UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

IGOR GUERRERO

DESENVOLVIMENTO DE UM *DISPLAY* ESFÉRICO ROTATIVO COM ATUALIZAÇÃO DINÂMICA

São Carlos
2012

IGOR GUERRERO

DESENVOLVIMENTO DE UM *DISPLAY* ESFÉRICO ROTATIVO COM ATUALIZAÇÃO DINÂMICA

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em Eletrônica

ORIENTADOR: Professora Mestre Luiza Maria Romeiro Codá

São Carlos
2012

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Guerrero, Igor
G924d Desenvolvimento de um display esférico rotativo com atualização dinâmica / Igor Guerrero; orientadora Luiza Maria Romeiro Codá. São Carlos, 2012.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. Imagem esférica. 2. Placa rotativa. 3. LabVIEW. 4. POV. 5. LED. I. Título.

FOLHA DE APROVAÇÃO

Nome: Igor Guerrero

Título: "Desenvolvimento de um display esférico obtido por rotação com controle em tempo real"

Trabalho de Conclusão de Curso defendido e aprovado
em 30 / 11 / 2012,

com NOTA 10,0 (dez), *pela Comissão Julgadora:*

Profa. Assistente Luiza Maria Romeiro Codá (Orientadora)
SEL/EESC/USP

Profa. Dra. Maria Stela Veludo de Paiva
SEL/EESC/USP

M.Sc. Henrique Cunha Pazelli
Opto Eletrônica S.A.

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

À minha família.

Agradecimentos

Agradeço a minha família, meus pais e meus irmãos.

À professora Luiza Maria Romeiro Codá pela orientação e permitir a utilização de seu laboratório para a construção do projeto.

Aos meus amigos Lucas Assis de Moraes, Rafael Santos Moura e João Paulo Vicentini Fracarolli por me apoiarem todo o tempo, dentro ou fora do laboratório.

Ao pessoal da empresa Opto Eletrônica S.A. pelos auxílios e dicas que tornaram esse projeto possível.

À Universidade de São Paulo por me permitir estudar na melhor universidade do hemisfério sul.

Resumo

GUERRERO, I. **Desenvolvimento de um display esférico rotativo com atualização dinâmica**. Trabalho de conclusão de curso de engenharia elétrica com ênfase em eletrônica da Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

Este trabalho consiste no desenvolvimento de uma placa circular que, ao ser rotacionada, cria a ilusão de uma imagem esférica luminosa. O sistema como um todo recebe informações do usuário sobre a imagem a ser exibida e envia os dados em tempo real de um computador para uma placa rotativa que, por sua vez, processa os dados para o acendimento das luzes corretas. O projeto compreende a utilização de um motor de corrente continua com velocidade controlada pelo usuário, para realizar a rotação da placa circular, comunicação serial através do protocolo RS-232 para transmissão dos dados que compõem a imagem, interpretação de dados seriais através da linguagem VHDL implementado em CPLDs e criação de um software de controle em um computador através do software LabVIEW.

Palavras-chave: Imagem esférica. Placa rotativa. LabVIEW. POV. LED.

Abstract

GUERRERO, I. **Development of a rotating spherical display with dynamic update.** Senior research project in electrical engineering with emphasis on electronics of the Escola de Engenharia de São Carlos (School of Engineering of São Carlos), Universidade de São Paulo (University of São Paulo), 2012.

This work describes the development of a circular board that, when rotated, creates the illusion of a spherical luminous image. The rotating system receives user information of which image to *display*. The image data are sent in real time from a computer to a rotating PCB's CPLD which processes the data for lighting the LEDs correctly. The project includes a dc motor with speed control through RS-232 serial protocol of communication, interpretation of serial data with VHDL language and control software using LabVIEW software.

Keywords: Spherical *display*. Rotating board. LabVIEW. POV. LED.

Lista de Figuras

Figura 1.1: Primeiro rosto exibido em um aparelho de televisão [1]. 25
Figura 1.2: *Display* rotativo em roda de veiculo [6].....26
Figura 1.3: Relógio rotativo criado pro Bob Blick [7]. 26
Figura 1.4: Outros projetos de *displays* rotativos [8]. 26
Figura 1.5: Globo rotativo – HackedGadgets [9] e SolderLab [10]. 27
Figura 2.1: Interface com conector DB9 [15]. 31
Figura 2.2: Componentes de um VI. 32
Figura 2.3: Paletas de acesso às funções e controles. 33
Figura 2.4: Comparação LabVIEW e linguagem C: Adição. 34
Figura 2.5: Comparação LabVIEW e linguagem C: Loop controlado pelo usuário. 34
Figura 2.6: Paralelismo em LabVIEW. 35
Figura 2.7: Exemplo de tipos de dados no LabVIEW. 35
Figura 2.8: Utilização de SubVIs: Ícones e conectores. 36
Figura 3.1: Esquema de conjunto do sistema. 38
Figura 3.2: Diagrama de relacionamento em blocos do projeto. 38
Figura 3.3: *Slip Rings* comerciais de quatro canais [21]. 39
Figura 3.4: Conector P2 utilizados e funcionalidade atribuída a cada faixa de contato do *Slip Ring*. 39
Figura 3.5: DB9 - Pinos utilizados para a comunicação serial. 40
Figura 3.6: Palavra serial para controle dos LEDs. 41
Figura 3.7: Palavra serial para controle do motor. 41
Figura 3.8: Motor CC. 42
Figura 3.9: *Encoder* rotacional de 1 *bit* e 13 *bits* utilizando código Gray [22]. 43
Figura 3.10: Sensor e receptor infravermelho próprio para *encoders*. 43
Figura 3.11: *Encoder* de fibra de vidro: Projeto e placa real. 44
Figura 3.12: *Encoder* de acetato finalizado com suportes de fibra de vidro. 45
Figura 3.13: Nível de tensão de linha: RS-232 e TTL respectivamente. 45
Figura 3.14: Circuito de conversão do nível de tensão da comunicação serial. 46
Figura 3.15: Esquema elétrico da CPLD na placa controle. 47
Figura 3.16: Esquema elétrico do *driver* de potência para o motor CC.....48
Figura 3.17: Condicionamento do sensor e receptor IV. 49
Figura 3.18: Comparador analógico utilizado para adequar a tenso sinal do *encoder*. 49
Figura 3.19: Placa controle: projeto. 50
Figura 3.20: Placa controle montada. 51
Figura 3.21: Esquema elétrico da CPLD na placa rotativa. 52

Figura 3.22: *Driver* de corrente para os LEDs da placa rotativa. 53
Figura 3.23: Placa rotativa: Projeto. 54
Figura 3.24: Destaque para os LEDs soldados em ambas as faces da PCI. 54
Figura 3.25: Placa rotativa montada. 55
Figura 3.26: Conector Motor - Placa Rotativa: Vistas de projeto e peça finalizada. 56
Figura 3.27: Suporte *Slip Ring* Fêmea: Vistas de projeto e peça finalizada. 57
Figura 3.28: Suporte *Slip Ring* Macho: Vistas de projeto e peça finalizada. 57
Figura 3.29: Suporte de Fixação à Caixa: Vistas de projeto e peça finalizada. 58
Figura 3.30: Suporte do Motor 58
Figura 3.31: Esquema em blocos da lógica da CPLD: Placa Controle. 60
Figura 3.32: Esquema em blocos da lógica da CPLD: Placa Rotativa. 61
Figura 3.33: Painel frontal do software de controle. 63
Figura 3.34: Sistema montado: vista 3D. 64
Figura 3.35: Conectores na parte traseira da caixa de suporte. 64
Figura 3.36: Placa controle ligada aos conectores externos. 65
Figura 3.37: Montagem Superior: Placa rotativa, *Slip Ring* e tubo de suporte. 65
Figura 3.38: Montagem inferior: Motor, PCI, *encoder* e sensor IV. 66
Figura 3.39: Montagem completa. 66
Figura 4.1: Simulação do código interpretador de mensagens serial da placa controle. 69
Figura 4.2: Simulação do código interpretador de mensagens serial da placa rotativa. 70
Figura 4.3: Software em funcionamento. 71
Figura 4.4: Sistema em funcionamento com todos os LEDs acessos. 72
Figura 4.5: Sistema em funcionamento com uma imagem. 72
Figura 4.6: Medida da velocidade final do sistema e atraso entre pedido e recepção dos dados. 74

Lista de Siglas

CC	Corrente continua
CI	Circuito integrado
CPLD	Dispositivo Lógico Complexo Programável
FPGA	Arranjo de Portas Programável em Campo
IV	Infravermelho
LED	Diodo Emissor de Luz
PCI	Placa de circuito impresso
PPR	Pulsos por revolução
PWM	Modulação por largura de pulso
RPM	Rotações por minuto
RPS	Rotações por segundo
RS	Recommended Standard
SMD	Tecnologia de montagem superficial
TRS	Ponta-Anel-Capa
TTL	Lógica Transistor-Transistor
USB	*Universal Serial Bus*
USP	Universidade de São Paulo
VHDL	Linguagem de descrição de hardware VHSIC
VHSIC	Circuito integrado de velocidade muito alta
VI	Instrumento Virtual

Lista de Símbolos

Hz	Hertz
Mm	Milímetros
Ms	Milissegundo
V	Volts

Sumário

1 Introdução 25
1.1 Motivação 25
1.2 Objetivos do Trabalho 27
1.3 Organização da Monografia 28
2 Revisão Teórica 29
2.1 Persistência da visão 29
2.2 Comunicação Serial 30
2.3 LabVIEW 31
3 Desenvolvimento do Projeto 37
3.1 Junta Rotativa de contato elétrico (Slip Ring) 39
3.2 Comunicação Serial 40
3.3 Controle do Motor 41
3.4 Encoder 42
3.5 Placa Controle 45
3.5.1 Condicionamento do sinal de comunicação serial 45
3.5.2 Recepção e envio dos dados seriais 46
3.5.3 *Driver* de corrente para o motor 48
3.5.4 Condicionamento do sensor e emissor infravermelho 48
3.5.5 Fabricação da placa de circuito impresso 50
3.6 Placa Rotativa 51
3.6.1 Recepção dos dados seriais 51
3.6.2 Acionamento dos LEDs 53
3.6.3 Fabricação da placa de circuito impresso 53
3.7 Sistema mecânico 55
3.7.1 Conexão Motor – Placa Rotativa 56
3.7.2 Suporte *Slip Ring* Fêmea 56
3.7.3 Suporte *Slip Ring* Macho 57
3.7.4 Suporte de Fixação à Caixa 58
3.7.5 Suporte do Motor 58
3.8 Temporização do sistema 59
3.9 Circuitos do CPLD 60

3.9.1 Lógica VHDL da Placa Controle 60
3.9.2 Lógica VHDL da Placa Rotativa 61
3.10 Software 62
3.11 Montagem 63
4 Resultados e Discussões 67
4.1 Simulação dos códigos em VHDL 67
4.1.1 Bloco de decodificação serial da placa controle 67
4.1.2 Bloco de decodificação serial da placa rotativa 68
4.2 Testes 71
5 Conclusões 75
5.1 Sugestões para trabalhos futuros 75
Referências Bibliográficas 77
Apêndice A – Desenhos das peças mecânicas 79
Apêndice B – Esquemas elétricos 85
Apêndice C – Placas de circuito impresso 89
Apêndice D – Descrições de hardware: Diagrama em blocos 91
Apêndice E – Descrição de hardware: VHDL 93
Apêndice F – Softwares em LabVIEW 99

1 Introdução

1.1 Motivação

Em 26 de janeiro de 1926 o inventor escocês John Logie Baird apresentou para um time de cientistas o primeiro aparelho de televisão capaz de transmitir uma imagem ao vivo. Curiosamente a primeira imagem a ser transmitida foi a cabeça de um boneco de ventríloquo sendo, logo em seguida, televisionado o primeiro rosto humano, que pode ser visto na Figura 1.1 [1] [2].

Figura 1.1: Primeiro rosto exibido em um aparelho de televisão [1].

As imagens transmitidas eram atualizadas a uma taxa de 12,5 quadros por segundo, recorde na época, e consistia na taxa mínima para o olho humano deixar de identificar imagens isoladas e conseguir criar a sensação de movimento continuo. Dez anos após essa data as olimpíadas de Berlim foram transmitidas através de um circuito fechado para 41 países [3] [4].

Menos de cem anos depois de sua invenção, a televisão se tornou um dos meios de informação e entretenimento doméstico mais popular no mundo, possuindo aproximadamente 1,4 bilhões de aparelhos em todo o mundo. Nas últimas décadas, os computadores pessoais ganharam mercado, porém sua interface de saída de dados para o usuário ainda é somente um monitor, não apresentando grandes diferenças com as telas dos aparelhos de televisão [5].

Várias alternativas ao clássico monitor retangular foram idealizadas e desenvolvidas, porém nenhuma conseguiu evoluir a um nível igualitário de qualidade. Uma das ideias desenvolvida é um mostrador composto de uma base com vários emissores de luz que, ao ser rotacionado sincronamente com a variação da intensidade de iluminação dos emissores de luz, é capaz de criar uma imagem. Essa ideia não é capaz de substituir um monitor de computador ou televisão, mas pode ser utilizada para fins simples e criativos, como criar efeitos em rodas em movimento como pode ser visto na Figura 1.2.

Figura 1.2: *Display* rotativo em roda de veiculo [6].

Uma das aplicações mais conhecidas desse sistema é o desenvolvimento de um relógio chamado *propeller clock* criado por Bob Blick em 1996 e composto de sete LEDs (Light-Emitting Diode) que, ao rodarem, criam linhas contínuas. Um microprocessador PIC16C84 secciona as linhas em 30 colunas e chaveando-as, cria-se a ilusão de um relógio que pode ser visto na Figura 1.3 [7]. Outros projetos foram desenvolvidos baseando-se na ideia de Bob Blick, porém retirando a função de relógio e adicionando imagens estáticas. Exemplos podem ser vistos na Figura 1.4.

Figura 1.3: Relógio rotativo criado pro Bob Blick [7].

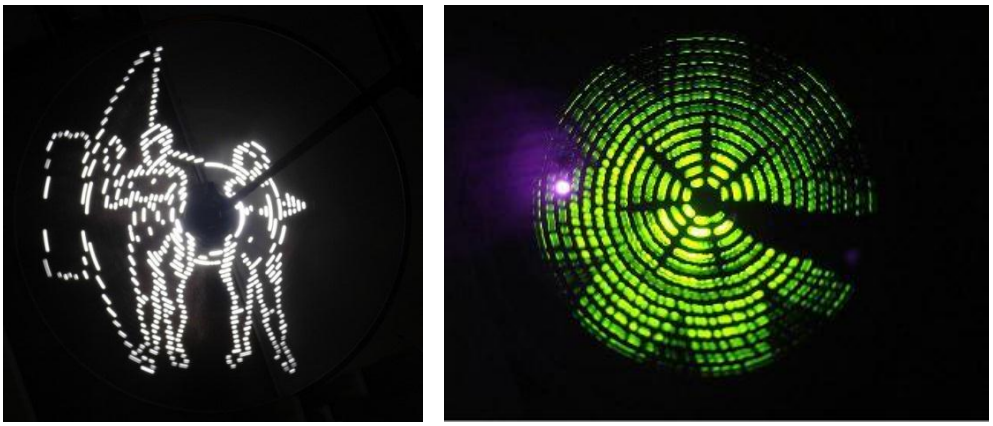


Figura 1.4: Outros projetos de *displays* rotativos [8].

Uma variação mais complexa pode apresentar um número maior de LEDs e uma base de formato circular de modo que, ao ser rotacionada, a imagem resultante seja um globo. Exemplos desse estilo podem ser vistos na Figura 1.5.

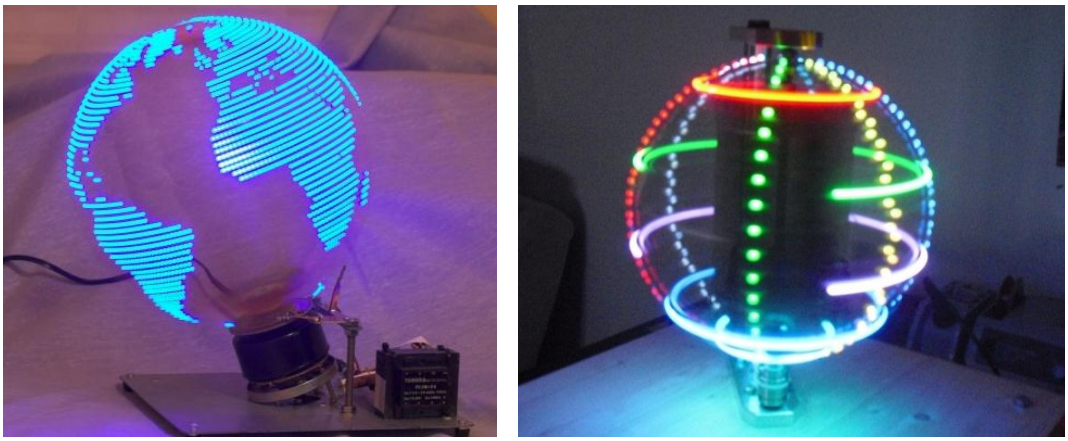


Figura 1.5: Globo rotativo – HackedGadgets [9] e SolderLab [10].

Todos os projetos mostrados nas Figuras de 1.2 a 1.5 são controlados por microprocessadores PIC ou ATmega da Atmel com imagens estáticas e previamente gravadas no microprocessador, não sendo possível altera-las sem reprogramar o microprocessador [8] [9] [10].

1.2 Objetivos do Trabalho

Tendo como referência os projetos citados na seção anterior, o objetivo desse trabalho é o projeto e construção de um *display* luminoso esférico capaz de exibir imagens em tempo real por meio de comunicação com um computador. Esse *display* é constituído por uma placa com formato circular que, ao ser rotacionada, causa a impressão de uma esfera. As principais diferenças deste trabalho com aqueles citados na seção 1.1 são o controle de velocidade de rotação e de imagem exibida em tempo real, não necessitando de dados previamente gravados em uma memória.

A imagem é formada por uma fileira de LEDs, com controles individuais, posicionados na borda externa de uma placa que, quando rotacionada, chaveia-os sincronamente com a posição angular do sistema, criando fusos esféricos nos quais a intensidade luminosa de cada LED é mantida.

Este projeto também apresenta objetivos didáticos, visando aumentar a experiência e nível técnico do autor, somado ao desafio de elaborar todas as peças mecânicas necessárias para a realização do sistema.

1.3 Organização da Monografia

O trabalho está dividido em cinco capítulos, os quais estruturados da seguinte maneira:

1. Introdução: Apresenta os projetos utilizados como motivação, os objetivos principais do trabalho e sua organização.
2. Revisão Teórica: É apresentada uma breve revisão teórica sobre alguns dos principais assuntos utilizados durante o projeto. A primeira seção mostra uma explicação para a sensação de continuidade criada pelo movimento de objetos luminosos ou imagens. A segunda seção relata sobre a teoria utilizada para a comunicação de dados e finalmente a terceira seção descreve o software utilizado para implementar o software de controle.
3. Desenvolvimento do Projeto: Descreve os passos utilizados para o desenvolvimento do projeto como o trabalho realizado para a idealização, projeto e, construção das peças do sistema. Algumas seções podem apresentar uma breve introdução teórica, utilizada apenas para contextualizar o assunto a ser tratado.
4. Resultados e Discussões: Os resultados obtidos são apresentados, incluindo os problemas encontrados e é realizada uma comparação desses resultados com outros projetos similares.
5. Conclusões: Apresenta as conclusões obtidas e sugestões para melhorias futuras.

Os apêndices apresentam todos os dados necessários para a reconstrução do projeto. No apêndice A são encontrados os desenhos relativos às partes mecânicas desenvolvidas para o suporte do sistema. O apêndice B apresenta os desenhos esquemáticos dos circuitos eletrônicos, enquanto o apêndice C mostra o projeto final das placas de circuito impresso que alocam esses circuitos. O código em diagrama de blocos e VHDL utilizados pelas CPLDs (Complex Programmable Logic Device) podem ser vistos nos apêndices D e E. Finalmente, o apêndice F apresenta o software desenvolvido em LabVIEW para controle da comunicação e interação com o usuário.

2 Revisão Teórica

Este capítulo apresenta uma breve revisão da teoria utilizada no desenvolvimento do projeto. A revisão se inicia com uma breve explicação sobre o suposto efeito da persistência da visão, amplamente estudado no inicio do desenvolvimento da indústria do cinema e televisão. Na sequencia é apresentado o conceito de comunicação serial com uma explicação mais detalhada do protocolo a ser utilizado no projeto. Finalmente é apresentada uma explicação sobre o motivo, vantagens e desvantagens do software utilizado para o processamento e envio dos dados.

2.1 Persistência da visão

Em 1824, Peter Mark Roget publicou seu trabalho *"Explanation of an optical deception in the appearance of the spokes of a wheel when seen through vertical apertures"* e criou a ideia de persistência da visão. Uma explicação típica para a persistência da visão é que o olho humano, ao ser apresentado a uma sucessão rápida de imagens, mantém por alguns milésimos de segundo o efeito da luz na córnea, criando um efeito de fusão dessa imagem com a próxima a ser captada, suavizando as transições e criando a sensação de movimento [11] [12].

