Dinesh Atul Rodrigues Trivedi

DESENVOLVIMENTO DE SOFTWARE PARA CONTROLE DE UMA BOMBA DE INFUSÃO DE INSULINA BASEADA EM MICROCONTROLADOR DA FAMÍLIA PIC

São José dos Campos, SP

Dinesh Atul Rodrigues Trivedi

DESENVOLVIMENTO DE SOFTWARE PARA CONTROLE DE UMA BOMBA DE INFUSÃO DE INSULINA BASEADA EM MICROCONTROLADOR DA FAMÍLIA PIC

Trabalho de conclusão de curso apresentado ao Instituto de Ciência e Tecnologia – UNIFESP, como parte das atividades para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal de São Paulo – UNIFESP

Instituto de Ciência de Tecnologia

Bacharelado em Ciência da Computação

Orientador: Prof. Dr. Luiz Eduardo Galvão Martins

São José dos Campos, SP

Julho de 2014

Dinesh Atul Rodrigues Trivedi

DESENVOLVIMENTO DE SOFTWARE PARA CONTROLE DE UMA BOMBA DE INFUSÃO DE INSULINA BASEADA EM MICROCONTROLADOR DA FAMÍLIA PIC

Trabalho de conclusão de curso apresentado ao Instituto de Ciência e Tecnologia – UNIFESP, como parte das atividades para obtenção do título de Bacharel em Ciência da Computação.

Trabalho aprovado em 01 de Julho de 2013:

Prof. Dr. Luiz Eduardo Galvão Martins
Orientador

Prof. Dr. Tiago de Oliveira
Convidado 1

Prof. Dr. Fabiano C. Paixão
Convidado 2

São José dos Campos, SP
Julho de 2014

Este trabalho é dedicado ...

Agradecimentos

Escreva aqui os agradecimentos ...

Ensinar é reensinar a pensar. Bom senso é o hábito de pensar, podemos dizer que é pensar de uma determinada forma. O primeiro passo é reensinar a pensar, construir pensamento. O Segredo é entender e não decorar, saber o porque das coisas. As perguntas são mais importantes que as respostas, as respostas mudam, mas as perguntas não. (Trivedi, Dinesh)

Resumo

Estima-se que o Diabetes Melito (DM) já afeta 246 milhões de pessoas em todo mundo. A estimativa é que aumente para 380 milhões até 2025, sendo que no Brasil esse número chegue à aproximadamente 7 milhões. Seu tratamento adequado, na maioria das vezes, é o uso contínuo da bomba de infusão de insulina, entretanto é inacessível a grande parte da população, devido ao seu alto custo, aproximadamente R$ 14.000,00. Sendo assim,este projeto tem como objetivo o desenvolvimento de um sistema embarcado crítico de controle de protótipo de bomba de infusão de insulina, baseado no microcontrolador da família PIC, PIC18F452. Neste trabalho utilizou-se o compilador MikroC que dá suporte aos periféricos necessários através de uma biblioteca completa em C. O desenvolvimento utilizou conceito de OOC, *Object Orientated Programming in ANSI-C*, para desacoplamento de todos os módulos existentes. Esse conceito possibilitou o uso de *Design Patterns*, ambos contribuíram para aumentar a capacidade de expandir e dar manutenção ao *software* de forma mais simples. Os testes do software foram realizados através do simulador Proteus, o que possibilita uma grande facilidade para o depuramento. O circuito montado nesse simulador contém todos os componentes reais necessários para o desenvolvimento do protótipo. E,além disso, optou-se pelo uso do Proteus devido as grandes vantagens e minimização de problemas de integração que um simulador proporciona. Por fim esse projeto serve como uma prova de conceito para o problema abordado, demonstrando a evolução e problemas encontrados durante o desenvolvimento. Servindo como base para projetos futuros.

Palavras-chaves: PIC, Microcontrolador, Sistema embarcado, Sistema crítico, Motor de passo, OOC.

Abstract

It is estimated that diabetes mellitus (DM) already affects 246 million people around world. It is estimated to increase to 380 million by 2025, and in Brazil this the number will reach about 7 million. Adequate treatment in most times, is the continuous use of insulin pump therapy, however it is inaccessible to a large part of the population due to its high cost approximately R$ 14,000.00. Thus, this project aims to develop an embedded system critical control prototype insulin infusion pump based microcontroller family PIC, PIC18F452. In this study we used the MikroC compiler that supports the required peripherals through a complete library in C. The development concept used OOC emph Object Orientated Programming in ANSI-C for decoupling of all the modules. This concept allowed the use of emph Design Patterns, both contributed to increase the ability to expand and maintain the emph software in a simpler way. The tests were performed using the software simulator Proteus, which allows a great facility for the depuration. The circuit used in simulator contains all the components necessary for real prototype development. Also, Proteus was chosen because of the great benefits and minimize integration problems that a simulator provides. Finally this project serves as a proof of concept for the addressed problem, demonstrating the progress and problems encountered during development. Serving as a base for future projects.

Key-words: PIC, Microcontroler, Embedded System, Critical System, Step Motor, OOC.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

μA	Microampére
ADC	*Analogic Digital Conversor*
A/D	*Analogic Digital*
CA	Corrente alternada
CC	Corrente contínua
CCP	*Capture* / *Compare* / *PWM*
DM	Diabetes Melito
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
I/O	Entrada/Saída
IDE	*Integrated Development Environment*
kHzs	*Kilohertz*
LCD	*Liquid Crystal Display*
LVD	*Low Voltage Detect*
mA	Miliampére
MCLR	*Master Clear or Reset*
MHz	*Megahertz*
MIPS	Milhões de Instruções por Segundo
MSSP	*Master Synchronous Serial Port*
OO	Orientado à Objetos
OOC	*Object Orientated Programming in ANSI-C*
PC	*Personal Computer*
PIC	*Programmable Interface Controller*
PSP	*Parallel Slave Port*

PWM	*Pulse-Width Modulation*
RAM	*Random Access Memory*
ROM	*Read-Only Memory*
TTL	*Transistor-transistor Logic*
V	Volts

Sumário

1 Introdução

1.1 O que é Diabetes Mellitus?

O diabetes mellitus(DM) é um doença causada por uma desordem metabólica, caracterizada por hiperglicemia crônica envolvendo distúrbios relacionados à carboidratos, resultado do metabolismo de gorduras e proteínas devido à defeitos na secreção de insulina, ação da insulina, ou ambos. Os efeitos da diabetes mellitus inclui danos a longo prazo, disfunção e falha de vários órgãos. Os sintomas da diabetes mellitus podem ser sede, urinar em excesso, embaçamento da visão e perda de peso. Em caso mais graves, cetoacidose ou um não-cetótica hiperosmolar podendo desenvolver e levar ao estado de consciência ou sensibilidade apenas parcial ou insensibilidade, coma e, na ausência de tratamento eficaz, a morte. Muitas vezes, os sintomas não são graves, ou podem estar ausentes. Os efeitos a longo prazo da diabetes mellitus incluem perda parcial da visão e até mesmo cegueira, nefropatia, que pode levar a insuficiência renal e/ou neuropatia com risco de úlceras nos pés, amputação, incluindo disfunção sexual. Pessoas com diabetes têm um risco aumentado de adquirir doenças cardiovasculares e cerebrovasculares (ALBERTI; ZIMMET, 1998)

Segundo (ALBERTI; ZIMMET, 1998), o DM possui duas classificações, Tipo 1 e Tipo 2. A primeira caracteriza-se por ser autoimune que lesa, de forma irreversível, as células do pâncreas, produtoras de insulina e conhecidas como células beta, e seu diagnóstico se dá durante a infância do portador. Enquanto a segunda é consequência da resistência do próprio organismo contra as ações da insulina, o principal fator para se desenvolver essa resistência é a obesidade.

1.2 Tipos de Tratamentos Diabetes Mellitus Tipo 1

As formas de terapias para a diabetes mellitus desenvolveu e mudou bastante desde 1921. Inicialmente as seringas eram a única forma de terapia de injeção de insulina, e até hoje continuam ser o principal método utilizado nos EUA. Os avanços tecnológicos levaram as seringas mais compactas e de diversos tamanhos, e uma ampla variedade de agulhas, ultrafina, microfina e outros sistemas de agulhas. Avanços nos quais aumentaram o conforto dos pacientes.

Toda melhoria de um sistema de administração de insulina possui dois objetivos comuns: conveniência e melhor controle glicêmico. Estudos clínicos demonstraram uma forte relação entre um rígido controle glicêmico e a redução do risco de complicações em pacientes com diabetes mellitus. As principais formas de tratamento são seringas e bomba de infusão de insulina (MAGNOLTI; RAYFIELD, 2007).

1.2.1 Seringas

As seringas dominam o mercado de dispositivos de injeção de insulina. Inicialmente eram grandes, pesadas, tubos de vidros reutilizáveis com um longa agulha que precisava de uma frequente amolação. Seringas de plástico modernas são leves, descartáveis e com agulhas ultrafinas para aumentar o conforto e conveniência dos pacientes. Seringas de insulina são diferenciadas de três maneiras:

- Espessura da agulha;
- Comprimento da agulha;
- Capacidade da agulha.

A maioria dos fabricantes oferecem uma variedade de tamanhos e estilos, estas variações aumentam o conforto e conveniência para o paciente. Sendo essa variedade de produtos a principal vantagem no uso de seringa. Por exemplo, existem seringas fáceis de ler suas medidas, baixas doses disponíveis para aqueles pacientes que têm problemas de visão e que requerem baixas quantidade de insulina. Além disso, os pacientes podem misturar diferentes tipos de insulina em uma seringa para satisfazer as suas necessidades de insulina individuais. Seringas tem alguns inconvenientes.

- Carregar um frasco volumosos e seringa pode ser complicado;
- Podem intimidar muitos pacientes;
- Técnica de injeção leva tempo e prática para se desenvolver;
- A manipulação da seringa e o frasco pode ser particularmente difícil para os pacientes com destreza e/ou diminuição da visão;
- Aplicação de uma dose de insulina e injetando-se com uma agulha e seringa em um ambiente social poderá causar constrangimento.

Além disso, a precisão da dose de uma seringa varia com seu tamanho. Estudos mostram que o erro na medição de doses de insulina com menos de 5 unidades foi de quase 10% (MAGNOLTI; RAYFIELD, 2007).

1.2.2 Bomba de Infusão de Insulina

Bomba de infusão de insulina é um pequeno aparelho eletrônico, do tamanho de um celular ou pager, que está ligado ao corpo do portador da doença por um finíssimo cateter com uma agulha flexível na ponta. Essa agulha é inserida no braço, coxa ou abdômen e deve ser

trocada em um período de 2 ou 3 dias. Essa bomba não mede o índice glicêmico ou a quantidade de insulina a ser utilizada, essa medição é feita através do glicosímetro. A Figura 1[1] representa um aparelho comercial (BODE; TAMBORLANE; DAVIDSON, 2002)

Figura 1 – Imagens de uma bomba de infusão de insulina

(PORTALDIABETES, 2008)

Seu funcionamento é bem simples, libera-se uma quantidade de insulina, programada pelo médico, durante o dia todo, simulando o funcionamento do pâncreas de uma pessoa saudável, entretanto existem cuidados a serem tomados: calcular a quantidade de carboidratos ingeridos a cada refeição e programar o aparelho para injetar uma quantidade de insulina com maior velocidade no organismo nos horário em que se faz as refeições principais. Quanto a quem pode usar, a pessoa deve cumprir alguns pré-requisitos que são:

- Conseguir medir o índice glicêmico no mínimo 4 vezes por dia;
- Durante a fase de adaptação e ajuste da dosagem a serem utilizadas pela bomba, fazer a medição glicêmica de 6 a 8 vezes por dia;
- Seguir as recomendações médicas além de manter contato e um constante *feedback* com os responsáveis pela bomba e, além de tudo, seguir a dieta recomendada, respeitando quantidades ingeridas;
- Ter condição financeira para custear o equipamento e o contato com os responsáveis por ele;
- Estar disposto ao uso da bomba durante o dia todo, 24 horas junto ao corpo;
- Aprender sobre contagem de carboidratos para saber seu consumo durante as refeições;
- Praticar exercícios.

Cumprindo os pré-requisitos citados temos as vantagens de seu uso que são:

- Maior flexibilidade no horário das refeições;

[1] http://www.diabetes.org.br/sala-de-noticias/2316-bombas-de-infusao-de-insulina

- Se usada corretamente o risco de hipoglicemia é reduzido, e a longo prazo as complicações devido a diabetes também;
- Melhora o controle glicêmico;
- Melhora no controle do fenômeno do amanhecer, responsável pelo aumento do índice glicêmico durante a manhã, entre as 4 e 8 horas da manhã, causador da hipoglicemia se o diabético não calculou a dose de insulina antes de dormir, ou não se levantou durante a noite para gerenciá-la.

Entretanto mesmo com as vantagens existentes em função de seu uso, caso o diabético seja obeso, ingira grandes quantidades de alimento ou açúcar, ou seja, carboidratos, não praticar atividades físicas, não fazer a medição do índice glicêmico na quantidade de vezes recomendada, ou até mesmo determinar por si só a quantidade de insulina a ser utilizada, essas vantagens deixam de existir.

É importante ter em mente que mesmo com toda facilidade e tecnologia existente o acompanhamento médico não deve ser deixado de lado. As principais indicações médicas para o uso do equipamento são:

- Fenômeno do amanhecer;
- Hipoglicemia;
- Diminuir a variação do índice glicêmico;
- Hiperglicemia;
- Recorrente ceatosidade, que é o acumulo de ceatócidos, pois o fígado quebra a gordura e proteína devido à falta de insulina, pois o corpo não consegue utilizar a glicose como energia;
- Flexibilidade, especialmente para crianças pequenas;
- Gestação, viagens e atividade físicas;
- Fobia de injeção;
- Desejo do diabético (SBC, 2014; DIABETES, 2013; PORTALDIABETES, 2009).

1.3 Motivação

Segundo a Sociedade Brasileira de Diabetes (SBC, 2014), diversos estudos realizados mostram que o tratamento feito através da Infusão de insulina tem diversas melhorias quando comparado com outros tratamentos existentes. Entretanto não é o mais utilizado devido ao seu

alto custo, devido a importação. Logo, esse projeto vem com foco social: facilitar o acesso da população brasileira de baixa renda ao equipamento, melhorando sua qualidade de vida dos portadores da doença que se encaixem nesse perfil.

1.4 Objetivos

Esse projeto tem como objetivo principal desenvolver um software para controle de uma Bomba de Infusão de insulina utilizando o microcontrolador da família PIC - *Programmable Interface Controller* -, PIC18F452.

1.4.1 Objetivos Secundários

Em segundo plano este trabalho foca em:

- Aprendizado sobre as características e funcionalidades disponíveis do microcontrolador escolhido;
- Aprendizado das tecnologias utilizadas como: compilador, simulador e bibliotecas disponíveis;
- Desenvolvimento das funcionalidades básicas de uma bomba de infusão de insulina;
- Aprendizado sobre a escolha e uso de um motor de passo;
- Aprendizado sobre a forma de uso de um *display* de LCD para comunicação com o usuário.

1.5 Procedimentos Metodológicos

Inicialmente foi feita uma pesquisa e levantamento bibliográfico sobre o tema em questão para adquirir um melhor entendimento da plataforma, tecnologias e, claro, do problema abordado. Juntamente com essa pesquisa, foi feito um estudo sobre o microcontrolador PIC escolhido, PIC18F452, a partir do material encontrado. O estudo e desenvolvimento foi feito utilizando o compilador MikroC e sua IDE - *Integrated Development Environment* -, já para os testes utilizou-se o simulador Proteus. Em seguida houve-se um levantamento de informações sobre sistemas embarcados, sistemas críticos e sistemas críticos de tempo real, e assim adquirindo-se conhecimento sobre os conceitos de desenvolvimento da área em questão, uma vez que a confiabilidade e segurança do problema proposto é de suma importantância.

O software de controle do protótipo da bomba de infusão de insulina foi desenvolvido na linguagem C, compilado para o microcontrolador já citado, PIC18F452. O sistema é responsável basicamente por gerenciar o perfil de infusão basal, calcular intervalos corretos e passos

necessários para uma infusão. Desta forma, o software foi dividido nos seguintes módulos: Config, LCD, InsulinPump, Motor, Menu, TimerMotor e principal.

Essa divisão por módulos é, e foi, extremamente importante para o desenvolvimento do sistema. Somando-a à um dos conceitos mais importante desse projeto - OOC, *Object-Oriented Programming With ANSI-C* - são as chaves para a facilidade de manutenção, entendimento do código e possível evolução do projeto.

