Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Relatório de Estágio Supervisionado
Laboratório EMBEDDED

Wislayne Dayanne Pereira da Silva

Campina Grande, PB

04 de dezembro de 2019

Wislayne Dayanne Pereira da Silva

Relatório de Estágio Supervisionado
Laboratório EMBEDDED

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduada em Engenharia Elétrica.

Orientador: Jaidilson Jó da Silva, D.Sc.

Supervisor: Saulo Oliveira Dornellas Luiz, D.Sc.

Campina Grande, PB

04 de dezembro de 2019

Wislayne Dayanne Pereira da Silva

Relatório de Estágio Supervisionado
Laboratório EMBEDDED

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduada em Engenharia Elétrica.

Trabalho aprovado em: ____ / ____ / ____

Jaidilson Jó da Silva, D.Sc.
Orientador

Gutemberg Gonçalves dos S. Júnior
Professor Convidado

Campina Grande, PB

04 de dezembro de 2019

Dedico este trabalho aos meus amados pais, Maria Juraneide Pereira da Silva e Eliosmar Bezerra da Silva, à meus irmãos Wirla Aparecida Pereira da Silva, Wesley Danilo Pereira da Silva e Wirlane Danise Pereira da Silva, e ao meu sobrinho Thiago Vinícius Pereira Belarmino.

Agradecimentos

Em primeiro lugar agradeço a Deus por me dar força e coragem diariamente para seguir minha jornada com fé e resiliência. Em seguida, agradeço aos meus pais, Eliosmar e Juraneide, por me mostrarem desde cedo que a educação é a única forma de transformar realidades e principalmente por todo amor e apoio emocional durante esses cinco anos de curso.

Gratidão aos meus irmãos, Wirla, Wesley e Wirlane, por todas as palavras de carinho e conforto faladas nos momentos de dificuldade. Ao meu sobrinho, Thiago, agradeço por ser minha fonte de motivação diária para não desistir dos meus objetivos.

Aos meus professores, por todos os ensinamentos e oportunidades que me foram dadas. Em especial aos professores: Jaidilson, que me orientou nesse trabalho e no trabalho de conclusão de curso e ao professor Saulo que supervisionou minhas atividades no projeto de Pesquisa e Desenvolvimento a qual o estágio foi realizado.

Aos amigos do laboratório Embedded e colegas de trabalho, Augusto, Bruna, Breno, Camila, Danilo, Jonatas, Lucas, Matheus, Pedro, Rony e Tony, agradeço por todos os aprendizados, conversas, conselhos e cafés. Com vocês eu me descobri como profissional e resolver problemas se tornou muito mais fácil.

"Acredite, pense e faça,
use sua intuição,
transforme sonho em suor,
pensamento em ação".
Bráulio Bessa

Resumo

Neste relatório estão descritas as atividades realizadas pela estudante de graduação em Engenharia Elétrica, Wislayne Dayanne Pereira da Silva, como parte da disciplina de Estágio Supervisionado com carga horária de 300 horas. O estágio foi realizado no período de 06 de agosto a 29 de novembro de 2019 no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded). No estágio supervisonado, a estagiária desenvolveu modelos no *simulink* para geração automática de código para sistemas embarcados.

Palavras-chaves: Comunicação Serial, Estágio Supervionado, Matlab Simulink, Modelos, Microcontroladores.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ADC	Analog to Digital Convert
ARM	Advanced RISC Machine
CEEI	Centro de Engenharia Elétrica e Informática
CATI	Comitê da Área de Tecnologia de Informação
DAC	Digital to Analog Converter
EMDEDDED	Laboratório de Sistemas Embarcados e Computação Pervasiva
IDE	Integrated Development Environment
IoT	Internet das Coisas
I2C	Inter-Integrated Circuit
LCD	Liquid Crystal Display
MIPS	Milhões de Instruções Por Segundo
PWM	Pulse Width Modulation
PD	Pesquisa e Desenvolvimento
RISC	Reduced Instruction Set Computer
SRAM	Dynamic Random Access Memory
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
UFCG	Universidade Federal de Campina
USB	Universal Serial Bus
USART	Universal Synchronous Asynchronous Receiver Transmitter

Sumário

1 Introdução

Conforme previsto no Projeto Pedagógico do Curso de Engenharia Elétrica do Centro de Engenharia Elétrica e Informática (CEEI) da Universidade Federal de Campina Grande (UFCG) é obrigatório a elaboração, desenvolvimento, orientação, apresentação e avaliação da disciplina Estágio Integrado ou Estágio Supervionado para os alunos concluintes. O objetivo é proporcionar ao estudante uma experiência profissional, em que seja possível conhecer e executar atividades associadas a Engenharia Elétrica, consolidando assim os conhecimentos apreendidos durante o curso.

Com a realização do estágio obrigatório é esperado que o estudante se torne apto a desenvolver as competências esperadas ao término do curso e se familiarize com o setor de atuação. Além disso, o cumprimento da carga horária do estágio é um dos requisitos necessários para obtenção do grau de Bacharel em Ciências no domínio da Engenharia Eletrica.

Nesse contexto, neste trabalho estão descritas as principais atividades realizadas pela estudante Wislayne Dayanne Pereira da Silva durante o estágio superviosionado no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded), que foi realizado no período de 06 de agosto a 29 de novembro de 2019, com uma carga horária de 20 horas semanais, totalizando 330 horas, sob a orientação do professor Jaidilson Jó da Silva e a supervisão do professor Saulo Oliveira Dornellas Luiz.

O Laboratório de Sistemas Embarcados e Computação Pervasiva fica localizado na UFCG e o estágio foi realizado em um projeto de Pesquisa e Desenvolvimento (P&D) do laboratório, no qual o tema central é o monitoramento de baterias chumbo-ácido estacionárias. No projeto a estagiária integrou a equipe de modelo e desenvolveu atividades relacionadas com: a criação de modelos no *simulink*; desenvolvimento de *software* para sistemas embarcados; integração de subsistemas de *software* e integração de subsistemas de *software* e *hardware*.

Os demais Capítulos desse trabalho estão organizados da seguinte forma: no Capítulo 2 encontra-se a apresentação da estrutura do laboratório e do projeto em que o estágio foi realizado; no Capítulo 3 concentra-se a fundamentação teórica dos conceitos utilizados nas atividades; no Capítulo 4 são descritas as atividades desenvolvidas; e por fim, o Capítulo 5 apresenta as considerações finais do trabalho.

2 Apresentação do Ambiente de Trabalho

Neste capítulo serão apresentados o laboratório Embedded e a estrutura organizacional do projeto de pesquisa e desenvolvimento em que o estágio foi realizado.

2.1 O Embedded

O Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded) foi fundado em dezembro de 2005 e integra o grupo de laboratórios do Centro de Engenharia Elétrica e Informática (CEEI) da Universidade Federal de Campina Grande (UFCG), em Campina Grande, Paraíba. Na Figura 1 é apresentada uma fotografia da fachada do laboratório.

O Embedded é credenciado no Comitê da Área de Tecnologia de Informação (CATI) para receber recursos da Lei de Informática, tendo o Parque Tecnológico da Paraíba como interveniente financeiro também credenciado no CATI. Diante disso, possui um histórico de parcerias com grandes empresas, em diversos projetos relacionados com sua área de atuação.

Atualmente o Embedded possui linhas de atuação nas áreas de: Internet das Coisas (IoT), *Software-Hardware Co-Design* e Microeletrônica. A equipe do laboratório é formada por professores e pesquisadores do CEEI, e incluem alunos de doutorado, mestrado e graduação.

Figura 1 – Fotografia da Fachada do Laboratório Embedded

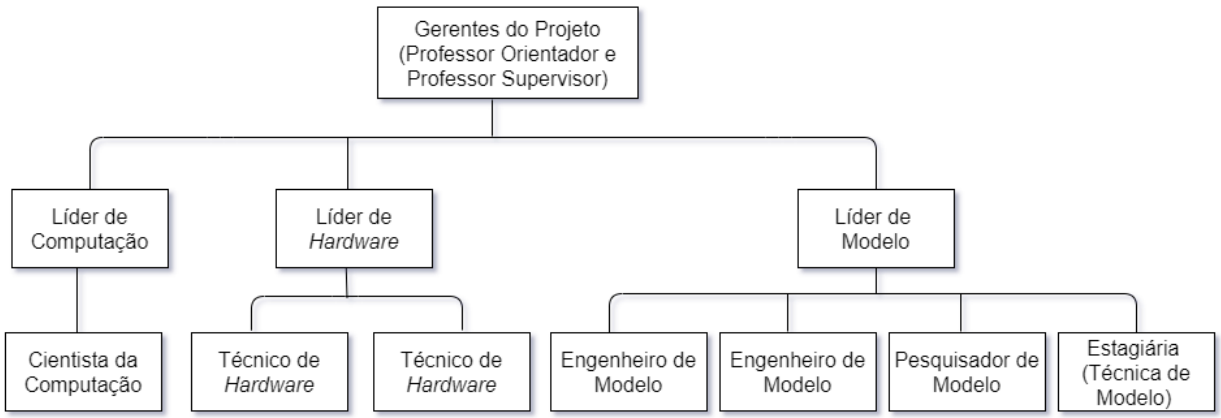
Fonte: embedded.ufcg.edu.br

A missão do Embedded é avançar no estado da arte nas áreas de sistemas embarcados e computação pervasiva, promovendo ações que permitam que tais avanços tragam benefícios para a sociedade por meio de parcerias com grandes empresas. Para isso, tem-se uma equipe formada por pesquisadores focados na produção de conhecimento e na aplicação deste conhecimento na resolução de problemas reais da indústria, equilibrando perspectivas acadêmicas com as necessidades de mercado [1].

2.2 Projeto de Pesquisa e Desenvolvimento

No contexto de atuação do Embedded, a estagiária realizou as atividades em um projeto de cooperação técnico e cientifíco entre o laboratório e uma empresa parceira. Nesse projeto são executadas atividades de pesquisa e desenvolvimento de *hardware* e *software*, incluindo investigação, definição de arquitetura, codificação, teste e implementação de soluções. Atualmente, a equipe do projeto é composta por 12 pessoas e organizadas de acordo com a estrutura hierárquica apresentada na Figura 2.

Figura 2 – Representação da Estrutura Hierárquica da Equipe do Projeto



Fonte: Autoria Própria

As tarefas e decisões do direcionamento do projeto eram elaboradas pelos líderes e repassadas aos outros membros. A estagiária respondia diretamente a Líder de Modelo, responsável por coordenar as atividades desenvolvidas pela equipe de modelo. Semanalmente ocorriam reuniões com o gerente do projeto e todos os desenvolvedores para acompanhamento das atividades executadas e direcionamento para novas atividades.

Neste Capítulo foram apresentados o ambiente de trabalho e a estrutura do projeto de coorperação técnico científico em que o estágio foi realizado. No Capítulo 3 serão abordados os principais conceitos necessários para o desenvolvimento das atividades.

3 Fundamentação Teórica

Neste capítulo serão apresentados os principais conceitos necessários para o desenvolvimento das atividades do estágio, no intuito de prover maior embasamento teórico ao que será explanado posteriormente. São aqui descritos as especificações dos microcontroladores utilizados, os conceitos e configurações necessárias para a geração automática de código com o MATLAB/Simulink e os conceitos acerca da comunicação serial.

3.1 Microcontrolador dsPIC 33EP512MU810 e Explorer 16/32

O microcontrolador dsPIC 33EP512MU810 (Figura 3), fabricado pela *Microchip Technology Inc*, é um microcontrolador integrado com um processador digital de sinais, concedendo-o sua característica de controlador digital de sinais. Seu núcleo, capaz de alcançar até 70 mega instruções por segundo (MIPS), trabalha com tensões elétricas entre 3,0 V e 3,6 V, em temperaturas entre -40 ^{o}C e +85 ^{o}C. Possui uma arquitetura de Harvard modificada, conjunto de instruções otimizadas em linguagem C, barramento de dados de 16 *bits*, entre outras características que o permitem obter alto desempenho e precisão quando utilizado em sistemas de controle e/ou de processamento de sinais [2].