Com o passar do tempo muitas pesquisas foram feitas sobre o assunto e, de acordo com Joseph e Barbara Anderson, a persistência da visão é um mito que foi difundido até os dias de hoje mesmo não sendo o real motivo pelo qual o movimento é percebido. Joseph e Barbara fazem duas perguntas em seu trabalho: Por que uma sucessão de imagens aparenta um movimento contínuo e porque essa imagem pode aparentar se movimentar? De acordo com eles o real motivo não está na córnea, e sim, no processamento realizado pelo córtex visual dos dados capturado. O cérebro processa as imagens captadas pelas córneas e as conecta, criando a ilusão de movimento. Essa lógica de interpretação é a mesma utilizada para o processamento de movimentos reais, ou seja, o cérebro confunde o movimento de um objeto físico, real e tangível com o movimento aparente causado pela variação de uma fonte estática [11] [12].

Movimento aparente é o nome dado quando nenhum estimulo visual realmente altera sua posição através do tempo dentro do campo visual. É uma ilusão, visto que essa percepção não condiz com as condições presentes no mundo real [13].

Um experimento foi feito e pode ser utilizado para ilustrar a ideia de que a ilusão do movimento aparente não condiz com a realidade, e comprovar a possibilidade da ilusão proposta por este trabalho. Duas imagens muito similares, ambas contendo um ponto desenhado, porém com posições absolutas levemente deslocadas entre si, são exibidas sucessivamente para um espectador. Se, o período de transição entre as duas imagens for maior que 200 ms o espectador observa uma sucessão de pontos, primeiro um depois o outro. Se o período for menor que 30 ms o espectador observa dois pontos parados e, se o período estiver entre 30 e 200 ms o espectador vivencia a ilusão de que somente um ponto se move de um lado para o outro [13]. Portanto, se imagens forem apresentadas

sucessivamente com uma frequência maior que 5 Hz a ilusão de movimento é criada, aparentando a ideia de persistência da visão.

2.2 Comunicação Serial

Comunicação serial é um método de conexão entre equipamentos que utiliza a variação de informação através do tempo, possibilitando o envio de dados utilizando um único canal. A utilização de um único canal é, ao mesmo tempo, a maior vantagem e desvantagem desse método. Em comparação com seu método oposto de comunicação, comunicação em paralelo, esta possibilita a transmissão de vários dados simultaneamente, atingindo uma maior taxa de transmissão de dados por tempo, porém para isso necessita de vários canais funcionando ao mesmo tempo, dificultando sua montagem física e aumentando a possibilidade de erros e problemas [14].

Na engenharia elétrica, o principal meio de dados é o fio, transmitindo sinais elétricos através da variação de tensão pelo tempo. A quantidade de *bits* transmitida por segundo (BPS) recebe o nome de *baud rate* e, quanto maior for o *baud rate,* mais *bits* serão transmitidos [14].

Dois métodos de sincronismo entre emissor e receptor são possíveis: comunicação síncrona e assíncrona. No método síncrono um canal extra de transmissão é necessário transmitindo um *clock*, informando ao dispositivo receptor quando os *bits* recebidos devem ser interpretados. No método assíncrono o próprio canal de dados informa ao receptor quando a interpretação de dados deve ser executada. Um *bit* de inicio (*start bit*) apresentando um nível de tensão oposto ao nível da linha inativa (*idle*) indica que os próximos sinais devem ser interpretados como dados. A quantidade de *bits* que compõe um dado deve ser previamente conhecida pelo receptor para uma correta interpretação, já que, após os *bits* de dado, pode se seguir um *bit* de paridade servindo para a verificação de erros durante a transmissão. O ultimo *bit*, *stop bit*, é transmitido utilizando o mesmo nível de tensão que o *start bit*, porém pode apresentar tamanho de 1, 1.5 ou 2 vezes maior que os *bits* de dados e indica o fim dos dados [14]. Várias arquiteturas de comunicação serial foram desenvolvidas com o tempo para diversas aplicações, entre essas se pode citar: SPI, I2C, ethernet, RS-422, RS-485 e, a que será utilizada por este projeto, RS-232.

Em 1962 a EIA (*Electronic Industries Alliance*) padronizou o protocolo de comunicação serial 232, voltado inicialmente à comunicação entre máquinas de escrever eletromecânicas com *modens*. Com o passar do tempo, os novos dispositivos criados mantiveram esse meio de comunicação para que pudessem continuar se comunicando com os outros aparelhos.

Atualmente a comunicação serial RS-232 pertence à padronização TIA-232-F de interface entre circuitos. De acordo com essa norma, um sinal equivalente a um nível lógico “0” deve ser representando por uma tensão positiva com variação de +5 a +15 V no circuito emissor e de +3 V a +15 V no receptor. Para o nível lógico “1”, vale a simetria desses sinais em relação ao 0 V, ou seja, o emissor deve enviar um nível de tensão entre -5V a -15 V e o receptor deve compreender sinais variando de -3V a -15 V [15].

Embora a padronização especifique a utilização de um conector de 25 pinos (com somente três inativos) é usualmente utilizado um conector DB9 com sinais simplificados [15]. A pinagem utilizada no conector DB9 pode ser vista na Figura 2.1.

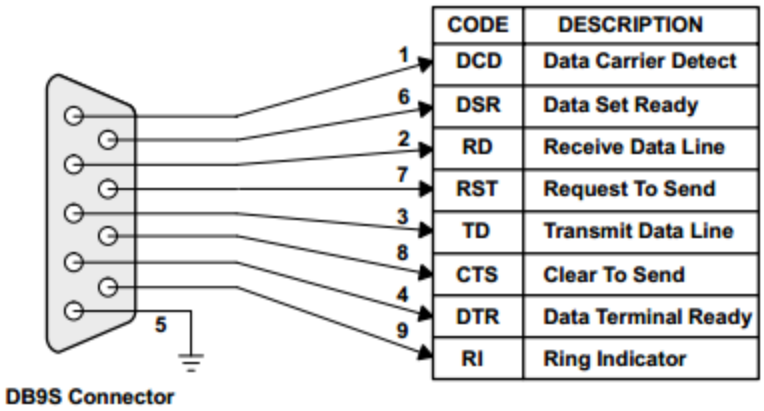


Figura 2.1: Interface com conector DB9 [15].

2.3 LabVIEW

LabVIEW, acrônimo para *Laboratory Virtual Instrument Engineering Workbench*, é um ambiente de desenvolvimento de aplicativos lançado em outubro de 1986 pela empresa National Instruments. Inicialmente foi desenvolvido como uma ferramenta de software de apoio para gerenciamento/supervisão de módulos de circuitos programáveis, porém seu desenvolvimento e aprimoramento tornaram-no um software de utilidade geral, intuitivo e eficiente [16] [17]. As datas de lançamento das versões desse software podem ser vistas na Tabela 1.

Tabela 1: LabVIEW: versões e datas de lançamento [17].

Versão	Lançamento
LabVIEW - inicio do projeto	abril-83
LabVIEW 1.0	outubro-86
LabVIEW 2.0	janeiro-90
LabVIEW 3.0	julho-93
LabVIEW 4.0	abril-96
LabVIEW 5.0	fevereiro-98
LabVIEW 6.0	julho-00
LabVIEW 7.0	abril-03
LabVIEW 8.0	setembro-05
LabVIEW 2009	abril-09
LabVIEW 2010	abril-10
LabVIEW 2011	junho-11
LabVIEW 2012	agosto-12

A linguagem G é o centro de funcionamento do LabVIEW e sendo construída graficamente, utiliza ícones, fios e nós ao invés de texto para a construção de aplicativos. Com o auxilio do *mouse* o

usuário arrasta ícones e fios pela tela, não sendo necessária a utilização de textos para a descrição do código. De acordo com especialistas, a curva de aprendizado para novos usuário é menor em linguagens gráficas comparadas com as textuais, pois se pode relacionar o fluxo de dados com outros processos gráficos já conhecidos, como por exemplo, diagrama em blocos. A linguagem gráfica se baseia no conceito de fluxo de dados para suas interações. Esse conceito de arquitetura de software baseia-se na ideia que a alteração do valor de uma variável acarretará automaticamente o recálculo de todas as variáveis dependentes desse valor [17] [18] [19].

O conceito de fluxo de dados é fundamentalmente diferente do modo de instruções em nível de máquina. Códigos em baixo nível descrevem uma lista sequencial de instruções a serem realizadas, uma de cada vez. Na computação baseada em fluxo de dados a instrução é executada quando todas as suas entradas estiverem disponíveis, não utilizando o conceito de controle de tráfego e contador de instrução. Consequentemente, com esse tipo de computação é possível a realização de funções em paralelo através do uso de vários processadores [19].

Os programas gerados em LabVIEW são chamados de instrumentos virtuais (VI) porque sua aparência e operação tentam aparentar instrumentos físicos [18]. Um VI contém 3 componentes que são descritos abaixo e podem ser visualizados na Figura 2.2:

- Painel frontal: Janela de interação com o usuário. Esse painel apresenta todos os controles e indicadores do aplicativo utilizado.
- Diagrama em blocos: Janela de desenvolvimento e processamento. Apresenta toda a lógica do código ficando transparente ao usuário após sua compilação.
- Ícone e conector: São as portas de entrada e saída do VI, possibilitando a conexão de vários VI.

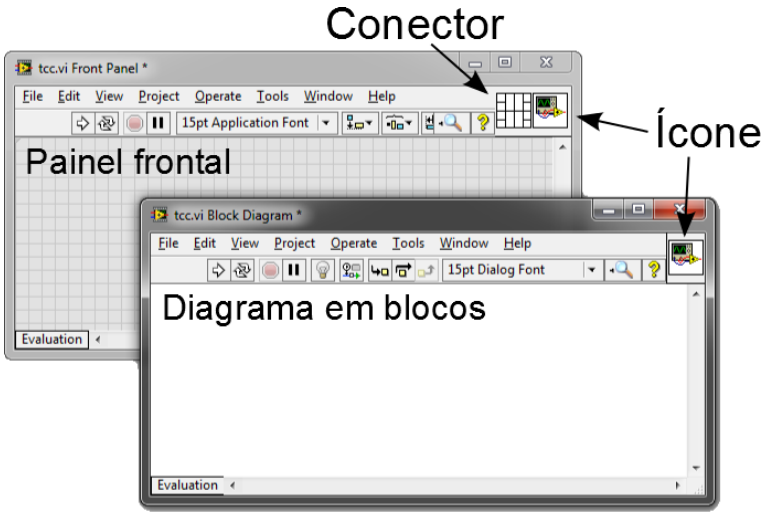


Figura 2.2: Componentes de um VI.

Para o desenvolvimento de um aplicativo o usuário deve arrastar das paletas o ícone que representa a função desejada. A Figura 2.3 a seguir indica as duas paletas disponíveis: funções e controles, relacionadas respectivamente à janela de diagrama em blocos e painel frontal.

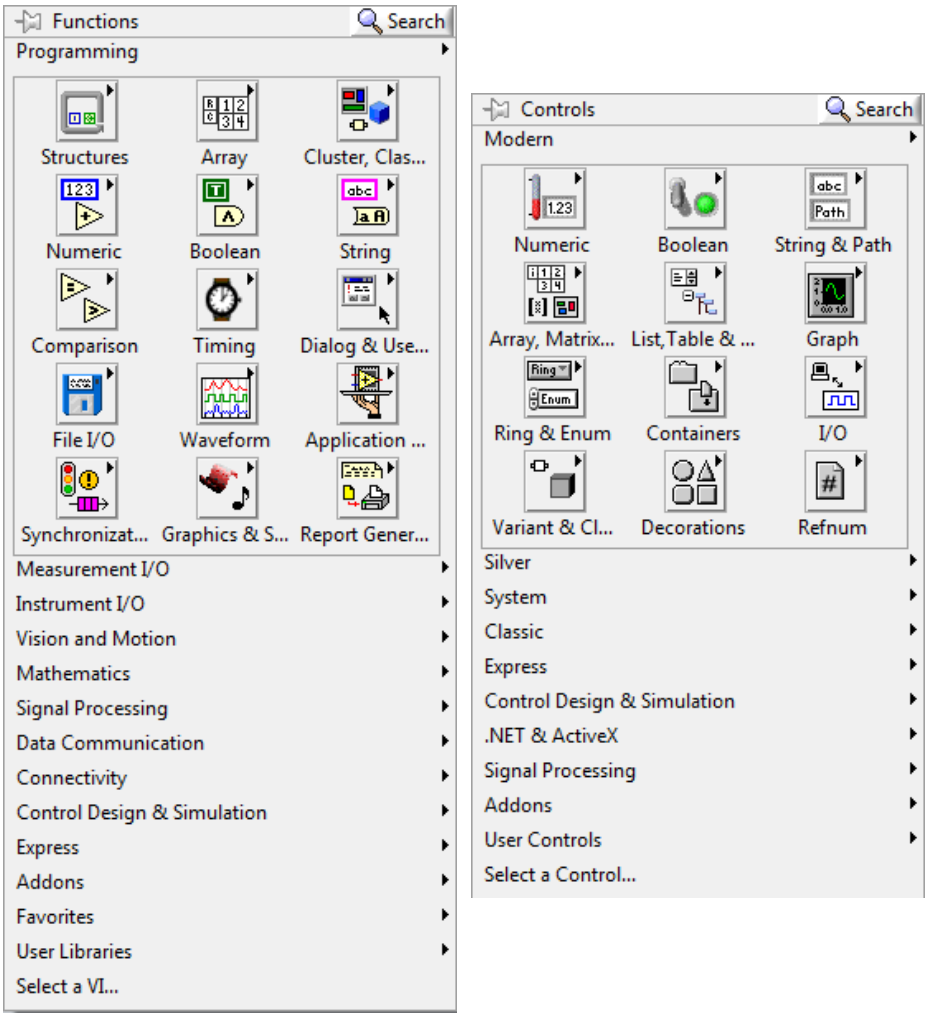
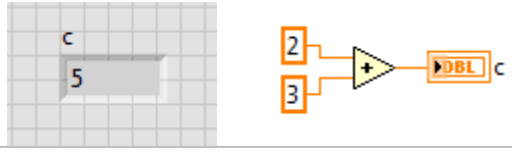


Figura 2.3: Paletas de acesso às funções e controles.

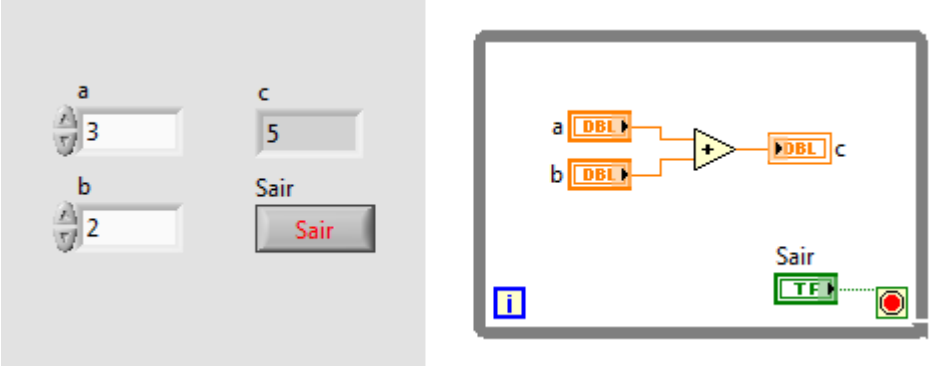
Devido ao uso de ícones, o uso do software se torna extremamente intuitivo. Uma operação de soma será usada para comparar o desenvolvimento do código em LabVIEW e em linguagem C. Para a realização de uma soma entre duas constantes é necessário somente encontrar o ícone de soma e a constante de número na paleta de funções, o ícone de indicador numérico na paleta de controle e conectar os valores através de fios ao bloco desejado. A Figura 2.4 exemplifica a comparação das linguagens, exibindo o painel frontal e diagrama de blocos do LabVIEW contraposto com o código em linguagem C para a realização da mesma tarefa.




```
#include <stdio.h>

int main()
{
    int c;

    c = 2 + 3;
    printf("soma = %d\n", c);

return 0;
}
```

Figura 2.4: Comparação LabVIEW e linguagem C: Adição.

Ao executar o software, o LabVIEW examina os dados presentes em ambas as suas janelas e transforma o código em linguagem de máquina, otimizando áreas não conectadas para que essas executem em paralelo. Outro exemplo mais complexo pode ser visto na Figura 2.5 que compara os códigos necessários para o cálculo da soma de dois valores inseridos pelo usuário, tantas vezes quanto o usuário desejar.




```
#include <stdio.h>

int main()
{
    int a, b, c, d;
do{
    scanf("%d",&a);
    scanf("%d",&b);
    c = a + b;
    printf("soma = %d\n", c);
    printf("0 -> Sair \n1 -> Outra soma\n");
    scanf("%d",&d);
}while(d != 0);

return 0;
}
```

Figura 2.5: Comparação LabVIEW e linguagem C: Loop controlado pelo usuário.

A estruturação em fluxo de dados permite maior flexibilidade e paralelismo ao código. Para ilustrar a Figura 2.6 indica dois loops executando em paralelo, cada um em um processador diferente. Se o computador utilizado não possuir vários processadores um erro será exibido ao usuário.

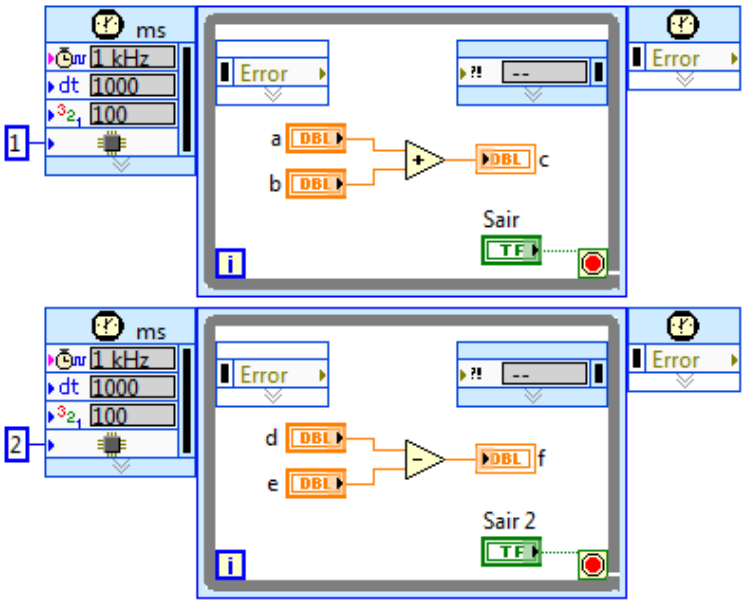


Figura 2.6: Paralelismo em LabVIEW.

Assim como outras linguagens, o LabVIEW apresenta a possibilidade do usuário utilizar vários tipos de dados para diferentes situações. Cada tipo de dado apresenta uma única cor e estilo de fio de conexão. Alguns dos tipos de dados permitidos podem ser vistos na Figura 2.7.

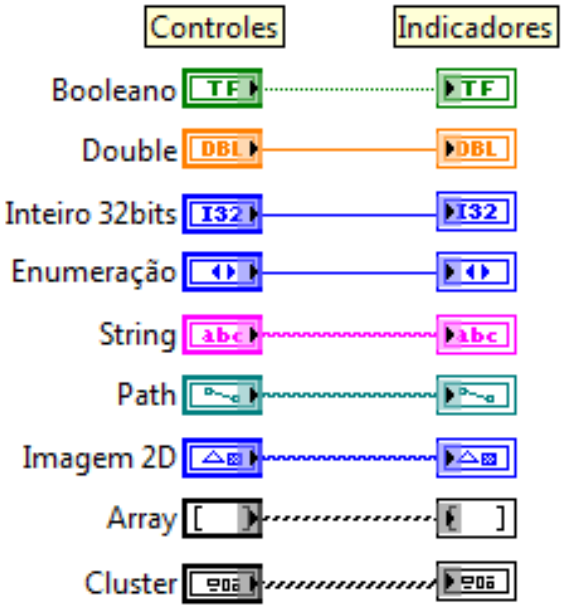


Figura 2.7: Exemplo de tipos de dados no LabVIEW.

Instrumentos virtuais podem utilizar outros instrumentos virtuais dentro de sua lógica, esses VI chamados recebem o nome de SubVI. Um SubVI exerce exatamente a mesma tarefa de quando executado isoladamente, a única diferença é o modo de entrada e saída de dados. A conexão dos dados é realizada da mesma maneira que as outras funções, através do ícone e conector.

Quando um SubVI é chamado, ele aparece na tela utilizando seu ícone, porém suas conexões devem ser feitas utilizando o padrão de seu conector. A Figura 2.8 exemplifica a utilização de um SubVI mostrando seus ícones e conectores equivalentes.

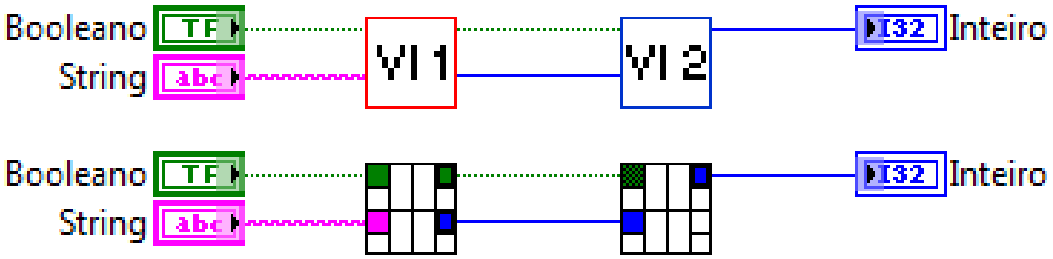


Figura 2.8: Utilização de SubVIs: Ícones e conectores.