As responsabilidade dos módulos são bem claras e isoladas:

Config: Centralizar configurações gerais do sistema; LCD: Abstrair uso do periférico para o resto do sistema; InsulimPump: Abstrair funcionamento, requisitos de segurança e outras particularidades da bomba; Motor: Abstrai como se controla o motor e qual tipo está sendo utilizado para infusão; TimerMotor: Separa o gerenciador de tempo das particularidade únicas do hardware e compilador; Menu: Facilitar navegação entre menus, simulando máquina de estado; Principal: Possui o *loop* principal para navegação simples entre os menus existentes.

Os testes do software foram feitos através do simulador Proteus, desenvolvido pela Labcenter. Ferramenta tão importante quanto o OOC para o caminhar deste trabalho. Permite um alto nível de testes, devido a sua facilidade de uso, além do grande auxílio à depuração. O uso do simulador durante o desenvolvimento do software acarreta na minimização dos problemas de integração entre hardware e software, complicações estas que são praticamente impossíveis de estimar.

E, finalizando, após o desenvolvimento, executou-se uma bateria de testes para se o funcionamento da bomba estava sendo de acordo coma forma especificada, levando em conta toda a integração necessária com os periféricos existentes.

1.6 Organização do Trabalho

@TODO: VOLTAR!!!!!!!!!!!!!!!

A organização desse trabalho tem como início considerações do estudo feito sobre sistemas embarcados, no capítulo 2. Assunto que abrange definições sobre tipos de sistemas embarcados, componentes como microcontroladores, sensores, atuadores e de interface de comunicações com o usuário. Em seguida, tem-se uma introdução do simulador Proteus, ferramenta de teste utilizada, em conjunto com o compilador. Para complementar as informações sobre o compilador, há uma explicação da organização do microcontrolador escolhido, como pinagem, contadores, barramentos e outros componentes. A partir dessas informações aborda-se o problema proposto, contextualizando-o, descrevendo requisitos e características.

Após a introdução do problema abordado e explicação de todas as tecnologias utilizadas, vem a descrição e organização do software, explicando cada módulo e forma de implementação. Assim sendo tem-se a discussão de resultados abordando detalhes, motivos e vantagens

da implementação para que seja possível finalizar descrevendo a conclusão de todo estudo e projetos futuros.

2 Sistemas Embarcados

Diferentemente dos computadores do dia-dia, ou seja, de propósito geral, em sistemas embarcados o software é acoplado à um hardware dedicado, sendo as funcionalidades implementadas de acordo com os requisitos e também com as características e particularidades de seus componentes físicos sendo utilizados. Sua principal característica é o fato de ser destinado a executar tarefas específicas (LI; YAO, 2003)

Segundo (CUNHA, 2013) a inteligência embarcada é uma tendência futura, cada vez mais inteligência será adicionada aos equipamentos do dia-a-dia, considera que um micro-ondas atual tem mais capacidade computacional do que tinha o projeto Apolo, que levou o homem a lua. Esta crescente utilização se dá basicamente pelo preço e consumo reduzido dos microcontroladores, além da grande flexibilidade ao atender os mais diversos problemas visto o vasto número de arquiteturas disponíveis: ARM, MIPS, Coldfire/68k, PowerPC, x86, PIC, 8051, Atmel AVR, Renesas H8, SH, V850, FR-V, M32R, Z80, Z8 e outras. Um contraste que atrai diversos desenvolvedores quando comparado com o número limitado de arquiteturas diponíveis para microprocessadores do mercado de computadores pessoais (GERMANO, 2011).

A comunicação dos microcontroladores com o meio externo, segundo (GERMANO, 2011), se dá pelos periféricos e o mais comuns são:

- Entrada de dados através de teclas, geralmente pelo de teclados feitos com varredura matricial;
- *Leds*;
- *Display's* de LCD, sendo os mais comuns os alfanuméricos como o HD44780;
- Interface serial, por exemplo RS232 e I2C;
- USB, *Universal Serial Bus*;
- TCP/IP.

Como dito anteriormente, esses sistemas estão cada vez mais no dia-a-dia das pessoas e, claro, facilitando a vida delas, mas muitas vezes não são percebidos. E cada vez mais estão mais acessíveis podendo automatizar funções até mesmo dentro das próprias casas.

2.1 Sistemas Embarcados de Tempo Real

O conceito Tempo Real vem da ideia básica em que é esperado que o software execute ou dê um retorno de sua execução até um limite de tempo bem definido. Normalmente pessoas

assumem que tempo real significa "muito rápido", entretanto não é verdade, tempo real simplesmente significa "rápido o suficiente"no contexto de operação do sistema. Um exemplo é a ação do motor, pode-se dizer que é "rápida", pois o sistema deve tomar decisões como - fluxo de combustível, tempo da faísca - toda vez que o motor completa um ciclo.

Sistemas de tempo real são baseados em previsibilidade e, segundo (FARINES; FRAGA; OLIVEIRA, 2000), essa previsibilidade de um sistema de tempo real é obtida quando independente de falhas, sobrecargas e variações de hardware, e assim é possível que seu comportamento seja antecipado antes de sua execução. Isso tem a finalidade de poder prever o funcionamento de um sistema de tempo real e garantir as suas restrições temporais, e para isso é necessário definir hipóteses em relação a carga e falhas em relação ao ambiente externo deste sistema (FARINES; FRAGA; OLIVEIRA, 2000). Segundo (MALL, 2009) os sistemas de tempo real são classificados em dois tipos:

- *Soft Real Time Systems*: Sistemas não críticos de tempo real, onde a ocorrência de uma falha temporal é da mesma ordem de grandeza que os resultados em que o funcionamento está correto, exemplos: Máquina de lavar e portão eletrônico de uma casa;
- *Hard Real Time Systems*: Sistemas Críticos de Tempo Real, onde a ocorrência de uma falha temporal complicam, e muito, os resultados quando comparado com seu funcionamento correto, exemplos: sistema de controle de um avião e um sistema de controle de semáforos.

2.2 Sistemas Embarcados Críticos

Sistema Crítico é um sistema no qual a confiança é fundamental, ou melhor, a questão mais importante em seu desenvolvimento. Isso porque sistemas críticos, em caso de falha, podem causar consequências gravíssimas para os humanos, economia e outras áreas. Pode-se dizer que seus indicadores são: Disponibilidade, confiabilidade, segurança e proteção. E para que essa confiança seja alcançada deve-se evitar erros durante seu desenvolvimento e realizar diversos testes para que seja possível detectar e corrigir os erros que passarem de forma que seja possível limitar os danos causados por falhar operacionais (SOMMERVILLE, 2004; FELDMANN et al., 2007; JORDAN, 2006).

Segundo (KOPETZ, 2011) as classificações dos sistemas embarcados críticos podem ser:

- *Fail Safe*: Classificação para sistemas onde o estado seguro pode ser atingido em caso de falha, como por exemplo, esgotar a bateria de uma bomba de insulina;
- *Fail Operational*: Classificação para sistemas que em caso de falhas ainda são capazes de fornecer algum tipo de serviço, mesmo que mínimo. Um exemplo é um sistema de

controle de voo que, mesmo em caso de falha, é capaz de fornecer serviços e ser seguro.

2.3 Microcontrolador

Segundo (GANSSLE, 1999), o microcontrolador é a parte mais importante de um sistema embarcado e sua principal diferença quando comparada com um microprocessador é o fato de ser um sistema computacional completo que integra todos as principais partes da arquitetura de Von Neumann em um único componente, as partes citadas são:

- CPU: *Central Processor Unit*;
- Memória RAM: *Random Access Memory*;
- Portas I/O: Portas de entrada e saída.

Além de ser composto por temporizadores, memória ROM (*Read Only Memory*), conversor AD, analógico – digital, e DA, digital – analógico. Comparados com microprocessadores, os microcotroladores possuem consumo e *clock*, processamento, reduzidos, isso devido ao fato que o primeiro é destinado a tarefas que necessita uma alta capacidade de processamento como, por exemplo, os microprocessadores do nosso PC do dia-a-dia. Por padrão, os microprocessadores são utilizados em situações que os requisitos são abrangentes, com entradas e saída variadas como: sensores, atuadores e periféricos de comunicação (LEE; SESHIA, 2011).

2.4 PIC18F452

O PIC18F452 é um microcontrolador fabricado pela empresa *Microchip Technology*. Possui tecnologia CMOS, como consequência tem um consumo baixíssimo, possui memória do tipo FLASH, um grande facilidade para desenvolvimento de protótipos, uma vez que para apagá-la não é preciso utilizar luz ultravioleta como em versões antigas, utilizavam memória EEPROM. Abaixo seguem as principais características desse microcontrolador:

- Microcontrolador de 40 Pinos;
- Memória de programa FLASH de 32Kbytes;
- Memória RAM de 1536 bytes;
- Memória EEPROM de 256 bytes;
- Processamento de até 10MIPS;
- Quatro *timers*, ou temporizadores, internos – um de 8 bits e 3 de 16 bits – TIMER0, TIMER1, TIMER2 e TIMER3;

- 2 canais capture/compare/PWM – Módulo CCP;
- Módulo *Master Synchronous Serial Port* (MSSP);
- *Unhaced Usart;*
- 8 canais A/D de 10 bits;
- Detector de baixa voltagem programável;
- Permite até 100.000 ciclos de escrita e leitura na memória FLASH;
- Permite 1.000.000 ciclos de escrita e leitura na memória EEPROM.
- Retenção de dados na FLASH por 40 anos;
- *Watchdog timer* com oscilador próprio e programável;
- Três pinos de interrupções externas: INT0, INT1 e INT2

A Figura 2 representa uma imagem do microcontrolador.

Figura 2 – PIC18F452

2.4.1 Descrição dos pinos

Dos 40 pinos desse microcontrolador 34 são pinos I/O, entrada e saída, divididos em 5 "PORT". A Figura 3 representa uma relação dos pinos do microcontrolador.

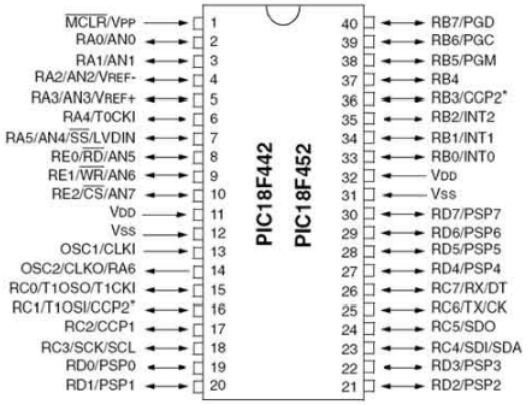

Figura 3 – Pinos PIC18F452

A divisão dos pinos I/O citada é da seguinte maneira:

- PORTA: São 7 pinos nomeados de RA0 a RA6. Podem ser utilizados como I/O geral ou conversor A/D, essa segundo opção tem exceção o pino RA4. Além de possuir a opção LVD, detecção de baixa tensão;
- PORTB: São 8 pinos nomeados de RB0 a RB7. Podem ser utilizados para I/O geral e, além disso, pode-se trabalhar com três interrupções externas, módulo CCP, pinos de gravação e debug;
- PORTC: São 8 pinos nomeados de RC0 a RC7. Podem ser utilizados para I/O geral, saída do oscilador do *timer*, módulo CCP, Clock e data(dados) para os modos SPI, I2C e UART;
- PORTD: São 8 pinos nomeados de RD0 a RD7. Podem ser utilizados para I/O geral ou como PSP para ter saída TTL, para interfaceamento com microprocessadores, por exemplo;
- PORTE: São 3 pinos nomeados de RE0 a RE2. Podem ser utilizados para I/O geral ou pinos de controle de acesso.

2.5 Sensores

Segundo (ALMEIDA, 2004), transdutores são equipamentos que transformam informações, ou seja, transformam um tipo de sinal em outro para permitir medições de grandezas, controles de outros equipamentos, entre outras formas de transformação. A definição de sensor pode ser a de um transdutor capaz de alterar sua característica física interna em resposta à

um fenômeno físico externo. Além disso, existem sensores considerados de operação indireta que são os quais alteram suas propriedades como capacitância, resistência, ou, até mesmo, sua indutância, sob ação de algum gradeza ou evento externo (ROSÁRIO, 2006).

Segundo (NOMADSUSP, 2012), os sensores são largamente utilizados na medicina, indústria, robótica, além de outras aplicações. Considerando que o sinal é sempre uma forma de energia, os sensores podem ser classificados em função da energia que é capaz de detectar, como:

- Sensores de luz: células solares, fotodíodos, fototransistores, tubos fotoelétricos, e outros;
- Sensores de som: microfones e hidrofone;
- Sensores de temperatura: termômetros e termopares;
- Sensores de resistência elétricas: ohmímetro.
- Outros;

2.5.1 Sensores Biológicos

Segundo (NOMADSUSP, 2012), os sensores citados anteriormente são corretamente chamados de sensores artificiais. Isto devido ao fato de existir sensores naturais ou biológicos, já que todos os organismos vivos possuem sensores capazes de agir da mesma forma que os artificias. Esses sensores biológicos são células especializadas, sensíveis a:

- Luz, movimento, temperatura, vibração, pressão, campos eléctricos, som, e outros aspectos físicos do ambiente;
- Grande variedade de moléculas ambientais, incluindo toxinas e nutrientes;
- Aspectos metabólicos, tais como os níveis de glicose e oxigênio;
- Até mesmo as diferenças entre proteínas do ambiente externo e do próprio organismo.

Esses sensores artificiais que imitam sensores biológicos, utilizando componentes biológicos, são chamados biossensores.

2.5.2 Atuadores

Segundo (CHIRONIS; SCLATER, 1991), dispositivos considerados atuadores são aqueles que transformam uma forma de energia em outra, causando mudanças no ambiente em que estão atuando, ou seja, de acordo com sinais, ou impulsos, recebidos realizam ações capazes

de alterar as grandezas físicas do ambiente em questão. Eles são capazes de converter energias como: energia elétrica, hidráulica e pneumática em energia mecânica. Segue exemplos de alguns tipos de atuadores:

- Atuadores eletromagnéticos: São os motores elétricos como motores de passos, servos;
- Atuadores hidráulicos: Utilizam um fluido submetido a uma pressão para movimentar um braço, são utilizados em robô que operam grandes cargas;
- Atuadores pneumáticos: Utilizam um gás submetido a uma pressão para movimentar o braço, possuem menor custo que os hidráulicos, sendo utilizados em robôs de menor porte;

2.5.3 Motor de Passo

Motor de passo é um dispositivo eletromecânico. Sua principal propriedade é sua habilidade de transformar pulsos elétricos em movimentos, que são precisamente incrementados na posição do rotor e são denominados "passos". Esse tipo de motor é caracterizado como máquina duplamente saliente, significando que possui dentes, compostos por matérias magnéticos nas duas partes que o compõe: A parte imóvel chamada estator e a móvel rotor (SANTOS, 2008; ACARNLEY, 2002).

Seu uso é interessante em situações em que precisão nos movimentos é necessária. Isso porque com ele é possível controlar: ângulo de rotação, velocidade, posição e sincronismo. Suas vantagens não são seu torque nem a capacidade de gerar movimentos de alta velocidade, mas sim a precisão em seus movimentos. Devido a essas características esse tipo de motor é amplamente utilizado em: câmeras de vídeo, robôs, brinquedos, *scanners*, impressoras, entre outros (SANTOS, 2008).

De forma simples o funcionamento de um motor de passo consiste no uso de materiais magnéticos, ou solenoides, como dito anteriormente, alinhados dois a dois, representando os polos norte e sul, que quando energizados atraem o rotor fazendo-o se alinhar as partes energizadas do estator, causando assim um pequeno movimento: o passo. Sua velocidade e sentido estão diretamente relacionados à forma com que os solenoides são acionados, o primeiro com a frequência e o segundo a ordem de acionamento (SANTOS, 2008; ACARNLEY, 2002).

2.5.4 LCD - *Liquid Crystal Display*

Um *display* de cristal líquido, LCD, é utilizado para exibir informações como texto, imagens e vídeos. É composto por um painel fino e, segundo (BARBACENA; AFONSO, 1996), é um componente, considerado interface de saída, muito utilizados em conjunto com microcontroladores. Exemplos de uso são: *cockpit* de aeronaves, *displays* em computadores de bordo de automóveis, dispositivos de jogos, relógios, calculadoras e outros.

Pode-se classificar os módulos de LCD em: exibição gráfica e exibição de caractere. Os módulos gráficos são encontrados com resoluções de 122x32, 128x64, 240x64 e 240x128 pixel, possuindo, geralmente, 20 pinos para conexão. Já os modelos comuns, tipo caractere, tem sua especificação em função dos números de linhas e colunas.