Figura 3 – Fotografia do Microcontrolador dsPIC 33EP512MU810

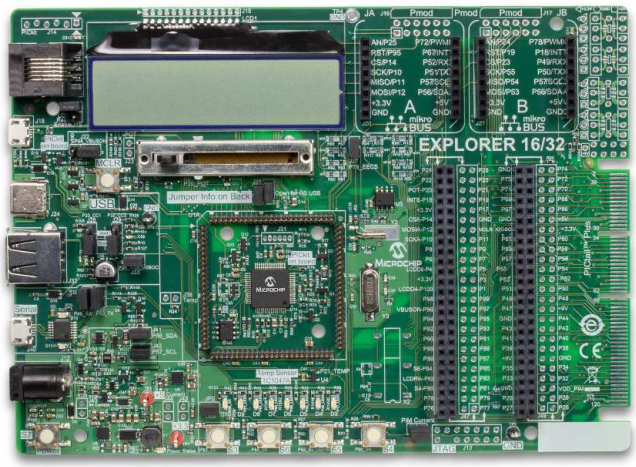
Fonte: Microchip Technology

Dentre seus periféricos, esse microcontrolador possui sete geradores de sinais PWM (*Pulse Width Modulation*), cada um com dois pinos de saída; uma interface de codificador em quadratura; dois módulos de conversores analógico-digital independentes, um configurável com resolução de 10 *bits*, ou 12 *bits*, com taxa de amostragem de até 500 mil amostras por segundo, o outro módulo trabalha com resolução de 10 *bits* e pode alcançar ate 1,1 milhões de amostras por segundo; 19 temporizadores com 16 *bits* de resolução; 15 canais de acesso direto à memó-

ria; dentre outras funcionalidades melhor detalhadas no *datasheet* do dispositivo fornecido pelo fabricante [2].

Para a integralização desse microcontrolador com o *hardware* desenvolvido no projeto, utilizou-se uma placa de desenvolvimento fabricada também pela Microchip. Esta placa é a Explorer 16/32 (Figura 4), uma plataforma flexível e conveniente para desenvolvimento, demonstração e testes. Nela está contido o *hardware* necessário para utilização de todas as funcionalidades do microcontrolador, bem como sua programação sem a necessidade de dispositivos adicionais. Esta placa pode ser alimentada via porta USB (*Universal Serial Bus*), com tensão de 5 V, ou via conector externo, com tensão maior que 8V e menor que 20 V. Ela alimenta o microcontrolador com tensão de 3,3 V e fornece um oscilador a cristal de 8 MHz para ser usado como relógio do microcontrolador.

Figura 4 – Fotografia da Placa de Desenvolvimento Explorer 16/32

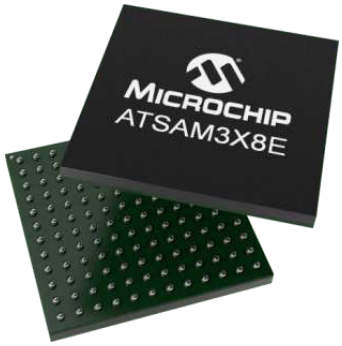


Fonte: Microchip Technology

3.2 Microcontrolador ATSAM3X8E e Arduino DUE

O Atmel SAM3X8E, conhecido como ATSAM3X8E, (Figura 5), é um microcontrolador baseado no processador de alto desempenho ARM Cortex -M3 RISC de 32 bits. Opera a uma velocidade máxima de 84 MHz e possui até 512 Kbytes de Flash e até 100 Kbytes de SRAM (*Dynamic Random Access Memory*). Possui um conjunto periféricos altamente integrado para conectividade e comunicação que inclui: USB 2.0 de alta velocidade, 4 USART (*Universal Synchronous Asynchronous Receiver Transmitter*), 4 SPI (*Serial Peripheral Interface*) e 2 I2C (*Inter-Integrated Circuit*), 8 geradores de PWM, Ethernet, um ADC/DAC (*Analog to Digital Convert* e *Digital to Analog Converter*) de 12 bits, sensor de temperatura, temporizadores de 32 bits, entre outros [3].

Figura 5 – Fotografia do Microcontrolador ATSAM3X8E



Fonte: Microchip Technology

Para a integralização desse microcontrolador com o *hardware* desenvolvido no projeto foi utilizada uma placa de desenvolvimento Arduino DUE (Figura 6), que é a primeira placa Arduino baseada em um microcontrolador de núcleo ARM de 32 bits. A placa possui: 54 pinos de entrada/saída digitais (dos quais 12 podem ser usados como saídas PWM); 12 entradas analógicas e 4 UARTs. A placa contém tudo o necessário para suportar o microcontrolador. Pode ser alimentada com um cabo micro-USB, com tensão de 3,3 V, ou via um conversor CA-CC ou bateria [3].

Figura 6 – Fotografia da Placa de Desenvolvimento Arduino Due

Fonte: https://www.arduino.cc/

3.3 Geração Automática de Código no MATLAB/Simulink

Os *Softwares* MATLAB e Simulink são excelentes ferramentas para otimizar processos de trabalho que envolvem geração de códigos nas linguagens C e C++. Atualmente, a geração de códigos a partir de modelos no Simulink é muito utilizada na indústria, uma vez que apresenta grandes ganhos nos processos de desenvolvimento. Para sistemas embarcados, a utilização desse método é muito vantajosa, uma vez que reduz a necessidade de conhecimentos especí-

ficos de programação. Todavia, existem casos em que os blocos do Simulink não oferecem suporte, sendo necessário o desenvolvimento das funções manualmente, é o caso da biblioteca da Microchip para Simulink.

Para o desenvolvimento desse trabalho, foi utilizada a geração de códido a partir de modelos no Simulink para embarcar nos microcontroladores dsPIC 33EP512MU810 e ATSAM3X8E. As configurações necessárias para cada caso são apresentadas a seguir.

3.3.1 Geração de Código para o dsPIC 33EP512MU810

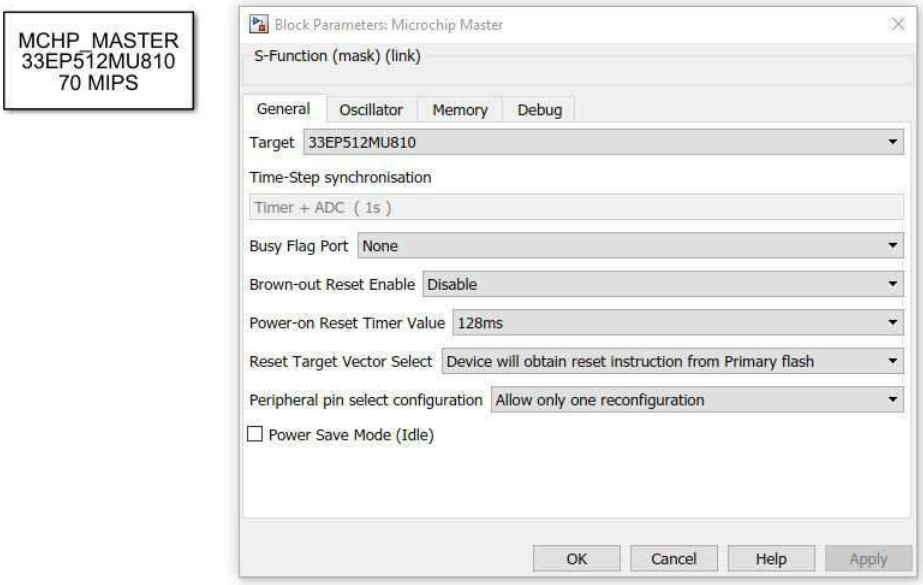
Para o desenvolvimento de algoritmos e geração automática de código em C no Simulink para dsPIC, é necessário a instalação de pacotes externos fornecidos pela Microchip. O primeiro deles é o *MPLAB X Integrated Development Environment* (IDE), um *software* existente para as plataformas Windows, Linux e Mac OS para o desenvolvimento de aplicações para microcontroladores da Microchip. É possível obter suporte de desenvolvimento com esta IDE para microcontroladores do tipo PIC, dsPIC, AVR, CEC e SAM.

O segundo pacote é o *MPLAB XC C Compilers*, o compilador necessário para interpretação dos códigos desenvolvidos para os microcontroladores da Microchip. Existem três versões disponíveis para uso, a MPLAB XC8 para PIC e AVR de 8 *bits*, a versão MPLAB XC16 para PIC e dsPIC de 16 *bits* e MPLAB XC32/32++ para PIC e SAM de 32 *bits*.

O terceiro pacote é o *MPLAB Device Blocks for Simulink*, uma biblioteca de blocos para desenvolvimento dos microcontroladores da Microchip em ambiente Simulink. As configurações básicas necessárias para criar o modelo no Simulink são feitas por meio dos blocos *Microchip Master* e *Compiler Options*, pertencentes a biblioteca instalada. Esses blocos permitem escolher o microcontrolador de destino, o oscilador, o compilador e as opções de saída, conforme apresentado nas Figuras 7 e 8.

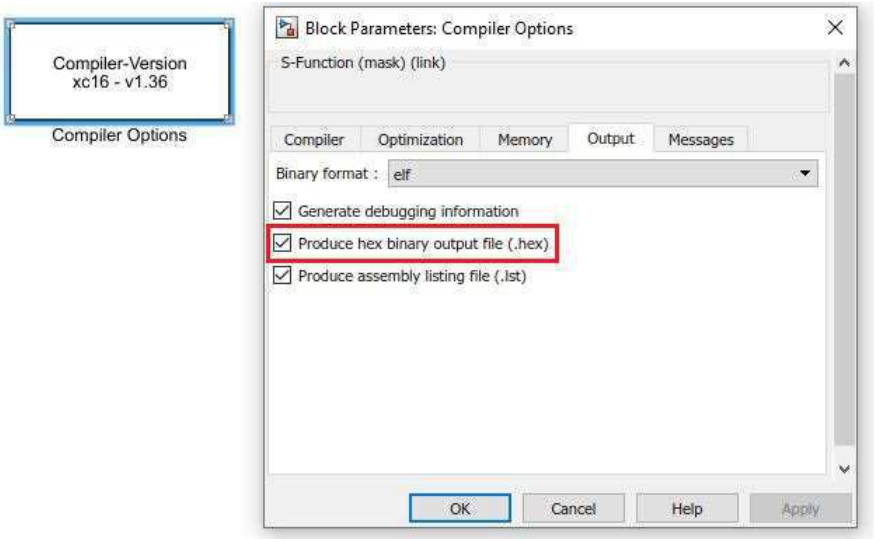
No bloco *Compiler Options* deve-se configurar a opção para produção do arquivo de saída binário .hex. Isso é feito na aba *Output* marcando a opção *Produce hex binary output file (.hex)*. Com isso, ao compilar o modelo será gerado o arquivo .hex que pode ser carregado diretamente no microcontrolador.

Figura 7 – Representação do Bloco *Microchip Master*



Fonte: Autoria Própria

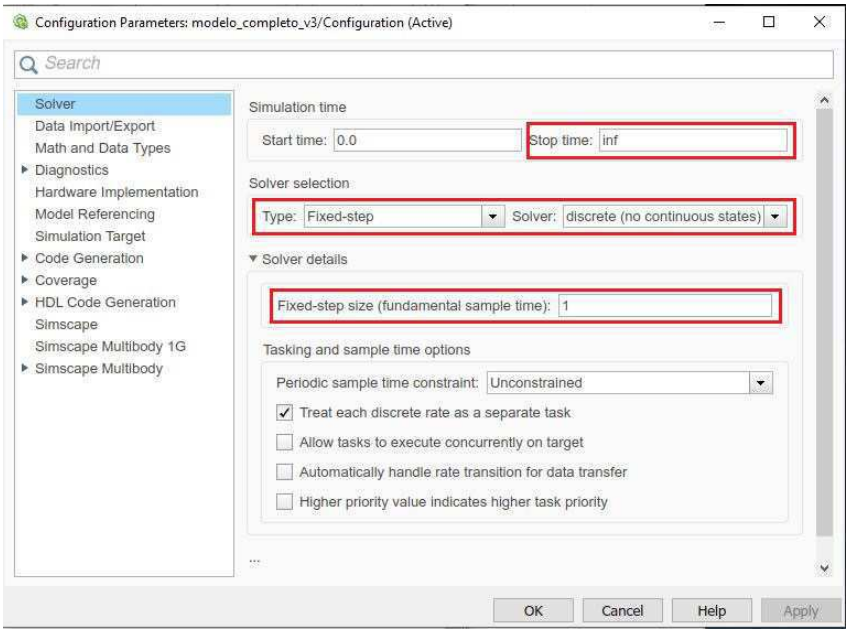
Figura 8 – Representação do Bloco *Compiler Options*



Fonte: Autoria Própria

O código C/C++ para a aplicação é gerado de forma automática a partir do modelo no Simulink. O código é automaticamente compilado e um arquivo binário é produzido para então ser carregado no microcontrolador de destino. As configurações necessárias no próprio Simulink são feitas na opção *Model Configuration Parameters* presente na barra de ferramentas do Simulink. Com o ambiente de configuração aberto no menu *Solver* deve-se fixar o *Stop time* para infinito; o *Solver selector* para discreto, sem estados contínuos; e *Sample time* para 1 segundo, como apresentado na Figura 9.