3 Desenvolvimento do Projeto

O objetivo deste trabalho é o desenvolvimento de uma placa com LEDs que, ao ser rotacionada, crie a ilusão de uma imagem luminosa. A placa é rotacionada por meio de um motor CC (corrente continua) e suportada por varias peças mecânicas para garantir sua estabilidade. Todas as peças mecânicas utilizadas foram desenvolvidas e fabricadas especialmente para esse projeto.

Um diferencial desejado para este projeto é a capacidade de atualização das imagens exibidas em tempo real, não sendo necessária a reprogramação de uma memória instalada na placa a ser rotacionada. Essa ideia foi crucial para o projeto, já que mudaria completamente sua forma de controle.

Para possibilitar o controle em tempo real, duas propostas de comunicação de dados foram levadas em consideração: Comunicação com e sem fio.

- Comunicação sem fio

  Utilizando esse modo de comunicação a placa rotativa precisaria apenas de dois contatos elétricos com o restante do sistema (alimentação e terra), não sendo necessário um sistema complexo de conexão, entretanto precisaria de um modo de comunicação sem fio (rádio, *bluetooth* ou módulos comerciais como ZigBee).

  Aumentar a lógica necessária na placa rotativa significa aumentar o número de componentes. Um alto número de componentes na placa não é desejável, já que a rotação e vibração podem danificá-los mecanicamente.
- Comunicação com fio

  Utilizando esse modo de comunicação a placa rotativa precisaria de três contatos elétricos (alimentação, terra e dados), dificultando sua conexão com os suportes mecânicos de apoio, porém diminuindo a complexidade da placa giratória. Tal conexão somente poderia ser feita utilizando uma junta rotativa de contato (*Slip Ring*).

Analisando as considerações propostas nos parágrafos anteriores foi escolhido trabalhar com o método de comunicação com fio devido à diminuição de complexidade da placa rotativa, a oportunidade de trabalhar com um *Slip Ring* e o já grande número de trabalhos realizados com comunicação sem fio. A escolha do método de comunicação permite criar um delineamento inicial de todos os componentes do projeto:

A parte principal é a placa rotativa, nela estão presentes todos os LEDs que formam as imagens e a lógica de interpretação das palavras de comunicação. A comunicação com o usuário e o envio dos dados das imagens foi realizada por um computador pessoal, sendo à saída de dados por meio de uma porta USB. A rotação da placa foi realizada por um motor DC sendo sua velocidade

controlada por uma placa de circuito impresso. Esta mesma placa condiciona o nível de tensão da comunicação serial e é denominada “Placa Controle”.

Um diagrama de alguns desses componentes e a visão geral do projeto podem ser vistos na Figura 3.1. As linhas tracejadas indicam os fios de comunicação entre os componentes e, as setas em azul o sentido de trasmissão de dados.

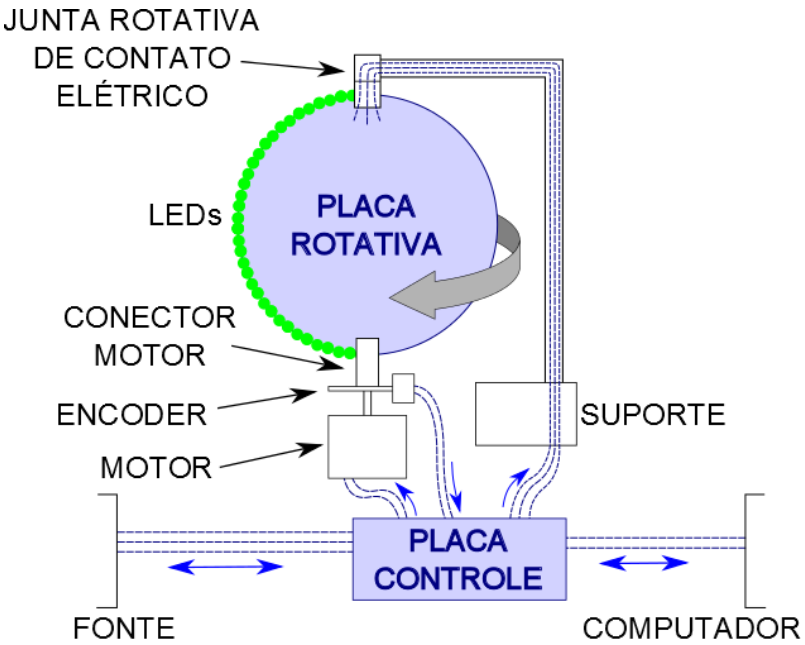


Figura 3.1: Esquema de conjunto do sistema.

A lógica de relacionamento entre as placas, o usuário e a imagem final estão representadas no diagrama da Figura 3.2:

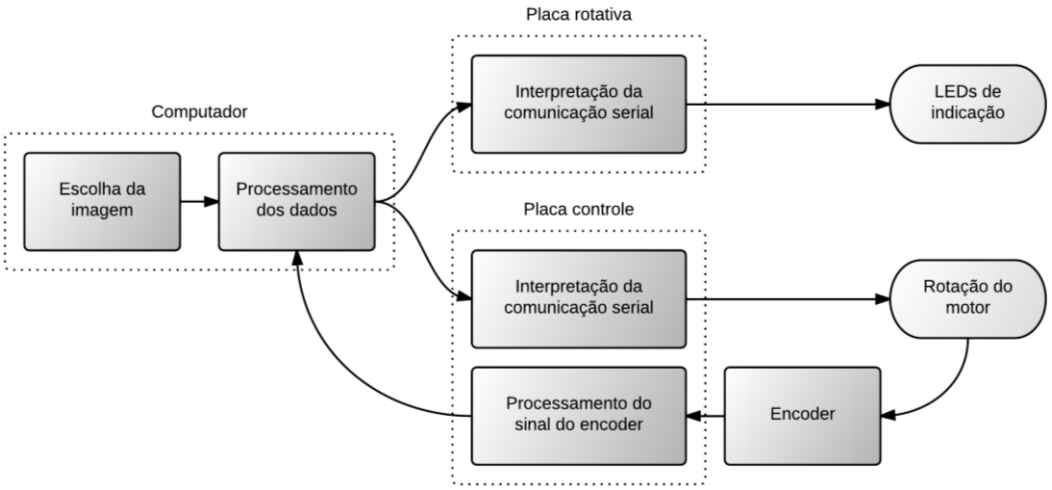


Figura 3.2: Diagrama de relacionamento em blocos do projeto.

As próximas seções relatam como foi projetado cada um dos blocos ou partes relativos as Figura 3.1 e Figura 3.2.

3.1 Junta Rotativa de contato elétrico (Slip Ring)

Um item crucial para um bom funcionamento do projeto é a junta rotativa de contato elétrico, também conhecido como *Slip Ring*. Esse dispositivo tem a função de manter o contato elétrico entre um ou vários canais possibilitando um movimento rotacional entre o estágio primário e secundário [20]. No projeto essa junta é utilizada para o contato elétrico dos três fios necessários para o funcionamento da placa rotativa, conectando-a com o tubo de suporte estático. Os três fios são: alimentação, terra e dados seriais.

Vários modelos de juntas rotativas são comercializados, porém, devido à complexidade de se desenvolver um contato confiável, seus preços são bem elevados, saindo da faixa de preço viável para esse projeto. Exemplos de *Slip Rings* comerciais podem ser vistos na Figura 3.3:

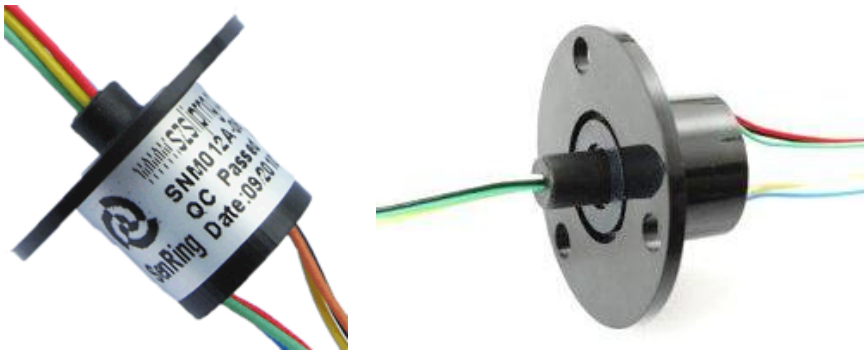


Figura 3.3: *Slip Rings* comerciais de quatro canais [21].

A solução escolhida para driblar o alto custo foi utilizar um substituto mais barato e fácil de ser encontrado: um conector TRS (*tip-ring-sleeve*) de 3.5mm, também conhecido como conector P2. Os conectores macho e fêmea, assim como a funcionalidade atribuída a cada contato pode ser visto na Figura 3.4.

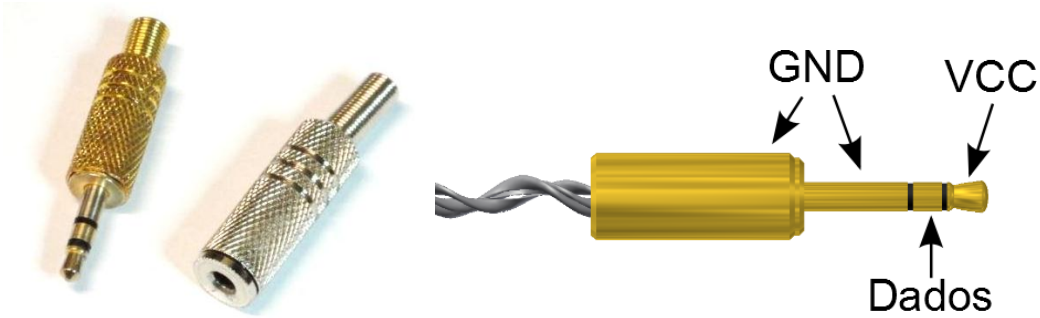


Figura 3.4: Conector P2 utilizados e funcionalidade atribuída a cada faixa de contato do *Slip Ring*.

Apesar das vantagens, a qualidade dos contatos durante a rotação se mostrou inferior ao desejável, também apresentando grande atrito entre os conectores macho e fêmea o que dificultava a rotação do sistema. Esses problemas foram amenizados lubrificando os contatos com grafite em pó.

3.2 Comunicação Serial

O envio dos dados a serem atribuídos às imagens geradas e ao controle do motor é fornecido por meio do protocolo RS232 de comunicação serial.

A comunicação é iniciada por um computador, utilizando o pacote VISA de comunicação serial presente no software LabVIEW. Como a maioria dos computadores e notebooks modernos não apresentam porta serial DB9, a alternativa foi utilizar a saída serial COM (USB) e um cabo conversor USB para RS-232. A conversão resulta em dados conforme o protocolo RS-232 simplificado, ou seja, somente 4 fios são necessário para a comunicação. Os sinais presentes nesses fios são: alimentação, terra, leitura e escrita. A pinagem pode ser vista na Figura 3.5:

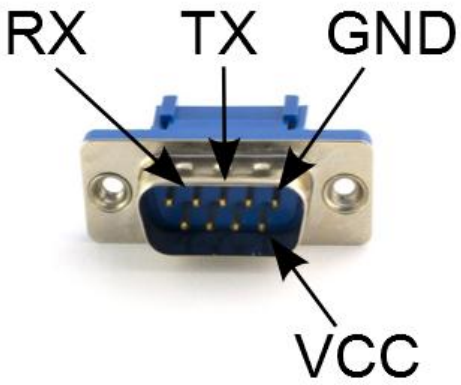


Figura 3.5: DB9 - Pinos utilizados para a comunicação serial.

O mesmo canal de comunicação controla dois receptores distintos: a placa giratória e a placa controle. Para cada uma das finalidades foram criadas palavras de comunicação diferentes, visando à robustez do sistema e à diminuição de interpretações erradas. A configuração da comunicação serial escolhida foi a utilização de palavras de 7 *bits*, não utilizando *bit* de paridade e *stop bit* de tamanho 1. A paridade é feita manualmente, se necessária.

A palavra de comunicação com a placa giratória é a mais simples, apresentando 3 *bits* de cabeçalho, 58 de dados e 2 de finalização.

- Os 3 *bits* de cabeçalho devem ser iguais a “010” para indicar que o destino dos dados é a placa giratória.
- Os 58 *bits* de dados indicam a atividade ou inatividade dos LEDs (1 e 0 respectivamente). O primeiro *bit* refere-se ao primeiro LED, de modo sucessivo até o ultimo *bit* e LED.
- Os 2 *bits* finais possuem somente a função de completarem um número de *bits* múltiplo de 7.

Um diagrama gráfico da palavra de comunicação com a placa giratória pode ser visto na Figura 3.6:

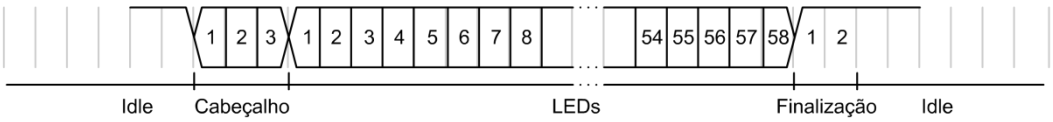


Figura 3.6: Palavra serial para controle dos LEDs.

Com o intuito de evitar erros de comunicação, a palavra de acionamento do motor DC presente na placa controle apresenta um *bit* de paridade. A paridade escolhida foi a impar, ou seja, o *bit* de paridade será "1" se o número de *bits* "1" na palavra for impar. A palavra completa possui 3 *bits* de cabeçalho, 8 de dados, 1 de paridade e 2 de finalização.

- Os 3 *bits* de cabeçalho devem ser iguais a "100" para indicar que o destino dos dados é o motor.
- Os 8 *bits* de dados indicam a velocidade de rotação do motor, variando de 0 a 255.
- O *bit* de paridade impar serve para garantir que a informação desejada chegue ao destino sem ser corrompida.
- Os 2 *bits* finais possuem somente a função de completarem um número de *bits* múltiplo de 7.

Um diagrama gráfico da palavra de comunicação com o motor pode ser visto abaixo:

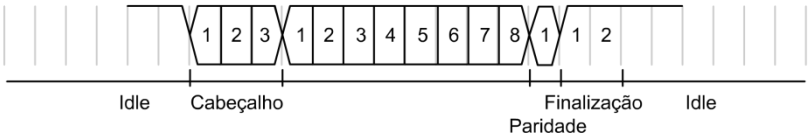


Figura 3.7: Palavra serial para controle do motor.

O protocolo RS-232 possui algumas taxas de transferência de *bits* (*baud rate*) padrão, não sendo possível alterá-las no software utilizado (LabVIEW). A seção 3.8 detalha o motivo da escolha da taxa de comunicação usada ser de 115200 *bits*/segundo.

3.3 Controle do Motor

O motor CC utilizando para a rotação da placa do LEDs é controlado utilizando modulação por largura de pulso (PWM - *Pulse-Width Modulation*). Esse método permite manter uma tensão fixa e variar somente o tempo em que a tensão fica em nível alto, não sendo necessário utilizar drivers analógicos. Esse método possui alta efetividade quando utilizado com cargas de alto fator indutivo, como um motor CC, porém possui a desvantagem de que a comunicação utilizada é digital o os

valores possíveis de *duty cycle* são discretizados e não contínuos como um controle mais fino requereria.

Foi escolhido 8 *bits* de dados para a palavra de comunicação com o motor, ou seja, 256 níveis possíveis de discretização (variando de 0 a 255). O motor escolhido apresenta uma boa relação custo/beneficio sendo de fácil acesso caso ocorra algum problema, e opera com tensão nominal de 12 V. O motor pode ser visualizado na Figura 3.8. A relação entre o *duty cycle* e o sinal de controle pode ser vista abaixo:

$$DutyCycle = [Controle] * \frac{100}{255} \qquad \text{[Eq. 1]}$$

Com essa equação determina-se que 8 *bits* de controle apresentam uma precisão de 0,39% do *duty cycle*.

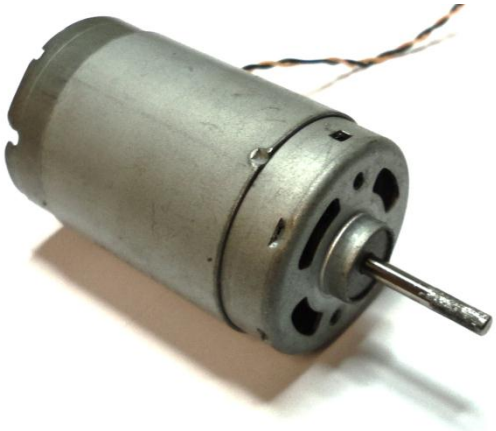


Figura 3.8: Motor CC.

O motor apresenta boa estabilidade quando excitado por uma fonte constante e sua velocidade pode ser controlada pelo usuário através do software em LabVIEW. O usuário deve escolher qual a porcentagem do *duty cycle* que o motor deve utilizar. Esse dado é enviado à placa controle onde é convertido em uma onda com modulação de pulso. Por sua vez, essa onda serve de controle para um circuito de potência que alimenta o motor.

Conectado ao eixo do motor está um *encoder* que gera um sinal digital com frequência proporcional à velocidade de rotação do motor. Esses pulsos ativam o envio de uma palavra de comunicação tendo como destino o software de controle. Essas palavras recebidas servem de *trigger* para o envio dos próximos dados de controle dos LEDs.

3.4 Encoder

Encoder rotativo é um método simples, barato e confiável da medição de velocidade angular por meio de ondas digitais geradas por um transdutor posicionado em um local onde faixas ativas e inativas passam sucessivamente em velocidade proporcional à desejada [22].

Sistemas mais simples e de medição de velocidade tendem a utilizar *encoders* de um único *bit*, ou seja, um único transdutor é utilizado, gerando uma única onda binária. Essa onda pode ser interpretada e indicar a velocidade instantânea ou posição angular caso seja utilizado juntamente com um sistema integrador. Sistemas mais complexos e que precisam de medição de ângulo tendem a utilizar *encoders* de vários *bits*, gerando várias ondas. Essas ondas juntas indicam em qual faixa de ângulo o sistema se encontra sem a necessidade de valores iniciais ou sistemas integradores.

Exemplos desses *encoders* podem ser vistos na Figura 3.9.

Figura 3.9: *Encoder* rotacional de 1 *bit* e 13 *bits* utilizando código Gray [22].

Vários tipos de transdutores podem ser utilizados para a geração das ondas digitais dependendo da situação em que são alocados e dos materiais utilizados nos discos. O transdutor utilizado nesse projeto é um sensor de radiação infravermelho próprio para ser utilizado como *encoder* rotativo.

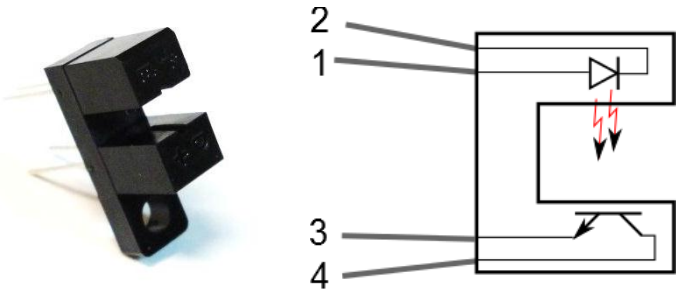


Figura 3.10: Sensor e receptor infravermelho próprio para *encoders*.

O sensor apresenta dois braços que são colocados envoltos da área ativa do disco. Um dos braços apresenta um LED emissor de radiação infravermelho próxima e o outro um componente semicondutor que conduz corrente quando excitado pela radiação. Um esquema simplificado do sensor e receptor pode ser visto na Figura 3.10.

Devido ao alto custo de um *encoder* comercial foi desenvolvido um especialmente para este projeto. O *encoder* rotativo utilizado foi inicialmente desenvolvido no software Altium Designer, voltado a fazer placas de circuito impresso. O projeto era composto de 128 faixas de furos espaçadas uniformemente, gerando outras 128 faixas sólidas, ou seja, 128 PPR. Cada faixa era composta por 4 furos de 0,7mm, resultando em um total de 512 furos.

O *encoder* foi confeccionado em uma placa de fibra de vidro para circuito impresso em uma máquina fresadora automática cedida pela empresa Opto Eletrônica S.A. e pode ser vista na Figura 3.11. A placa teve seu metal condutor removido para não causar reflexão, já que o transdutor utilizado é um sensor e receptor infravermelho.

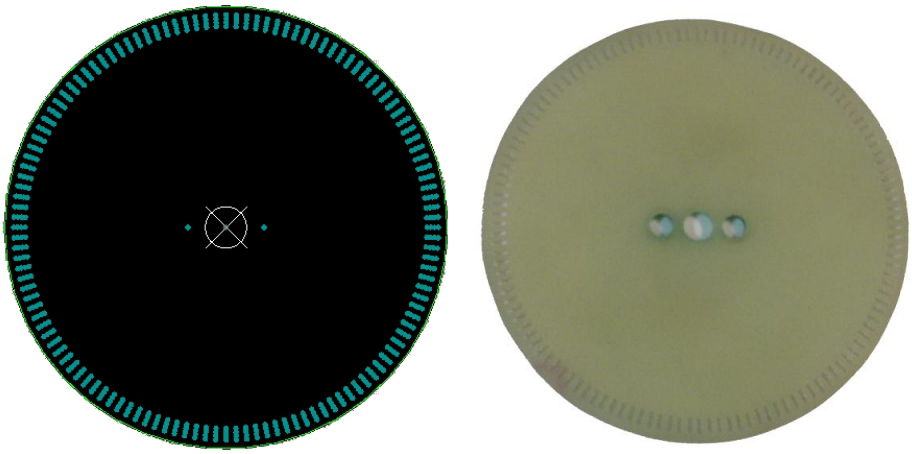


Figura 3.11: *Encoder* de fibra de vidro: Projeto e placa real.