2.6 PROTEUS *DESIGN SUITE*

Proteus é um suíte, conjunto de *features*, desenvolvido pela Labcenter Eletronics Ltd. É um software para: simulação de microcontroladores, de circuitos eletrônicos e para desenvolvimento de placa de circuito impresso. Os componentes desse sistema são:

- ISIS *Schematic Capture*: Ferramenta utilizada para se adicionar objetos, componentes para simulação;
- PROSPICE *Mixed mode SPICE simulation*: Simulador industrial padrão SPICE3F5, combinado com simulador digital de alta velocidade. *Spice, Simulation Program with Integrated Circuit Emphasis*, é um simulador para propósitos gerais para sistemas eletrônicos analógicos.
- ARES PCB *Layout*: Sistema de PCB, *Printed Circuit Board*, ou melhor, placa de circuito impresso, design de alto desempenho com posicionamento automático de componente, auto-roteamento, entre outras *features* relacionadas ao desenvolvimento de placas de circuito impresso.
- VSM - *Virtual System Modelling*: Permite simular software embarcado para microcontroladores, disponíveis em suas bibliotecas, ao lado de seu projeto de hardware (LABCENTER, 2013; WIKIPEDIA, 2013).

2.6.1 ISIS *SCHEMATIC CAPTURE*

Como já foi dito, essa *feature* se trata da funcionalidade de simulação e montagem de circuitos eletrônicos. Permite que o circuito em questão seja depurado de forma simples e, além disso, possibilita uma integração com códigos desenvolvidos para microcontroladores suportados por ele. Seus componentes são:

- Circuitos integrados das famílias: 74ALS, 74AS, 74F, 74HC, 74HCT, 74LS, 74S e 74STD;
- Componentes analógicos;
- Medidores de corrente e tensão para serem utilizados durante a depuração;
- Geradores como fontes e *clock*, ambos ajustáveis para o valor desejado;

- Semicondutores como diodo, transistor, LEDs e outros;
- Componentes básicos como chaves, resistores, capacitores e indutores;
- Atuadores como motor DC, motor de passo e outros;
- Microcontroladores da família 80XXX, AT e PIC;
- Memórias, *displays*, CMOS e outros.

Seu uso é bem simples, pois é como se estivesse desenhando o circuito em um papel. Após ser desenhado é possível utilizar as ferramentas de auxilio para depurar e assim coletar dados como voltagem e corrente do circuito e até mesmo se o funcionamento está conforme o esperado.

2.7 MIKROC

MikroC é um *toolchain* poderoso, uma ferramenta de desenvolvimento completa para microcontroladores da família PIC. Criado de forma que disponibilize ao usuário a forma mais fácil de desenvolver suas aplicações para sistemas embarcados, sem compromisso de desempenho ou controle. MikroC permite que seja feito um rápido desenvolvimento e embarcar aplicações complexas:

- Escrever códigos em C utilizando um editor de código avançado;
- Utilizar as bibliotecas do próprio MikroC para acelerar o desenvolvimento: aquisição de dados, memória, *display*, comunicação, entre outros;
- Auxilia na evolução e desenvolvimento do código monitorando a estrutura do programa, variáveis e funções. Gera um código comentado, humanamente legível em assembler, e um padrão HEX compatível com o padrão conhecido;
- Possui um *debugger* integrado capaz de gerar relatórios detalhados, estatísticas, pilha de execução, entre outras opções que auxiliam analisar o fluxo do programa (MIKROELETRONIKA, 2006).

2.8 *OOC* - Object Orientated Programming in ANSI-C

Segundo (SCHREINER, 1993), OOC, *Object Orientated Programming in ANSI-C*, é uma forma de se aproveitar as diversas vantagens da Programação orientada objeto. Vantagens essas proporcionadas por linguagens como C++, java, python, smalltalk e outras. OOC, necessita da separação em dois arquivo distintos, semelhante a do c++: o primeiro possui a extensão

por padrão ".h"e contém apenas a declarações de interfaces e classes, já o segundo, com extensão ".c", possui a devida implementação. Este é um conceito considerado mais denso do que Orientação Objeto propriamente dito, isso se deve ao fato de que o usuário implemente componentes como:

- Interface: Define-se interface como uma *struct* com ponteiros para funções, de forma que o *bind* das funções é feito na hora da criação de um objeto de uma classe que implemente essa interface;
- Classe: Da mesma forma que as interfaces, é uma *struct* em que os atributos são ponteiros para função, entretanto tem os *binds* definidos em uma função de criação(construtor) pertencente a essa classe, ou módulo;
- Construtor: Função da classe em que faz o *bind* de cada ponteiro para cada função da classe, alocações dinâmicas e outras operações que podem ser definidas de acordo com a necessidade de cada classe;
- Destrutor: Basicamente a liberação de todos os recursos alocados pelo construtor e funções que a classe utilizou;
- Public: Declaração da variável como atributo da *struct* ou no arquivo *header*;
- Private: Declaração da variável fora da *struct*, no arquivo que tem a implementação das funções da classe.

O conceito de polimorfismo e abstração de dados se dá pela associação de um mesmo ponteiros de função de uma classe ou interface à funções diferente, variando em função da classe mais especializada que está sendo abstraída.

Obviamente é uma simulação da consolidada programação orientada objeto e possui limitações como: Programação genérica sem validação em tempo de compilação, sintaxe mais complexa, geração de código implícito, entre outros. Entretanto ainda permite reutilização de código e, como dito anteriormente, abstração de dados.

2.9 *Design Patterns*

Na engenharia de *software*, um *Design Patterns* ou padrão de projeto é uma solução repetível geral para um problema comumente ocorre em *design* de *software*. Um padrão de projeto não é um projeto acabado que pode ser transformado diretamente em código. É uma descrição ou modelo de como resolver um problema que pode ser utilizado em diversas situações diferentes. (SHALLOWAY; TROTT, 2004).

2.9.1 Usos de *Design Patterns*

Os *Design Patterns* podem acelerar o processo de desenvolvimento, fornecendo testados e comprovados paradigmas de desenvolvimento. O *design* de *software* eficaz requer considerar as questões que não podem tornar-se visível até mais tarde na implementação. Reutilizar *Design Patterns* ajuda a prevenir situações que podem causar grandes problemas e melhora a legibilidade do código para programadores e arquitetos familiarizados com os padrões.

Muitas vezes, as pessoas só entendem como aplicar certas técnicas de *design* de *software* para determinados problemas. Estas técnicas são difíceis de aplicar a uma ampla gama de problemas. Os *Design Patterns* fornecem soluções gerais, documentadas em um formato que não requer especificidades ligadas a um problema particular (SHALLOWAY; TROTT, 2004).

Segundo (SHALLOWAY; TROTT, 2004), além disso, os padrões permitem que os desenvolvedores se comuniquem usando nomes bem conhecidos e bem compreendidos para interações de software. *Design Patterns* comuns podem ser melhorados ao longo do tempo, tornando-os mais robustos do que os projetos ad-hoc, ou seja, problemas específicos. Os *Design Patterns* podem ser divididos em:

- *Creational design patterns*: Estes *Design Patterns* tem tudo a ver com padrões de instanciação de classe. Esse padrão pode ser dividido em padrões de criação de classe e de criação de objetos. Enquanto os padrões de criação de classe usam a herança de forma eficaz no processo de instanciação, os padrões de criação de objeto usam a delegação de forma eficaz para ter o trabalho de criação feito.

- *Structural design patterns*: Este *Design Pattern* tem tudo a ver com composição de classe e objetos. Padrão estrutural de classe usa a herança para compor interfaces. Padrão estrutural de objeto define formas de compor objetos para obter novas funcionalidades.

- *Behavioral design patterns*: São os padrões que são mais especificamente relacionadas com a comunicação entre objetos.

2.9.2 *Factory Method*

Segundo (SHALLOWAY; TROTT, 2004), *factory method* foi um dos *Design Patterns* escolhidos para esse projeto. Seus objetivos principais são:

- Definir uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe instanciar. Factory Method permite adiar a instanciação da classe para a subclasses;

- Definir um construtor *virtual*;

- Operador *new* é considerado perigoso.

Esse *Design Patterns* foi utilizado no módulo que faz uso do *display* de LCD para que a troca de seu tipo ficasse isolado do restante do sistema e que um futura troca possa ser feita com o menor impacto possível. A Figura 4 representa sua etsrutura em UML.

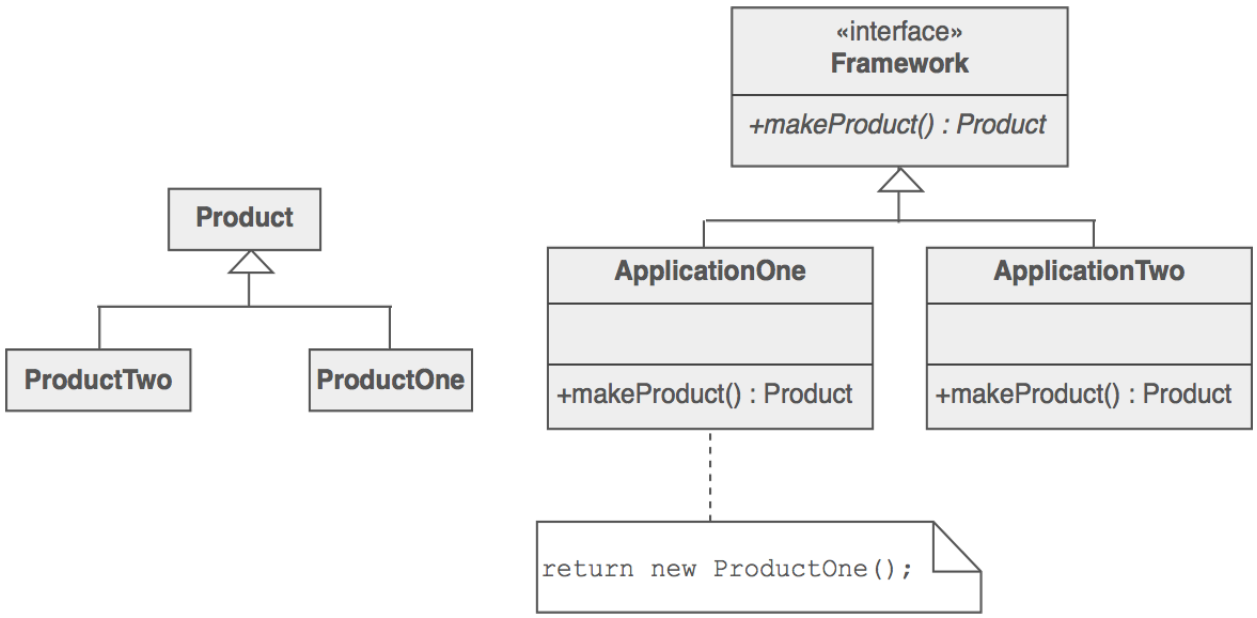

Figura 4 – Estrutura do padrão *Factory Method*

(SHALLOWAY; TROTT, 2004)

3 Bomba de infusão de insulina

3.1 Contextualização

Glicose, a principal fonte de energia do corpo humano, é absorvida a partir dos alimentos e distribuída ao corpo através da corrente sanguínea. Uma vez que está no sangue ela pode ser absorvida pelo fígado, temporariamente, utilizada pela células ou, em último caso, ser eliminada através da urina. Processos do corpo regulados pelos hormônios, que mantém a quantidade de glicose estável na corrente sanguínea. Dos hormônios existentes os mais importante é a insulina. Sua produção é feita pelo pâncreas e sua função é controlar absorção de glicose pelas células (SBC, 2014)

Segundo (SBC, 2014) existem conjunto de doenças crônicas, chamadas diabetes, que dificultam o metabolismo da glicose e da insulina. O diagnóstico dessa doença geralmente pode ser feita através da aferição da concentração de glicose na corrente sanguínea. O resultado positivo se dá em caso de ser constatado hiperglicemia, ou seja, elevada concentração de glicose.

Dentre os tipos de diabetes o mais raro é o tipo 1. O paciente com esse tipo de doença faz com que seja necessário a aplicações diárias de insulina, consequência da secreção desse hormônio. Esse tipo da doença ocorre por causa desconhecida ou, ainda sim, pela destruição das células betas pelo pâncreas, ocasionada por um processo auto-imune (MARTINS et al., 2012). Os principais tipos tratamentos para o quadro citado são:

- Infusão Contínua Subcutânea de Insulina através de uma bomba de infusão, situação abordada neste trabalho;
- Múltiplas Doses de Insulina (MDI).

3.2 Características gerais da bomba de infusão

Segundo (MINICUCCI, 2008), para gerenciar o processo de infusão de insulina no paciente é utilizado um dispositivo eletromecânico portátil conhecido como: bomba de infusão de insulina. A ideia desse aparelho é atuar como um pâncreas artificial, ou seja, atuar de forma similar ao organismo de uma pessoa que não possui diabetes, liberando insulina durante o dia todo e no horário das refeições. A primeira forma de funcionamento, liberação de insulina entre as refeições, é chamada de basal e a segunda bolus.

Bomba de infusão de insulina é um pequeno aparelho eletrônico, do tamanho de um celular ou pager, que está ligado ao corpo do portador da doença por um finíssimo cateter com

uma agulha flexível na ponta. É posicionada externamente ao corpo e possui um peso entre 80 e 100 gramas (MINICUCCI, 2008).

Segundo (AMORIM, 2008), o software embarcado, responsável pelo controle da bomba possui algumas funcionalidades como:

- avisos para monitoração da glicose;
- programação de doses de taxas basais para cada hora do dia;
- programação de doses de taxas bolus para certa quantidade de refeições;
- possibilidade de bloqueio do sistema voltados principalmente para utilização em crianças;
- possibilidade de escolha de menus operacionais;
- programação de quantidade de insulina a ser injetada;
- alarmes vibratórios e/ou sonoros para quantidade de insulina no reservatório;
- avisos para monitoração da glicose;
- ajuda de dolus que auxilia no cálculo da dose bolus necessária para correção de hiperglicemia e/ou alimentação (essa funcionalidade mais difícil de ser encontrada);
- dentre outras verificações de segurança.

Hoje em dia, a configurabilidade das funcionalidades da bomba de infusão é muito mais do que as funcionalidades básicas, o que aumentam seu custo e podem não agregar muito valor ao consumidor final. Funcionalidade essas como:

- integração com algum sistema de monitoração contínuo de glicose;
- sistema *bluetooth* onde o paciente utiliza um dispositivo externo para o controle da bomba;
- sistema de transferência de dados para um computador;
- lembretes;
- personalizações de menu;
- gráficos;
- visores coloridos;
- entre outros.

3.3 Aspectos de segurança da bomba de infusão

A bomba de infusão é considerado um sistema embarcado de tempo real crítico uma vez que dever ser extremamente seguro e fornecer respostas em prazos precisos e determinados. Considerações essas que implicam em diferenças importantes de projeto em relação a sistemas mais simples, pois complicações em sistemas desse tipo podem causar até mesmo a morte. Logo, esses dispositivos devem ser desenvolvidos com critérios de segurança robustos e requisitos funcionais e não-funcionais bem definidos (SOMMERVILLE, 2004).

É importante que o sistema seja confiável ao fornecer a quantidade correta de insulina solicitada, ou seja, ter disponibilidade sempre que for requisitada e, não menos importante, a bomba deve ser segura tratando falhas que possam suspender o fornecimento de insulina ou a infusão demasiada da mesma. Segundo (SOMMERVILLE, 2004), todas as condições anteriores retratam alguns requisitos de sistemas críticos, que no contexto deste trabalho podem ser representados por algumas funcionalidades essenciais, tais como:

- Alarmes para o nível da bateria;
- Sensor de pressão;
- Redundância de partes do hardware essenciais;
- Rotinas de verificação do funcionamento interno em geral (hardware e software);
- Implementação de rotinas que preveem iterações de forma errada por parte do paciente;
- Inacessibilidade de algumas configurações avançadas por meio do paciente;
- Alarmes para o nível de reservatório;
- Hardware de boa qualidade com as devidas certificações.

Segundo (ZHANG; JONES; JETLEY, 2010) pode-se listar situações referentes ao uso da bomba que podem causar alguma falha, podendo levar o paciente a perder a consciência no caso de uma hipoglicemia e cetoacidose para uma considerável hiperglicemia. A divisão é feita em seis grupos principais: causas operacionais, falhas de software, de hardware, causas ambientais, elétricas e químicas.