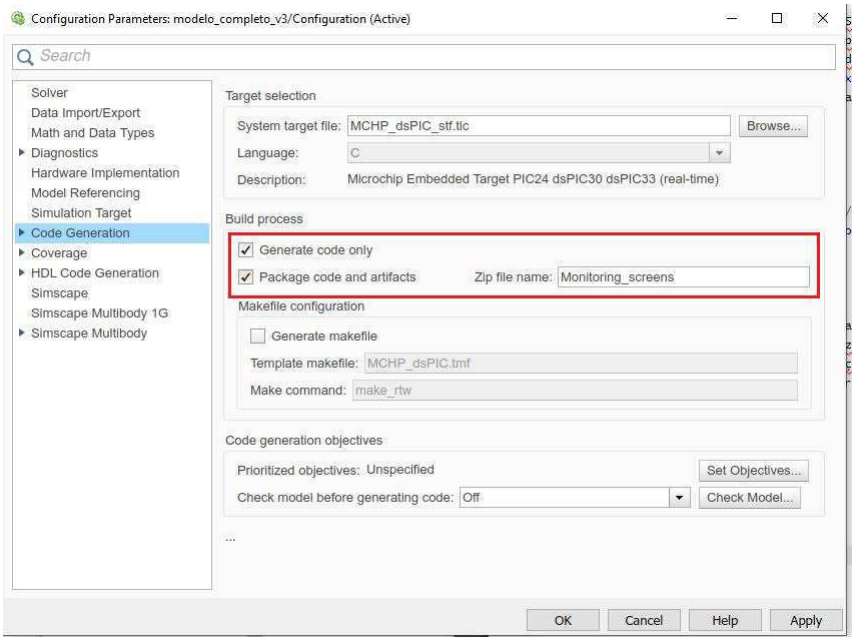
Figura 9 – Representação da Configuração dos Parâmetros Gerais do Modelo



Fonte: Autoria Própria

É necessário ainda configurar as opções para o *packNGo*. Essas configurações são realizadas no menu *Code Generation*. Para isso deve-se marcar a opção de *Package code and artifacts* e nomear o arquivo. Assim é habilitada a opção de baixar o código em C gerado em um arquivo .zip, como apresentado na Figura 10.

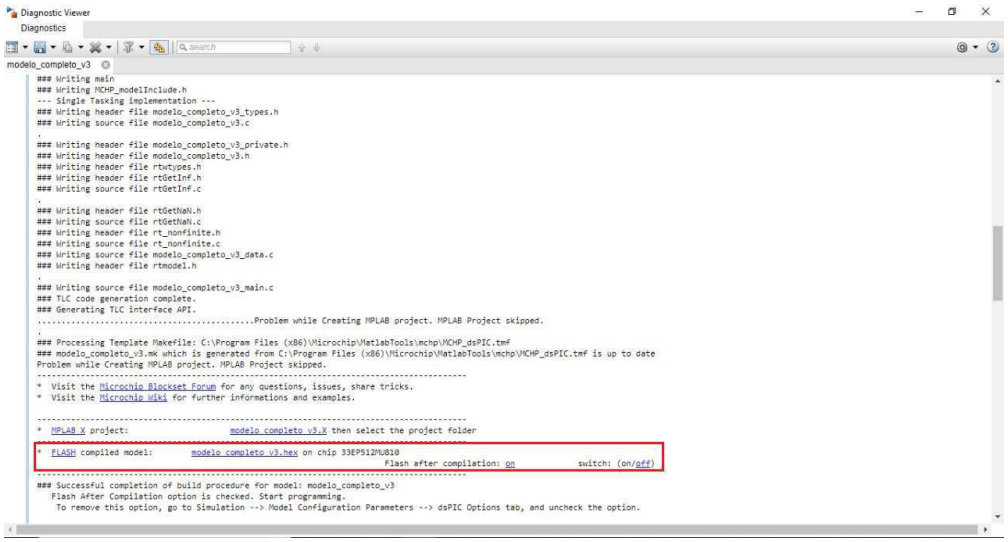
Figura 10 – Representação da Configuração da Geração Automática de Código (dsPIC)



Fonte: Autoria Própria

Ao compilar o modelo, será gerado o arquivo .hex que pode ser carregado diretamente no microcontrolador via Simulink, bastando apenas clicar no *link* do arquivo .hex, como é mostrado na Figura 11.

Figura 11 – Representação do Carregamento do Modelo no Microcontrolador



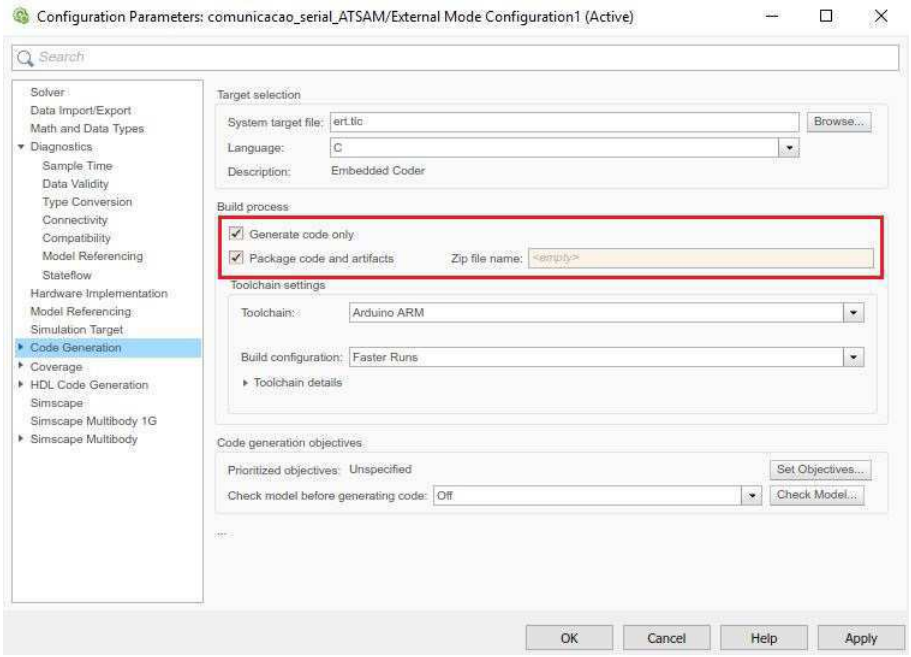
Fonte: Autoria Própria

3.3.2 Geração de Código para o ATSAM3X8E

Para geração automática de código C no Simulink para o Arduino é necessário instalar a biblioteca *Simulink Support Package for Arduino Hardware*. A instalação é feita no próprio MATLAB por meio da opção *Get Hardware Support Packages*. Depois da instalação é possível ter acesso e fazer a comunicação do *software* com a plataforma por meio dos blocos encontrados na biblioteca instalada.

A configuração do processo de geração automática de código é realizada na janela de configurações dos parâmetros, que é encontrada no ambiente principal do Simulink, na aba *Simulation - Model Configuration Parameters - Code Generation Options*, conforme ilustrado na Figura 12.

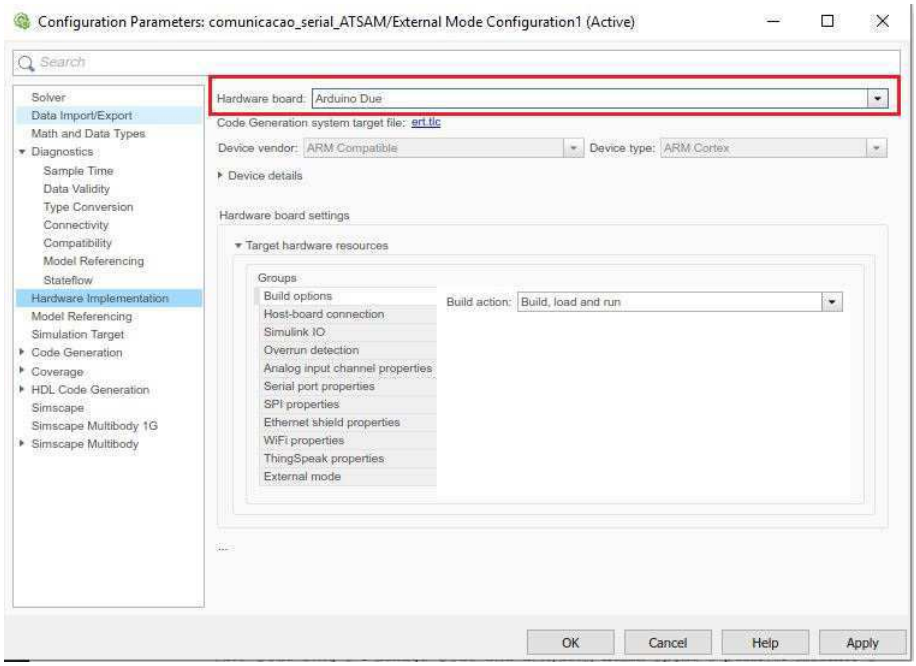
Figura 12 – Representação da Configuração da Geração Automática de Código (Arduino)



Fonte: Autoria Própria

Por fim, é necessário informar ao *software* a plataforma que será utilizada, isso é feito no menu *Hardware Implemetation*, conforme apresentado na Figura 13.

Figura 13 – Representação da Configuração do *Hardware*



Fonte: Autoria Própria

3.3.3 Comunicação Serial

A comunicação serial é um recurso utilizado para transmissão e recepção de dados digitais. Existem duas formas básicas de comunicação serial: síncrona e assíncrona. A transmissão serial síncrona requer que o emissor e o receptor compartilhem um *clock* entre si, ou que o remetente forneça um sinal de tempo para que o receptor saiba quando deve ler o próximo *bit* dos dados. Já a transmissão serial assíncrona permite que os dados sejam transmitidos sem que o emissor tenha que enviar um sinal de relógio ao receptor. Em vez disso, o remetente e o receptor devem concordar com os parâmetros de tempo de antecedência e *bits* especiais são adicionados a cada palavra, os quais são usados para sincronizar as unidades de envio e recebimento.

Um dispositivo de *hardware* utilizado para realizar a comunicação serial assíncrona é o UART, acrônomo para *Universal Asynchrounous Receiver/Transmiter* e em português Receptor/Transmissor Universal Assíncrono. Os UARTs convertem os dados paralelos para a forma serial, para possibilitar sua transmissão e vice versa. Seu funcionamento se dá da seguinte forma, a UART de origem pega *bytes* de dados e transmite os *bits* individuais de forma sequencial. No destino, uma segunda UART recebe os *bits* e os reúne em *bytes* completos para formar a informação.

Na realização do estágio se fez necessário utilizar a comunicação serial assíncrona para realizar transmissão e recepção de dados entre os microcontroladores presentes no sistema de *hardware* desenvolvido no projeto. Para isso, foram desenvolvidas UARTs virtuais para o microcontrolador dsPIC 33EP512MU810, tendo em vista que a biblioteca da Microchip para Simulink não oferece blocos de comunicação serial confiáveis e eficientes.

Neste Capítulo foi apresentada uma fundamentação teórica sobre os microcontroladores dsPIC 33EP512MU810 e ATSAM3X8E e suas respectivas placas de desenvolvimento; os conceitos e configurações acerca da geração de código automática no MATLAB/Simulink; e uma introdução sobre comunicação serial. Os conceitos apresentados foram utilizados no desenvolvimento das atividades do estágio, que serão apresentadas no Capítulo 4.

4 Atividade Desenvolvidas

Neste capítulo serão apresentadas as principais atividades desenvolvidas durante o estágio, que foram: desenvolvimento de funções em C para uma UART virtual; desenvolvimento de modelos para testes de validação de comunicação serial e implementação de uma estrutura de exibição de informações em um LCD (*Liquid Crystal Display*). Todas as atividades foram implementadas no ambiente MATLAB/Simulink utilizando a ferramenta de geração de código automática e embarcadas nos microcontroladores dsPIC 33EP512MU810 ou ATSAM3X8E.