Após o sistema montado com motor, encoder e sensores, foi observado que o encoder não funcionava como esperado devido às imperfeições nos furos que criavam as faixas abertas do *encoder*. O sensor captava grande quantidade de ruido, tornando impossível utilizar o sinal com precisão. Melhoras mecânicas foram tentadas, como lixar os vãos a fim de remover as imperfeições, mas devido ao tamanho da área de trabalho, melhores resultados não foram obtidos e, logo um novo projeto de *encoder* teve que ser desenvolvido.

A segunda alternativa apresentou uma ideia mais conservadora, consistindo em um disco de acetato transparente com segmentos radiais impressos em preto. Para melhorar sua rigidez mecânica o *encoder* final ganhou duas camadas protetoras de fibra de vidro uma de cada lado, criando um sanduíche de acetato. As peças de fibra ao acetado foram conectadas juntamente com a fixação do *encoder* na peça mecânica que o acopla ao motor: o conector inferior fêmea. O novo *encoder*, que pode ser visto na Figura 3.12, apresentou boa estabilidade mecânica à rotação e não apresentou ruídos significativos.

Figura 3.12: *Encoder* de acetato finalizado com suportes de fibra de vidro.

3.5 Placa Controle

Para controlar o motor CC e converter o nivel de tensão da comunicação serial foi projetada uma placa de circuito impresso denominada placa controle.

Essa placa apresenta os seguintes subsistemas:

- Conversão da comunicação serial de nivel RS-232 para TTL;
- Interpretação dos dados, tanto da palavra serial de controle do motor quanto do sinal do encoder;
- *Driver* de corrente para o motor;
- Alimentação e condicionamento de sinal do sensor IV.

Cada um desses subsistemas será discutido na sequência.

3.5.1 Condicionamento do sinal de comunicação serial

Como discutido na seção 2.2 o protocolo RS-232 utiliza uma faixa de tensão não compatível com circuitos TTL, não sendo conveniente utilizar esses níveis de tensão em um circuito de processamento de dados. Devido a isso foi necessário a sua conversão para uma faixa de tensão mais simples de ser trabalhado. A faixa escolhida foi a faixa de 0 a 5 V com lógica positiva, ou seja, uma tensão mais alta (5 V) indica um nível lógico “1” e uma tensão menor (0 V) indica um nível “0”, como pode ser visto na Figura 3.13.

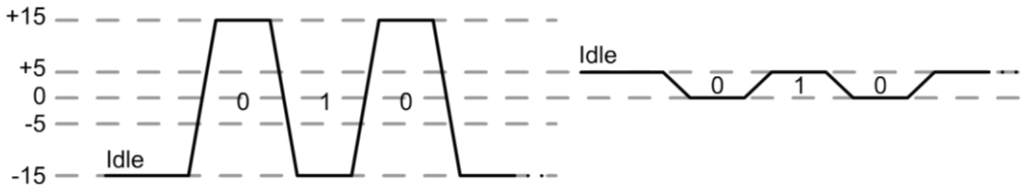


Figura 3.13: Nível de tensão de linha: RS-232 e TTL respectivamente.

Essa conversão de nível de tensão é bastante utilizada e possui circuitos integrados próprios para sua realização sendo o CI (circuito integrado) escolhido o MAX232 criado pela Maxim Integrated Products. O circuito utilizando de conversão do nível de tensão pode ser visto na Figura 3.14.

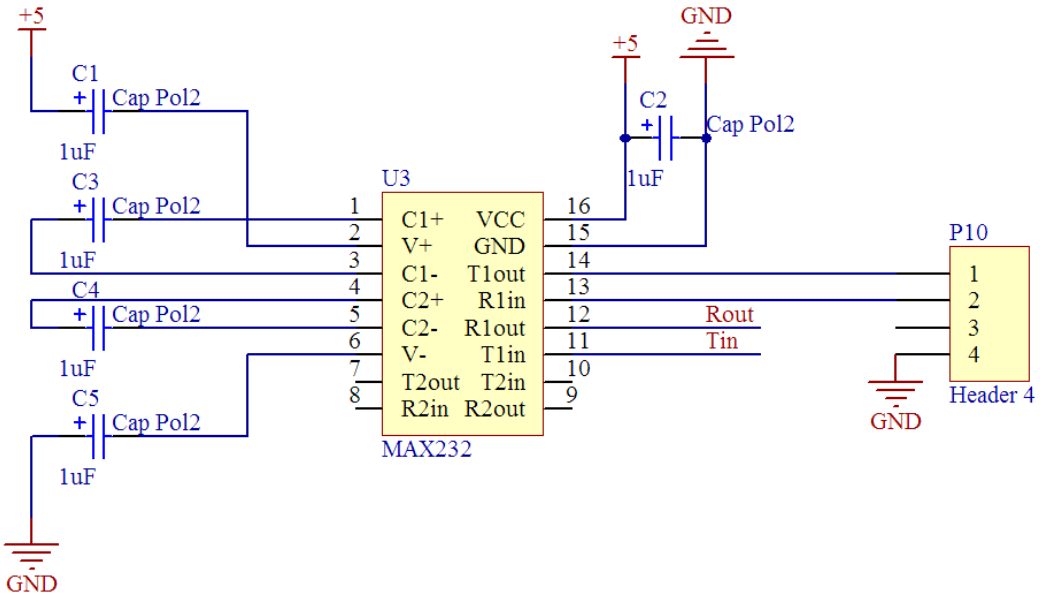


Figura 3.14: Circuito de conversão do nível de tensão da comunicação serial.

O circuito apresenta dois possíveis canais de conversão, porém somente um foi utilizado. Os dados seriais entram pela porta 13 (R1in) e saem pela porta 11 (T1in) já com nível compatível com lógica TTL. A mesma conversão é feita no sentido contrário, o sinal retorna em nivel TTL na porta 12 (R1out) e sai na porta 14 (T1out) com nivel compativel com RS-232.

Os cinco capacitores tem função de elevar a tensão ao nivel RS-232, já que o CI é alimentado com 5 V.

3.5.2 Recepção e envio dos dados seriais

Com os dados seriais já em nível de tensão baixo pode-se projetar a lógica de interpretação de dados. Devido a motivos históricos do laboratório de eletrônica digital, todo o processamento foi realizado em uma CPLD Altera EPM7128SL84-15 série MAX 7000. O esquema de ligações elétricas utilizado pode ser visto na Figura 3.15, não apresentando grandes problemas ou dificuldades.

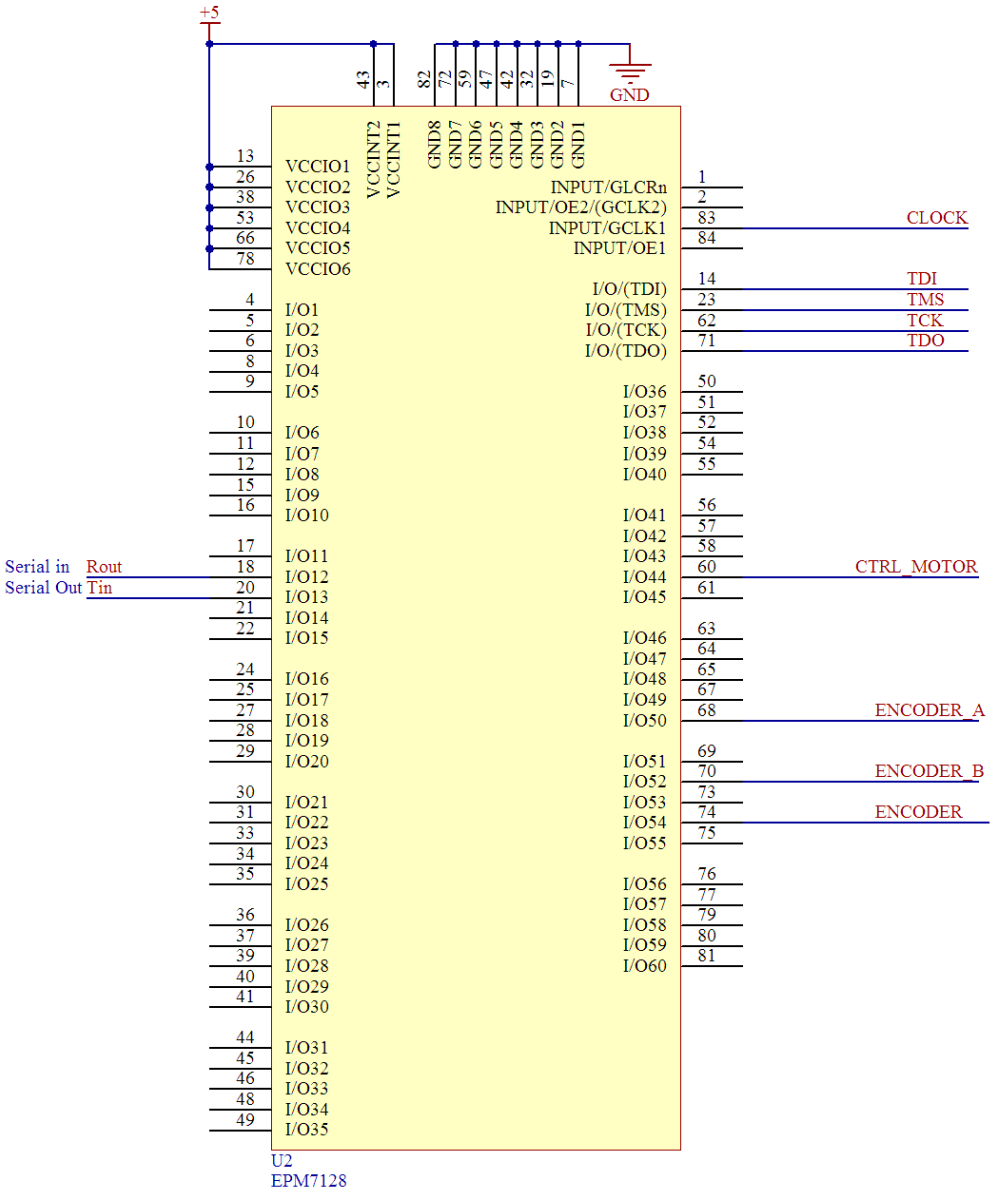


Figura 3.15: Esquema elétrico da CPLD na placa controle.

O esquema apresenta 11 pinos de entrada e saída. Quatro desses pinos (TDI, TMS, TCK e TODO) são utilizados na programação da CPLD. Rout e Tin são utilizados para leitura e escrita de dados seriais respectivamente. CTRL_MOTOR é o pino utilizado para o controle do motor CC através de PWM. ENCODER A e B são entradas para os sinais do *encoder*, foram projetados duas para o caso de mais de um *encoder* ser utilizado.

O pino CLOCK serve como entrada para o *clock* de funcionamento da CPLD. Como a função principal da placa é o processamento da comunicação serial, a frequência escolhida foi de 11.0592MHz. Essa frequência especifica, quando dividida por um número inteiro, possibilita exatamente a geração das frequências padrões de comunicação serial. Por exemplo, para gerar um *baud rate* de 9600 bps, divide-se o *clock* por exatamente 11520. A descrição de hardware utilizada na CPLD será apresentada na seção 3.9.1.

3.5.3 *Driver* de corrente para o motor

O sinal de controle do motor CC oriundo da CPLD apresenta saída de no máximo 5 V e 25 mA, não sendo o suficiente para ativar um motor, por esse motivo foi projetado um *driver* de corrente. O circuito escolhido baseia-se em um transistor TIP122 com capacidade de 100 V de tensão entre emissor e coletor e 5 A de corrente de coletor. O sinal de controle entra na base do transistor que, devido a sua polarização funciona como uma chave, abrindo quando a tensão de excitação é nula e conduzindo quando 5 V. Em paralelo aos braços coletor e emissor do transistor encontra-se um diodo retificador com alta capacidade de corrente para barrar qualquer corrente de fuga e não deixar que estas entrem no transistor sobrecarregando-o. O motor encontra-se conectado diretamente no emissor do transistor como pode ser visto no esquema apresentado na Figura 3.16.

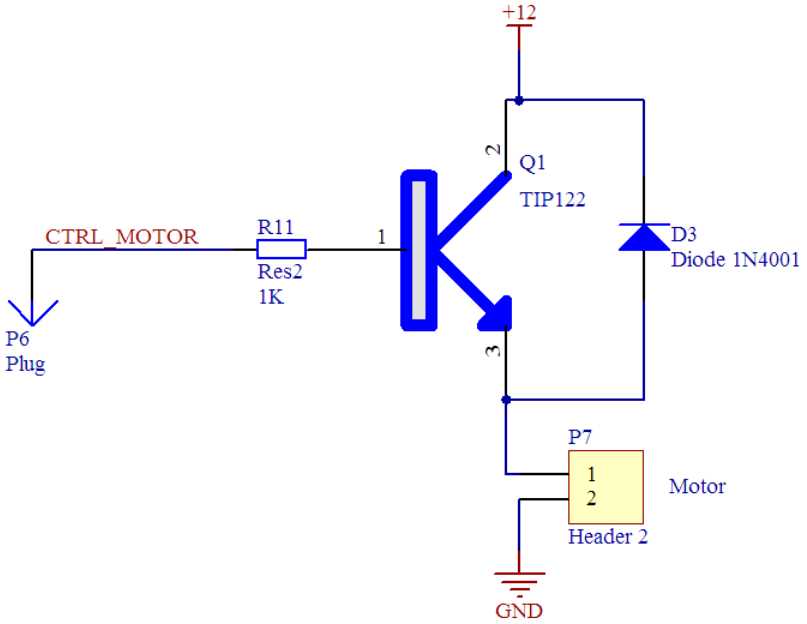


Figura 3.16: Esquema elétrico do *driver* de potência para o motor CC.

Vale ressaltar que a alimentação do transistor, e logo do motor, é de 12 V, sendo de uma fonte diferente da alimentação de 5 V do restante do circuito.

3.5.4 Condicionamento do sensor e emissor infravermelho

O sensor e receptor infravermelho (IV), cujos esquemas elétricos equivalentes estão mostrados na Figura 3.10, foram polarizados pelo circuito representado na Figura 3.17 o qual possibilita manter tensões factíveis de serem trabalhadas. Os pinos 1, 2 3 e 4 da Figura 3.10 são os mesmos pinos do conector P4 que pode ser visto na Figura 3.17 abaixo:

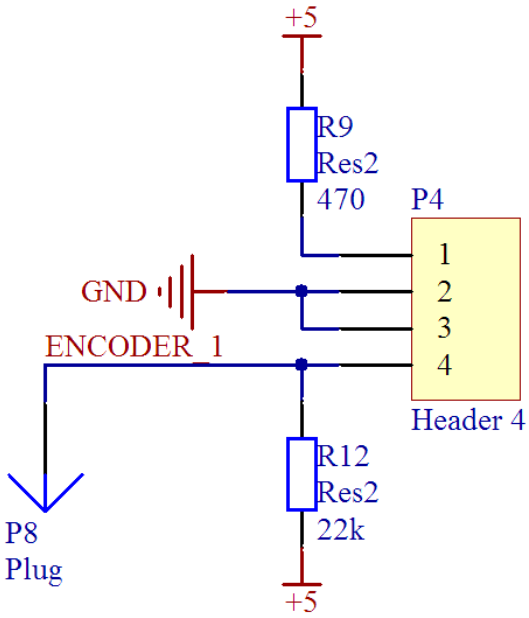


Figura 3.17: Condicionamento do sensor e receptor IV.

A saída do sensor IV é indicada pelo nome ENCODER_1 e pela ponta de testes P8. O sinal resultante apresenta uma onda com frequência proporcional à velocidade de rotação do motor, porém não apresenta nível de tensão compatível com a lógica digital utilizada. Para adequar o sinal do *encoder* ao nível de tensão utilizado é necessário a utilização de um circuito comparador que pode ser visto na Figura 3.18.

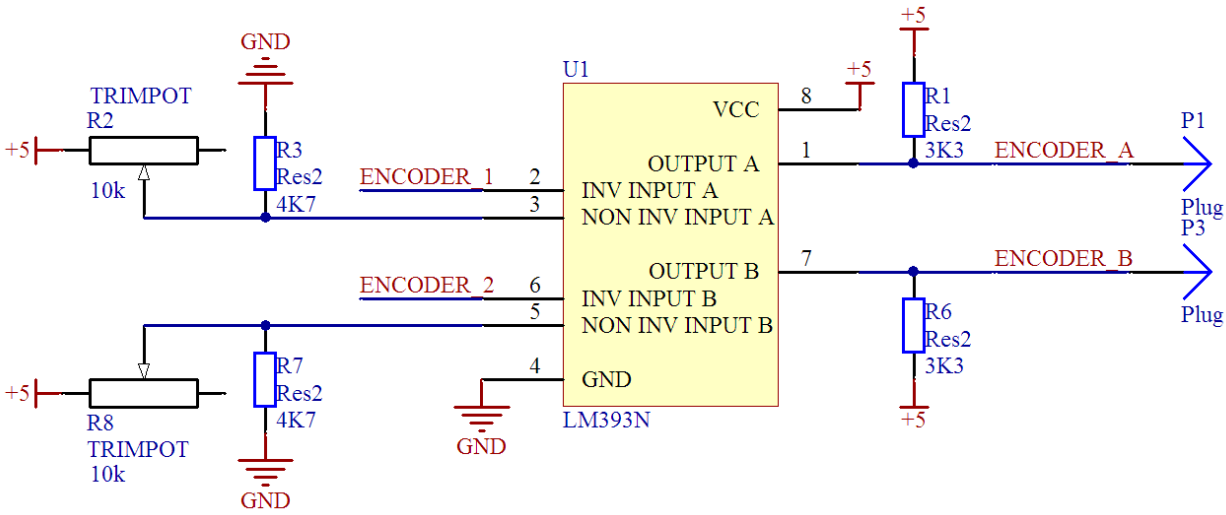


Figura 3.18: Comparador analógico utilizado para adequar a tenso sinal do *encoder*.

O circuito baseia-se na comparação do sinal oriundo do *encoder* com uma tensão de referência previamente definida. Se o sinal do *encoder* for superior à tensão de referência, a saída encontra-se em nível alto, senão em nível baixo.

3.5.5 Fabricação da placa de circuito impresso

O projeto da placa de circuito impresso foi desenvolvido no software Altium Designer Summer 09 e pode ser visto na Figura 3.19.

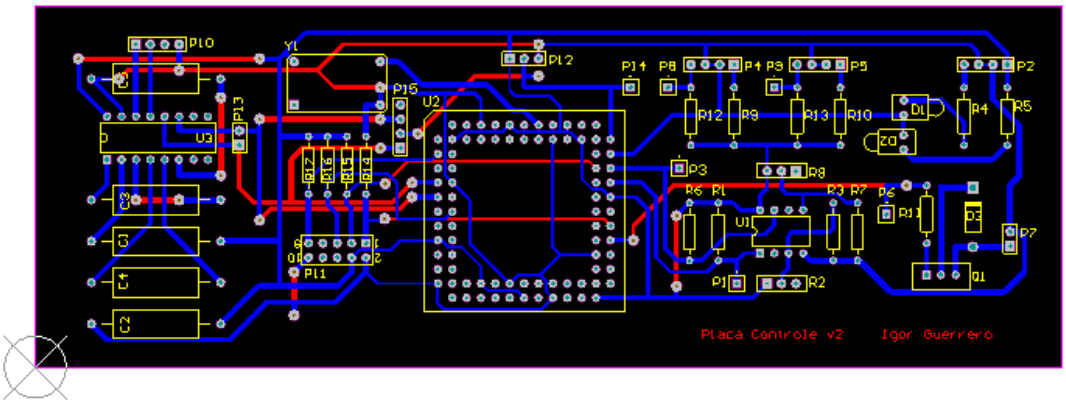


Figura 3.19: Placa controle: projeto.

A fabricação foi feita em fibra de vidro através de uma fresadora automática cedida pelo departamento de engenharia elétrica e de computação e manipulada pelo técnico de manutenção Odair da Silva. Uma camada de verniz Isotec para acabamento em circuitos impresso foi utilizada para proteger o cobre contra corrosão e oxidação.

Além dos circuitos apresentados da Figura 3.14 à Figura 3.18 a placa inclui dois LEDs de indicação de alimentação, uma para a fonte de 5 V e outro para a fonte de 12 V. O *clock* foi gerado utilizando um circuito oscilador comercial com encapsulamento DIP14. Devido a problemas de aquecimento do transistor foi criada uma extensão para possibilitar o posicionamento do dissipador em frente a uma ventoinha. Para facilitar a troca do transistor foram utilizados três conectores no lugar de solda. A placa finalizada pode ser vista na Figura 3.20 e seu esquema elétrico completo pode ser visto no Apêndice B – Esquemas elétricos.