Levantamento este importantíssimo para servir como guia do projeto, de forma que se possa medir e garantir os mais altos níveis de segurança tanto no *software* quanto no *hardware*. Ainda existe classificações de riscos que podem ser energéticas, mecânicas, biológicas, químicas, ambiental e terapêutica (ZHANG; JONES; JETLEY, 2010).

Exemplos de causas operacionais que levam a situação de riscos:

- Bomba está desconectada do conjunto de infusão sem o conhecimento do paciente;

- Excessiva administração da taxa de bolus devido a várias requisições do paciente;
- Vazamento da bomba;
- Taxa atual de infusão não está de acordo com o programado;

Exemplos de falhas de *software* que levam a situação de riscos:

- *Looping* infinito;
- Acesso indevido da memória;
- Taxa incorreta de bolus recomendada pelo cálculo do sistema;
- Estouro de pilha.

Exemplos de falhas de *hardware* que levam a situação de riscos:

- Falha na memória ROM ou flash;
- Falhas do motor;
- Falha no sensor de reservatório de insulina;
- Falha no sensor de bateria;
- Falha no microcontrolador.

Exemplos de causas ambientais que levam a situação de riscos:

- Aquecimento da bomba durante o funcionamento;
- Alta diferença de pressão entre o interior da bomba e o ambiente externo;
- Uso da bomba em temperaturas fora do especificado;
- Uso da bomba em ambientes com alta umidade.

Exemplos de causas elétricas que levam a situação de riscos:

- Interferência eletromagnética vinda de outros aparelhos eletrônicos;
- Bateria desconectada;
- Vazamento de corrente pela superfície da bomba;
- Nível de bateria baixo;

- Descarga eletrostática.

Exemplos de causas químicas e/ou biológicas que levam a situação de riscos:

- Material do equipamento de baixa qualidade, problemas alérgicos;
- Infecção na região (pele) da infusão;
- Perda das propriedades bioquímicas da insulina durante a infusão, hiperglicemia;
- Precipitação química dentro do cateter, problemas alérgicos, infecções;
- Equipamento não higienizado, problemas alérgicos, infecções.

3.4 Requisitos da bomba de infusão

A bomba será implementada utilizando um microcontrolador de baixo custo da família PIC, de forma a manter as características mais importante, qualidade e segurança do dispositivo. A Figura 5 representa o modelo do contexto da bomba de infusão de insulina, com todas as entidades que deverão ser monitoradas e/ou controladas pelo software.

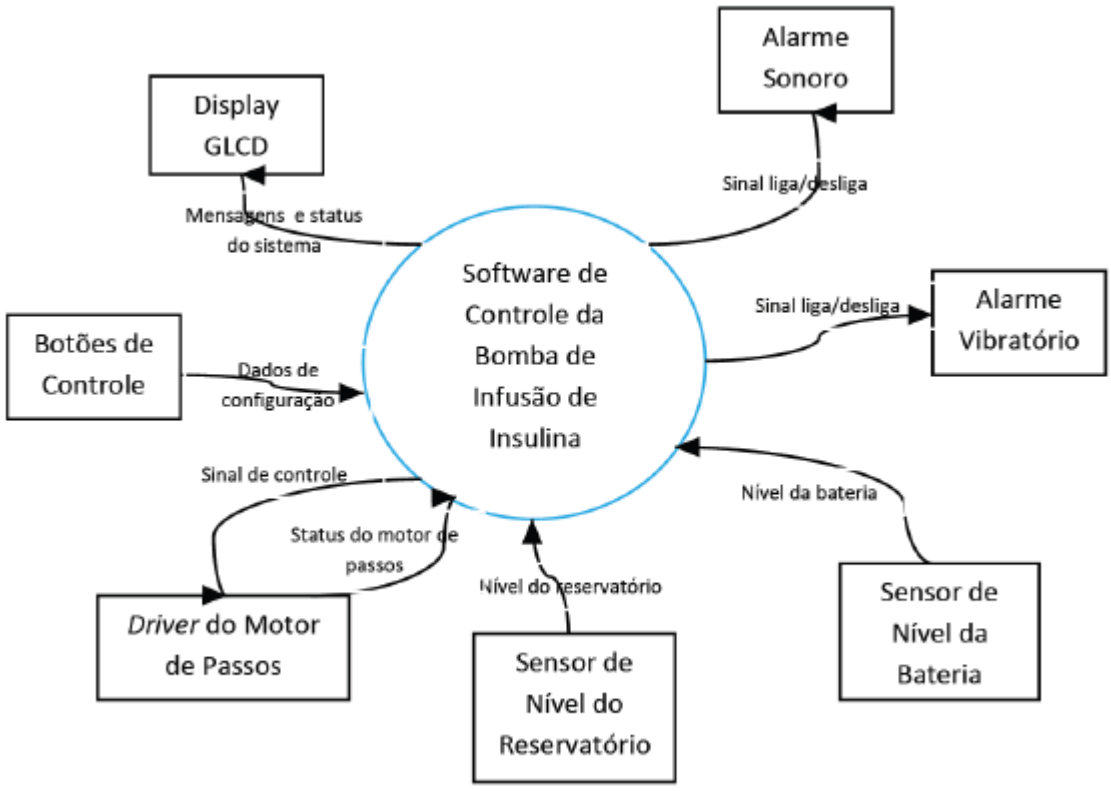

Figura 5 – Modelo de contexto da bomba de infusão de insulina

(MARTINS et al., 2012)

A divisão dos requisitos funcionas da bomba foi feita da seguinte forma:

- Módulo de interface com o usuário;
- Módulo de controle de infusão;
- Módulo de monitoramento de sensores.

Abaixo segue figuras para representar os módulos do sistema citados anteriormente. A Figura 6 representa o módulo de interface com o usuário, pode-se observar as possíveis ações que o usuário pode realizar de forma a interagir com o sistema. A Figura 7 representa o segundo módulo, controle de infusão, que tem a responsabilidade de administrar as duas formas de infusão: basal e bolus. O último, módulo de monitoramento dos sensores, está representado na Figura 8 e é responsável pela verificação do sistema como um todo.

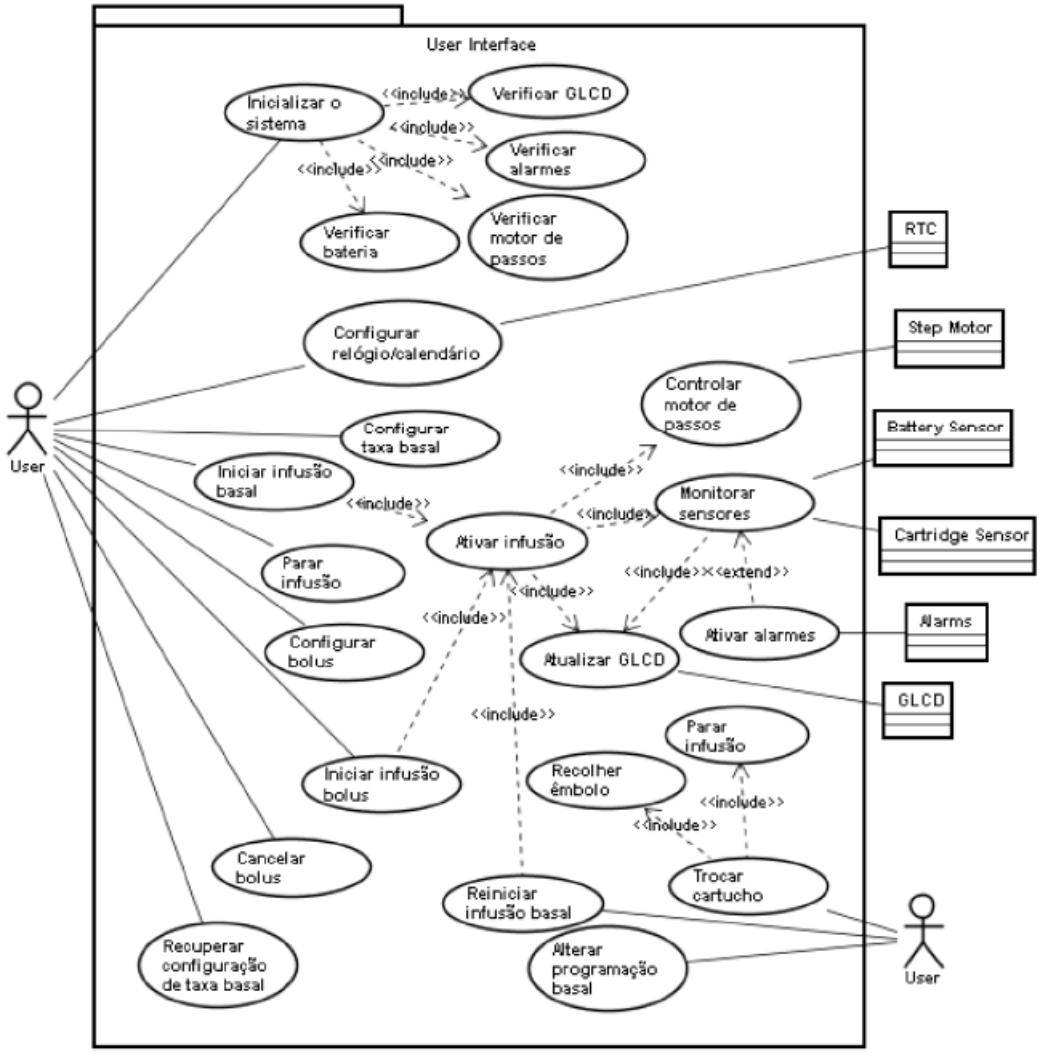

Figura 6 – Caso de uso do módulo de interface com o usuário

(MARTINS et al., 2012)

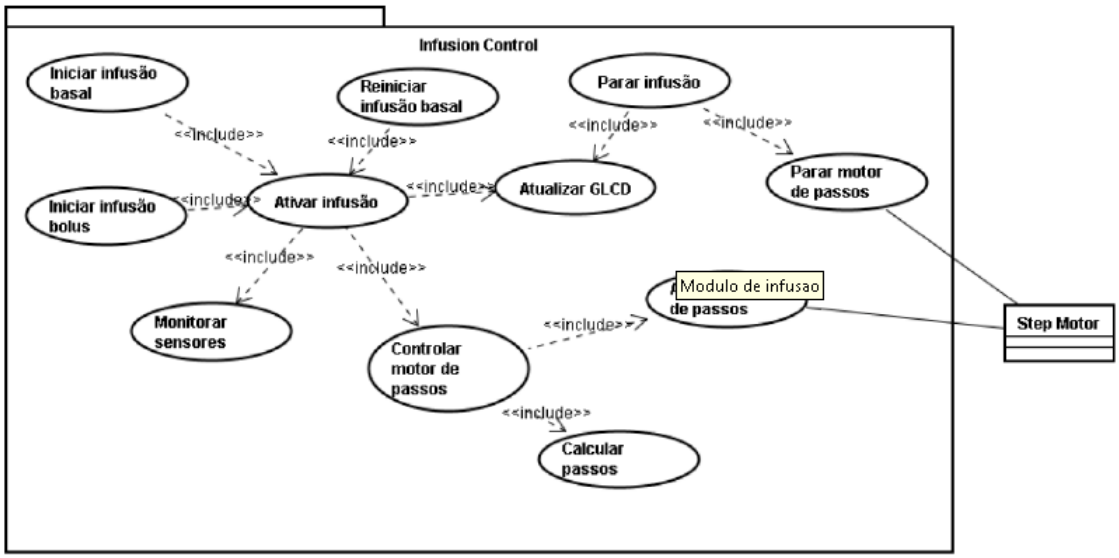

Figura 7 – Caso de uso do módulo de controle de infusão

(MARTINS et al., 2012)

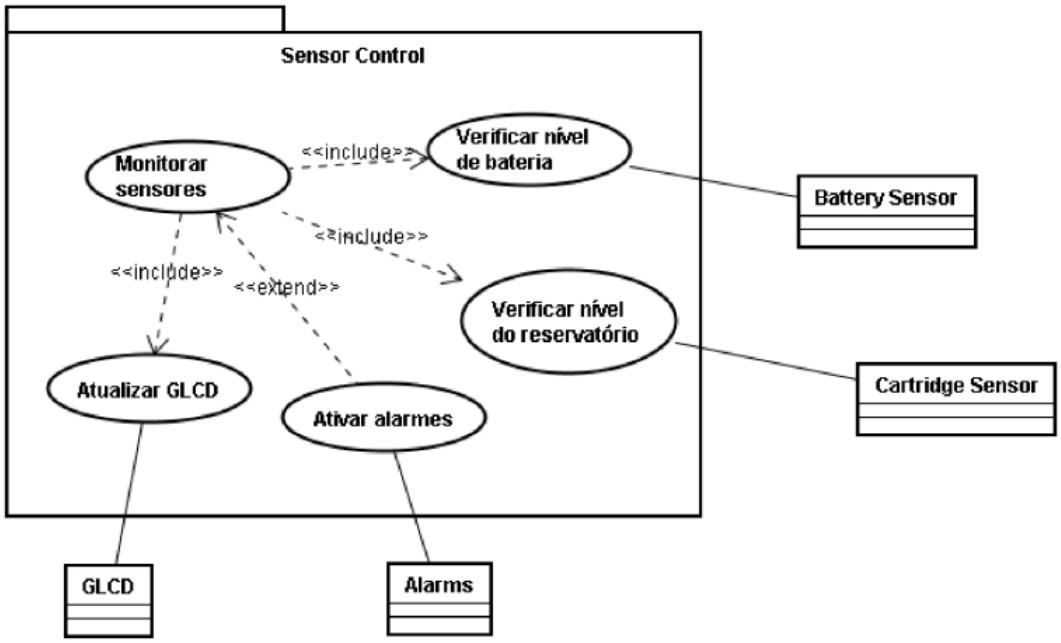

Figura 8 – Caso de uso do módulo de monitoramento dos sensores

(MARTINS et al., 2012)

4 Módulo de controle de infusão de insulina

Módulo desenvolvido é responsável por controlar o funcionamento da bomba em função dos parâmetros recebidos e pré estabelecidos. É possível configurar a quantidade a ser infundido no período de 24 horas, infusão basal, e, além disso e mais importante, o desenvolvimento foi feito em cima do simulador proteus e devido a possibilidade de mudanças possíveis o sistema foi montado de forma que deixa os módulos desacoplados. Para isso foi utilizado o conceito OOC, *Object Orientated Programming in ANSI-C*, o que possibilitou o uso de *Design Patterns*.

4.1 Proteus

No desenvolvimento deste TCC o simulador Proteus foi usado para montar o ambiente de testes. Para testes, o circuito teve seu desenvolvimento em função dos seguintes componentes: microcontrolador da família PIC, PIC18F452, motor de passo, *display* de LCD 2x16, botões e driver para o motor de passo. A Figura 9 representa a imagem do *Software*.

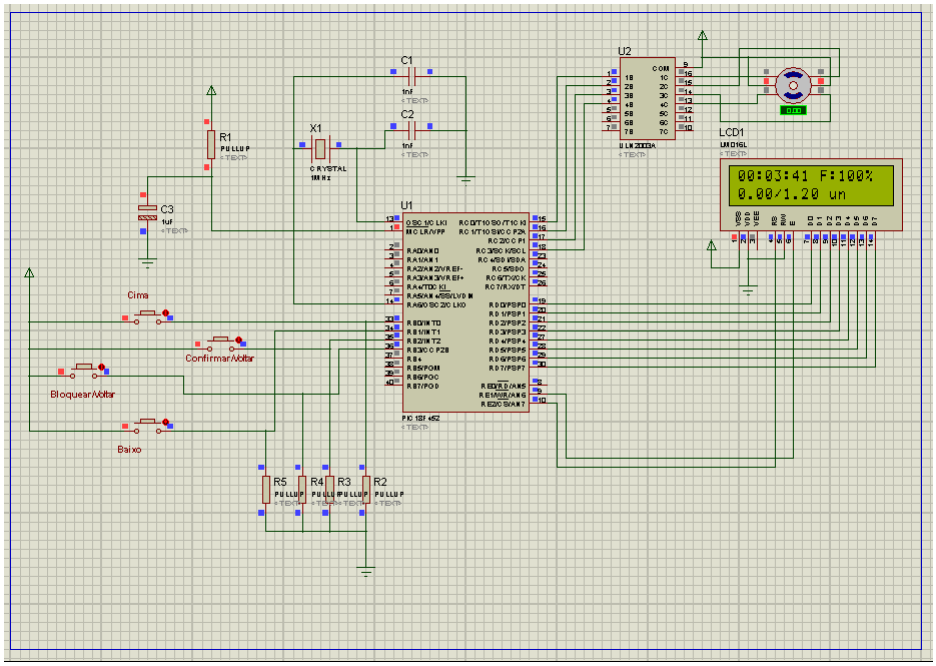

Figura 9 – *Software* Proteus

4.2 Arquitetura do Software

O *Software* foi desenvolvido de forma modular para que as funções fossem facilmente testadas, modificadas e evoluídas, a Figura 10 representa a estrutura citada. A divisão do *Software* foi feita da seguinte forma: config, insulimpump, lcd, menu, motor, timerMotor e principal.