4.1 Desenvolvimento de uma UART Virtual

A biblioteca *MPLAB Device Blocks for Simulink* fornecido pela Microchip fornece um conjunto de blocos para periféricos, que incluem blocos de configuração, transmissão e recepção de dados via serial. Todavia a partir de testes de validação foi percebido que as configurações permitidas pelos blocos de desenvolvimento não eram suficientes para atender os requisitos da aplicação. Além disso, para o projeto são utilizadas três UARTs e o *hardware* da placa Explorer 16/32 apresentava muita perda de dados e falhas de comunicação ao utilizar mais de uma UART simultameamente.

Como solução, foi adaptado um código em C com funções específicas para uma UART implementada via *software*. UARTs implementadas via *software* são denominadas UARTs virtuais e são muito utilizadas quando é preciso estabelecer comunicação em dispositivo que não possuem periféricos UART ou como no caso do projeto, as UARTs do dispositivo apresentam falhas em determinadas situações de interesse.

Com UARTs virtuais, é possível utilizar a classe de dispositivo de comunicação USB para fazer um dispositivo USB parecer uma conexão RS-232. Dessa forma, o dispositivo de destino interage com esse tipo de UART como qualquer outra, transmitindo e recebendo dados atráves da porta serial. A diferença é que a UART virtual ocupa parte do espaço de memória da aplicação.

O *software* UART implementado possui três funções principais, que são: *InitSoftUART*, *UART_Write*, *UART_Receive*. A função *InitSoftUART* inicializa os pinos de TX e RX da UART de acordo com a necessidade da comunicação, conforme ilustrado na Figura 14.

Figura 14 – Representação da Função para Inicialização da UART

```
void InitSoftUART(void)
{
    UART_TX = 0;
    __delay_ms(1000);
    UART_TX = 1;

    UART_RX_DIR = 1;
    UART_TX_DIR = 0;
}
```

Fonte: Autoria Própria

Na *UART_Write* (Figura 15) é implementada a lógica para a transmissão de um dado do tipo *char* sem sinal. Inicialmente é enviado um *bit* em nível lógico baixo, em seguida é feita uma verificação em todos *bits* do *byte* a ser transmitido, por meio de uma estrutura de repetição do tipo *for*. Se o *bit* estiver em nível lógico alto, o pino correspondente a TX é setado em 1 (um), caso contrário é setado em 0 (zero). Após a transmissão, é enviado um *bit* de parada em nível lógico alto.

Figura 15 – Representação da Função para Transmissão

```
void UART_Write(unsigned char DataValue)
{

    UART_TX = 0;
    __delay_us(OneBitDelay);

    unsigned int i = 0;

    for (i = 0; i < DataBitCount; i++ )
    {
        if( ((DataValue>>i)&0x1) == 0x1 )
        {
            UART_TX = 1;
        }
        else
        {
            UART_TX = 0;
        }
        __delay_us(OneBitDelay);

    }

    UART_TX = 1;
    __delay_us(OneBitDelay);

}
```

Fonte: Autoria Própria

A função *UART_Receive* corresponde a função de recepção serial, que possibilita o envio de um dado do tipo *char* sem sinal. A lógica é semelhante a da função *UART_Write*. É verificado o estado do pino de RX para todo o *byte*, e se o pino estiver em nível lógico alto, é deslocada uma possível para esquerda da variável de armazena o *byte* recebido. Isso é realizado até que todo o *byte* seja preenchido e a função retorna esse valor, conforme descrito na Figura 16.

Figura 16 – Representação da Função para Recepção

```
unsigned char UART_Receive(void)
{
    unsigned char DataValue = 0;

    __delay_us(OneBitDelay);
   __delay_us(OneBitDelay/2);

    unsigned int i;

    for (i = 0; i < DataBitCount; i++ )
    {
        if ( UART_RX == 1 )
        {
            DataValue += (1<<i);
        }
        __delay_us(OneBitDelay);
    }

    if ( UART_RX == 1 )
    {
        __delay_us(OneBitDelay/2);
        return DataValue;
    }
    else
    {
        __delay_us(OneBitDelay/2);
        return '\0';
    }

}
```

Fonte: Autoria Própria

O código foi implementado no ambiente MATLAB e foi aplicado para a criação de três UARTs, utilizadas para comunicação serial entre o microcontrolador dsPIC 33EP512MU810 e outros três microcontroladores, conforme apresentado na Tabela 1.

UART	Microcontrolador de Origem	Microcontrolador de Destino
UART 1	dsPIC 33EP512MU810	ESP 8266
UART 2	dsPIC 33EP512MU810	Raspberry Pi
UART 3	dsPIC 33EP512MU810	PIC 18F25k80

Tabela 1 – UARTs Virtuais Utilizadas

Os pinos da placa de desenvolvimento Explorer 16/32 utilizados como RX e TX das três UARTs são descritos na Tabela 2.

Pinos	UART 1	UART 2	UART 3
RX	52	39	49
TX	53	40	50

Tabela 2 – Pinos Configurados para RX e TX

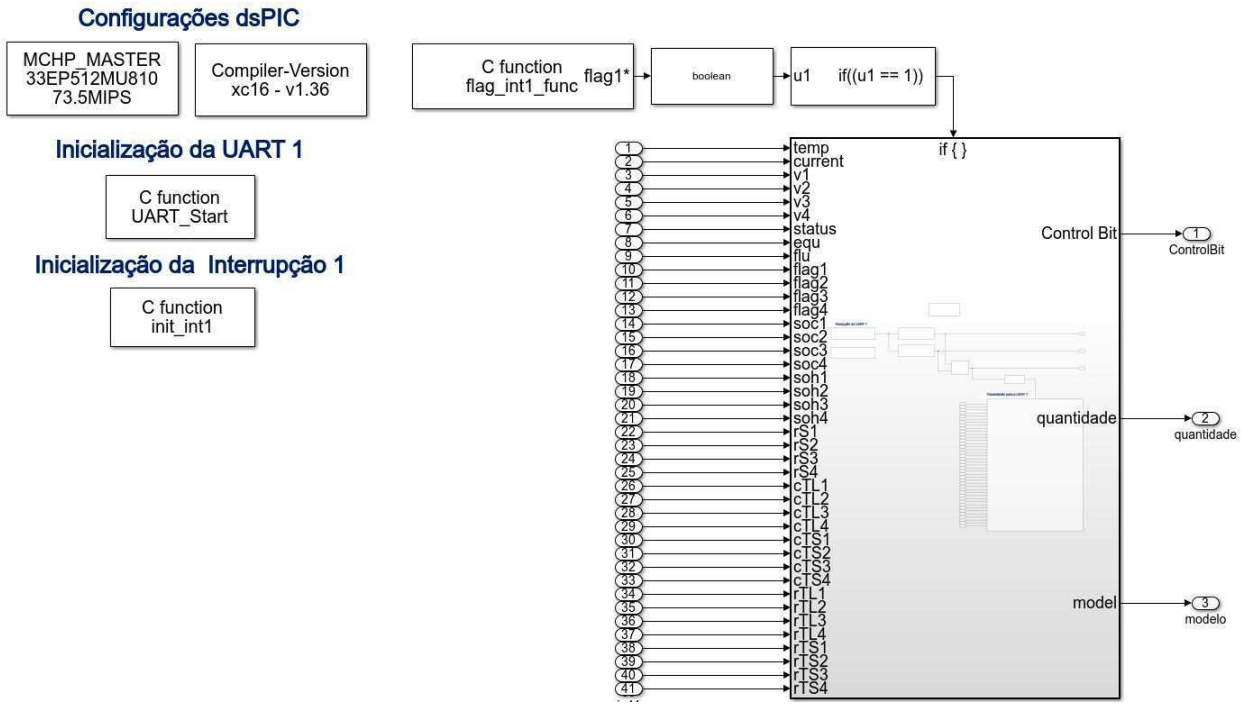
4.2 Modelos para Comunicação Serial Desenvolvidas para o dsPIC

Foram implementados modelos no Simulink para testes de validação da comunicação serial entre o microcontrolador dsPIC e outros três microcontroladores, que são: ESP 8266, Raspberry Pi, PIC 18F25k80. Para o desenvolvimento dos modelos foi utilizada o código da UART virtual abordada na seção anterior e os códigos gerados foram embarcados no dsPIC.

4.2.1 Modelo para Teste de Comunicação entre os Microcontroladores dsPIC 33EP512MU810 e ESP 8266

Para a aplicação do projeto, o microcontrolador dsPIC se comunica via serial com um microcontrolador ESP 8266, recebendo e transmitindo dados. Para validar essa comunicação de recepção e transmissão, foi implementado no Simulink, utilizando a UART virtual descrita na seção anterior, um modelo de teste de comunicação. A UART destinada a essa comunicação é a UART1. O diagrama em blocos do modelo desenvolvido é apresentado na Figura 17. Nesse modelo, são feitas as configurações padrão do microcontrolador e implementada a lógica do teste.

Figura 17 – Diagrama em Blocos do Modelo para Teste de Comunicação entre o dsPIC e o ESP

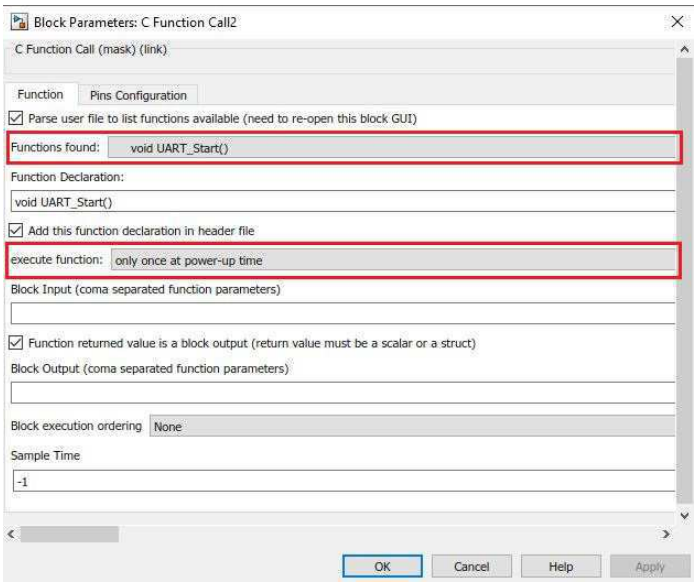


Fonte: Autoria Própria

As configurações do dsPIC são feitas utilizando os blocos *Microchip Master* e *Compiler Options*. É necessário também inicializar a UART Virtual antes de realizar qualquer comunicação. Para isso faz-se uso do bloco *C Function Call*, em que é possível selecionar a função

UART_Start que está contida em um arquivo C localizado na mesma pasta do modelo. Essa função apenas realiza a chamada da função *InitSoftUART* descrita na seção anterior. Como se trata se uma função de inicialização, deve-se selecionar a opção de execução "apenas uma vez no momento da inicialização", conforme ilustrado na Figura 18.

Figura 18 – Representação da Chamada da Função de Inicialização da UART



Fonte: Autoria Própria

Da mesma forma é realizada a chamada da função de inicialização da interrupção externa 1. Essa interrupção se refere a interrupção programada para a recepção de dados via a UART1. Na Figura 19 é apresenta a função de iniciação, na qual são configurados os registradores de *enable*, do pino a ser monitorado, de prioridade e de borda de subida.

Figura 19 – Representação da Função de Inicialização da Interrupção Externa 1

```
// ----- Inicialização da Interrupção Externa - UART1_RX (Recepção ESP)  ------
void init_int1()
{
    IEC1bits.INT1IE = 1; // registrador referente a enable
    RPINR0bits.INT1R = 0x0062; // registrador para o pino 52, RP98(RF2)
    IPC5bits.INT1IP = 2; //registrador de prioridade
    INTCON2bits.INT1EP = 1; // registrador referente a borda negative edge ou positive edge

}
```

Fonte: Autoria Própria

A interrupção acontece quando algum dado é recebido pelo pino RX. O dado é recebido utilizando a função de recepção implementada para a UART virtual, é armazenado numa variável global e a *flag* da interrupção é setada para 1 (um), conforme descrito na Figura 20.

