A3_Read();
  A2 = A2_Read();
  A1 = A1_Read();
  A0 = A0_Read();
  Gain = A3 * 0.5;
  Gain1 = A2 * 0.25;
  Gain2 = A1 * 0.125;
  Sum1 = Gain;
  Sum2 = Gain1;
  Sum11 = (+Sum1+Sum2);
  Sum12 = Gain2;
  Sum21 = (+Sum11+Sum12);
  Gain3 = A0 * 0.0625;
  Sum22 = Gain3;
  Product1 = (+Sum21+Sum22);
  Product2 = Vref;
  Vout = (Product1 * Product2);
  DacValue = Vout*1000;
  DacMsbReg0_Write(DacValue >> 4);
  DacMsbReg1_Write((DacValue >> 4) + 1);
  PwmMsb_WriteCompare((uint8)(DacValue & 0X0F));
 }
}
```

APÊNDICE C – CÓDIGO GERADO PELA FERRAMENTA MS^{2}PSoC PARA O SISTEMA DE CONTROLE DE TEMPERATURA VEICULAR.

```
// ========================================================
//
// UNESP - Universidade Estadual Paulista
// FEIS - Faculdade de Ilha Solteira
// DEE - Departamento de Engenharia Eletrica
// LPSSD - Laboratorio de Sinais e Sistemas Digitais
//
// ========================================================

#include <device.h>
#include <stdio.h>
#include <math.h>

 float cabine = 293;
 float Out_occupants;
 float entrada_celsius;
 float saida_kelvin;
 float In_celsius;
 float Out_kelvin;
 float vel_motor_ac;
 float torque_ac;
 float ventil_ac;
 float temp_int_ac;
 float tem_saida_ac;
 float mud_temp;
 float vent_vel;
 float temp_ext_aq;
 float tout;
 float SetPTemp;
 float IntTemp;
```

```
 float HeaterAct = 0;
 float ACAct = 0;
 float RequiredTemp;
 float BlowerOut = 0;
 float ent_int;
 float sai_int=293;
 float entrada_kelvin;
 float saida_celsius;

void Calor()
{
  float Calor_ocupante = 100;
  float N_ocupantes;

  N_ocupantes = Calor_ocupante * 1;
  Out_occupants = N_ocupantes;
}

void Celsius_Kelvin()
{
  float Sum12;
  float Constante = 273;
  float Sum11;

  Sum12 = entrada_celsius;
  Sum11 = Constante;
  saida_kelvin = (+Sum11+Sum12);
}

void Celsius_Kelvin1()
{
  float Constante = 273;
  float Sum11;
  float Sum12;

  Sum11 = Constante;
  Sum12 = In_celsius;
  Out_kelvin = (+Sum11+Sum12);
}

void Controle_AC()
{
  float Sum2;
```

```
  float Sum1;
  float Product12;
  float taxa;
  float Product11;
  float efic;
  float Product2;
  float Product1;

  taxa = ventil_ac * 50;
  Product12 = taxa;
  Product2 = torque_ac;
  Product1 = vel_motor_ac;
  efic = (Product1 * Product2) * 0.80;
  Product11 = efic;
  Sum2 = (Product11 / Product12);
  Sum1 = temp_int_ac;
  tem_saida_ac = (+Sum1-Sum2);
}

void Controle_Aquecedor()
{
  float efeito_fan2;
  float vel_fan = 0.12;
  float efeito_fan1;
  float Air_Density;
  float gain;
  float Mux3;
  float cst = 350;
  float Mux2;
  float Mux1;
  float Product1;
  float func_troca_calor;
  float ganho;
  float Product2;

  efeito_fan2 = vent_vel;
  efeito_fan1 = vel_fan;
  Air_Density = (efeito_fan1 * efeito_fan2) * 10;
  gain = Air_Density/100;
  Mux3 = temp_ext_aq;
  Mux2 = cst;
  Mux1 = gain;
  func_troca_calor = Mux2-(Mux2-Mux3)*exp(-(3.14159265*0.004*0.05*23.8)/(Mux1*1007));
```

```
  Product1 = func_troca_calor;
  ganho = mud_temp * 2;
  Product2 = ganho;
  tout = (Product1 * Product2);
}

void F_Controle()
{
  float absTempDiff;