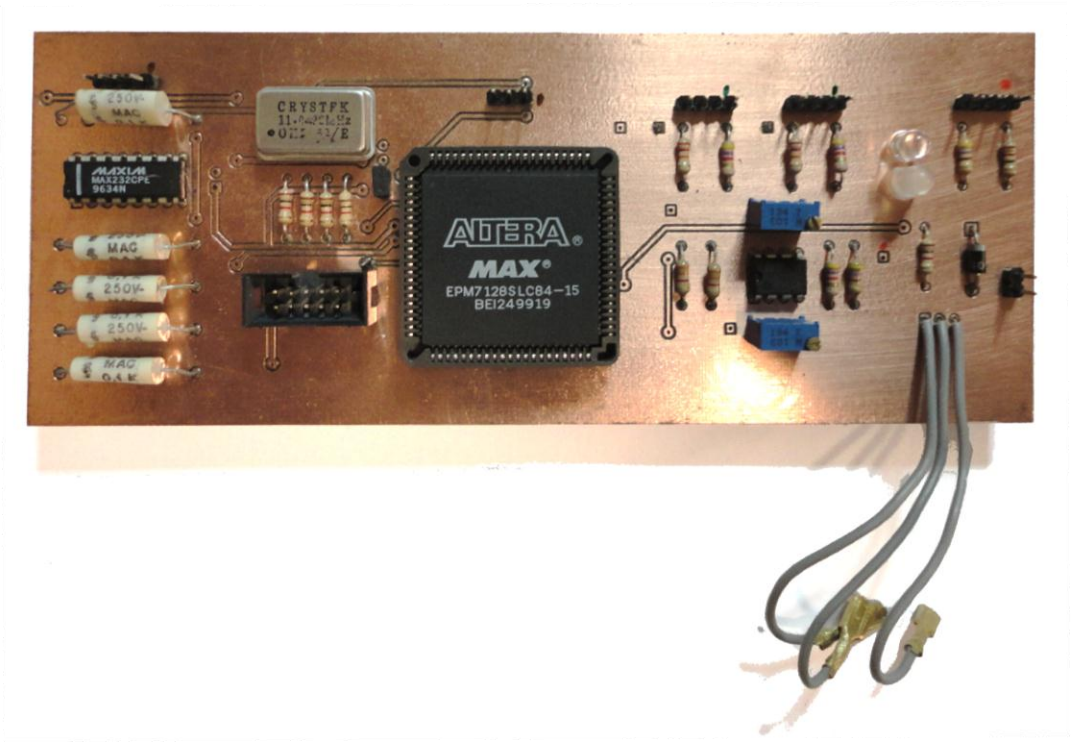


Figura 3.20: Placa controle montada.

3.6 Placa Rotativa

A placa rotativa é a parte móvel do sistema, sendo fixada somente em dois pontos: o acoplamento com o motor e o conector *Slip Ring*.

A placa possui duas funções:

- Interpretação dos dados seriais e
- Acionamento dos LEDs.

Cada uma dessas funções será descrita abaixo:

3.6.1 Recepção dos dados seriais

Utilizando a mesma lógica de circuito da placa controle, a interpretação dos dados nesta placa é realizada por uma CPLD Altera EPM7128SL84-15 série MAX 7000. O circuito utilizado pode ser visto na Figura 3.21.

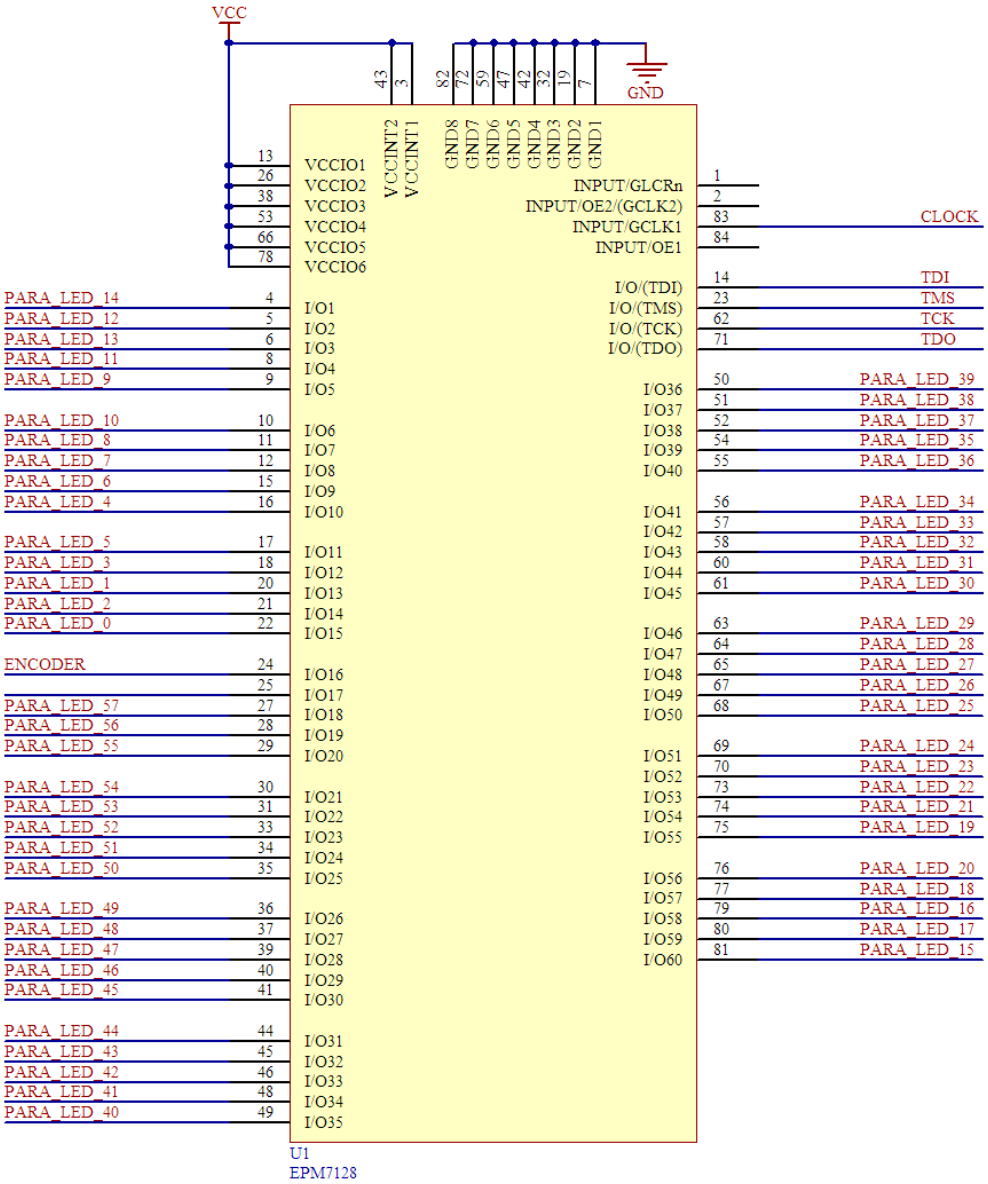


Figura 3.21: Esquema elétrico da CPLD na placa rotativa.

A CPLD utilizada apresenta 84 pinos ao total, sendo que 60 podem ser utilizados com entrada e saída de dados. Devido ao número de pinos e o formato mecânico da placa foi decidido utilizar 58 LEDs de indicação, sobrando dois pinos livres. Um desses pinos é utilizado como entrada de dados serial. Vale ressaltar que esses dados estão em um nível de tensão compatível com lógica TTL já que passam pelo condicionamento na placa controle. O pino restante não foi utilizado na montagem final, sendo necessário somente nas etapas de *debug* da linguagem de descrição de hardware inserida na CPLD.

O *clock* utilizado possui a mesma frequência de 11.0592MHz que a placa controle.

3.6.2 Acionamento dos LEDs

Os 58 LEDs apresentam controle individual, cada um com seu sinal de controle próprio. Para não sobrecarregar a CPLD foi projetado um pequeno *driver* de corrente para cada LED.

O sinal de saída da CLPD ativa a base de um transistor possibilitando uma alteração em seu modo de condução, fechando ou abrindo o anodo do LED. Um dos 58 circuitos montados pode ser visto na Figura 3.22.

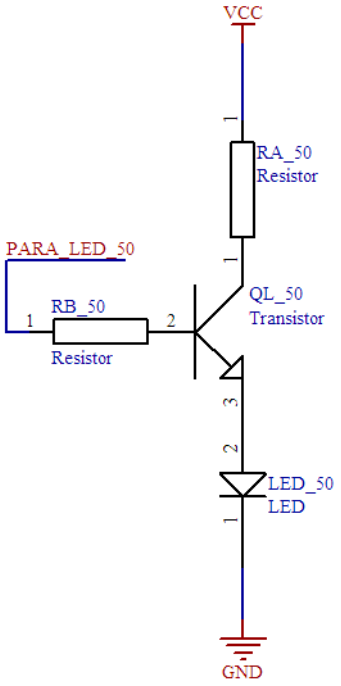


Figura 3.22: *Driver* de corrente para os LEDs da placa rotativa.

A placa rotativa não apresenta grande complexidade em circuito, apresentando somente um oscilador gerador do *clock* e um conector voltado para a programação da CPLD.

3.6.3 Fabricação da placa de circuito impresso

O projeto da placa rotativa também foi desenvolvido no software Altium Designer Summer 09 e pode ser visto na Figura 3.23. Para que os LEDs que formam as imagens fossem bem visíveis, estes foram posicionados na lateral da placa tendo um contato em cada face condutora. Foram utilizados LEDs de 3 mm devido ao seu menor tamanho, se comparado aos LEDs convencionais de 5 mm e maior facilidade de solda quanto aos com encapsulamento SMD. Uma parte da solda pode ser vista na Figura 3.24.

Como a placa anterior, essa também foi confeccionada em fibra de vidro na fresadora automatica do departamento de engenharia elétrica e de computação e pode ser vista na Figura 3.25.

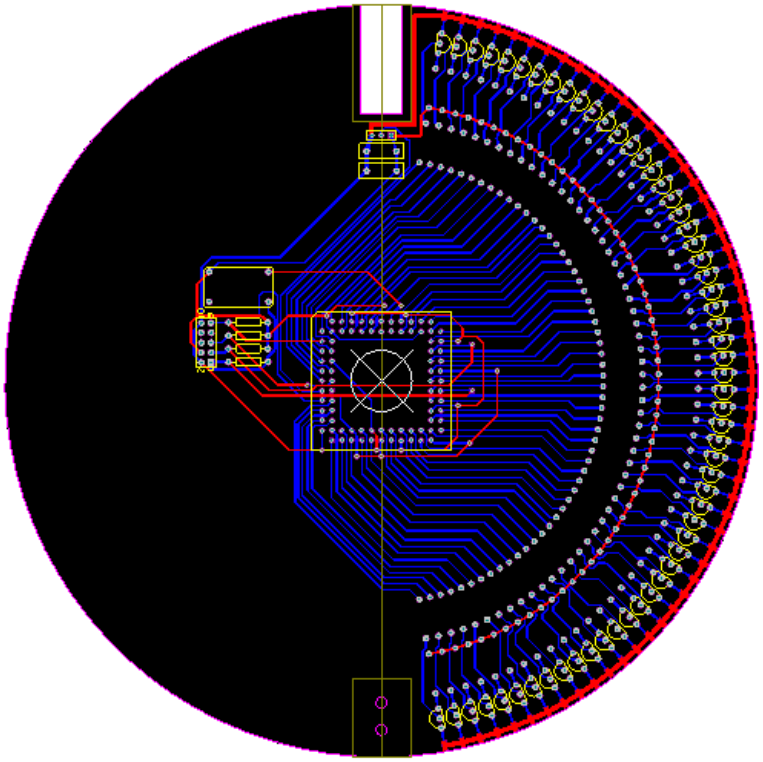


Figura 3.23: Placa rotativa: Projeto.

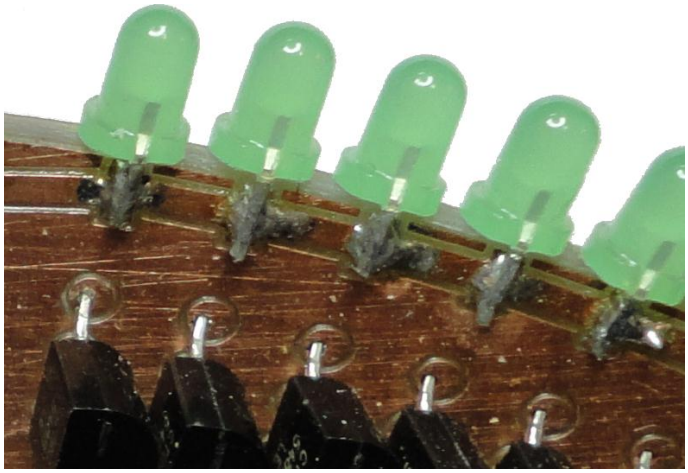


Figura 3.24: Destaque para os LEDs soldados em ambas as faces da PCI.

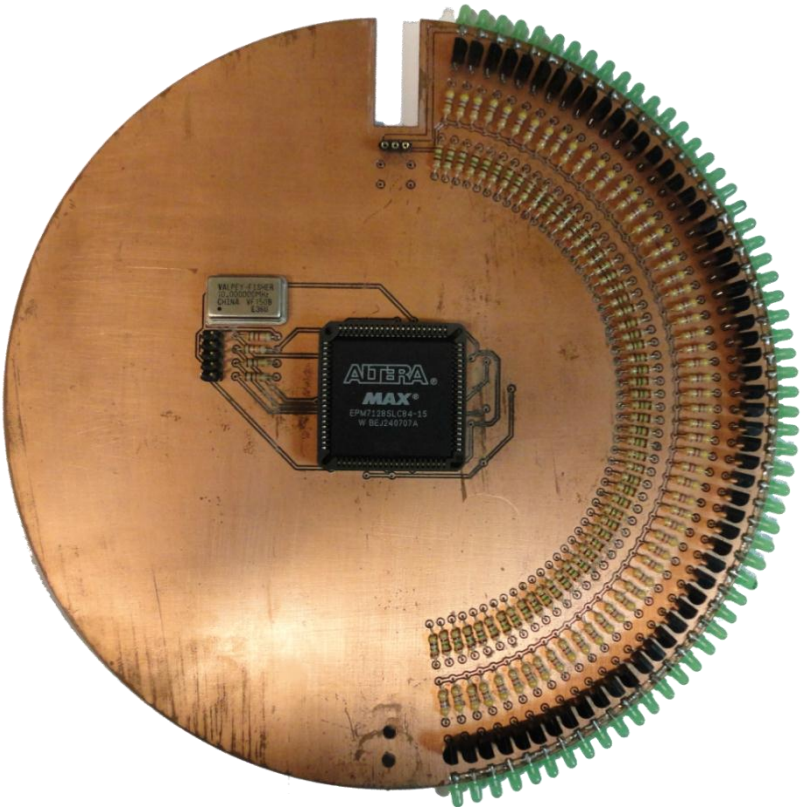


Figura 3.25: Placa rotativa montada.

Uma camada de verniz Isotec também foi utilizada para proteger o cobre contra corrosão e oxidação, porém essa camada não apresentou qualidade comparável à placa controle, sendo facilmente danificada, mas sem o comprometimento da qualidade dos contatos.

O rasgo na parte superior e os buracos na parte inferior são voltados à fixação mecânica da placa em seus suportes.

3.7 Sistema mecânico

Com as partes elétricas projetadas foi realizado o projeto das peças mecânicas que dão segurança e estabilidade ao sistema. Ao total serão necessárias cinco partes mecânicas distintas:

- Conexão Motor – Placa Rotativa
- Suporte *Slip Ring* Fêmea
- Suporte *Slip Ring* Macho
- Suporte de Fixação à Caixa
- Suporte do Motor

Para reduzir os gastos foi utilizada uma caixa comercial para suportar todo o sistema.

As cinco partes projetadas serão descritas a seguir:

3.7.1 Conexão Motor – Placa Rotativa

A placa rotativa precisa ser fixada ao motor CC de um modo seguro e estável. O motor utilizado, como pode ser visto na Figura 3.8, apresenta eixo cilíndrico e liso. A placa rotativa foi desenvolvida com dois furos de 3m de diâmetro centrados em seu eixo de rotação para possibilitar a confecção de uma peça de conexão. Essa peça assim como todas as subsequentes, foi projetada utilizando o software Inventor Professional 2011, da Autodesk.

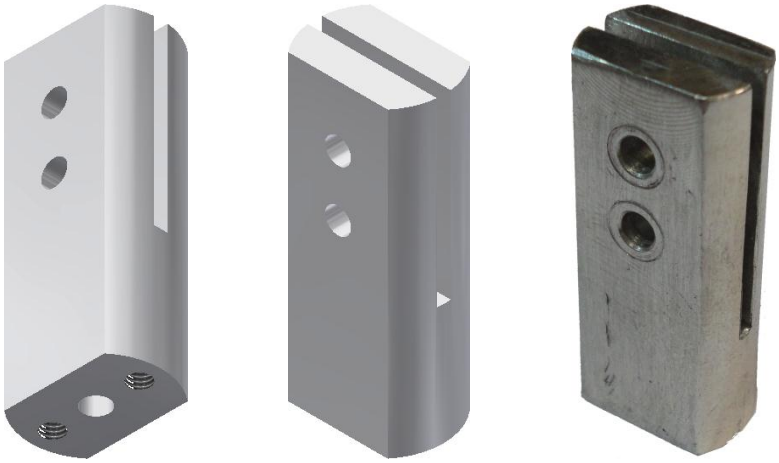


Figura 3.26: Conector Motor - Placa Rotativa: Vistas de projeto e peça finalizada.

O conector, que pode ser visto na Figura 3.26, apresenta um rasgo em quase toda sua extensão possibilitando o posicionamento da placa rotativa. Dois parafusos fixam a PCI (placa de circuito impresso) ao conector. A parte inferior apresenta um furo para o encaixe do eixo do motor CC e dois encaixes de parafuso para a fixação do *encoder* descrito no item 3.1. Essa peça foi fabricada em alumínio e não houve a necessidade de tratamento superficial.

3.7.2 Suporte *Slip Ring* Fêmea

O rasgo na parte superior da placa rotativa foi feito para o encaixe da parte fêmea do conector P2 que é utilizado como *Slip Ring*. A peça apresenta vincos de 2 mm em três lados para o seu posicionamento na placa rotativa. A parte inferior é vazada para a saída de três fios que levam os contatos do *Slip Ring* até a PCI. A parte superior apresenta uma rosca para a fixação do conector P2. Assim como o encapsulamento do conector P2, a peça, que é feita de alumínio, está aterrada em relação às conexões elétricas e pode ser vista na Figura 3.27.

Figura 3.27: Suporte *Slip Ring* Fêmea: Vistas de projeto e peça finalizada.

3.7.3 Suporte *Slip Ring* Macho

A parte superior do conector P2 necessita de um suporte para não ser rotacionado juntamente com a sua parte inferior, para isso foi projetado uma peça que possui duas funções: impedir essa rotação e apresentar resistência mecânica aumentando a estabilidade e impossibilitando que o placa giratória saia do eixo de rotação.

A peça feita em alumínio mantem o conector seguro com a ajuda de dois parafusos exercendo forças em direções opostas, segurando o conector. A estabilidade mecânica é obtida através de um tubo em formato de L que conecta a peça, e logo a PCI rotativa, com a caixa de suporte geral do sistema através do suporte de fixação à caixa. Na Figura 3.28 abaixo pode ser visto a peça finalizada já com o conector P2 macho posicionado e assegurado pelos parafusos.

Figura 3.28: Suporte *Slip Ring* Macho: Vistas de projeto e peça finalizada.

3.7.4 Suporte de Fixação à Caixa

O tubo em L que pode ser visto na Figura 3.28 e conecta as pecas do *Slip Ring* à caixa de suporte apresenta seu outro extremo de conexão nessa peça.

A única função dessa peça é manter o tubo de conexão estável e conectado à caixa de suporte. A peça apresenta três braços de conexão com a tampa da caixa, impossibilitando que o tubo se movimente em dois eixos. O eixo vertical é assegurado utilizando-se dois parafusos na parte traseira da peça. Assim como as peças anteriores essa também foi fabricada em alumínio e pode ser vista na Figura 3.29.

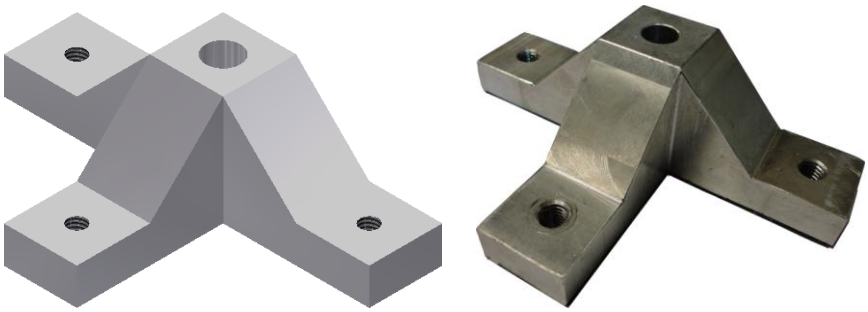


Figura 3.29: Suporte de Fixação à Caixa: Vistas de projeto e peça finalizada.

3.7.5 Suporte do Motor

O motor CC gera toda a energia de rotação do sistema e, para que tal tarefa seja realizada com sucesso, o motor deve estar seguramente acoplado à caixa de suporte.

Um segmento de tubo de PVC encobre o motor e, esse tubo, por sua vez, é conectado à caixa de sustentação através de um bloco de madeira.