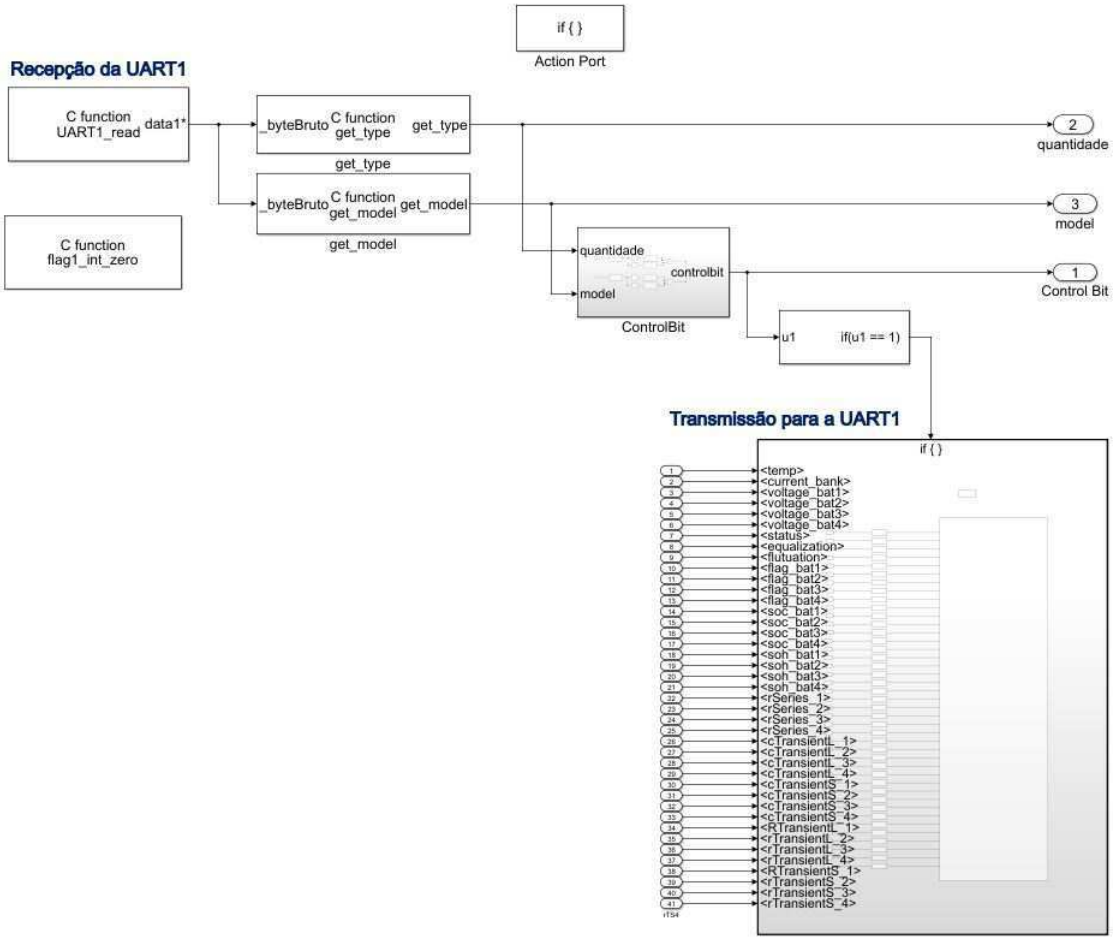
Figura 20 – Representação da Função da Interrupção Externa 1

```
/*----- Interrupção Externa 1 EX_INT0 - INT0 - (UART1_RX) ------------
  Recepção do Modelo e Quantidade de Baterias do ESP (UART1_RX)
*/
void __attribute__ ( ( interrupt, no_auto_psv ) ) _INT1Interrupt(void)
{

    data_UART1 = UART1_Receive();
    flag1_int = 1;

    EX_INT1_CallBack();
    IFS1bits.INT1IF = 0;

}
```

Fonte: Autoria Própria

Dessa forma, no modelo para o teste de comunicação, a *flag* da interrupção da função de recepção serial é verificada continuamente por um bloco de *if*. Caso assuma o valor lógico 1, significa que um dado foi recebido na serial, assim o modelo dentro do bloco de ação do *if* é executado. O bloco de ação possui o diagrama de blocos apresentado na Figura 21.

Figura 21 – Diagrama em Blocos do Modelo de Recepção e Transmissão da UART1



Fonte: Autoria Própria

Nesse modelo de teste, o dsPIC recebe na UART1 um *byte* no formato *uint8*, equivalente a um *char* sem sinal em C. Os quatro primeiros *bits* correspondem a quantidade de baterias e os últimos quatro *bits* correspondem a um modelo de baterias . De acordo com o valor recebido é verificado se as informações são válidas, caso positivo uma variável nomeada como *Control Bit* retornará o valor 1 (um). Quando *Control Bit* é igual a 1, é habilitada a transmissão do dsPIC para o ESP.

Como pode-se observar são utilizados quatro blocos *C Function Call*. Um se refere a chamada da função *UART*1*_read*, que é uma função que armazena o valor recebido na serial em uma variável do tipo *char* e retorna esse valor. Os dois outros blocos realizam as chamadas das funções *get_type* e *get_model*, ambas recebem o valor retornado pela *UART1_read* e realizam a quebra dos quatro *bits* mais significativos e dos quatro menos significativos, respectivamente, conforme descrito na Figura 22. O retorno da primeira corresponde a quantidade de baterias e da última se refere ao modelo. A função *flag1_int_zero* zera a *flag* da interrupção da UART1.

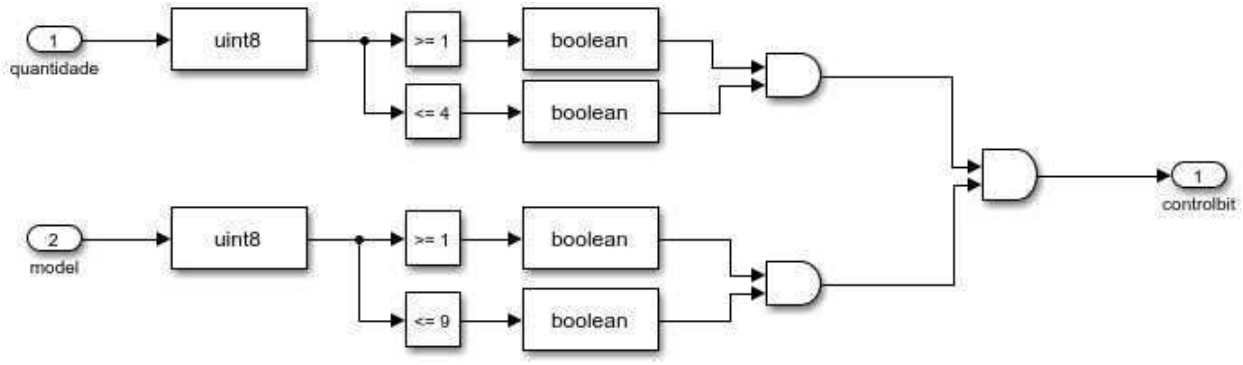
Figura 22 – Representação das Funções para Separar o *Byte* Recebido

```
 // --- Quatro primeiros bits do byte recebido (quantidade de baterias) ----
unsigned char get_type(unsigned char _byteBruto)
{
   unsigned char type = (unsigned char)(_byteBruto&0xf0)/16;
   return type;

}
 // ------- Quatro últimos bits do byte recebido (modelo) ---------
unsigned char get_model(unsigned char _byteBruto)
{
    unsigned char model = (unsigned char)(_byteBruto)&0xf;
    if(model!=15){
        UART3_Write(model);
        lastvalue = model;
    }

   modelo_recebido = model;
   return model;
}
```

Fonte: Autoria Própria

Com os valores do modelo e quantidade de baterias é verificado se essas informações são válidas. A quantidade de bateriais possíveis para compor um banco de baterias é de 1 a 4, e os modelos disponíveis são de 1 a 9. Na Figura 23 é apresentado o diagrama de blocos do subsistema *ControlBit*.

Figura 23 – Diagrama em Blocos do Subsistema *ControlBit*

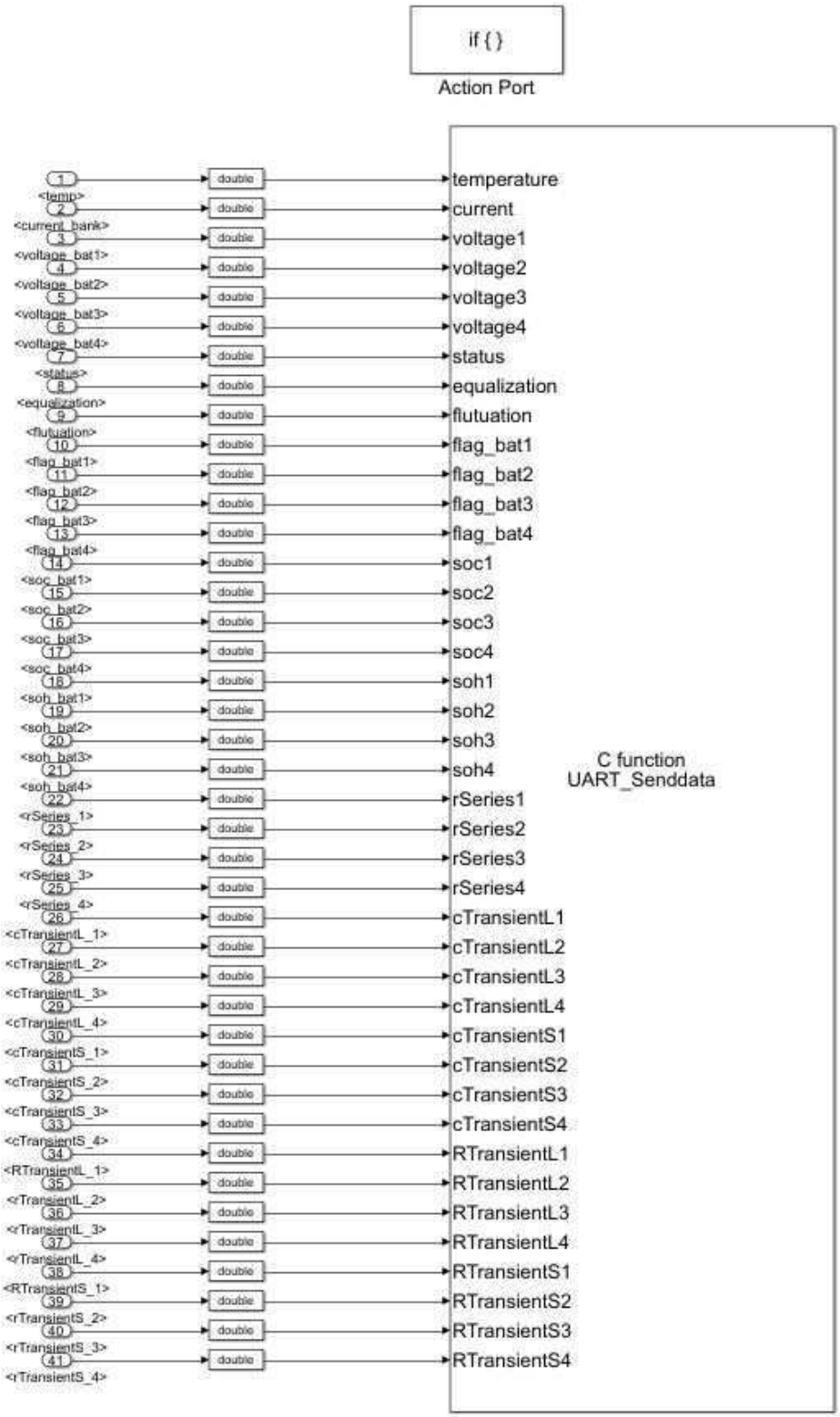


Fonte: Autoria Própria

O subsistema de transmissão recebe 41 variáveis de entrada e transmite para o microcontrolador de destino. Os valores de entradas correspondes a parâmetros de *status*, parâmetros elétricos e estimação de estado de saúde e carga das baterias. Na Figura 24 é apresentado o diagrama de blocos do subsistema de transmissão.

Como é observado é utilizado um bloco *C Function Call* para chamar a função *UART_Senddata*, que recebe as variáveis de entrada como parâmetros, as coloca em um *buffer* utilizando a função em C *sprintf* e as transmite utilizando a função *UART_Write*.