  ACAct = 0;
  HeaterAct = 0;
  BlowerOut = 0.2;
  RequiredTemp = SetPTemp-IntTemp;
  if(RequiredTemp<0)
  {
      absTempDiff = -RequiredTemp;
  }
  else
  {
      absTempDiff = RequiredTemp;
  }
  if(absTempDiff>2030 || absTempDiff<=5)
  {
      BlowerOut=0;
  }
  if(absTempDiff>15 && absTempDiff<50)
  {
    BlowerOut=0.2;
  }
  if(absTempDiff>70 && absTempDiff<100)
  {
    BlowerOut=0.4;
  }
  if(absTempDiff>120 && absTempDiff<150)
  {
    BlowerOut=0.6;
  }
  if(absTempDiff>170 && absTempDiff<200)
  {
    BlowerOut=0.8;
  }
  if(absTempDiff>220 && absTempDiff<2000)
```

```
  {
    BlowerOut=1;
  }
  if(RequiredTemp<−5)
  {
      ACAct = 1;
  }
  if(RequiredTemp>=−5)
  {
  }
  if(RequiredTemp>5)
  {
      HeaterAct = 1;
  }
  if(RequiredTemp<=5)
  {
  }
}

void Interior()
{
  float ganho;

  ganho = ent_int/1000;
  cabine = cabine + ganho;
  sai_int = cabine;
}

void Kelvin_Celsius()
{
  float Sum12;
  float constante = 273;
  float Sum11;

  Sum12 = entrada_kelvin;
  Sum11 = constante;
  saida_celsius = (−Sum11+Sum12);
}

int main()
{
  uint16 DacValue;
  uint8 Clk;
```

```
 float Sum23;
 float Sum11;
 float Sum2;
 float Temp_ext = 30;
 float Sum1;
 float taxa_ar;
 float Sum12;
 float vel_prop2;
 float Tout_H=0;
 float Sum22;
 float Sum21;
 float Torque = 25;
 float Tout_AC=0;
 float Vel_motor = 200;
 float Setpoint = 20;
 float vel_prop1;
 int Controle_Aquecedor_enable;
 int Controle_AC_enable;
 float vent;
 float Sum3;
 float Temperatura;
 VDAC8_Start();
 PwmMsb_Start();
 Amplifier_Start();

for(;;)
{
 Clk = Clk_Read();
 if(Clk == 1 || Clk == 0)
 {
      tout = 0;
      tem_saida_ac = 0;
      entrada_celsius = Setpoint;
      In_celsius = Temp_ext;
      Celsius_Kelvin();
      SetPTemp = saida_kelvin;
      IntTemp = sai_int;
      F_Controle();
      Controle_Aquecedor_enable = HeaterAct;
      Controle_AC_enable = ACAct;
      mud_temp = RequiredTemp;
      vel_prop1 = BlowerOut;
      ventil_ac = BlowerOut;
```

```
        vent_vel = BlowerOut;
        Calor();
        Sum2 = Out_occupants;
        Celsius_Kelvin1();
        temp_ext_aq = Out_kelvin;
        if(Controle_Aquecedor_enable == 1)
        {
          Controle_Aquecedor();
        }
        Tout_H = tout;
        torque_ac = Torque;
        vel_motor_ac = Vel_motor;
        temp_int_ac = sai_int;
        if(Controle_AC_enable == 1)
        {
          Controle_AC();
        }
        Tout_AC = tem_saida_ac;
        Sum23 = sai_int;
        Sum22 = Tout_H;
        Sum21 = Tout_AC;
        vel_prop2 = (+Sum21+Sum22−Sum23);
        vent = (vel_prop1 ∗ vel_prop2) ∗ 120;
        Sum11 = sai_int;
        Sum12 = Out_kelvin;
        taxa_ar = (−Sum11+Sum12) ∗ 0.1;
        Sum1 = taxa_ar;
        Sum3 = vent;
        ent_int = (+Sum1+Sum2+Sum3);
        Interior();
        entrada_kelvin = sai_int;
        Kelvin_Celsius();
        Temperatura = saida_celsius;
        DacValue = Temperatura∗100;
        DacMsbReg0_Write(DacValue >> 4);
        DacMsbReg1_Write((DacValue >> 4) + 1);
        PwmMsb_WriteCompare((uint8)(DacValue & 0X0F));
    }
  }
}
```