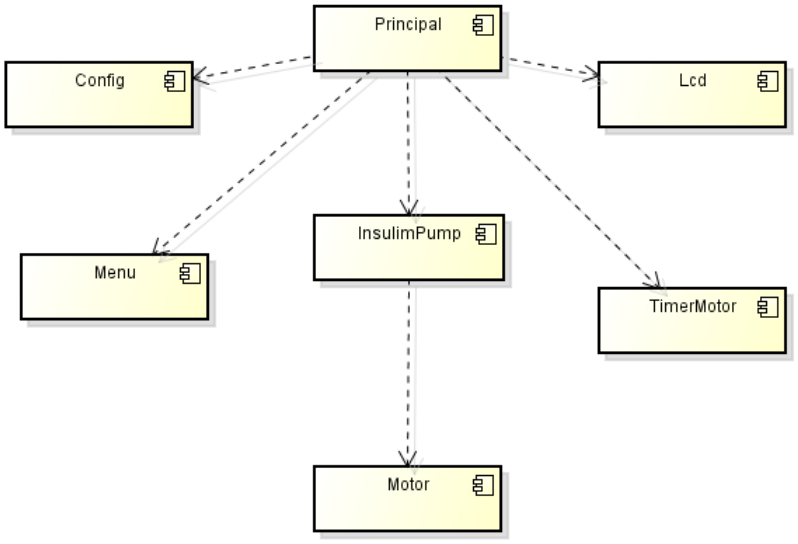

Figura 10 – Estrutura do *Software*

4.2.1 Módulo Config

Módulo feito para centralizar todas as configurações do sistema, utilizado, basicamente, por quase todos os demais módulos. Seguindo parte do conceito de OOC, pois foi possível utilizar a ideia de *private* e *public* para as variáveis. Entretanto não foi criada nenhuma *struct* de forma que representasse uma classe. A Figura 11 representa o diagrama de classe desse módulo.

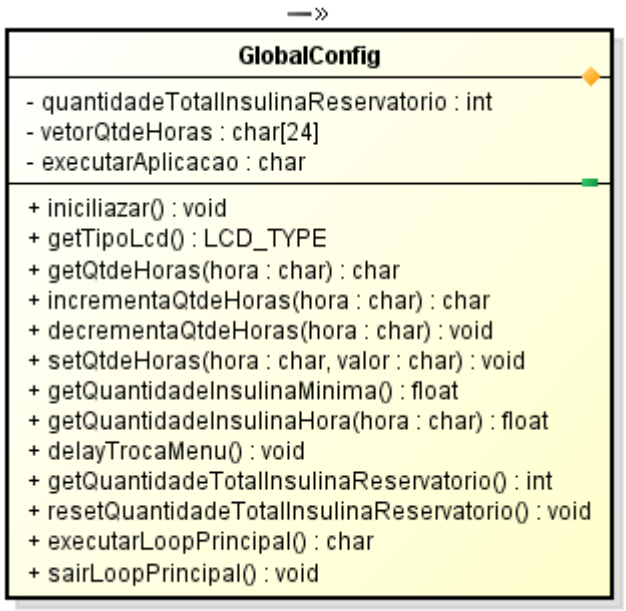

Figura 11 – Diagrama de classe GlobalConfig

Quando se diz *private* e *public* é o fato de adicionar as variáveis no arquivo GlobalConfig.h, *private*, ou GlobalConfig.c, *public*. A principal parametrização desse módulo é a quantidade que o repositório de insulina suporta. Para modificar seu valor basta alterar o *define* QUANTIDADE_TOTAL_RESERVATORIO_INSULINA. Esse valor representa a quantidade de insulina existente no reservatório na escala da infusão mínima da bomba.

4.2.2 Módulo InsulinPump

O módulo InsulinPump é responsável pela abstração das funções da bomba para o resto do sistema. Funções simples para o resto do sistema como: inicializar variáveis de controle, iniciar operação, parar operação - principalmente para configurações da bomba -, injetar e retornar a quantidade inserida naquela hora. Seguindo o conceito OOC foi criada uma "interface", IInsulinPump, e a "classe"concreta InsulinPump. A Figura 12 representa a relação e diagrama de classe dos elementos citados anteriormente.

4.2.3 Módulo Lcd

Esse módulo é responsável pelo isolamento do *display* de LCD do resto do sistema. Composto por um *factory*, que retorna um obeto de controle de acordo com o parâmetro passado, uma "interface"para possibilitar essa abstração e as classes concretas, no caso existe apenas a classe concreta para *display* 2x16(2 linhas por 16 colunas). A Figura 13 representa a relação e diagrama de classe dos elementos citados anteriormente.

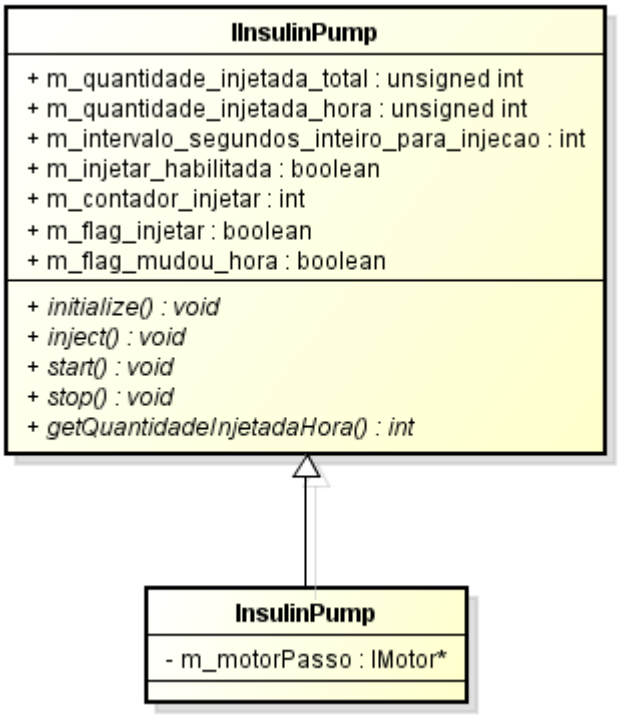

Figura 12 – Diagrama de classe InsulinPump

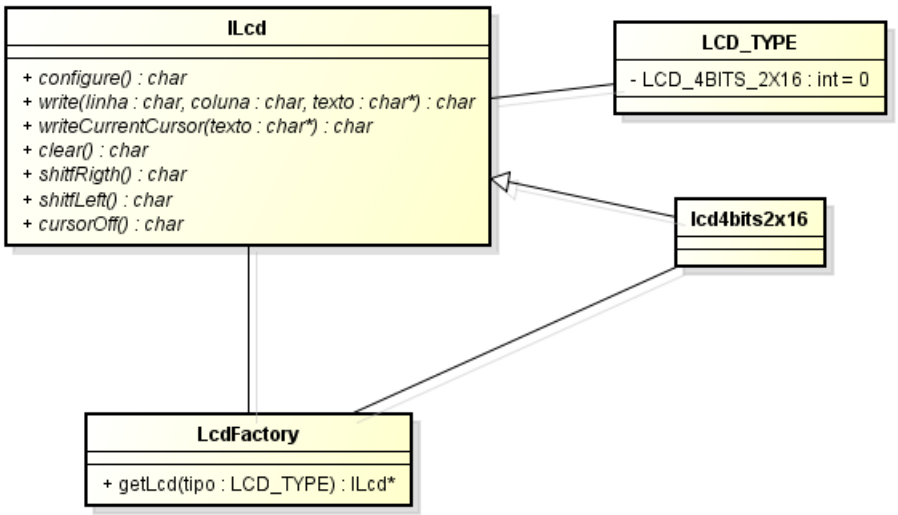

Figura 13 – Diagrama de classe Lcd

4.2.4 Módulo Menu

O módulo de menu foi criado para abstrair todos as possíveis situações e estados da bomba. Foi criado de uma forma muito simples e é equivalente a uma máquina de estado. Utiliza uma "inteface"que é implementado por todos os tipos de menu existentes como: menu de confuguração da bomba, menu da bomba em execução, menu de confirmação do estado do reservatório e outros. A Figura 14 representa as relações entre os componentes, ou "classes", do módulo e demonstra como é simples a criação de novos menus.

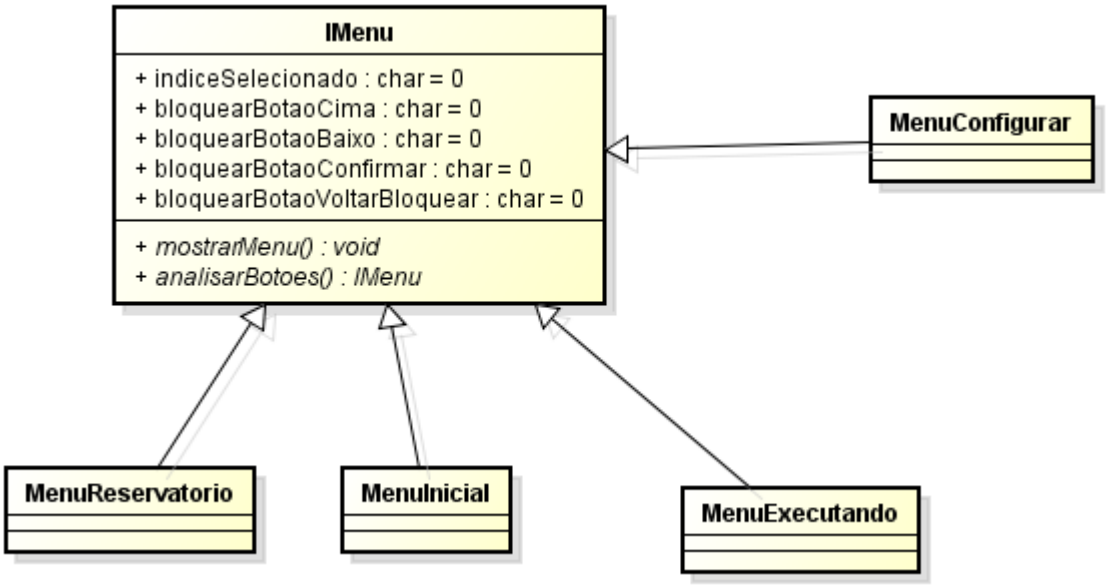

Figura 14 – Diagrama de classe Menu

4.2.5 Módulo Motor

O módulo motor, tem como resposabilidade abstrair todas as operações necessárias para o controle do motor de forma que o resto do sistema não saiba qual motor está utilizando. Isso é possível devido a "interface"criada para abstração e o uso de um *Factory* que retorna o motor que deve ser utilizado de acordo com um parâmetro global, localizado no módulo config. A figura 15 representa as relações existentes nesse módulo.

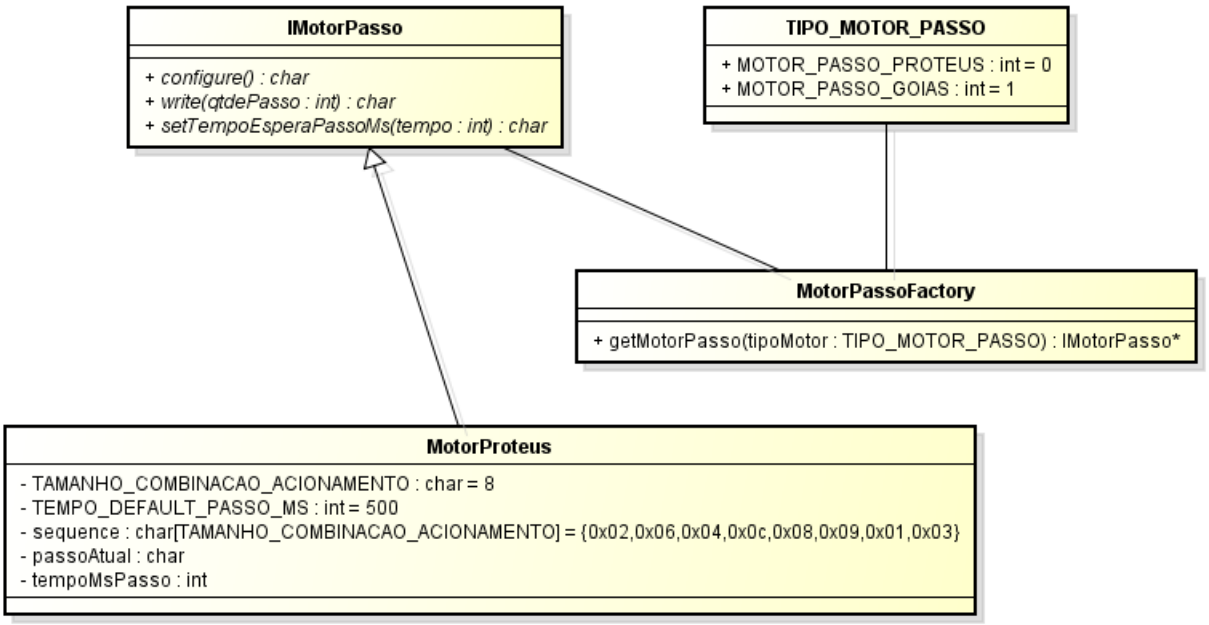

Figura 15 – Diagrama de classe Motor

4.2.6 Módulo TimerMotor

O módulo TimerMotor armazena todos os dados do contador para o uso dos motores. Abstrai a configuração do timer em função do *hardware* para o resto do sistema. Dessa forma o sistema só precisa se preocupar em: fazer configuração inicial(inicializar variáveis), iniciar timer, parar timer. A Figura 16 representa a relação citada acima.

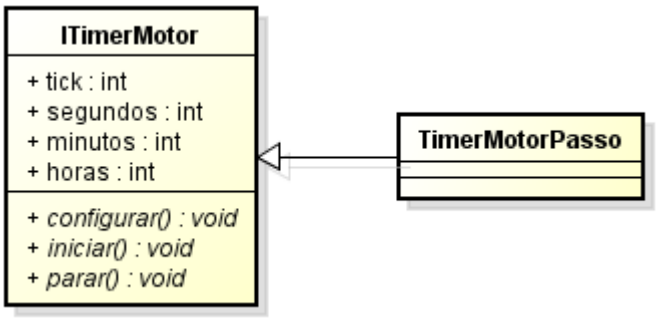

Figura 16 – Diagrama de classe TimerMotoror

4.2.7 Módulo Principal

Devido a forma modular com que foi implementado o sistema esse tornou-se o módulo mais simples. Ficou responsável apenas pela interrupção, específico do hardware, chamar os métodos de incialização dos módulos do lcd, TimerMotor e InsulinPump, e um loop principal, extremamente simples, que exibe o Menu, correspondente ao estado atual, e recebe o retorno do próprio menu em questão para navegar entre os menus existente.

5 Análise e discussão dos resultados

5.1 Simulador

O uso do simulador foi importantíssimo para o desenvolvimento desse trabalho. Ele possibilitou iniciar o desenvolvimento sem uma placa física. Utilizando-o montou-se um circuito exemplo utilizando o microcontrolador escolhido e, de forma iterativa, era possível evoluir tanto o circuito quanto o software, por exemplo: montou-se o circuito com o LCD e implementou-se o módulo de controle do mesmo, adicionou-se os componentes de menu e a análise de botões, e outros passos. Vendo esses passos é possível perceber que o desenvolvimento foi feito por funcionalidade isoladas de forma que fosse possível testar cada "parte"do software e assim minimizar os problemas do *software* como um todo e assim sempre ter um versão estável, mesmo que mínima.

Além de todas essas facilidades, a gama de testes e depuração é maior em um simulador do que no *hardware*, como: o simulador indica más práticas existentes, permite simular o uso da bomba por dias em questão de minutos, basta configurar o tempo de execução, e outros. Graças aos exemplos citados anteriormente foi possível encontrar diversos problemas que só seriam descobertos quando estivesse testando diretamente no *hardware* e mesmo assim surgiria a dúvida: É o *hardware* ou o *software*.

O circuito elétrico montado no simulador para testes foi desenvolvido com base na placa Microgenios, representada pela Figura 17. Todos os componentes utilizados e suas ligações são fiéis à placa de referência, salvo o uso do motor de passo que não faz parte do conjunto de referência. Com isso, uma migração para a placa física da Microgenios manteria o funcionamento de todos os componentes exceto o motor de passo.