Figura 24 – Diagrama em Blocos da Função de Transmissão para a UART1

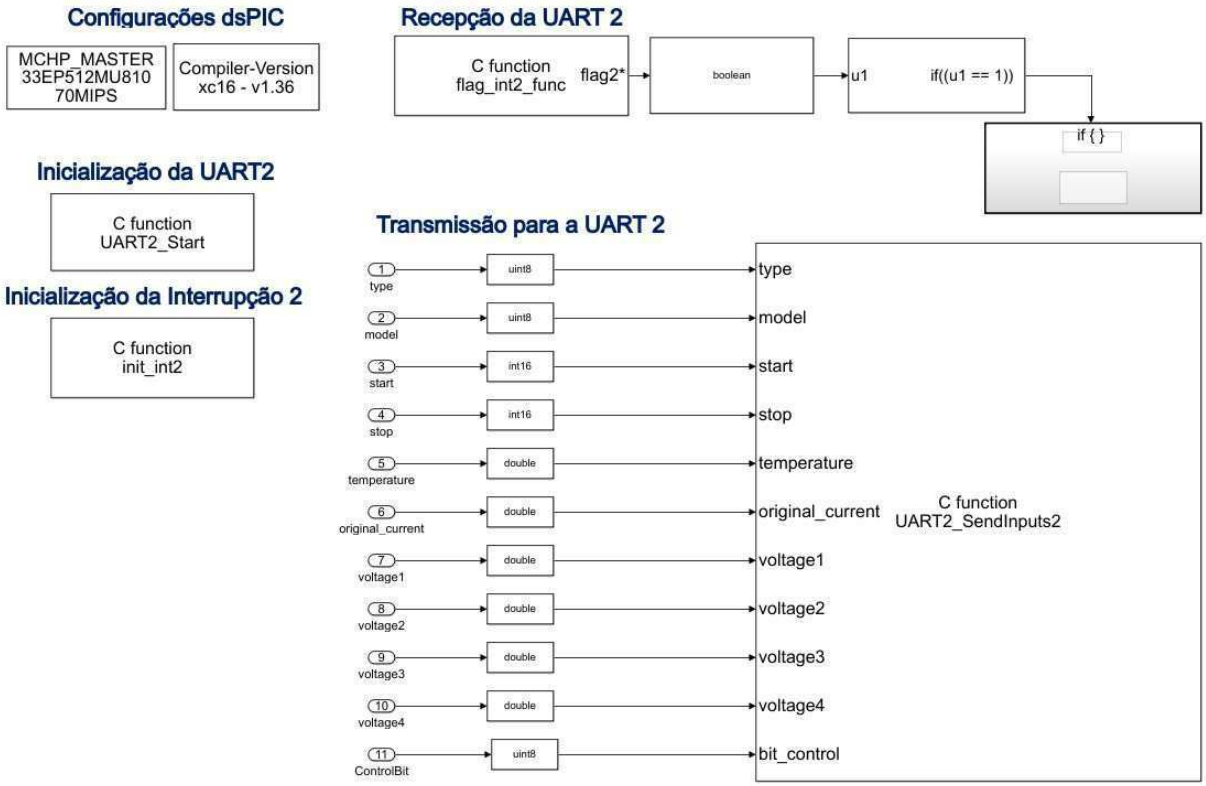


Fonte: Autoria Própria

4.2.2 Modelo para Teste de Comunicação entre os Microcontroladores dsPIC 33EP512MU810 e Raspberry Pi

Para a comunicação do dsPIC com o microcontrolador Raspeberry Pi foi utilizada a UART2. Sendo necessário realizar o mesmo procedimento de inicialização da UART e da interrupção descrito para a UART1. O dsPIC recebe da UART2 um *array* de *char* de 250 posições, correspondentes as informações de estimação de estado de carga, estado de saúde e parâmetros elétricos de baterias. Em contrapartida, transmite um conjunto de informações referentes a modelo e quantidade de baterias, entradas analógicas e de entradas digitais. Na Figura 25 é apresentado o diagrama em blocos do modelo implementado para validar essa comunicação.

Figura 25 – Diagrama em Blocos do Modelo para Teste de Comunicação entre os Microcontroladores dsPIC e Raspberry Pi



Fonte: Autoria Própria

Na parte do modelo referente a recepção é realizada a verificação contínua da *flag* da interrupção dois, essa *flag* é atualizada para 1 (um) sempre que chega um dado na UART2. Caso a UART2 receba alguma informação, o modelo no interior do bloco de ação do *if* é executado. Nesse caso, o modelo corresponde apenas a chamada de um fução que seta a *flag* para zero.

O *array* recebido é formatado para iniciar com o caractere "<"e terminar com o caracter ">". Dessa forma, na função da interrupção 2, é realizada a verificação de todo o *array* para

armazenar cada *char* recebido na sua posição correspondente, conforme ilustrado na Figura 26.

Figura 26 – Representação da Função da Interrupção Externa 2

```
void __attribute__ ( ( interrupt, no_auto_psv ) ) _INT2Interrupt(void)
{

    unsigned char data_UART2[250];
    int i_flag = 0;

    data_UART2[i_flag] = UART2_Receive();

    if(data_UART2[i_flag] == '>')
    {
            flag2_int = 1;
            i_flag = 0;
    }

    i_flag += 1;
    EX_INT2_CallBack();
    IFS1bits.INT2IF = 0;

}
```

Fonte: Autoria Própria

Ao que refere a transmissão de dados, a função *UART2_SendInputs2* recebe um conjundo de informações, as armazena em um *buffer* por meio da função *sprintf* e as transmite utilizando a função de escrita da UART virtual, conforme descrito na Figura 27.

Figura 27 – Representação da Função de Transmissão para a UART2

```
void UART2_SendInputs2(unsigned char type, unsigned char model, int start,
                       int stop, double temperature, double original_current,
                       double voltage1, double voltage2, double voltage3,
                       double voltage4, unsigned char bit_control){

    int i;
    char bufferTX[1000];
    sprintf(bufferTX, "< %d %d %d %d %9.2f %9.2f %9.2f %9.2f %9.2f %9.2f %d >",
                      type, model, start, stop, temperature, original_current,
                      voltage1, voltage2, voltage3, voltage4, bit_control);

    for(i=0; bufferTX[i]!='\0'; i++){
        UART2_Write(bufferTX[i]);
    }
    __delay_ms(1000);

}
```

Fonte: Autoria Própria

4.2.3 Modelo pata Teste de Comunicação entre os Microcontroladores dsPIC 33EP512MU810 e PIC 18F225k80

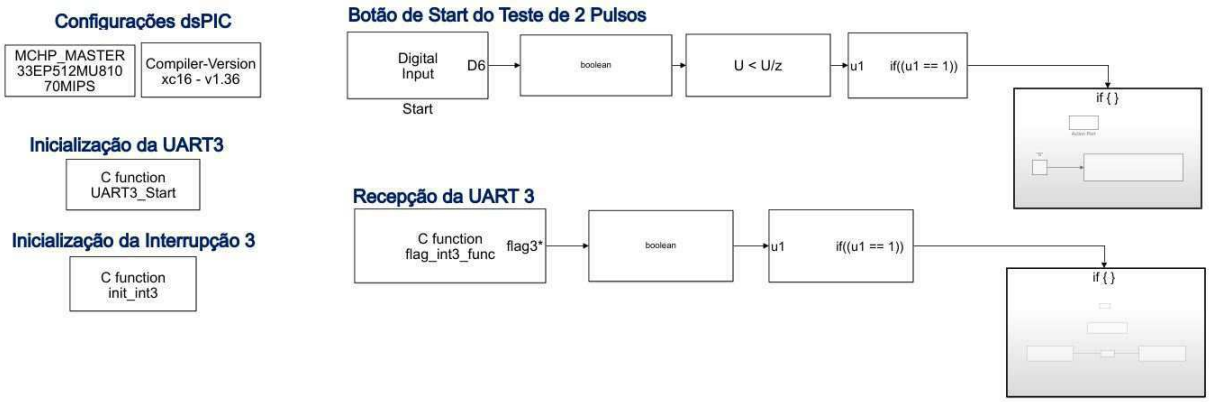
O dsPIC realiza troca de mensagens de comando com o microcontolador PIC 18F225k80. Para realizar essa comunicação foi utilizada a UART3. As mensagens recebidas na UART3 estão descritas na Tabela 3.

Palavra Recebida	Significado
M	Modelo recebido com sucesso
m	Modelo não foi recebido
F	Teste de dois Pulsos Finalizado
E	Erro
R	Retransmitir o último comando

Tabela 3 – Palavras Recebidas na UART3

O dsPIC transmite pela UART3 o modelo de baterias, o comando de iniciar o teste de dois pulsos e respostas para as palavras recebidas. Na Figura 28 é apresentado o diagrama em blocos do modelo implementado para o teste de validação dessa comunicação.

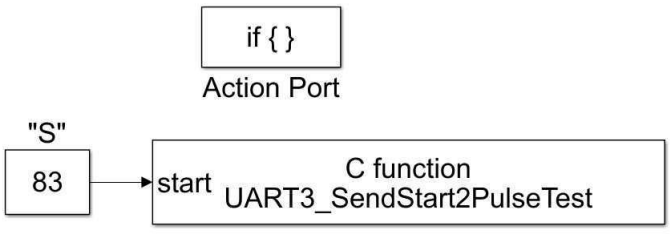
Figura 28 – Diagrama em Blocos do Modelo para Teste de Comunicação entre os Microcontroladores dsPIC e PIC 18F225k80



Fonte: Autoria Própria

São realizados os mesmos procedimentos para a configuração do dsPIC e as inicializações da UART3 e da interrupção externa 3, referente a recepção. Nesse modelo, é feita a verificação da entrada digital correspondente ao botão de *start* do teste de dois pulsos. Caso o botão esteja em nível lógico alto, é transmitido pela UART 3 o caractere "S", sinalizando que o teste deve ser iniciado (Figura 29). A função *UART3_SendStart2PulseTest* transmite o valor recebido como parâmetro por meio da função de escrita da serial virtual.

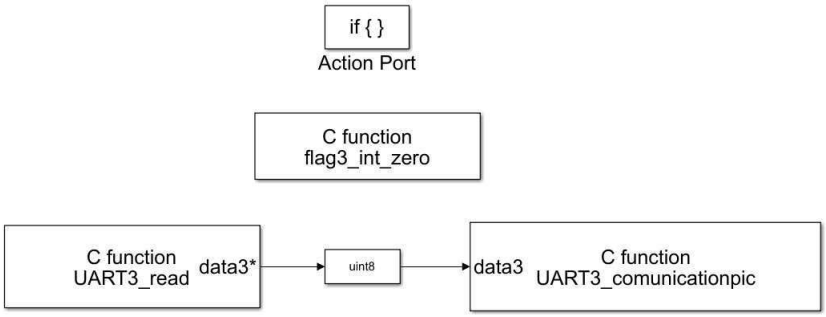
Figura 29 – Representação do Envio do Comando de *Start* do Teste de Dois Pulsos



Fonte: Autoria Própria

Nesse modelo é verificado também o valor da *flag* da interrupção três, essa *flag* é atualizada para 1 (um) sempre que chega um dado na UART3. Caso a UART3 receba alguma informação, o modelo no interior do bloco de ação do *if* é executado, esse modelo é apresentado na Figura 30. Conforme observado, o valor recebido na UART 3 é repassado como entrada da função *UART3_comunicationpic*.

Figura 30 – Representação do Tratamento do Valor Recebido



Fonte: Autoria Própria

A função *UART3_comunicationpic* verifica qual foi o valor recebido e transmite a resposta correspondente. Caso o dado recebido seja "m", significa que o microcontrolador PIC não recebeu o modelo de baterias, sendo assim, é transmitido utilizando a função de escrita da UART. Caso receba "E", significa que aconteceu algum erro, nesse caso é transmitido com o resposta o caractere "e". Caso seja "F", significa que o teste de dois pulsos foi finalizado, a resposta correspondente é "f"e caso seja "R", o último comando enviado é retransmitido, conforme descrito na Figura 31.