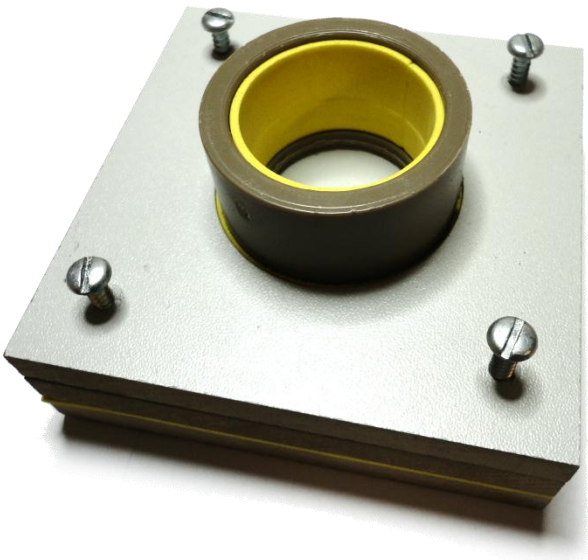


Figura 3.30: Suporte do Motor

3.8 Temporização do sistema

Neste capítulo é mostrado o calculo da temporização do sistema, incluindo o cálculo da taxa de transferência dos *bits* na comunicação serial e da velocidade mínima desejada para o motor.

Como detalhado no item 3.2 a maior palavra de comunicação utilizada é a palavra destinada ao controle dos LEDs. Essa palavra contém 63 *bits* de dados e, de acordo com o protocolo utilizado e o software LabVIEW, a palavra é dividida em 9 bytes de 7 *bits* de dados cada. Além dos sete *bits* de dado, cada *byte* apresenta um *bit* de inicio e um de finalização. O intervalo entre os bytes apresenta um *bit* extra com a mesma tensão do sinal em *idle*, logo o número de *bits* usado no sinal é:

$$9 * (7 + 2) + 8 * 1 = 89 bits = maior_palavra \qquad \text{[Eq. 2]}$$

Usando uma taxa de T *bits* por segundo uma palavra completa leva 89/T segundos para ser transmitida.

Considerando que o motor tenha uma velocidade angular de V RPS = V*60 RPM e um *encoder* de C PPR, tem-se:

$$V\ rotações \rightarrow 1\ segundo \qquad \text{[Eq. 3]}$$

$$C\ pulsos \rightarrow 1\ rotação \qquad \text{[Eq. 4]}$$

$$1\ pulso \rightarrow \frac{1}{C * V}\ segundo \qquad \text{[Eq. 5]}$$

O trem de pulsos gerado pelo *encoder* ativa o envio de mensagens seriais com destino ao computador de processamento. O computador por sua vez espera essas mensagens em um loop continuo e, quando a mensagem do e*ncoder* é recebida, ativa o envio da próxima palavra de dados.

Com essa logica tem-se que o tempo entre os pulsos do *encoder* deve ser maior que o tempo de envio de uma palavra serial:

$$\frac{1}{C * V} > \frac{89}{T} \qquad \text{[Eq. 6]}$$

O *encoder* projetado apresenta 64 PPR e, como discutido no item 3.2 as taxas de comunicação serial são discretizadas nos seguintes valores: 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200, 300, 110 *bits*/segundo.

Utilizando 115200 *bits*/segundo tem-se:

$$\frac{1}{64 * V} > \frac{89}{115200} \rightarrow V < \frac{115200}{64 * 89} \rightarrow V < 20{,}2\ RPS \qquad \text{[Eq. 7]}$$

Com essa velocidade máxima é possível obter um aspecto quase estático da imagem para o olho humano, possibilitando uma boa visualização.

3.9 Circuitos do CPLD

Esse item aborda a lógica e a utilização de descrição de hardware utilizada nos CPLDs. A linguagem de descrição de hardware utilizada foi VHDL e todos os códigos foram gerados e programados utilizando o software Quartus II 11.1 Web Edition. Os códigos completos em VHDL podem ser encontrados no Apêndice E – Descrição de hardware: VHDL. Para simplificar a compreensão de desenvolvimento os códigos foram segmentados em blocos com tarefas específicas. Esses blocos são unidos utilizando o recurso diagrama em blocos do programa Quartus II e podem ser vistos no Apêndice D – Descrições de hardware: Diagrama em blocos.

3.9.1 Lógica VHDL da Placa Controle

Como descrito na seção 3.5.2, a CPLD presente na placa de controle possui três entradas funcionais e somente duas saídas. A diagramação em blocos da lógica utilizada por essa CPLD pode ser vista na Figura 3.31.

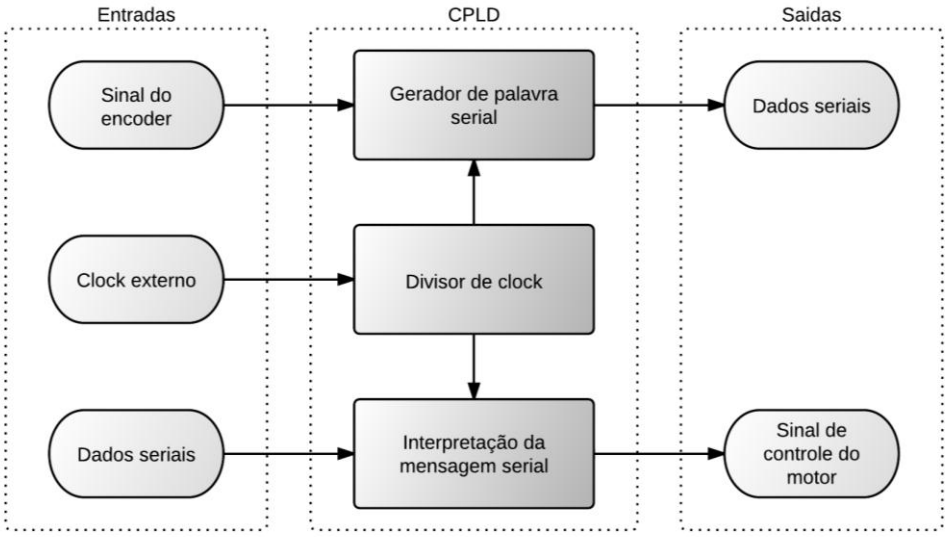


Figura 3.31: Esquema em blocos da lógica da CPLD: Placa Controle.

A entrada *clock* externo injeta no hardware um *clock* muito mais alto que o necessário para a geração do *baud rate*. Para que a taxa de transferência de *bits* seja atingida, esse *clock* deve passar por uma divisão, diminuindo sua frequência de oscilação, casando o *clock* gerado com o necessário. Essa nova frequência de excitação gerada interage com dois outros subsistemas: a interpretação e a geração de palavras seriais.

A interpretação das palavras seriais tem como objetivo receber os sinais puros e liberar um sinal PWM para a excitação do motor. A CPLD exerce essa tarefa através de um código VHDL que segue o seguinte procedimento:

- Aguardar o sinal serial entrar em nível “0”, significando o *bit* de inicio;
- Utilizando *flip flops* registar todos os *bits* de dados e de paridade. Esse registro transforma os dados seriais em paralelos;
- Verificar o *bit* de paridade;
- Se a paridade estiver correta enviar todos os *bits* em paralelo para um bloco gerador de PWM, se incorreta ignorar os dados;
- O bloco gerador de PWM adquire os *bits* e, utilizando uma frequência maior, gera um sinal de período fixo, mas com o *duty cycle* proporcional aos *bits* de dados adquiridos;
- Voltar ao estado inicial.

A última entrada da CPLD apresenta o sinal de saída do *encoder* já condicionado em uma onda quadrada. O objetivo desse sinal é sincronizar o recebimento do próximo dado serial, para isso esse sinal ativa um envio serial utilizando a seguinte lógica:

- Aguardar a subida de borda do sinal do *encoder*;
- Um bloco gerador de dados serial adquire das portas internas os *bits* em paralelo necessário para gerar uma palavra;
- Sincronamente com a taxa de transferência, *bit* a *bit* é enviado;
- Ao fim do envio o bloco volta à sua tarefa inicial.

3.9.2 Lógica VHDL da Placa Rotativa

Comparativamente a lógica necessária na CPLD da placa rotativa é mais simples que na placa controle. Somente dois blocos são necessários: Divisor de *clock* e interpretação das mensagens seriais. Um esquema pode ser visto na Figura 3.32.

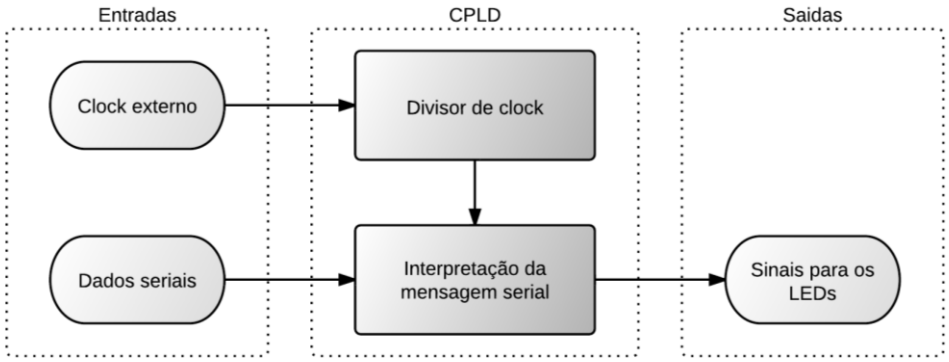


Figura 3.32: Esquema em blocos da lógica da CPLD: Placa Rotativa.

O divisor de *clock* é idêntico ao citado anteriormente, transformando 11,0592 MHz na taxa de transferência de *bits* necessária.

O interpretador de mensagens é uma versão simplificada, porém maior do que seu similar na placa controle. Esse bloco apresenta menos funções, não sendo necessário processar os dados após a transformação de serial em paralelo, porém o número de *flip flops* necessário para o registro de todos os *bits* é muito maior.

Originalmente o projeto abrangeria vários níveis de intensidade para cada LED em vez dos dois níveis utilizados, porém essa ideia necessitaria de mais macro células na CPLD do que as utilizadas atualmente suportam. Para possibilitar um aumento de capacidade seria necessário substituir a CPLD por uma de maior capacidade ou alterar o tipo de componente para utilizar uma FPGA (*field-programmable gate array*), extrapolando a lista de componentes disponíveis no laboratório e sendo necessário um nível de precisão de montagem maior que o atual, não sendo esse o objetivo do projeto.

3.10 Software

O software de controle foi completamente desenvolvido em LabVIEW 2011 para cumprir os seguintes objetivos:

- Possibilitar uma interface amigável ao usuário, possibilitando-o escolher qual imagem exibir;
- Criar um canal de comunicação com a porta USB de maneira transparente;
- Fazer toda a comunicação serial necessária para o controle do motor e dos LEDs;

O software, que pode ser visto integralmente no Apêndice F – Softwares em LabVIEW apresenta três secções principais: inicialização, processamento e finalização. A área de inicialização tem como função criar o canal de comunicação serial com todas as configurações necessárias. A área de finalização tem como objetivo fechar esse canal.

A secção central é a mais importante, onde ocorre todo o processamento de dados. O software foi desenvolvido utilizando a logica de dois *loops* executando em paralelo. O *loop* 1 apresenta um bloco controlador de eventos, recebendo e processando todos os dados de entrada de usuário. O usuário possui o controle de quatro ações distintas:

- Abrir uma imagem do arquivo através do botão Imagem.
- Alterar a velocidade do motor digitando um número de 0 a 255 na caixa de texto.
- Alterar a imagem manualmente, selecionando os nos pixels que deseja ativar ou desativar.
- Sair do software e finalizar o canal de comunicação através do botão Sair.

A tela frontal do software, que apresenta os botões citados, pode ser vista na Figura 3.33.

Quando o usuário pressiona o botão Imagem, uma tela de busca do Windows aparece para a escolha de uma imagem *bitmap* de dimensões 64x58 pixels. Com a imagem escolhida o software passa-a por uma binarização por meio da comparação da intensidade de cor com o nível médio possível.

Um segundo *loop* controla o envio dos dados destinados aos LEDs da placa rotativa. Uma variável, botão Iniciar, controla se os dados devem ou não ser enviados à porta serial, possibilitando o início e término da transmissão em tempo real.

O loop fica em constante ativação, verificando se algum dado foi recebido serialmente. Enquanto nenhum dado chegar o *loop* fica inativo, porém quando uma palavra é recebida significa que o *encoder* detectou um deslocamento angular e que a próxima coluna da imagem deve ser enviada para os LEDs. O loop automaticamente seleciona a coluna da imagem a ser enviada, adiciona o cabeçalho, pads e *bit* de paridade e a envia serialmente.

Figura 3.33: Painel frontal do software de controle.

3.11 Montagem

O modelo 3D do projeto da montagem pode ser visto na Figura 3.34. Para possibilitar a conexão de elementos externos foram adicionados alguns componentes para realizar a interface com a fonte de alimentação, o computador e o usuário. Esses componentes são: uma ventoinha, uma chave liga/desliga e dois conectores elétricos, uma para as fontes de 5 e 12 V e outro para a comunicação RS-232. O projeto com os conectores montados pode ser visto na Figura 3.35.

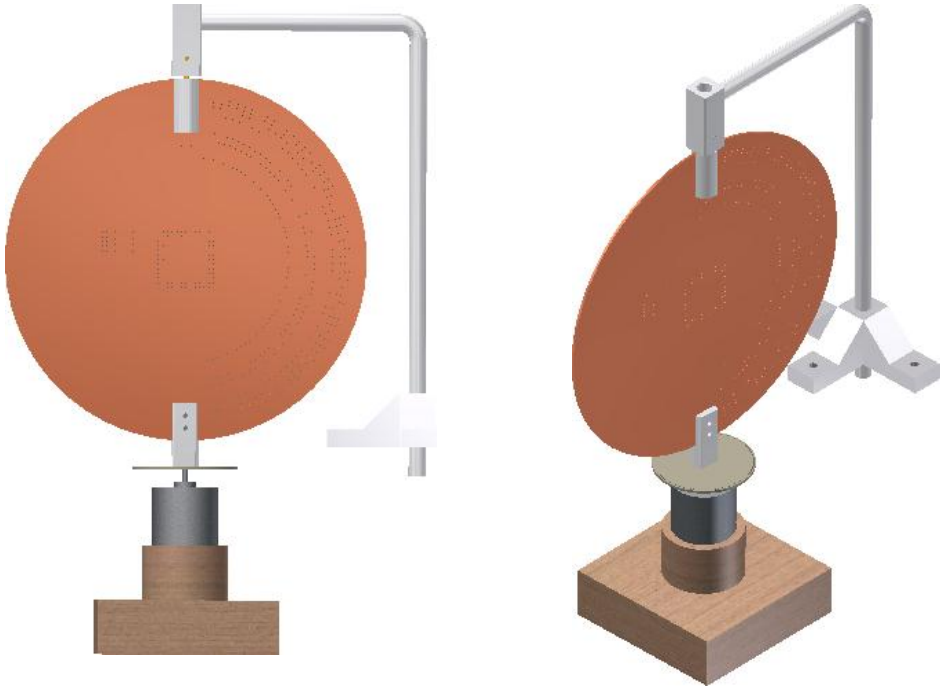


Figura 3.34: Sistema montado: vista 3D.

Figura 3.35: Conectores na parte traseira da caixa de suporte.

Nos primeiros testes o circuito controlador do motor CC apresentou bom funcionamento, porém após um longo período de funcionamento o transistor de potência superaquecia e queimava. A solução para esse problema foi conectar o transistor mecanicamente a um dissipador de calor utilizando pasta térmica para aumentar a transferência de calor. Após a montagem final do sistema o dissipador ficou posicionado na frente de uma ventoinha para aumentar a taxa de troca de calor.

Da Figura 3.36 à Figura 3.39 são mostradas fotos do sistema montado.

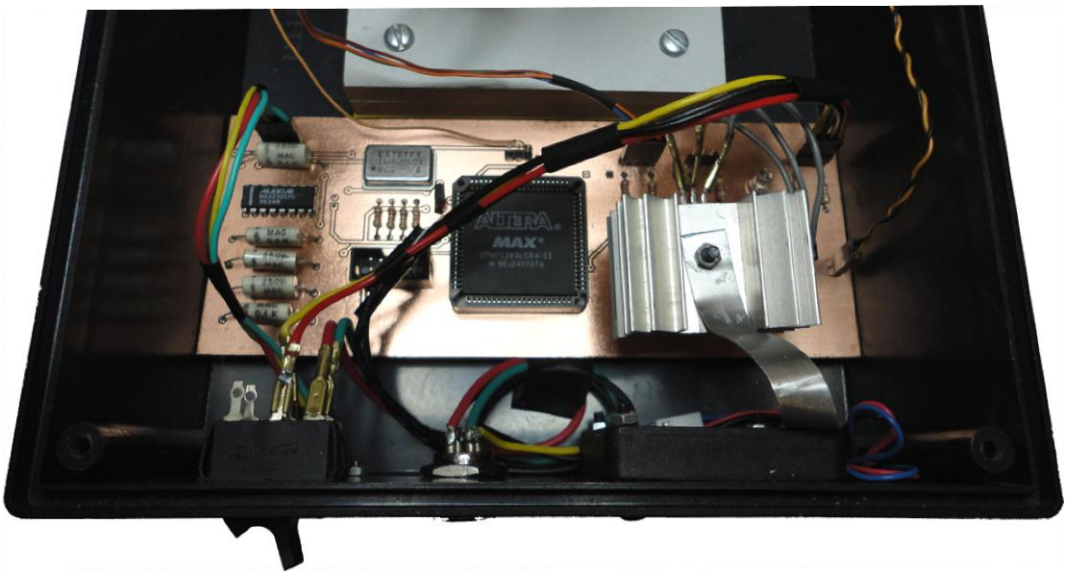


Figura 3.36: Placa controle ligada aos conectores externos.

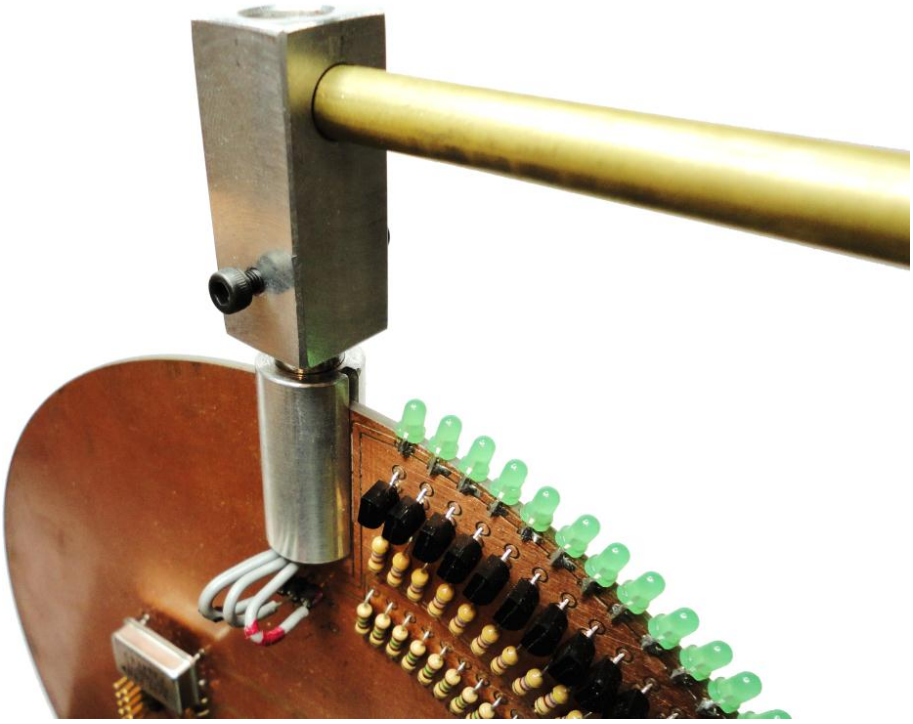


Figura 3.37: Montagem Superior: Placa rotativa, *Slip Ring* e tubo de suporte.

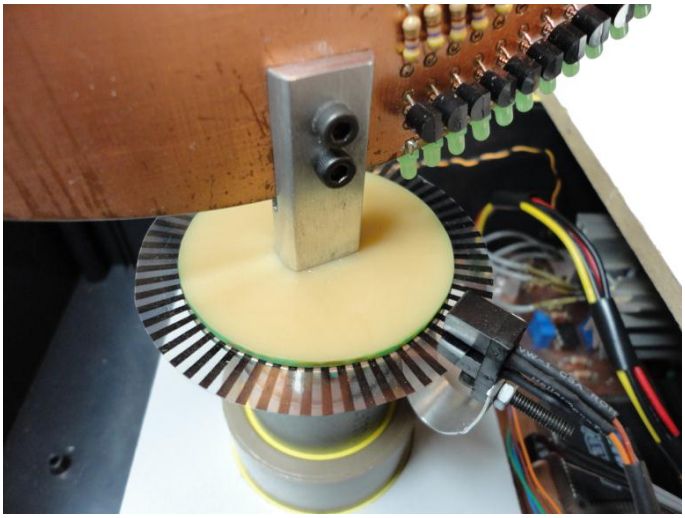


Figura 3.38: Montagem inferior: Motor, PCI, *encoder* e sensor IV.