E, além disso, como os testes e validações do *hardware* feito através simulador são extremamente válidas e consistentes e o desenvolvimento foi baseado em uma placa consolidada, uma integração com uma placa diferente da utilizada como referência será muito mais simples. Isso deve-se ao fato através do simulador diminui-se os riscos e problemas no período de integração, onde são encontrados as maiores e inesperadas dificuldades do desenvolvimento.

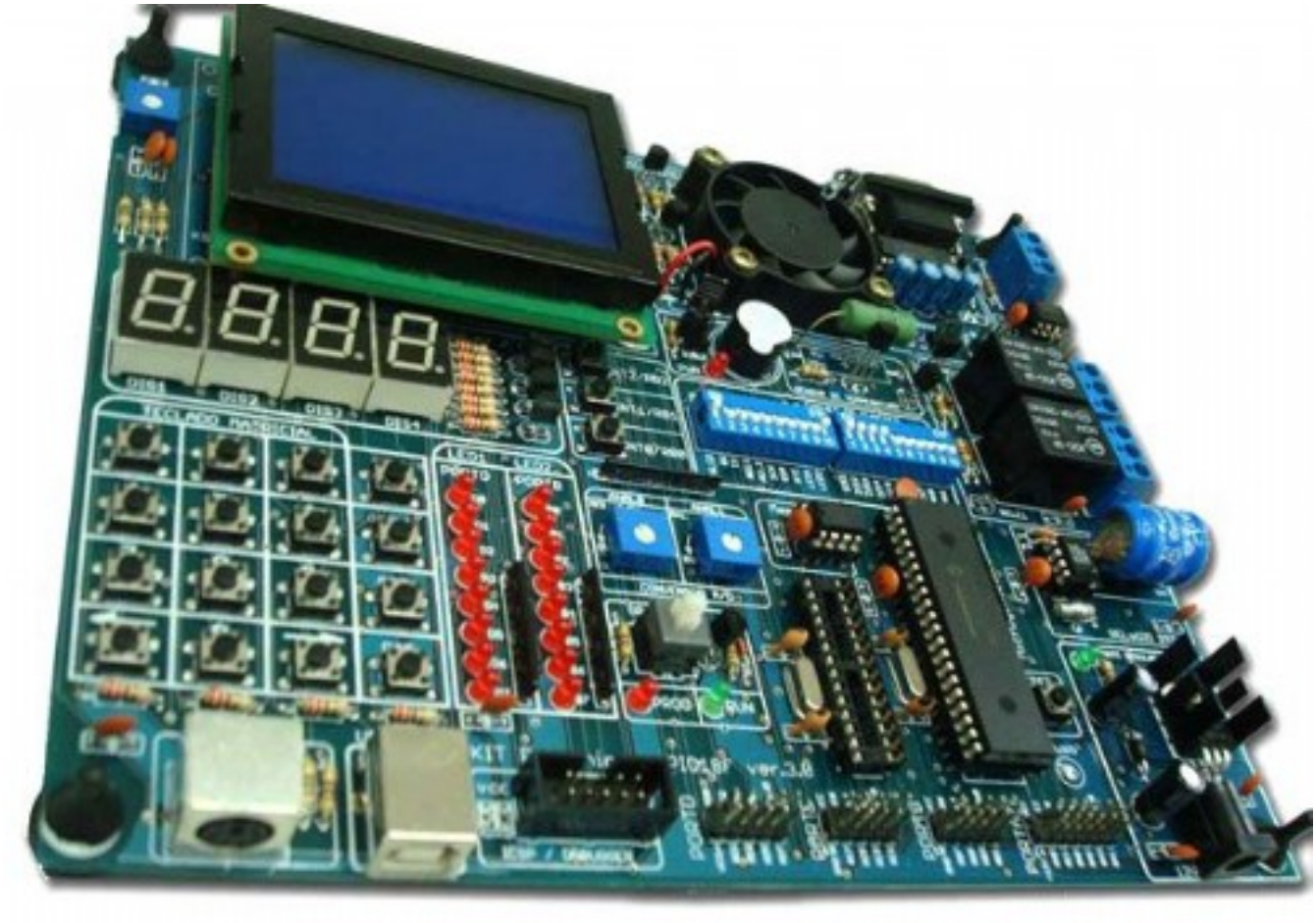

Figura 17 – Placa Microgenios PIC18F4452

(ELETRONICABRASIL, 2014)

5.2 Cálculo por Referência

Os cálculos do sistema foram todos baseados em número inteiros. Isso devido ao fato de que o uso de ponto flutuante pode causar imprecisões além de consumir mais memória. Para isso toda internamente toda contagem das infusões está em função da quantidade mínima de infusão, 0,1, ou seja, caso seja preciso infundir 1 unidade de insulina internamente o sistema intende como 10. Isso também permite trocar o valor de quantidade minima a ser infundida sem impactar no funcionamento do resto do sistema.

5.3 Arquitetura modularizada

A organização do código é o grande destaque e vantagem desse trabalho. O desacoplamento foi o foco principal durante todo o desenvolvimento isso para facilitar mudanças no próprio *software* ou do *hardware*.

Esse desacoplamento alcançado deveu-se graças ao uso do conceito OOC, *Object Orientated Programming in ANSI-C*. Através de seu uso pode-se aproveitar das diversas vantagens que a Programação Orientada Objetos proporciona, embora seja um conceito considerado mais volumoso já que seu uso requer um conhecimento de como as linguagens OO abstraem os operadores desse paradigma, pois são implementados manualmente, como:

- *Bind*: É a associação de uma chamada a um método ou função com sua implementação, de forma mais específica é definir para quem o ponteiro da função está endereçado;
- Interface: Definição dada por uma *struct* com ponteiros para funções. O *bind* é feito no momento da criação do objeto da classe que implemente essa interface;
- Classe: Análogo às interfaces, é uma *struct* em que os atributos são ponteiros para função, entretanto tem os *binds* definidos em uma função de criação(construtor) pertencente a essa classe, ou módulo;
- Construtor: Função da classe responsável por realizar o *bind* de cada ponteiro para cada função da classe, alocações dinâmicas e outras operações que podem ser definidas de acordo com a necessidade de cada classe;
- Destrutor: Basicamente a liberação de todos os recursos alocados pelo construtor e funções que a classe utilizou;
- Public: Para declarar um atributo com esse operador de acesso deve-se declarar a variável como atributo da *struct* ou no arquivo header, seja da classe ou da interface;
- Private: Para esse operador a declaração da variável se dá fora da *struct*, não pode ser no *header*, pois é o arquivo conhecido pelo resto do sistema.
- Polimorfismo e abstração de dados: Resultado da associação de um mesmo ponteiros de função de uma classe ou interface à funções diferente, variando em função da classe mais especializada que está sendo abstraída.

Mas obviamente OOC é limitado e algumas complicações em sua adaptação para o paradigma OO. Validações em tempo de compilação são menos específicas, programação genérica, sintaxe mais complexa, geração de código implícito, entre outros.

Essa abordagem foi importantíssima, pois é possível fazer testes, mocks no sistema, pelo fato de simular programação orientação à objetos, possibilitando realizar testes unitários e utilizando um PC. Uma vez que as classes e interfaces possuem ponteiros de funções, para que o *bind* seja feito nas funções de criação, a realização do mock é muito simples, pois basicamente é fazer com que o ponteiro da função a ser mockada seja endereçada para outra de acordo com o teste a ser realizado. Ao realizar os testes e mocks o acesso aos periféricos, todos os

acionamentos e comunicações, ou seja, tudo vinculado ao *hardware* foram redirecionados para um arquivo log de acompanhamento. A definição da arquitetura foi totalmente desenvolvida em função da possibilidade de realização de testes unitários, uma vez que para se realizar os testes é preciso um alto nível de desacoplamento. Os testes foram realizados antes da unificação de todos os módulos do projeto, pois o módulo TimerMotor causou complicações uma vez que não se alcançou o desacoplamento total do *hardware* devido ao *timer* do microcontrolador. A seguir está descrito as vantagens principais de cada módulo devido a forma de implementação.

O módulo Config é o mais simples de todos, sua principal vantagem é o fato de centralizar todas as informações de configuração, fazendo com que o sistema fique mais claro. Além disso, criar casos de testes torna-se simples, pois mudando qualquer uma de suas informações já se reflete no sistema como um todo. É importante lembrar também que ele não carrega nenhuma dependência do compilador ou do hardware, foi implementado em ANSI-C, o que permite que seja utilizado por qualquer compilador e *hardware*.

Os demais módulos foram criados para seguir a ideia de isolação do módulo config. Para isso o que foi feito é deixar as interfaces e funções comuns independente de *hardware* e compilador utilizando apenas ANSI-C. Dessa forma, caso precise de alguma mudança mais drástica relacionada aos dois itens citados o impacto seja mínimo, precisando fazer apenas uma equivalência das funcionalidades específicas.

Tendo dito as vantagens com relação à mudanças *hardware* e compilador é importante lembrar que a expansibilidade e manutenibilidade do software ficou incrivelmente simples. A ideia foi deixar o "motor"ou "coração"da bomba ter conhecimento apenas das "interface"do sistemas. Dessa forma os detalhes de funcionamento, requisitos de segurança, configurações específicas de hardware para controle dos periféricos e outros, podem ser modificadas sem impactar o funcionamento básico da bomba de infusão. O mais importante é que essas alterações são parametrizáveis, localizando-se no módulo config, e o software sabe quais objetos utilizar e de onde recuperá-los, em algum *Factory* na maioria dos casos. Devido a isso tudo é possível manter mais de um produto em um único código e mudanças importantes no *core* do sistema não precisa ser replicadas e correr risco de conflito entre projetos. Graças ao uso do OOC para criar algo novo basta implementar as funções da interface do módulo em questão e adicionar ao *factory*, caso exista. Segue uma breve explicação do que pode ser feito em cada módulo.

O módulo InsulinPump, responsável por abstrair particularidades de funcionalidades e segurança da bomba para o resto do ambiente, permite criar diversos tipos de bomba. Essa criação leva em conta mudanças nos dois itens citados, por exemplo, bomba europeia, americana, brasileira, que podem ter requisitos de segurança e funcionalidades distintas.

O módulo LCD, responsável por abstrair a forma de uso e qual *display* está sendo utilizado. Ele permite a troca do tipo de LCD utilizado seja simples, por exemplo, trocar o utiliza 2x16, por um 2x12.

O módulo Menu, lembra uma máquina de estado, pois retorna um próximo Menu(estado) ou ele mesmo caso a mudança não seja necessária. Para adicionar um novo menu basta colocá-lo como retorno em alguma situação dentro da função de análise de botões que todos os Menus possuem.

O módulo Motor segue a mesma linha do LCD, permite a troca do periférico por outro motor de passo ou até mesmo um outro tipo de motor. Esse módulo possui um *factory* para que seja possível recuperar o objeto de abstração do motor desejado.

Por fim, o módulo TimerMotor foi criado para desvincular o *timer* que é extremamente dependente do *hardware* e compilador. Foi o módulo mais complicado de se isolar e é o único que não está totalmente isolado devido a algumas limitações da execução de interrupções.

6 Conclusão

Este trabalho teve como base o desenvolvimento de um *software* de controle para um protótipo de bomba de infusão de insulina baseado em um microcontrolador da família PIC. Além disso, desenvolveu-se juntamente com o software de controle um módulo de comunicação com periféricos para interação com o usuário. O *software* desenvolvido é responsável por executar os perfis de infusão configurados pelo o usuário, ativar o motor de passos, dar *feedback* do status da bomba e de sua execução ao paciente. Os objetivos desse trabalho foram alcançados ao desenvolver o *software* utilizando o microcontrolador de baixo custo da família PIC: PIC18F452.

A compreensão das tecnologias utilizadas, simulador e, é claro, da natureza do problema proposto foram imprescindíveis. O presente trabalho pode ser considerado uma prova de conceito do módulo desenvolvido, e pode ser continuado e aprimorado sem grandes problemas devido a sua alta modularidade e simplicidade do código. E com isso realizar testes mais precisos e factíveis com relação a realidade do problema proposto.

Considerando como projeto futuro seria os testes utilizando um *hardware*, placa da Microgenios citada anteriormente, composto pelos componentes abordados e comprovar as vantagens do uso do simulador, demonstrar que testes de *stress*, ou seja, por longos períodos pode sem simulado no PC, graças ao OOC. E, por fim, talvez menos importante, conseguir isolar completamente a dependência do compilador e *hardware* devido à forma de uso da interrupção para contagem de tempo para infusão.

Por fim, o código é aberto, licença MIT e pode ser encontrado em: `https://gitlab.com/dinesh/insulin_pump`.

Referências

ACARNLEY, P. P. *Stepping motors: a guide to theory and practice*. [S.l.]: Iet, 2002. Citado na página 37.

ALBERTI, K. G. M. M.; ZIMMET, P. f. Definition, diagnosis and classification of diabetes mellitus and its complications. part 1: diagnosis and classification of diabetes mellitus. provisional report of a who consultation. *Diabetic medicine*, Wiley Online Library, v. 15, n. 7, p. 539–553, 1998. Citado na página 23.

ALMEIDA, P. A. d. O. *TRANSDUTORES PARA MEDIDA DE DESLOCAMENTOS LINEARES*. 2004. Acessado em 20/07/2014. Disponível em: <http://www.lem.ep.usp.br-/pef5003/TRANSDUTORES-1.pdf>. Citado na página 35.

AMORIM, A. C. P. d. Novas abordagens em insulinoterapia. Universidade da Beira Interior, 2008. Citado na página 46.

BARBACENA, I.; AFONSO, C. F. *DISPLAY LCD*. UNICAMP, 1996. Acessado em 20/06/2014. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/ea079/complementos/Lcd.pdf>. Citado na página 37.

BODE, B. W.; TAMBORLANE, W. V.; DAVIDSON, P. C. Insulin pump therapy in the 21st century. *Postgrad Med*, v. 111, n. 69-77, p. 9, 2002. Citado na página 25.

CHIRONIS, N. P.; SCLATER, N. *Mechanisms & mechanical devices sourcebook*. [S.l.]: McGraw-Hill New York, 1991. Citado na página 36.

CUNHA, A. *O que são sistemas embarcados?* 2013. Acessado em 20/05/2014. Disponível em: <http://www.techtraining.eng.br/files/uploads/2013/04/19/artigo-sist-emb.pdf>. Citado na página 31.

DIABETES. *Bomba de infusão de insulina*. 2013. Acessado em 08/03/2013. Disponível em: <http://www.diabetes.org.br/sala-de-noticias/2316-bombas-de-infusao-de-insulina>. Citado na página 26.

ELETRONICABRASIL. *pic18f452 microgenios*. 2014. Acessado em 02/02/2014. Disponível em: <http://www.eletronicabrasil.com.br/>. Citado na página 60.

FARINES, J.-M.; FRAGA, J. d. S.; OLIVEIRA, R. d. Sistemas de tempo real. *Escola de Computação*, v. 2000, p. 201, 2000. Citado na página 32.

FELDMANN, R. L. et al. A survey of software engineering techniques in medical device development. In: IEEE. *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007. HCMDSS-MDPnP. Joint Workshop on*. [S.l.], 2007. p. 46–54. Citado na página 32.

GANSSLE, J. *The art of designing embedded systems*. [S.l.]: Newnes, 1999. Citado na página 33.

GERMANO, A. *Você sabe o que são sistemas embarcado?* 2011. Acessado em 20/05/2014. Disponível em: <http://www.gruponetcampos.com.br/2011/05/voce-sabe-o-que-sao-sistemas-embarcados/>. Citado na página 31.

JORDAN, P. Standard iec 62304-medical device software-software lifecycle processes. In: IET. *Software for Medical Devices, 2006. The Institution of Engineering and Technology Seminar on*. [S.l.], 2006. p. 41–47. Citado na página 32.

KOPETZ, H. *Real-time systems: design principles for distributed embedded applications*. [S.l.]: Springer, 2011. Citado na página 32.

LABCENTER. *Labcenter Eletronics*. 2013. Acessado em 05/03/2013. Disponível em: <http://www.labcenter.com/index.cfm>. Citado na página 38.

LEE, E. A.; SESHIA, S. A. *Introduction to embedded systems: A cyber-physical systems approach*. [S.l.]: Lee & Seshia, 2011. Citado na página 33.

LI, Q.; YAO, C. *Real-time concepts for embedded systems*. [S.l.]: CRC Press, 2003. Citado na página 31.

MAGNOLTI, M. A.; RAYFIELD, E. J. An update on insulin injection devices. *Insulin*, Elsevier, v. 2, n. 4, p. 173–181, 2007. Citado 2 vezes nas páginas 23 e 24.