Figura 31 – Representação da Função *UART_comunicationspic*

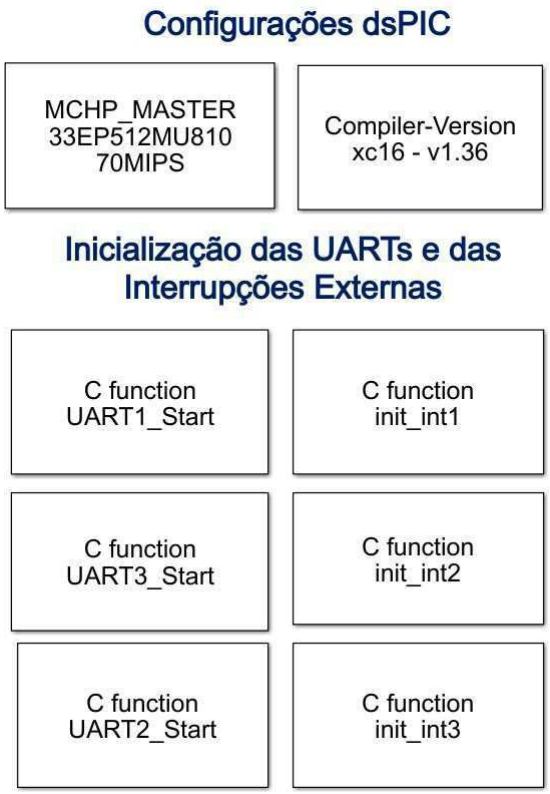
```
void UART3_comunicationpic(unsigned char data3)
{

   if(data3 == 'm'){
       UART3_Write(modelo_recebido);
       lastvalue = modelo_recebido;
   }

   else if (data3 == 'E'){
       UART3_Write('e');
       lastvalue = 'e';
   }

   else if (data3 == 'F'){
       UART3_Write('f');
       lastvalue = 'f';
   }

   else if (data3 == 'R'){
        __delay_ms(500);
       UART3_Write(lastvalue);
    }

}
```

Fonte: Autoria Própria

4.2.4 Integração dos Modelos de Comunicação Serial

Após validar todas as comunicações separadamente, foi implementado um modelo de integração, em que as três UARTs funcionam simultanemente recebendo e transmitindo dados. Na Figura 32 é apresentado as configurações do modelo de integração.

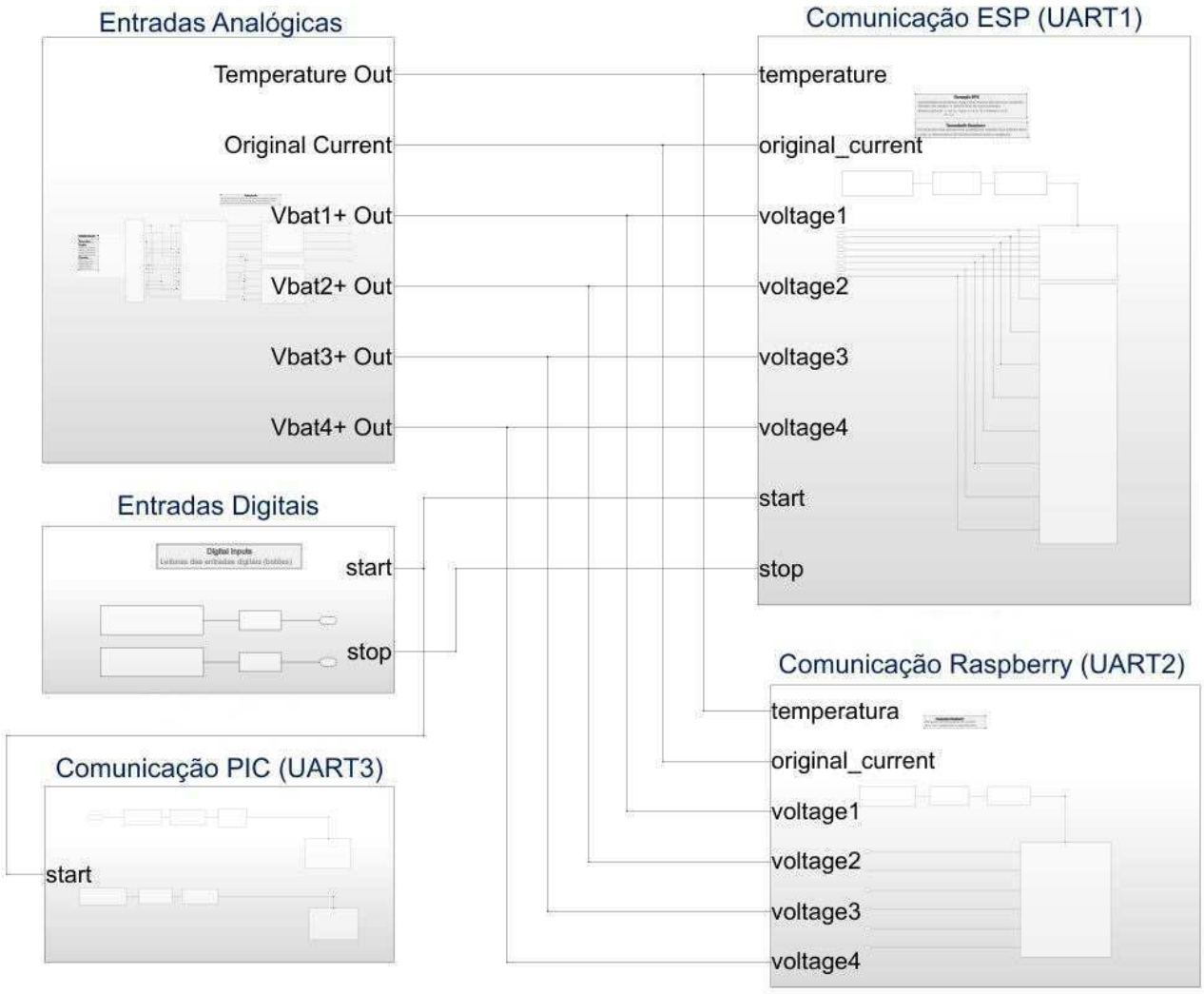
Figura 32 – Representação das Configurações do Modelo de Integração



Fonte: Autoria Própria

Como é observado é realizada inicialização das três UARTs e das respectivas funções de interrupções externas. O modelo de integração também inclui o modelo da aquisição das leituras analógicas dos sensores do *hardware* desenvolvido para o projeto, modelo este que não foi desenvolvido pela estagiária. Na Figura 33 é apresentado o modelo de integração. Não será detalhado cada subsistema devido a complexidade, entretanto trata-se de uma junção dos modelos implementados para as três comunicações.

Figura 33 – Diagrama em Blocos do Modelo de Integração

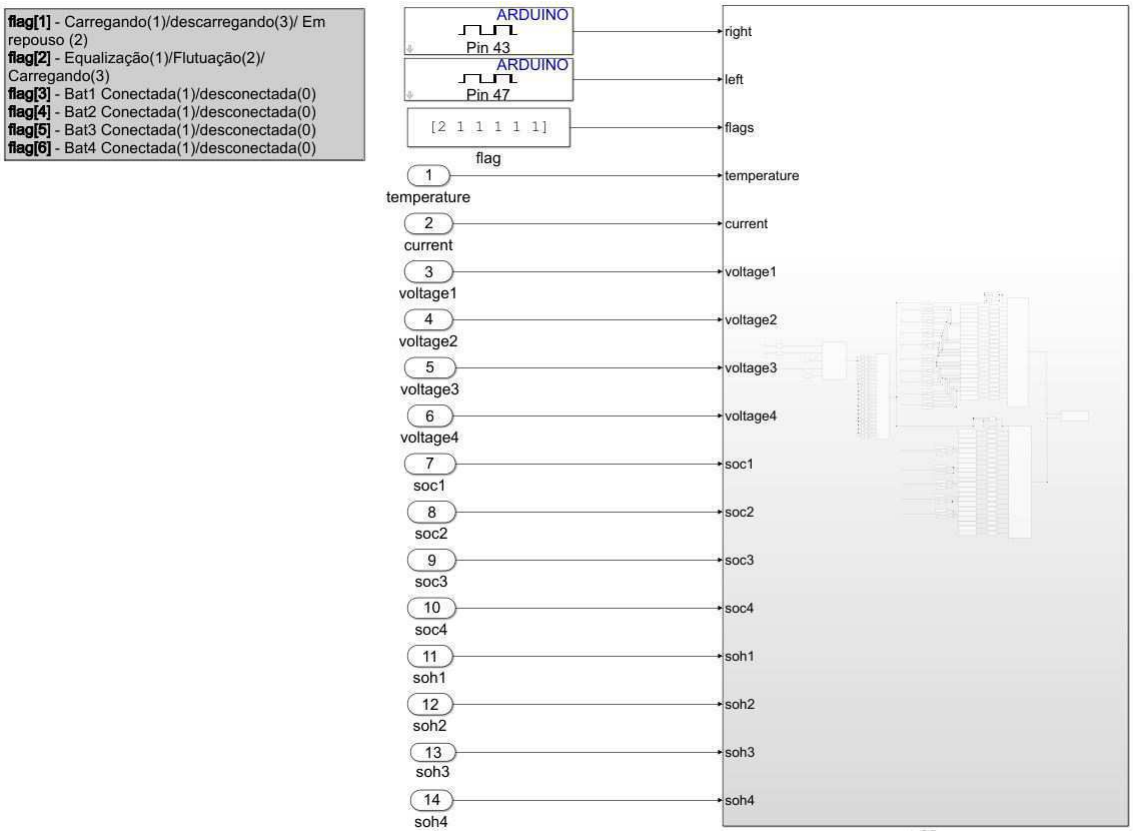


Fonte: Autoria Própria

4.3 Modelo para LCD para o Microcontrolador ATSAM3X8E

Foi implementado um modelo no Simulink para a visualização das informações de monitoramento do sistema desenvolvido no projeto. Esse modelo é embarcado em um microcontrolador ATSAM3X8E e com ele são exibidas em um *Display* LCD 16x2 informações referentes a um banco de baterias e informações referentes à cada bateria conectada ao banco. Essas informações são organizadas em telas e com o acionamento de dois botões é possível realizar a transição entre elas. Para implementar esse modelo, foi utilizado o *Driver* de LCD para Arduino e a biblioteca *LiquidCrystal* para Simulink. Na Figura 34 é apresentado o diagrama de blocos do modelo desenvolvido.

Figura 34 – Diagrama em Blocos do LCD



Fonte: Autoria Própria

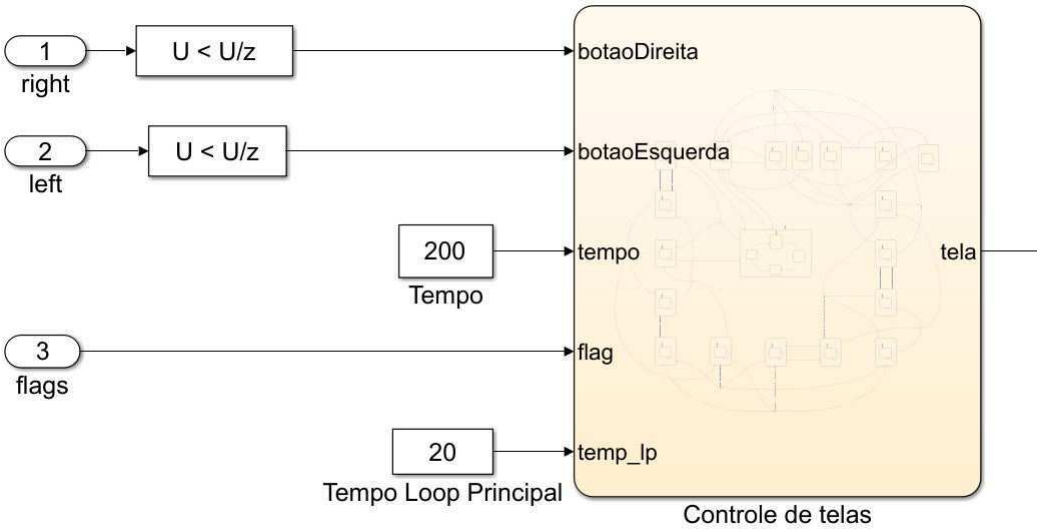
Não é necessário fazer nenhuma configuração via blocos para o microcontrolador. O subsistema principal do modelo recebe como entradas as seguintes informações: duas entradas digitais, referentes aos botões de mudança de tela para direita e para esquerda (Pinos 43 e 47, respectivamente); um *array* de *flags* correspondentes aos *status* das baterias, conforme descrito no quadro de comentários apresentado no diagrama da Figura 34; e as variáveis que serão exibidas no LCD.

As informações que são exibidas no LCD são referentes ao banco de baterias, que são:

temperatura, *status* do banco (carregando ou descarregando), flutuação ou equalização (para o caso de o banco estar em carga), além de informações de cada bateria conectada, que são: estado de carga, estado de saúde, tensão e corrente. Se a bateria não estiver conectada, será exibida apenas uma tela com essa informação.

O modelo no interior do subsistema principal se refere a lógica da transição entre as telas e a organização das informações em cada tela, que ao todo totalizam vinte e duas. Para realizar o processo de transição, é utilizada uma máquina de estados que recebe como parâmetros de entrada os estados dos botões, variáveis de tempo e o *array* de *char* com os *stutus* do banco, conforme apresentado na Figura 35 . A máquina de estado possui um estado para tela, de forma que sua saída é uma variável indicando o número da tela que deve ser exibida, de acordo com as entradas dos botões.