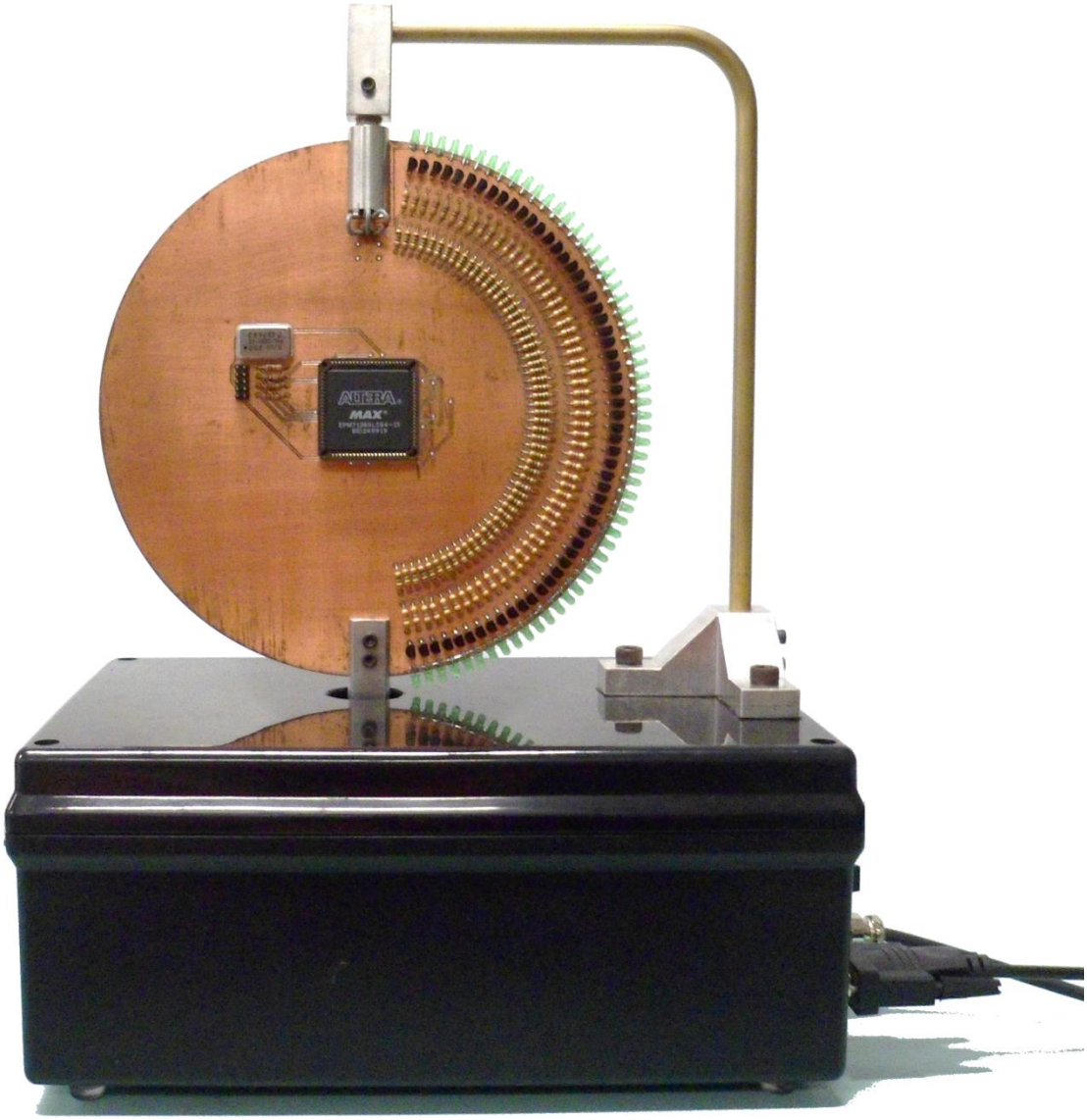


Figura 3.39: Montagem completa.

4 Resultados e Discussões

Este capítulo mostra todos os resultados obtidos durante as simulações e testes com o sistema, sejam esses resultados positivos ou negativos. Também apresenta uma breve discussão comparativa dos resultados obtidos com os esperados.

4.1 Simulação dos códigos em VHDL

Os códigos em VHDL que geraram a síntese do circuito de controle e circuito da placa rotativa nas CPLDs foram simulados utilizando a ferramenta nativa de simulação presente no Quartus II 9.1 Web Edition. Os sinais de entrada foram criados manualmente em um arquivo estilo Vector Waveform.

Ambas as CPLDs apresentam mais códigos dos que os simulados nesse capitulo, porém os simulados são os que apresentaram maior complexidade e necessidade da simulação. Os códigos mais simples foram testados nas próprias CPLDs e não apresentaram problemas.

4.1.1 Bloco de decodificação serial da placa controle

O código em VHDL que gerou a síntese do circuito de controle é apresentado no item 3.9.1, e tem como tarefa principal a decodificação dos dados seriais de entrada. A simulação desse código foi dividida em três etapas de verificação: verificar se o código interpreta os dados reais corretamente e sua imunidade, tanto a ruído quanto as mensagens seriais com destino a outra placa.

Inicialmente foi criado um sinal de *clock* com a mesma frequência que pode ser utilizada na montagem real. Na linha de entrada (dados_in) foi inserida a palavra serial indicativa de velocidade máxima do motor, ou seja, o número 255 com *bit* de paridade, *padding*, *start* e *stop bit*. A imagem “a” presente na Figura 4.1 mostra tanto esses sinais quanto a saída paralela resultante (dados_out). É possível observar que cerca de cinco ciclos de *clock* após o final da mensagem serial o processamento é finalizado e o sinal de saída tem seu valor alterado de 0 para o 255, como esperado.

A verificação da imunidade ao ruído foi realizada inserindo um sinal aleatório na linha de entrada durante um intervalo de 10 ms. O sinal aleatório foi gerado por uma função própria do software de simulação e apresenta frequência duas vezes maior que a do *clock* de sincronismo do código. A imagem “b” da Figura 4.1 indica este teste e seu resultado: em nenhum momento o sinal de saída foi modificado devido a uma má interpretação dos dados ruidosos de entrada. Assim como no inicio do teste o sinal de saída é finalizado com valor 0, mostrando a robustez do código a ruído.

Finalmente a imagem “c” presente na Figura 4.1 mostra na linha de entrada uma palavra serial com destino ao controle dos LEDs da placa rotativa. A palavra é composta por todos os dados de luminosidade, *padding*, *start* e *stop bit*, formando uma palavra completa. Do mesmo modo que o teste anterior em nenhum momento os dados de saída são alterados.

Todos os testes foram refeitos com outros valores de entrada, e os resultados foram sempre positivos indicando um correto funcionamento deste bloco.

4.1.2 Bloco de decodificação serial da placa rotativa

O código em VHDL que gerou a síntese do circuito de controle da placa rotativa e apresentado no item 3.9.2 também tem como tarefa principal a decodificação dos dados seriais de entrada, porém voltado ao controle dos LEDs. A simulação desse código foi dividida nas mesmas três etapas mostradas no item anterior.

A simulação da palavra serial de controle dos LEDs foi realizada inserindo esta na linha de entrada de dados do código. A palavra inserida tem a função de acender todos os LEDs, logo a saída paralela que vai diretamente ao *drive* de potência deve ficar completamente em nível 1.

A imagem “a” da Figura 4.2 apresenta o resultado dessa simulação e pode-se observar que logo após o ultimo *bit* de dados a saída para de ser alterada e passa a apresentar somente níveis lógicos altos, indicando o correto funcionamento da conversão serial paralelo.

O teste de ruído foi realizado da mesma maneira e tempo que o relativo do item anterior. O resultado, que pode ser visto na imagem “b” da Figura 4.2 e comprova a imunidade do código a ruído, não apresentando nenhuma alteração da saída durante toda a duração do teste.

Finalmente a imagem “c” da Figura 4.2 mostra a simulação da interpretação da palavra serial destinada à placa controle. A palavra transmitida é a mesma utilizada no primeiro teste do item anterior, porém nesse teste ela deve ser completamente ignorada pelo código da placa rotativa, mantendo a saída com um valor imutável durante toda a simulação. A imagem “c” comprova o correto funcionamento do código mostrando que os dados de entrada são iguais aos dados de saída do canal de saída do bloco.

Assim como os testes do item anterior, esses foram refeitos com outros valores de entrada retornando sempre resultados positivos.

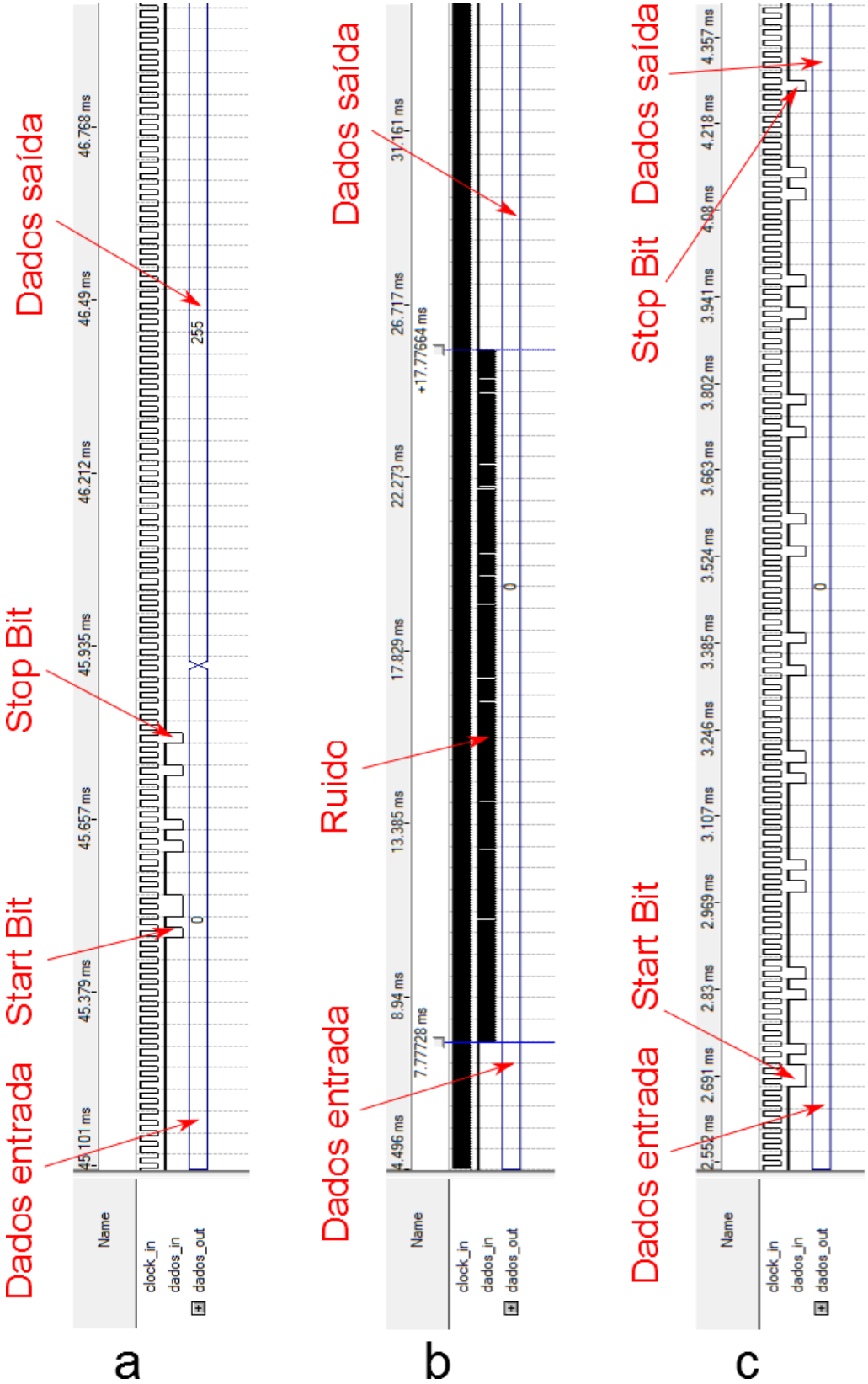


Figura 4.1: Simulação do código interpretador de mensagens serial da placa controle.

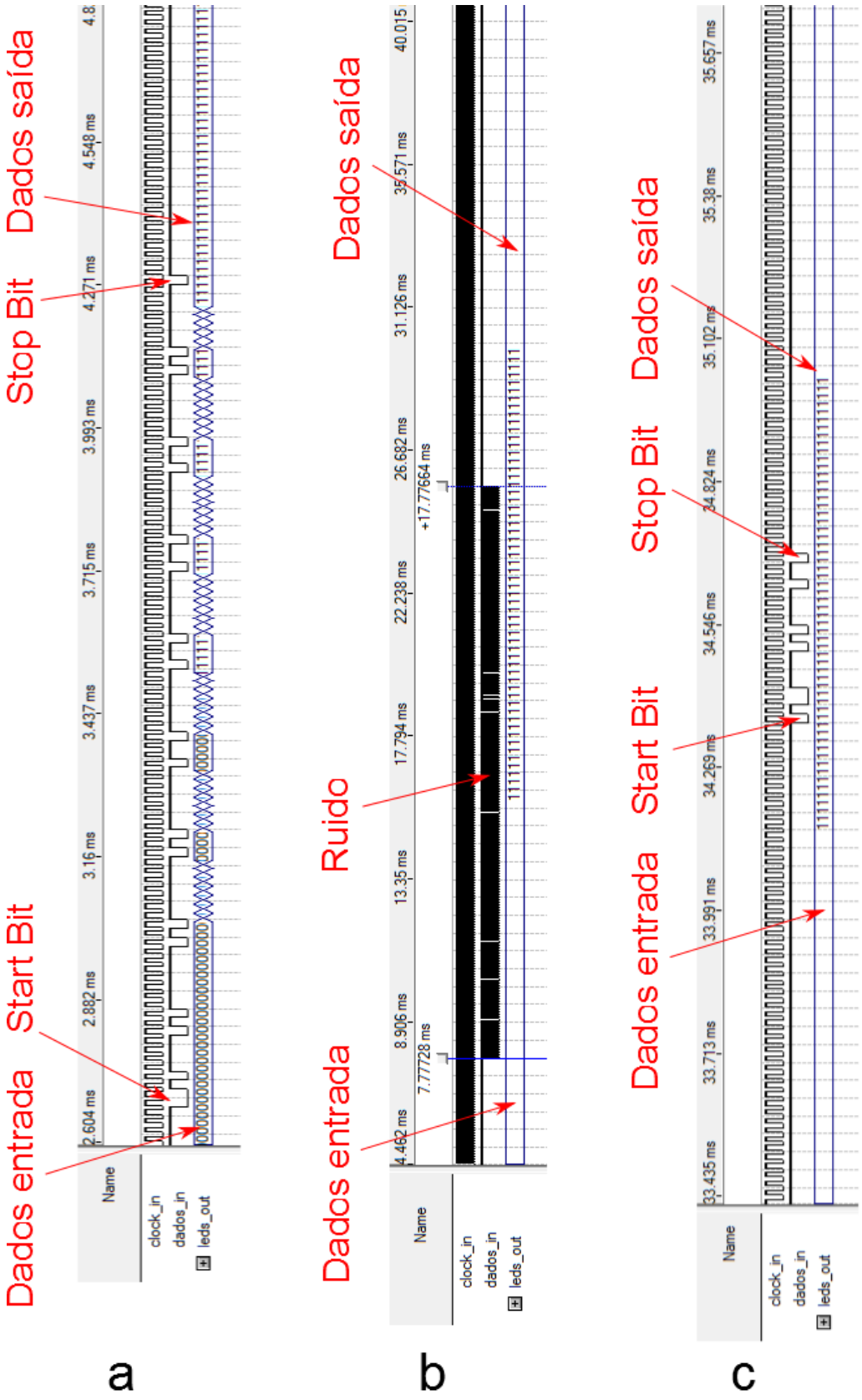


Figura 4.2: Simulação do código interpretador de mensagens serial da placa rotativa.

4.2 Testes

O sistema montado, em geral, apresentou um funcionamento estável, apresentando problemas de ruído e sincronismo causados principalmente do componente *Slip Ring*.

Após a montagem e simulações o sistema foi testado como um todo. As peças de sustentação mecânica projetadas funcionaram sem falhas, mantendo a placa rotativa estável e sem trepidações durante o funcionamento. O software de controle funcionou exatamente como esperado: não apresentando problemas ou dificuldades com a escolha da velocidade de rotação, processamento interno das imagens escolhidas e envio dos dados seriais. O software em funcionamento pode ser visto na Figura 4.3.

Figura 4.3: Software em funcionamento.

A Figura 4.4 apresenta o sistema em funcionamento mostrando uma imagem inteira de uma cor, ou seja, todos os LEDs acessos. Pode-se observar o reflexo da esfera na superfície da caixa de sustentação e, observando a leve variação de intensidade luminosa presente nas linhas horizontais podem ser observados os fusos esféricos de 5,625° em que cada LED mantém sua intensidade luminosa.

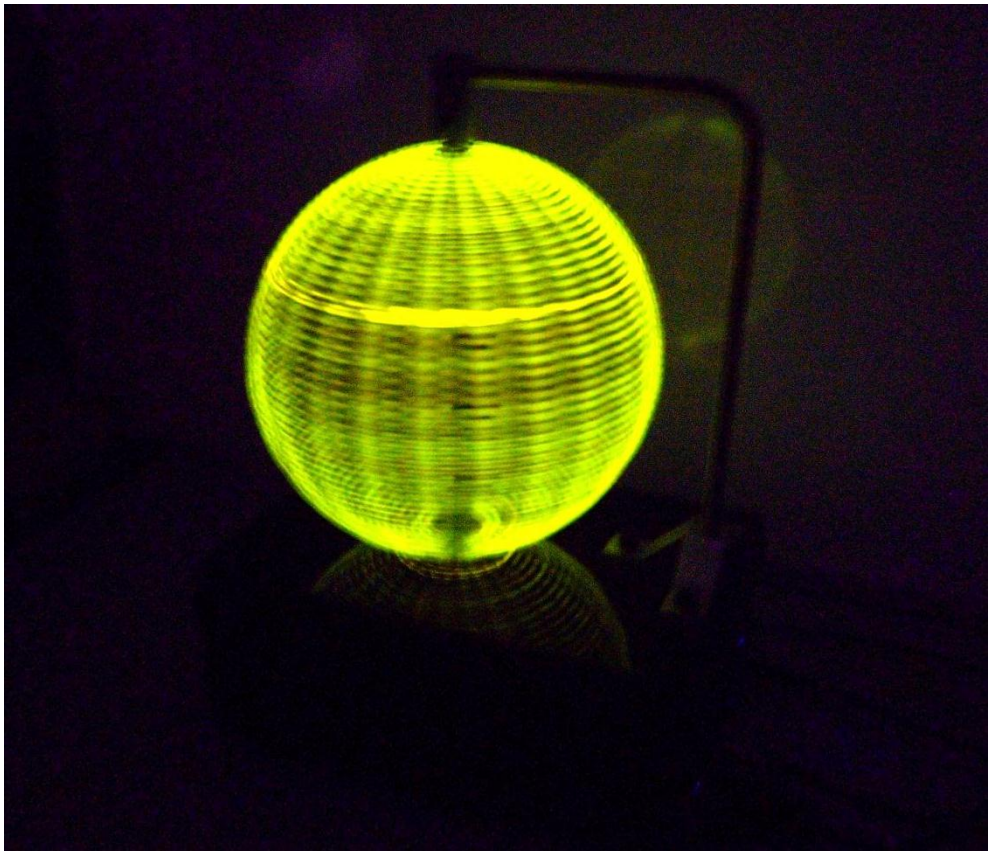


Figura 4.4: Sistema em funcionamento com todos os LEDs acessos.

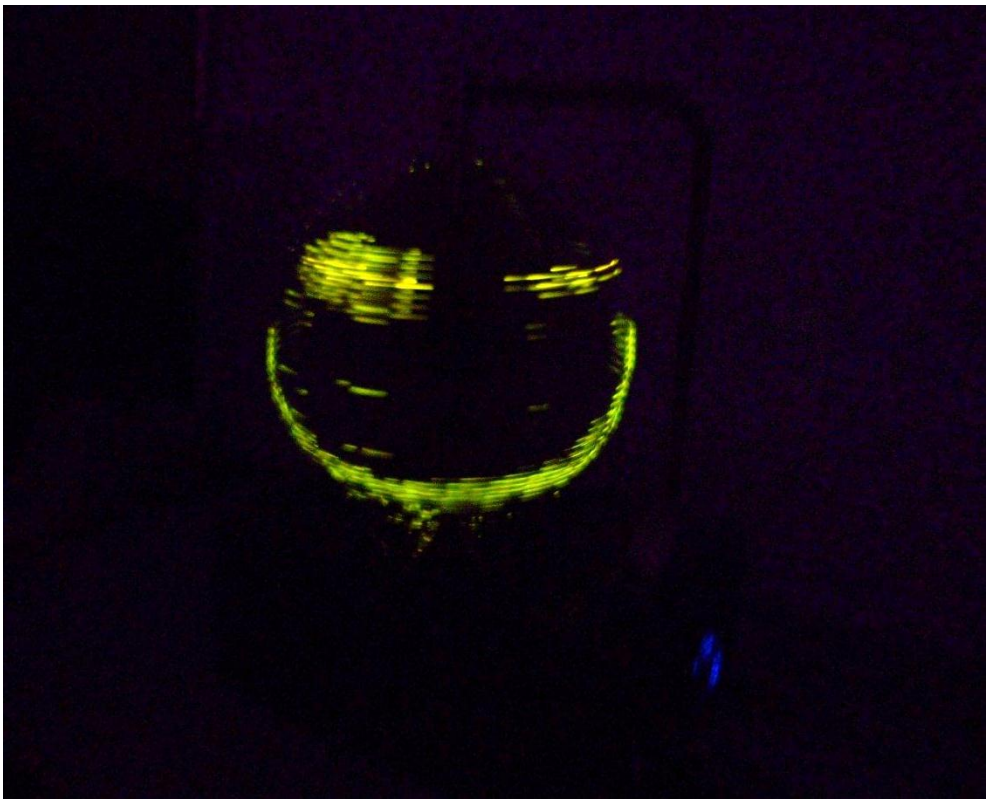


Figura 4.5: Sistema em funcionamento com uma imagem.