MALL, R. *Real-Time Systems: Theory and Practice*. [S.l.]: Pearson Education India, 2009. Citado na página 32.

MARTINS, L. E. G. et al. Desenvolvimento de um protótipo de bomba de infusão de insulina de baixo custo: em busca de uma alternativa para a população de baixa renda portadora de diabetes. Projeto submetido ao MCTI, 2012. Citado 4 vezes nas páginas 45, 49, 50 e 51.

MIKROELETRONIKA. *MikroC Making it Simple User's Manual*. [S.l.], 2006. Disponível em: <http://www.mikroe.com/pdf/mikroc/mikroc_manual.pdf>. Acesso em: 16.4.2013. Citado na página 39.

MINICUCCI, W. J. Uso de bomba de infusão subcutânea de insulina e suas indicações:[revisão]. *Arq. bras. endocrinol. metab*, v. 52, n. 2, p. 340–348, 2008. Citado 2 vezes nas páginas 45 e 46.

NOMADSUSP. *Sensores*. 2012. Acessado em 02/04/2014. Disponível em: <http://www-.nomads.usp.br/pesquisas/design/dos/Capacitacao/arquivos/sensores.pdf>. Citado na página 36.

PORTALDIABETES. *Iniciando insulinoterapia no diabetes mellitus tipo 2*. 2008. Acessado em 04/03/2013. Disponível em: <http://www.portaldiabetes.com.br/conteudocompleto-.asp?idconteudo=3267>. Citado na página 25.

PORTALDIABETES. *Bombas de Infusão de Insulina*. 2009. Acessado em 10/03/2013. Disponível em: <http://www.portaldiabetes.com.br/conteudocompleto.asp?idconteudo=3267>. Citado na página 26.

ROSÁRIO, J. M. *Princípios de mecatrônica*. [S.l.]: Pearson Prentice Hall, 2006. Citado na página 36.

SANTOS, V. P. de A. Motor de passo. 2008. Citado na página 37.

SBC. *Tudo sobre diabetes*. 2014. Acessado em 20/05/2014. Disponível em: <http://www-.diabetes.org.br>. Citado 2 vezes nas páginas 26 e 45.

SCHREINER, A.-T. Object oriented programming with ansi-c. Hanser, 1993. Citado na página 39.

SHALLOWAY, A.; TROTT, J. R. *Design patterns explained: a new perspective on object-oriented design*. [S.l.]: Pearson Education, 2004. Citado 3 vezes nas páginas 40, 41 e 43.

SOMMERVILLE, I. *Software Engineering. International computer science series*. [S.l.]: Addison Wesley, 2004. Citado 2 vezes nas páginas 32 e 47.

WIKIPEDIA. *SPICE*. 2013. Acessado em 06/03/2013. Disponível em: <http://en.wikipedia-.org/wiki/SPICE>. Citado na página 38.

ZHANG, Y.; JONES, P. L.; JETLEY, R. A hazard analysis for a generic insulin infusion pump. *Journal of diabetes science and technology*, SAGE Publications, v. 4, n. 2, p. 263–283, 2010. Citado na página 47.

Apêndices

APÊNDICE A – Config

A.1 Config.c

```
#include "GlobalConfig.h"

#define QUANTIDADE_TOTAL_RESERVATORIO_INSULINA 100
int quantidadeTotalInsulinaReservatorio =
   QUANTIDADE_TOTAL_RESERVATORIO_INSULINA;
char vetorQtdeHoras[24];
char executarAplicacao = 1;

void iniciliazar(){
  vetorQtdeHoras[0] = 0;
  vetorQtdeHoras[1] = 0;
  vetorQtdeHoras[2] = 0;
  vetorQtdeHoras[3] = 0;
  vetorQtdeHoras[4] = 0;
  vetorQtdeHoras[5] = 0;
  vetorQtdeHoras[6] = 0;
  vetorQtdeHoras[7] = 0;
  vetorQtdeHoras[8] = 0;
  vetorQtdeHoras[9] = 0;
  vetorQtdeHoras[10] = 0;
  vetorQtdeHoras[11] = 0;
  vetorQtdeHoras[12] = 0;
  vetorQtdeHoras[13] = 0;
  vetorQtdeHoras[14] = 0;
  vetorQtdeHoras[15] = 0;
  vetorQtdeHoras[16] = 0;
  vetorQtdeHoras[17] = 0;
  vetorQtdeHoras[18] = 0;
  vetorQtdeHoras[19] = 0;
  vetorQtdeHoras[20] = 0;
  vetorQtdeHoras[21] = 0;
```

```
  vetorQtdeHoras[22] = 0;
  vetorQtdeHoras[23] = 0;
}

#ifdef __cplusplus
extern "C"
#endif
LCD_TYPE getTipoLcd(){
    return LCD_4BITS_2X16;
}

char getQtdeHoras(char hora){
  return vetorQtdeHoras[hora];
}

char incrementaQtdeHoras(char hora){
    ++vetorQtdeHoras[hora];
    return vetorQtdeHoras[hora];
}

char decrementaQtdeHoras(char hora){
    if(vetorQtdeHoras[hora] > 0){
       --vetorQtdeHoras[hora];
    }

    return vetorQtdeHoras[hora];
}

void setQtdeHoras(char hora, char valor){
  vetorQtdeHoras[hora] = valor;
}

float getQuantidadeInsulinaMinima(){
    return 0.1f;
}

float getQuantidadeInsulinaHora(char hora){
    return getQuantidadeInsulinaMinima() * ( (float)
       vetorQtdeHoras[hora]);
```

```
}

void delayTrocaMenu(){
    Delay_ms(50);
}

int getQuantidadeTotalInsulinaReservatorio(){
    return quantidadeTotalInsulinaReservatorio;
}

void resetQuantidadeTotalInsulinaReservatorio(){
      quantidadeTotalInsulinaReservatorio =
         QUANTIDADE_TOTAL_RESERVATORIO_INSULINA;
}

char executarLoopPrincipal(){
      return executarAplicacao;
}

void sairLoopPrincipal(){
      executarAplicacao = 0;
}
```

A.2 Config.h

```
#ifndef GLOBAL_CONFIG_H
#define GLOBAL_CONFIG_H

#include "../lcd/lcd.h"

#define TRUE 1
#define FALSE 0
typedef char boolean;

void iniciliazar();
LCD_TYPE getTipoLcd();
char getQtdeHoras(char hora);
char incrementaQtdeHoras(char hora);
char decrementaQtdeHoras(char hora);
void setQtdeHoras(char hora, char valor);
```

```
float getQuantidadeInsulinaMinima();
float getQuantidadeInsulinaHora(char hora);
void delayTrocaMenu();
int getQuantidadeTotalInsulinaReservatorio();
void resetQuantidadeTotalInsulinaReservatorio();
char executarLoopPrincipal();
void sairLoopPrincipal();
#endif
```

APÊNDICE B – InsulinPump

B.1 IInsulinPump.h

```
#ifndef IINSULIN_PUMP_H
#define IINSULIN_PUMP_H

#include "../config/GlobalConfig.h"

extern unsigned int m_quantidade_injetada_total;
extern unsigned int m_quantidade_injetada_hora;
extern int m_intervalo_segundos_inteiro_para_injecao;
extern boolean m_injetar_habilitada;
extern int m_contador_injetar;
extern boolean m_flag_injetar;
extern boolean m_flag_mudou_hora;

typedef struct{
    void (*initialize)();
    void (*inject)();
    void (*start)();
    void (*stop)();
    int (*getQuantidadeInjetadaHora) ();
} IInsulinPump;

#endif
```

B.2 InsulinPump.c

```
#include "InsulinPump.h"
#include "../config/GlobalConfig.h"
#include "../timerMotor/TimerMotorPasso.h"
#include "../Motor/MotorPassoFactory.h"
#include "../lcd/lcd.h"
#include "../lcd/lcdFactory.h"
```

```
#define QUANTIDADE_SEGUNDOS_EM_HORA 3600
#define QUANTIDADE_PASSOS_INFUSAO_MINIMA 8

unsigned int m_quantidade_injetada_total;
unsigned int m_quantidade_injetada_hora;
int m_intervalo_segundos_inteiro_para_injecao;
boolean m_injetar_habilitada;
int m_contador_injetar;
boolean m_flag_injetar;
boolean m_flag_mudou_hora;

// publicas
void inicializePump();
void pumpInject();
void startInject();
void stopInject();
int getQuantidadeInjetadaHora();

void configuraHoraAtual();

IMotorPasso* m_motorPasso;

IInsulinPump InsulinPump = {
    &inicializePump,
    &pumpInject,
    &startInject,
    &stopInject,
    &getQuantidadeInjetadaHora
};

void inicializePump(){
      m_quantidade_injetada_total = 0;
      m_quantidade_injetada_hora = 0;
      m_intervalo_segundos_inteiro_para_injecao = 0;
      m_injetar_habilitada = FALSE;
      m_contador_injetar = 0;
      m_flag_injetar = FALSE;
      m_motorPasso = getMotorPasso( MOTOR_PASSO_PROTEUS );
```

```
      m_motorPasso->configure();
      m_flag_mudou_hora = FALSE;
      startInject();
}

void pumpInject(){
  if( m_flag_mudou_hora == TRUE ){
    configuraHoraAtual();
  }

  if( m_injetar_habilitada == TRUE ){
       if( m_flag_injetar == TRUE ){
           ++m_quantidade_injetada_total;
           ++m_quantidade_injetada_hora;
           m_flag_injetar = FALSE;
           m_motorPasso->write(QUANTIDADE_PASSOS_INFUSAO_MINIMA
              );
       }
     }
}

void startInject(){
      configuraHoraAtual();
      m_injetar_habilitada = TRUE;
}

void stopInject(){
      m_injetar_habilitada = FALSE;
}

int getQuantidadeInjetadaHora() {
    return m_quantidade_injetada_hora;
}

void configuraHoraAtual(){
      // a cada quantos segundos intejar
      char qtdeHora = getQtdeHoras(horas);
      if( qtdeHora > 0 )
           m_intervalo_segundos_inteiro_para_injecao =
```

```
                QUANTIDADE_SEGUNDOS_EM_HORA/qtdeHora;
        else
            m_intervalo_segundos_inteiro_para_injecao = 0;

        m_contador_injetar = m_contador_injetar %
           m_intervalo_segundos_inteiro_para_injecao;

}
```

B.3 InsulinPump.h

```
#ifndef INSULIN_PUMP_H
#define INSULIN_PUMP_H

#include "IInsulinPump.h"

extern IInsulinPump InsulinPump;

#endif
```

APÊNDICE C – Lcd

C.1 lcd.h

```
#ifndef LCD_H
#define LCD_H

typedef struct{
    char (*configure)();
    char (*write)( char, char, char* );
    char (*writeCurrentCursor)( char* );
    char (*clear)();
    char (*shitfRigth)();
    char (*shitfLeft)();
    char (*cursorOff)();
} ILcd;

typedef enum  { LCD_4BITS_2X16 = 0 } LCD_TYPE;

#endif
```

C.2 lcdFactory.h

```
#ifndef LCDFACTORY_H
#define LCDFACTORY_H

#include "lcd.h"

#ifdef __cplusplus
extern "C"
#endif
ILcd* getLcd( LCD_TYPE tipo );

#endif // LCDFACTORY_H
```

C.3 lcdFactory.c

```
#include "lcdFactory.h"
#include "lcd4bits2x16/lcd4bits2x16.h"

#ifdef __cplusplus
extern "C"
#endif
ILcd* getLcd( LCD_TYPE tipo )
{
    ILcd* retorno;

    switch( tipo )
    {
    case LCD_4BITS_2X16: retorno = &lcd4bits2x16; break;
    default: retorno = 0;
    }

    return retorno;
}
```

C.4 lcd4bits2x16

C.4.1 lcd4bits2x16.h

```
#ifndef LCD4BITS2X16_H
#define LCD4BITS2X16_H

#include "../lcd.h"

extern ILcd lcd4bits2x16;

#endif // LCD4BITS2X16_H
```

C.4.2 lcd4bits2x16.c

```
#include "Lcd4Bits2x16.h"

// LCD module connections
sbit LCD_RS at RE2_bit;
sbit LCD_EN at RE1_bit;
```

```
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;

sbit LCD_RS_Direction at TRISE2_bit;
sbit LCD_EN_Direction at TRISE1_bit;
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;

char configureLcd4Bits2x16();
char writeLcd4Bits2x16( char, char, char* );
char writeCurrentCursorLcd4Bits2x16( char* );
char clearLcd4Bits2x16();
char shitfRigthLcd4Bits2x16();
char shitfLeftLcd4Bits2x16();
char cursorOffLcd4Bits2x16();

ILcd Lcd4Bits2x16 = {
    &configureLcd4Bits2x16,
    &writeLcd4Bits2x16,
    &writeCurrentCursorLcd4Bits2x16,
    &clearLcd4Bits2x16,
    &shitfRigthLcd4Bits2x16,
    &shitfLeftLcd4Bits2x16,
    &cursorOffLcd4Bits2x16
};

/*
obs: a emissao do registrador ADCON1 nos programas, acarreta o
   nao acionamento adequado do PORTA e consequentemente provoca
    o mal funcionamento do programa.
O pino RA4 do PIC eh dreno aberto, isto implica que devemos
   conectar resitor de pull-up quando formos trabalhar este
   pino como I/O de uso geral.
O PORTE tambem possui a funcao de conversores analogicos ou
```

```
   digitais tambem. Devemos configurar o ADCON1 para
   trabalharmos o PORTE como IO de uso geral.
*/
char configureLcd4Bits2x16(){
  trise = 0; // pinos D como saida
  ADCON1 = 0x06; // Todos os pinos A/D como I/O uso geral
    Lcd_Init();
    //Lcd8_Config(&PORTE,&PORTD,2,1,0,7,6,5,4,3,2,1,0);
    return 0;
}

char writeLcd4Bits2x16( char linha, char coluna, char* texto ){
     trisd = 0; // pinos D como saida
     Lcd_Out( linha, coluna, texto );
     trisd = 255; // pinos D como entrada
     return 0;
}

char writeCurrentCursorLcd4Bits2x16( char* texto ){
    trisd = 0; // pinos D como saida
    Lcd_Out_CP( texto );
    trisd = 255; // pinos D como entrada
    return 0;
}

char clearLcd4Bits2x16(){
    trisd = 0; // pinos D como saida
    Lcd_Cmd( _LCD_CLEAR );
    trisd = 255; // pinos D como entrada
    return 0;
}

char shitfRigthLcd4Bits2x16(){
    trisd = 0; // pinos D como saida
     Lcd_Cmd(_LCD_SHIFT_RIGHT);
    trisd = 255; // pinos D como entrada
     return 0;
}
```

```
char shitfLeftLcd4Bits2x16(){
     trisd = 0; // pinos D como saida
      Lcd_Cmd(_LCD_SHIFT_RIGHT);
     trisd = 255; // pinos D como entrada
      return 0;
}

char cursorOffLcd4Bits2x16(){
     trisd = 0; // pinos D como saida
      Lcd_Cmd(_LCD_CURSOR_OFF);
     trisd = 255; // pinos D como entrada
      return 0;
}
```

APÊNDICE D – Menu

D.1 IMenu.h

```
#ifndef IMENU_H
#define IMENU_H

static char indiceSelecionado = 0;
static char bloquearBotaoCima = 0;
static char bloquearBotaoBaixo = 0;
static char bloquearBotaoConfirmar = 0;
static char bloquearBotaoVoltarBloquear = 0;

typedef struct stIMenu{
    void (*mostrarMenu)();
    struct stIMenu* (*analisarBotoes)();
} IMenu;

#endif
```

D.2 MenuConfigurar.h

```
#ifndef MENU_CONFIGURAR_H
#define MENU_CONFIGURAR_H

#include "IMenu.h"

extern IMenu MenuConfigurar;

#endif
```