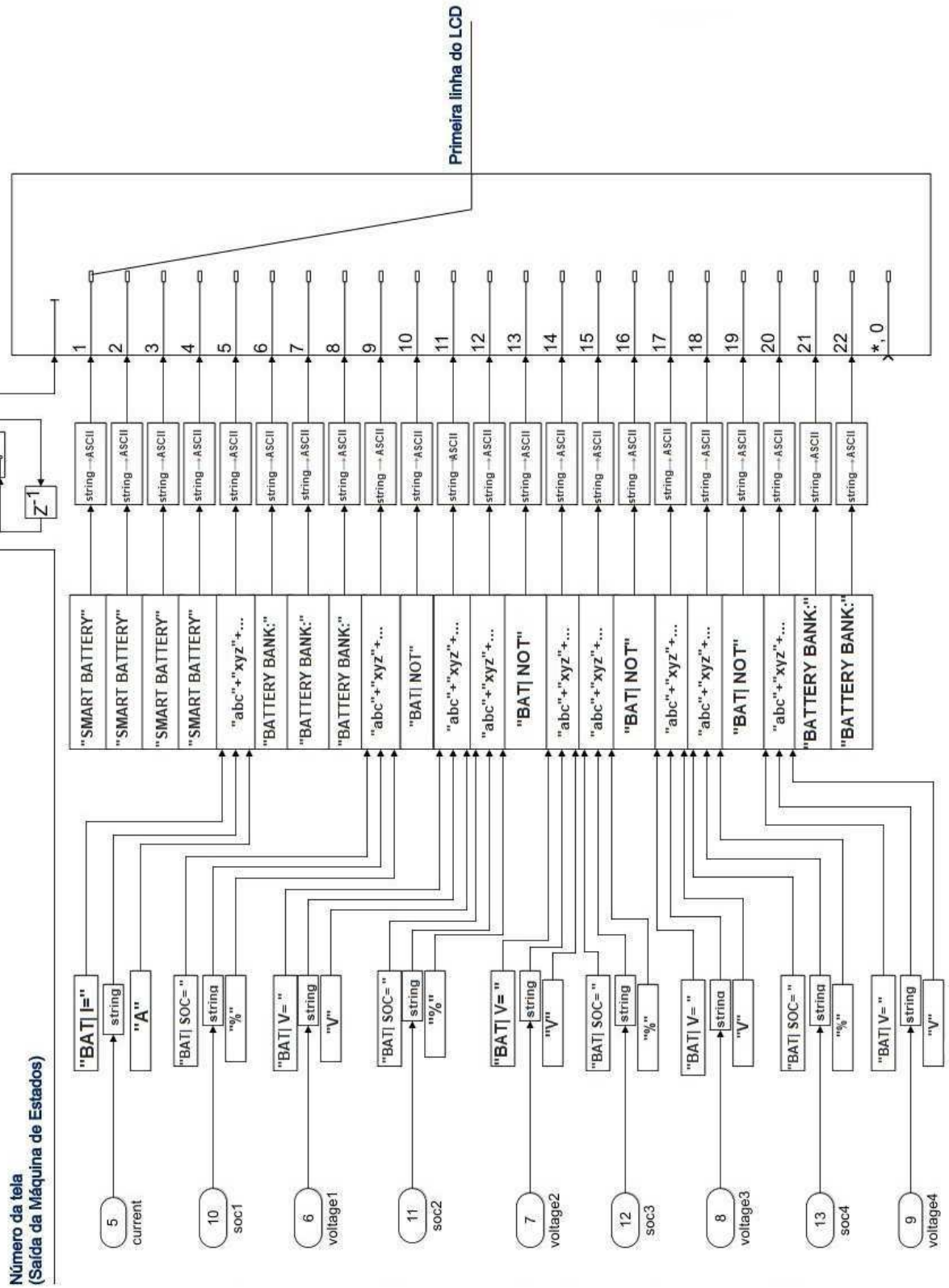
Figura 35 – Representação da Lógica da Transição das Telas do LCD



Fonte: Autoria Própria

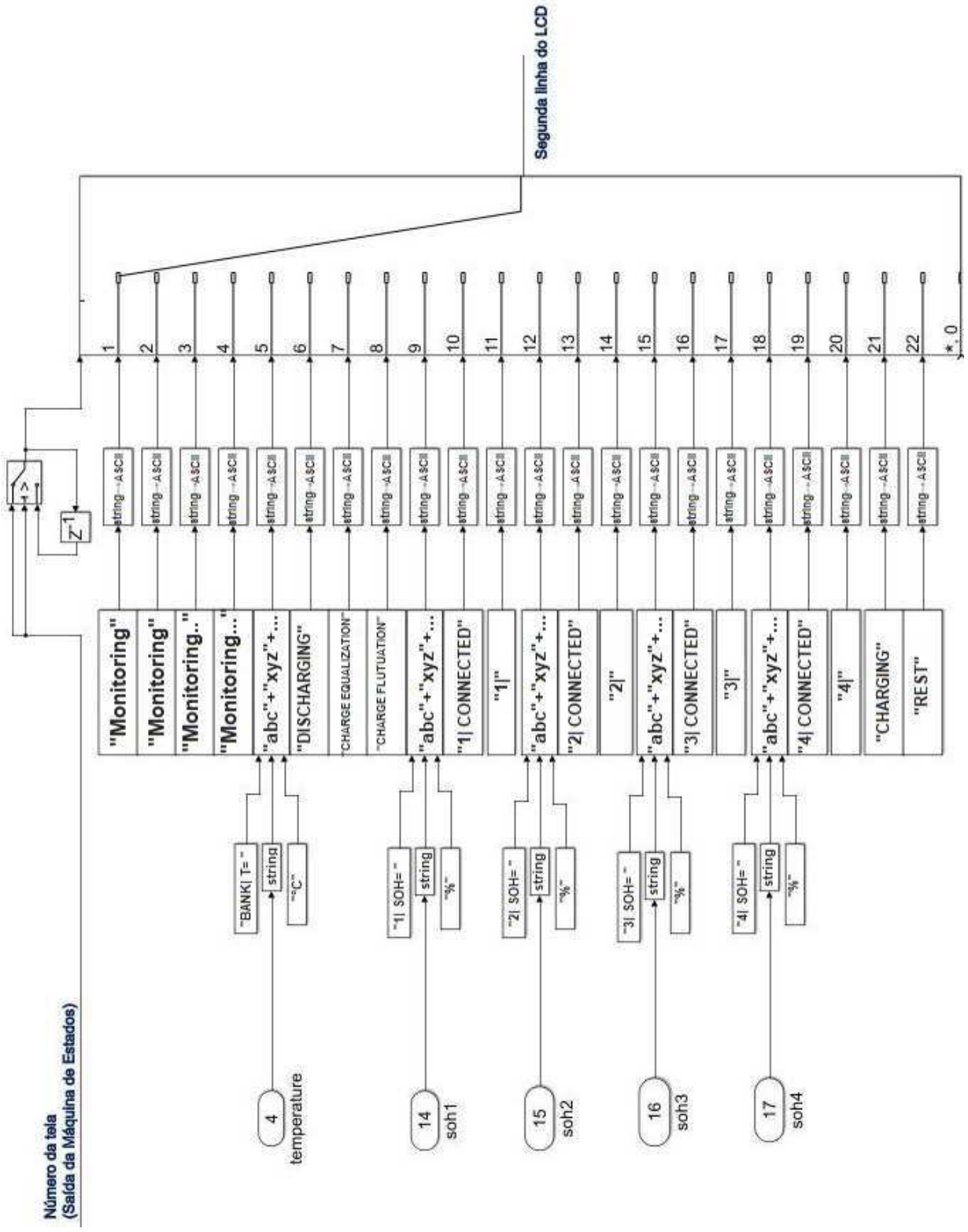
O número da tela atual é a entrada de dois blocos *Multiport Switch*, referentes a lógica de implementação das telas, em que o primeiro se refere a primeira linha do LCD e a última a segunda linha. Cada bloco contém vinte e dois casos, correspondentes a cada tela de informação. Dessa forma, de acordo com a saída da máquina de estados é realizada a seleção da tela que deve ser exibida. Nas Figuras 36 e 37 são ilustradas esse procedimento.

Figura 36 – Representação da Lógica para a Primeira Linha do LCD



Fonte: Autoria Própria

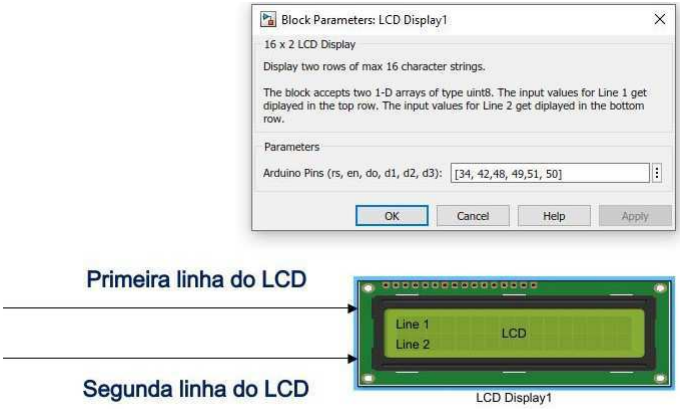
Figura 37 – Representação da Lógica para a Segunda Linha do LCD



Fonte: Autoria Própria

A saída de cada bloco *switch* correspondem as *strings* da primeira e segunda linha que deve ser exibida no LCD. Essas informações são as entradas do bloco *LCD Display*, que configura os pinos de conexão do arduino com o LCD, conforme apresentado na Figura 38. O código gerado a partir do modelo é carregado no ATSAM3X8E presente no sistema de *hardware* desenvolvido para o projeto.

Figura 38 – Representação do Bloco LCD Display



Fonte: Autoria Própria

Neste Capítulo foram apresentadas as atividades desenvolvidas durante o período de estágio, que se referem ao desenvolvimento de modelos no Simulink para geração automática de códigos na linguagem C para sistemas embarcados. Serão apresentadas no Capítulo 6 as considerações finais deste trabalho.

5 Considerações Finais

Neste relatório foram descritas as principais atividades desenvolvidas em um projeto de pesquisa e desenvolvimento como parte da disciplina de Estágio Supervisionado. A aluna desenvolveu suas atividades no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded). Foi cumprida a carga horária de aproximadamente 330 horas, ultrapassando a carga horária mínima para o estágio supervisionado matriculado (300 horas).

As atividades desenvolvidas foram solicitadas pela líder de modelo de acordo com as demandas do projeto. Muitos desafios foram encontrados no desenvolvimento dos modelos para o microcontrolador dsPIC 33EP512MU810, tendo em vista as limitações dos blocos fornecidos pela Microchip e pela falta de suporte da biblioteca no ambiente Simulink. Todavia, o desempenho da estagiária foi satisfatório perante a equipe com a qual trabalhou e todas as atividades solicitadas foram executadas, atingindo assim os objetivos propostos.

Participar de uma equipe formada por profissinais de Engenharia Elétrica e Ciência da Computação foi uma experiência extremamente enriquecedora, na qual a aluna pode vivenciar aspectos práticos acerca do desenvolvimento de projetos, como: trabalhar em equipe, cumprir *deadlines*, participar de reuniões e tomar decisões acerca da melhor solução a ser implementada. Além disso, o projeto em que o estágio foi desenvolvido é alinhado com os interesses de mercado de uma grande empresa, o que propocionou conhecer a um pouco da realidade da indústria.

Dessa forma, foi possível aplicar vários dos conceitos de engenharia adquiridos durante o curso, de forma a complementar a formação acadêmica da aluna. Todavia muitos conhecimentos específicos foram necessários e adquiridos ao longo do projeto.

Referências

[1] Embedded Lab. **LABORATÓRIO DE SISTEMAS EMBARCADOS E COMPUTAÇÃO PERVASIVA**. Disponível em: <https://www.embedded.ufcg.edu.br/>. Acesso em: outubro de 2019.

[2] Microchip Technology Inc. **dsPIC33EPXXX (GP/MC/MU) 806/810/814 and PIC24EPXXX (GP/GU) 810/814**. Datasheet. DS70616G, 2009-2012.

[3] Atmel Inc. **SAM3X / SAM3A Series**. Datasheet. Atmel 11057C ATARM SAM3X SAM3A, 2015.

[4] Arduino Due. Disponível em: <https://store.arduino.cc/usa/due>. Acesso em: novembro de 2019.