D.3 MenuConfigurar.c

```
#include "MenuConfigurar.h"
#include "../lcd/lcd.h"
```

```
#include "../lcd/lcdFactory.h"
#include "../config/GlobalConfig.h"
#include "MenuInicial.h"
#include "../config/GlobalConfig.h"

void mostrarMenuConfiguracao();
IMenu* analisarBotoesMenuConfiguracao();

IMenu MenuConfigurar = {
    &mostrarMenuConfiguracao,
    &analisarBotoesMenuConfiguracao
};

void mostrarMenuConfiguracao(){
   ILcd* lcd = getLcd( getTipoLcd() );
   char convercao[7];
   float quantidade;

   sprintf(convercao, "Hora: %d", indiceSelecionado+1);
   lcd->write(1,1, convercao);
   quantidade = ((float) getQtdeHoras(indiceSelecionado)) *
                        getQuantidadeInsulinaMinima();
   sprintf(convercao, "%.2f un", quantidade);
   lcd->write(2,1, convercao);
}

IMenu* analisarBotoesMenuConfiguracao(){
   ILcd* lcd = getLcd( getTipoLcd() );
       // pra cima
    if(button(&PORTB,0,1,0)){
       bloquearBotaoCima = 1;
    }
    else if(bloquearBotaoCima && button(&PORTB,0,1,1)){
       incrementaQtdeHoras(indiceSelecionado);
       bloquearBotaoCima = 0;
    }
   // Baixo
    if(button(&PORTB,1,1,0)){
       bloquearBotaoBaixo = 1;
```

```
    }
    else if(bloquearBotaoBaixo && button(&PORTB,1,1,1)){
        decrementaQtdeHoras(indiceSelecionado);
        bloquearBotaoBaixo = 0;
    }

    // confirmar
    if(button(&PORTB,2,1,0)){
        bloquearBotaoConfirmar = 1;
    }
    else if(bloquearBotaoConfirmar && button(&PORTB,2,1,1)){
        bloquearBotaoConfirmar = 0;
        if(indiceSelecionado >= 23) {
            indiceSelecionado = 0;
            lcd->clear();
            delayTrocaMenu();
            return &MenuInicial;
        }
        else{
            ++indiceSelecionado;
        }
    }

    // voltar-bloquear
    if(button(&PORTB,3,1,0)){
        bloquearBotaoVoltarBloquear = 1;
    }
    else if(bloquearBotaoVoltarBloquear && button(&PORTB,3,1,1)
        ){
        bloquearBotaoVoltarBloquear = 0;
        if(indiceSelecionado > 0) {
            --indiceSelecionado;
        }
        else{
            lcd->clear();
            indiceSelecionado = 0;
            delayTrocaMenu();
            return &MenuInicial;
        }
```

```
    }

    return &MenuConfigurar;
}
```

D.4 MenuExecutando.h

```
#ifndef MENU_EXECUTANDO_H
#define MENU_EXECUTANDO_H

#include "IMenu.h"

extern IMenu MenuExecutando;

#endif
```

D.5 MenuExecutando.c

```
#include "MenuExecutando.h"
#include "../lcd/lcd.h"
#include "../lcd/lcdFactory.h"
#include "../config/GlobalConfig.h"
#include "MenuInicial.h"
#include "../timerMotor/TimerMotorPasso.h"
#include "../insulinPump/InsulinPump.h"

void mostrarMenuExecutando();
IMenu* analisarBotoesExecutando();

IMenu MenuExecutando = {
    &mostrarMenuExecutando,
    &analisarBotoesExecutando
};

void mostrarMenuExecutando(){
  int porcentagem = 100*(getQuantidadeTotalInsulinaReservatorio
     () - m_quantidade_injetada_total)/
     getQuantidadeTotalInsulinaReservatorio();
  ILcd* lcd = getLcd( getTipoLcd() );
  unsigned char converter[15];
```

```
  memset(&converter, 0x00, sizeof(converter));

  sprintf(converter,"%02d:%02d:%02d F:%02d%%", horas, minutos,
     segundos, porcentagem);
  lcd->write(1,1, converter);

  sprintf(converter,"%.2f/%.2f un", (float)((float)InsulinPump.
     getQuantidadeInjetadaHora()*getQuantidadeInsulinaMinima())
     ,
                                    getQuantidadeInsulinaHora(
                                       horas));
  lcd->write(2,1, converter);
}

IMenu* analisarBotoesExecutando(){
  ILcd* lcd = getLcd( getTipoLcd() );
  // confirmar
  if(button(&PORTB,2,1,0)){
     bloquearBotaoConfirmar = 1;
  }
  if(bloquearBotaoConfirmar && button(&PORTB,2,1,1)){
    lcd->clear();
    bloquearBotaoConfirmar = 0;
    delayTrocaMenu();
    InsulinPump.stop();
    return &MenuInicial;
  }

   // voltar-vloquear
  if(button(&PORTB,3,1,0)){
     bloquearBotaoVoltarBloquear = 1;
  }
  if(bloquearBotaoVoltarBloquear && button(&PORTB,3,1,1)){
    lcd->clear();
    bloquearBotaoVoltarBloquear = 0;
    delayTrocaMenu();
    InsulinPump.stop();
    return &MenuInicial;
  }
```

```
    delayTrocaMenu();
    return &MenuExecutando;
}
```

D.6 MenuInicial.h

```
#ifndef MENU_INICIAL_H
#define MENU_INICIAL_H

#include "IMenu.h"

extern IMenu MenuInicial;

#endif // LCD4BITS2X16_H
```

D.7 MenuInicial.c

```
#include "MenuInicial.h"
#include "../lcd/lcd.h"
#include "../lcd/lcdFactory.h"
#include "../config/GlobalConfig.h"
#include "MenuConfigurar.h"
#include "MenuExecutando.h"
#include "../insulinPump/InsulinPump.h"

void mostrarMenuInicial();
IMenu* analisarBotoesMenuInicial();

IMenu MenuInicial = {
    &mostrarMenuInicial,
    &analisarBotoesMenuInicial
};

void mostrarMenuInicial(){
    ILcd* lcd = getLcd( getTipoLcd() );

    if( indiceSelecionado == 0 ){
        lcd->write(1,1, "- Iniciar");
    }
```

```
    else{
        lcd->write(1,1, "  Iniciar");
    }

    if( indiceSelecionado == 1 ){
        lcd->write(2,1, "- Config. basal");
    }
    else{
        lcd->write(2,1, "  Config. basal");
    }
}

IMenu* analisarBotoesMenuInicial(){
    ILcd* lcd = getLcd( getTipoLcd() );
        // pra cima
     if(button(&PORTB,0,1,0)){
        bloquearBotaoCima = 1;
     }
     if(bloquearBotaoCima && button(&PORTB,0,1,1)){
        bloquearBotaoCima = 0;
        if(indiceSelecionado == 0)
            indiceSelecionado = 1;
        else if(indiceSelecionado == 1)
            indiceSelecionado = 0;
     }
    // Baixo
     if(button(&PORTB,1,1,0)){
        bloquearBotaoBaixo = 1;
     }
     if(bloquearBotaoBaixo && button(&PORTB,1,1,1)){
        bloquearBotaoBaixo = 0;
        if(indiceSelecionado == 0)
            indiceSelecionado = 1;
        else if(indiceSelecionado == 1)
            indiceSelecionado = 0;
     }

     // confirmar
     if(button(&PORTB,2,1,0)){
```

```
        bloquearBotaoConfirmar = 1;
    }
    if(bloquearBotaoConfirmar && button(&PORTB,2,1,1)){
      bloquearBotaoConfirmar = 0;
      lcd->clear();
       if(indiceSelecionado == 0){
          delayTrocaMenu();
          InsulinPump.start();
          return &MenuExecutando;
       }
       else if(indiceSelecionado == 1) {
          indiceSelecionado = 0;
          delayTrocaMenu();
          return &MenuConfigurar;
       }
    }

    delayTrocaMenu();
    return &MenuInicial;
}
```

D.8 MenuReservatorio.h

```
#ifndef MENU_RESERVATORIO_H
#define MENU_RESERVATORIO_H

#include "IMenu.h"

extern IMenu MenuReservatorio;

#endif
```

D.9 MenuReservatorio.c

```
#include "MenuReservatorio.h"
#include "../lcd/lcd.h"
#include "../lcd/lcdFactory.h"
#include "../config/GlobalConfig.h"
#include "MenuInicial.h"
#include "../timerMotor/TimerMotorPasso.h"
```

```
char reservatoriCheioSim = 0;

void mostrarMenuReservatorio();
IMenu* analisarBotoesMenuReservatorio();

IMenu MenuReservatorio = {
    &mostrarMenuReservatorio,
    &analisarBotoesMenuReservatorio
};

void mostrarMenuReservatorio(){
  ILcd* lcd = getLcd( getTipoLcd() );
  unsigned char converter[15];
  memset(&converter, 0x00, sizeof(converter));

  sprintf(converter,"%02d:%02d:%02d␣F:%.0f%%", horas, minutos,
     segundos, 100);
  lcd->write(1,1, "Insulina␣cheia");
  if(reservatoriCheioSim == 0){
    lcd->write(2,1, "(␣)␣sim,␣(-)␣nao");
  }
  else{
    lcd->write(2,1, "(-)␣sim,␣(␣)␣nao");
  }
}

IMenu* analisarBotoesMenuReservatorio(){
  ILcd* lcd = getLcd( getTipoLcd() );
      // pra cima
    if(button(&PORTB,0,1,0)){
       bloquearBotaoCima = 1;
    }
    if(bloquearBotaoCima && button(&PORTB,0,1,1)){
       bloquearBotaoCima = 0;
       if(reservatoriCheioSim == 0)
          reservatoriCheioSim = 1;
       else if(reservatoriCheioSim == 1)
          reservatoriCheioSim = 0;
```

```
    }
   // Baixo
    if(button(&PORTB,1,1,0)){
        bloquearBotaoBaixo = 1;
    }
    if(bloquearBotaoBaixo && button(&PORTB,1,1,1)){
        bloquearBotaoBaixo = 0;
        if(reservatoriCheioSim == 0)
            reservatoriCheioSim = 1;
        else if(reservatoriCheioSim == 1)
            reservatoriCheioSim = 0;
    }

  // confirmar
  if(button(&PORTB,2,1,0)){
     bloquearBotaoConfirmar = 1;
  }
  if(bloquearBotaoConfirmar && button(&PORTB,2,1,1)){
    if(reservatoriCheioSim == 1)
        resetQuantidadeTotalInsulinaReservatorio();

    lcd->clear();
    bloquearBotaoConfirmar = 0;
    delayTrocaMenu();
    return &MenuInicial;
  }

  delayTrocaMenu();
  return &MenuReservatorio;
}
```

APÊNDICE E – Motor

E.1 IMotorPasso.h

```
#ifndef IMOTORPASSO_H
#define IMOTORPASSO_H

typedef struct{

    char (*configure)();
    char (*write)( int );
    char (*setTempoEsperaPassoMs)( int );

} IMotorPasso;

typedef enum  { MOTOR_PASSO_PROTEUS = 0, MOTOR_PASSO_GOIAS = 1
   } TIPO_MOTOR_PASSO;

#endif // IMOTORPASSO_H
```

E.2 MotorPassoFactory.h

```
#ifndef MOTORPASSOFACTORY_H
#define MOTORPASSOFACTORY_H

#include "IMotorPasso.h"

#ifdef __cplusplus
extern "C"
#endif
IMotorPasso* getMotorPasso( TIPO_MOTOR_PASSO tipoMotor );

#endif // MOTORPASSOFACTORY_H
```

E.3 MotorPassoFactory.c

```
#include "MotorPassoFactory.h"
#include "MotorProteus/MotorProteus.h"

#ifdef __cplusplus
extern "C"
#endif
IMotorPasso* getMotorPasso( TIPO_MOTOR_PASSO tipoMotor )
{
    IMotorPasso* retorno;

    switch( tipoMotor )
    {
    case MOTOR_PASSO_PROTEUS: retorno = &motorProteus; break;
    case MOTOR_PASSO_GOIAS: retorno = 0; break;
    default: retorno = 0;
    }

    return retorno;
}
```

E.4 MotorProteus

E.4.1 MotorProteus.h

```
#ifndef MOTORPROTEUS_H
#define MOTORPROTEUS_H

#include "../IMotorPasso.h"

extern IMotorPasso motorProteus;

#endif // MOTORPROTEUS_H
```

E.4.2 MotorProteus.c

```
#include "MotorProteus.h"

const char TAMANHO_COMBINACAO_ACIONAMENTO = 8;
const int TEMPO_DEFAULT_PASSO_MS = 500;

char configureMotorProteus();
```

```
char writeMotorProteus( int qtdePasso );
char setTempoEsperaPassoMsMotorProteus( int tempoMs );

char sequence[TAMANHO_COMBINACAO_ACIONAMENTO] = {0x02,0x06,0x04
   ,0x0c,0x08,0x09,0x01,0x03};
char passoAtual;
int tempoMsPasso;

IMotorPasso motorProteus = {
            &configureMotorProteus,
            &writeMotorProteus,
            &setTempoEsperaPassoMsMotorProteus
};

void delayEntrePassosDoMotor(){
     Delay_ms( 50 );
}

char configureMotorProteus(){
     TRISC = 0x00;
     passoAtual = 0;
     tempoMsPasso = TEMPO_DEFAULT_PASSO_MS;
}

char writeMotorProteus( int qtdePasso ){

//      const int teste = tempoMsPasso;
     for(;qtdePasso > 0; --qtdePasso )
     {
       PORTC = sequence[passoAtual];
       delayEntrePassosDoMotor();
       passoAtual = (passoAtual+1)%
          TAMANHO_COMBINACAO_ACIONAMENTO;
     }
}
char setTempoEsperaPassoMsMotorProteus( int tempoMs ){
     tempoMsPasso = tempoMs;
}
```

APÊNDICE F – TimerMotor

F.1 ITimerMotor.h

```
#ifndef ITIMERMOTOR_H
#define ITIMERMOTOR_H

extern volatile int tick;
extern volatile int segundos;
extern volatile int minutos;
extern volatile int horas;

typedef struct{
    void (*configurar)();
    void (*iniciar)();
    void (*parar)();
} ITimerMotor;

#endif
```

F.2 TimerMotorPasso.h

```
#ifndef TIMERMOTORPASSO_H
#define TIMERMOTORPASSO_H

#include "ITimerMotor.h"

extern ITimerMotor timerMotorPasso;

#endif
```

F.3 TimerMotorPasso.c

```
#include "ITimerMotor.h"
```

```
volatile int tick;
volatile int segundos;
volatile int minutos;
volatile int horas;

void configruarTimerMotorPasso();
void iniciarTimerMotorPasso();
void pararTimerMotorPasso();

ITimerMotor timerMotorPasso = {
    &configruarTimerMotorPasso,
    &iniciarTimerMotorPasso,
    &pararTimerMotorPasso
};

void configruarTimerMotorPasso(){
      tick = 0;
      segundos = 0;
      minutos = 0;
      horas = 0;

    TMR0L =  192;
    INTCON = 0xA0;
}
void iniciarTimerMotorPasso(){
    T0CON = 0xC4;             // Enable watchdog timer
}
void pararTimerMotorPasso(){
    T0CON = 0x00;             // Disable watchdog timer
}
```

APÊNDICE G – Principal

G.1 insulin_pump.c

```
#include "src/lcd/lcdFactory.h"
#include "src/config/GlobalConfig.h"
#include "src/insulinPump/InsulinPump.h"
#include "src/timerMotor/TimerMotorPasso.h"
#include "src/menu/MenuReservatorio.h"

//verifica se o contador de segundos chegou no intervalo que
   deve ocorrer a infusao
void verifica_infusao(){
  /*incrementa o contador que e utilizado pra infundir a
     insulina*/
  ++m_contador_injetar;
  /*caso tenha chegado no intervalo que deve ocorrer a infusao
     */
  if(m_intervalo_segundos_inteiro_para_injecao > 0 &&
     m_contador_injetar >=
     m_intervalo_segundos_inteiro_para_injecao){
    m_flag_injetar = TRUE;
    m_contador_injetar = m_contador_injetar %
       m_intervalo_segundos_inteiro_para_injecao;
  }
}

void interrupt() {
  tick++;
  // 400
    if (tick >= 10){
       ++segundos;
       verifica_infusao();
        if(segundos > 59){
            segundos = 0;
```

```
            ++minutos;
            if (minutos > 59) {
                minutos = 0;
                m_contador_injetar = 0;
                m_quantidade_injetada_hora = 0;
                m_flag_mudou_hora = TRUE;
                ++horas;
                if (horas > 23) {
                    horas = 0;
                    }
            }
        }

        tick = 0;
    }
  TMR0L = 192;                    //   on time
  INTCON = 0x20;
}

void main() {

  unsigned char converter[15];
  ILcd* lcd = getLcd( getTipoLcd() );
  IMenu* menu = &MenuReservatorio;

  InsulinPump.initialize();
  timerMotorPasso.configurar();
  timerMotorPasso.iniciar();

  iniciliazar();

  lcd->configure();                          // Initialize LCD

  lcd->clear();
  lcd->cursorOff();

  while(executarLoopPrincipal()) {
      menu->mostrarMenu();
      menu = menu->analisarBotoes();
```

```
        InsulinPump.inject();
    }

}
```