Um problema encontrado foi o funcionamento da junta rotativa de contato elétrico apresentada na Figura 3.4. Apesar de lubrificada, seu atrito mecânico impedia o motor de rodar suavemente e com sua velocidade máxima. A conexão elétrica durante a rotação em alta velocidade introduziu muito ruído na linha de dados alterando os dados de controle dos LEDs em alguns momentos. Vários conectores diferentes foram testados até ser encontrado um com maior estabilidade, menos atrito e melhor conexão. Na Figura 4.5 é possível observar uma imagem sendo exibida (a mesma imagem utilizada no software da Figura 4.3). Também é possível observar alguns LEDs acessos erroneamente devido ao ruído da má conexão da junta rotativa.

Outro problema encontrado foi a não estabilidade da imagem. Apesar de o sistema enviar os dados sincronamente com o *encoder* como pode ser visto no Figura 4.6, devido ao ruído do *Slip Ring* a imagem não permaneceu estática em um ponto, mas apresentando um lento deslocamento, girando em torno de seu próprio eixo.

A Figura 4.6 mostra dois sinais capturados por um osciloscópio durante o funcionamento do sistema. O canal 1 mostra a onda quadrada de saída do *encoder*, sinal utilizado como *trigger* de envio das mensagens seriais para o computador. O canal 2 mostra a resposta do computador, ou seja, a linha serial contendo as palavras com os dados de luminosidade dos LEDs.

Com o osciloscópio pode ser analisado que o tempo decorrido entre a subida de borda do sinal do *encoder* e *o start bit* da palavra serial com os dados dos LEDs é de 280 µs. Esse tempo é pequeno se comparado com o período total do sinal do encoder, 3.410 ms, não causando atrasos entre o ângulo do sistema e o dado requerido.

Apesar de o sistema possibilitar velocidade de rotação de até 20 RPS, que seria o ideal para o projeto como relatado no item 3.8 e, o motor possibilitar essa rotação, o atrito do *Slip Ring* somado ao peso da placa rotativa impossibilitou que o motor apresentasse uma rotação maior que 4,58 RPS. Esse valor pode ser medido analisando o período do sinal presente na Figura 4.6:

Período de 1 pulso do *encoder*: 3,410 ms.

Período de 64 pulsos do encoder: 218,24 ms.

Rotações por segundo: $(218{,}24\ ms\ )^{-1}$ = 4,58 RPS

Essa velocidade de rotação é inferior ao desejado pelo sistema e inferior ao mínimo de 12,5 RPS necessário para causar a impressão de imagem continua ao olho humano, porém suficiente para possibilitar a visualização das imagens da uma maneira não continua, sendo a imagem desenhada e apagada a cada volta.

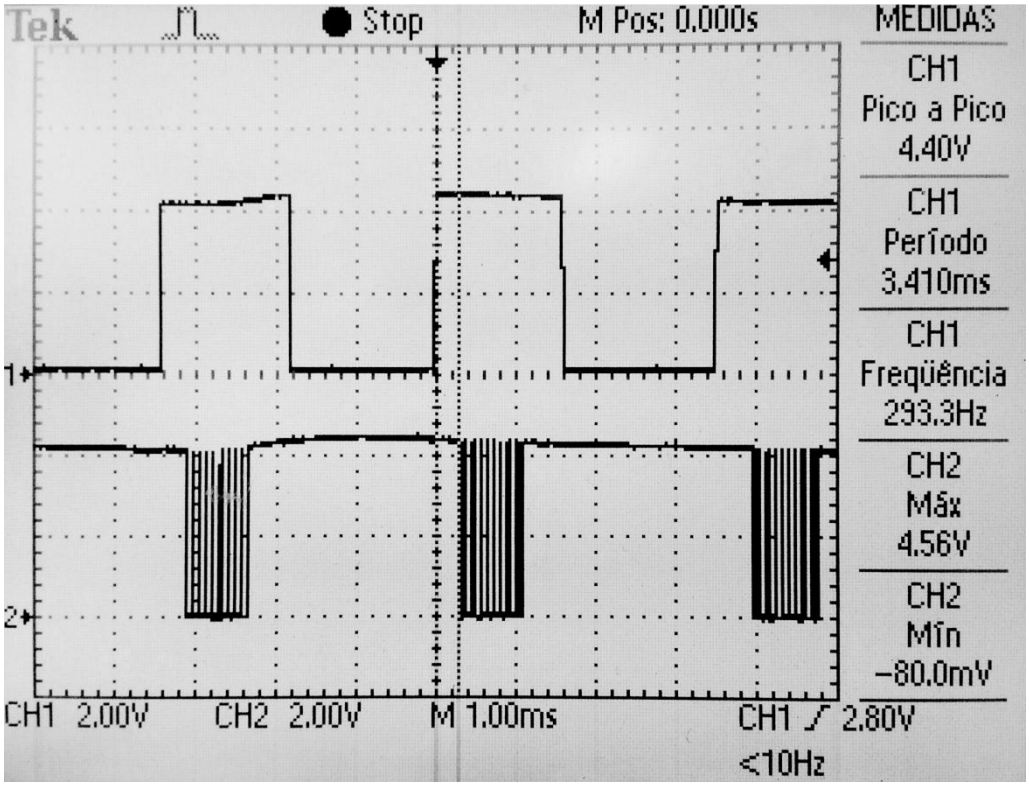


Figura 4.6: Medida da velocidade final do sistema e atraso entre pedido e recepção dos dados.

Comparando este trabalho com outros projetos encontrados na literatura e citados no capítulo 1, este projeto apresenta a vantagem de controle das imagens em tempo real, diferentemente dos outros que apresentam os dados gravados em memórias. Comparando o sistema mostrado na Figura 1.5 com o aqui proposto, Figura 4.5, a imagem obtida no primeiro apresenta qualidade melhor e mais nítida, não possuindo ruídos nem erros no chaveamento dos LEDs, entretanto não apresentam a alteração da imagem em tempo real. O sistema desenvolvido neste projeto dispensa o uso de memórias e possibilita que qualquer imagem possa ser exibida, possibilitando inclusive que o usuário edite dinamicamente os pixels da imagem para realizar alterações. Considerando todos os resultados obtidos pode-se deduzir que o sistema funcionaria sem erros apenas trocando a junta rotativa por outra de melhor qualidade.

5 Conclusões

Pode-se concluir que o trabalho atendeu o objetivo definido inicialmente de projetar e construir um *display* esférico capaz de reproduzir uma imagem esférica luminosa com controle de imagem em tempo real. A reprodução da imagem apresentou resultados inferiores aos projetos comparativos da Figura 1.5 devido aos problemas encontrados com o *Slip Ring* de baixa qualidade, não possibilitando a exibição dos resultados com a precisão e velocidade esperadas. Outros *Slip Rings* não foram utilizados devido ao alto custo do componente, dificuldade de encontrar fabricantes com um curto prazo de entrega e até mesmo a necessidade de comprovante de usuário final. Em contrapartida a utilização da técnica de controle da imagem em tempo real foi bem sucedida, apresentando uma interface simples e intuitiva ao usuário. Ainda, todas as partes mecânicas e elétricas desenvolvidas funcionaram de maneira satisfatória, desempenhado seus papeis sem falhas.

Com apenas algumas modificações este projeto pode ser utilizado nas aulas de graduação do curso de engenharia elétrica voltadas ao ensino da linguagem VHDL. Essas modificações seriam a automação do circuito de controle dos LEDs ou do circuito de controle do motor, deixando esses módulos invisíveis para o aluno, voltando o aprendizado ao outro modo.

5.1 Sugestões para trabalhos futuros

Para trabalhos futuros é principalmente sugerida a utilização de componentes de melhor qualidade como:

Uma junta rotativa de contato elétrico profissional, de alta qualidade e fabricada para o número de linhas e velocidade desejadas. Outra solução seria a utilização de comunicação sem fio, porém deve ser avaliado se a rotação do sistema não prejudicaria a qualidade e temporização dos dados transmitidos.

As CPLDs utilizadas podem ser substituídas por modelos mais modernos com maior capacidade lógica e mais pinos de entrada e saída como, por exemplo, a CPLD EPM3512AFI256-10 da série MAX3000A com 512 macrocélulas e 256 pinos. Um maior número de pinos possibilitaria a utilização de mais LEDs, melhorando a resolução da imagem ou possibilitando a utilização de LEDs RGB deixando a imagem colorida, ou mesmo numa melhora do circuito sintetizado no CPLD podendo ser acrescido um *buffer* de espera para o caso de um dado vir antes do esperado ou uma lógica de verificação de dados, tornando o sistema mais robusto a erros e ruído.

Para a utilização de CPLDs ou FPGAs de grande quantidade de pinos seria necessária a modernização da técnica de manufatura e solda das placas de circuito impresso do laboratório de eletrônica digital. Placas com alta densidade de componentes com montagem em superfície necessitam de uma precisão maior nas trilhas e conectores do que encontrado nas placas feitas por uma fresadora automática ou utilizando corrosão por percloreto. Uma solução seria a manufatura das placas através

da utilização do processo fotográfico, utilizando luz para corroer o cobre da placa, melhorando a precisão das trilhas e conectores.

Para aumentar a taxa de transmissão de dados poderia ser utilizado um hardware especifico para o envio dos dados removendo a dependência da porta serial do computador e do sistema operacional. Por exemplo, poderia ser utilizada uma placa comercial de aquisição e envio de dados de alta velocidade, como uma placa PCI-6541 da National Instruments, ou um hardware próprio com um microprocessador para armazenar e enviar os dados no momento correto.

Referências Bibliográficas

1. THE TIMES. The "Televisor" successful test of new apparatus. **The Times**, Londres, 28 Janeiro 1926.

2. BBC. http://www.bbc.co.uk/. **BBC**, 2012. Disponivel em: <http://www.bbc.co.uk/history/historic_figures/baird_logie.shtml>.

3. JOHN Logie Baird. **Wikipedia:** the free encyclopedia, 2012. Disponivel em: <http://en.wikipedia.org/wiki/John_Logie_Baird>.

4. 1936 Summer Olympics. **Wikipedia:** the free encyclopedia. Disponivel em: <http://en.wikipedia.org/wiki/1936_Summer_Olympics>.

5. CIA World Factbook. **Nation Master**, 2003. Disponivel em: <http://www.nationmaster.com/graph/med_tel-media-televisions>.

6. CUSTOMWHEEL LLC. Custom Wheel, 2012. Disponivel em: <http://customwheel.com>.

7. BLICK, B. Propeller Clock, 1996. Disponivel em: <http://www.bobblick.com>.

8. PATERSON, I. Ian's Spoke POV. **Ian Paterson`s site**, 2006. Disponivel em: <http://www.ianpaterson.org>.

9. PAREKH, A. Rotating LED Globe. **Hacked Gadgets**, 2010. Disponivel em: <http://hackedgadgets.com>.

10. SOLDERLAB. RGB LED Globe - Rotating Sphere. **SolderLab**, 2012. Disponivel em: <http://www.solderlab.de>.

11. ANDERSON, J.; FISHER, B. The Myth of Persistence of Vision. **Journal of the University Film Association**, v. 30, n. 4, p. 3-8, 1978.

12. ANDERSON, J.; ANDERSON, B. The Myth of Perssitense of Vision Revisited. **Journal of Film and Video**, v. 40, n. 1, p. 3-12, 1993.

13. STEINMAN, R.; PIZLO, Z.; PIZLO, F. Phi is not beta, and why Wertheimer's discovery launched the Gestalt revolution. **Vision Research**, v. 40, n. 17, p. 2257-2264, Agosto 2000.

14. AWADHI, A. eFarabi Educations and Knowledge. **What is Serial communication?** Disponivel em: <http://www.efarabi.com>.

15. TEXAS INSTRUMENTS. **Interface Circuits for TIA/EIA-232-F**. [S.l.]: [s.n.], 2002.

16. AGUIAR, M. L. D. Introdução ao LabVIEW. In: _____ **Notas de aulas - Sel0359 Controle Digital**. [S.l.]: [s.n.], 2012.

17. HOGG, S. What is LabVIEW? **NI Developer Zone**, 2011. Disponivel em: <http://www.ni.com>.

18. NERY, R. T. D. C. **Introdução ao LabVIEW - Grupo de vibrações e acústica**. Universidade

Federal do Paraná. [S.l.], p. 65.

19. DENNIS, J. B. Data Flow Supercomputers. **Computer - IEEE Journals & Magazines**, MIT Laboratory for Computer Science, v. 13, n. 11, p. 48-56, Novembro 1980.

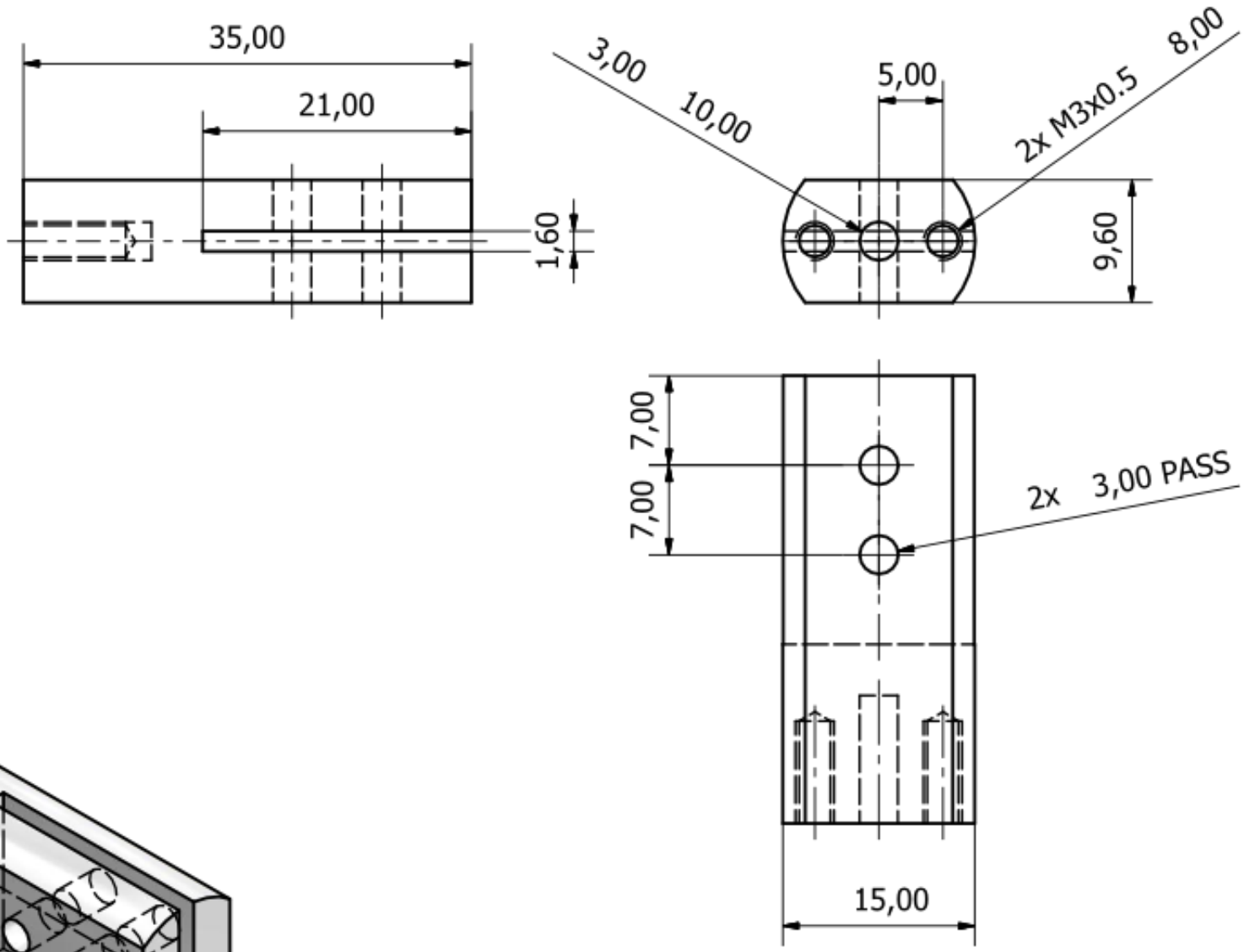
20. COLLINS ENGLISH DICTIONARY. **Collins English Dictionary - Complete and Unabridged**. [S.l.]: HarperCollins Publishers, 2003.

21. SENRING Electronics, 2011. Disponivel em: <http://www.senring.com/>.

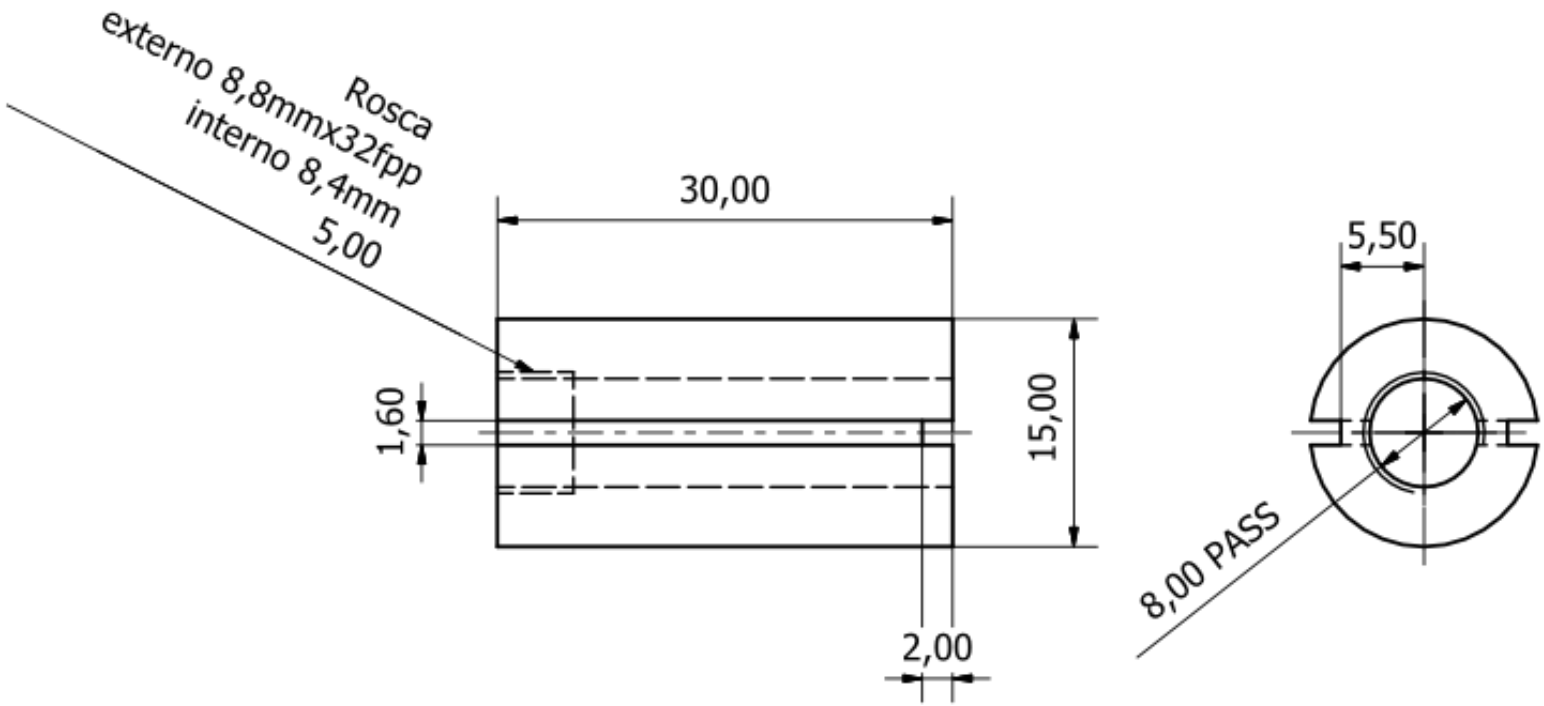
22. TRAHEY, S. Choosing a code wheel: A detailed look at how encoders work. **Small Form Factors**, Março 2008.

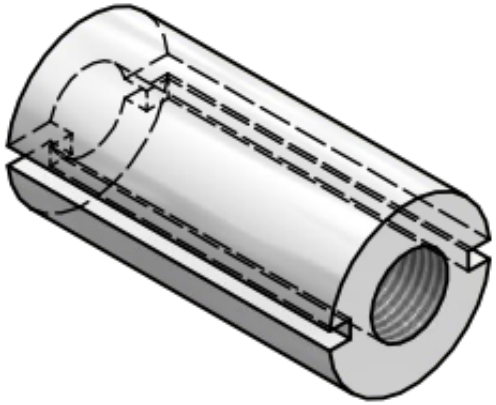
23. UNIVERSIDADE DE SÃO PAULO. **Diretrizes para Apresentação de Dissertações e Teses da USP - ABNT**. 2ª. ed. São Paulo: Sistema Integrado de Bibliotecas - SIBi/USP, 2009.

Apêndice A – Desenhos das peças mecânicas

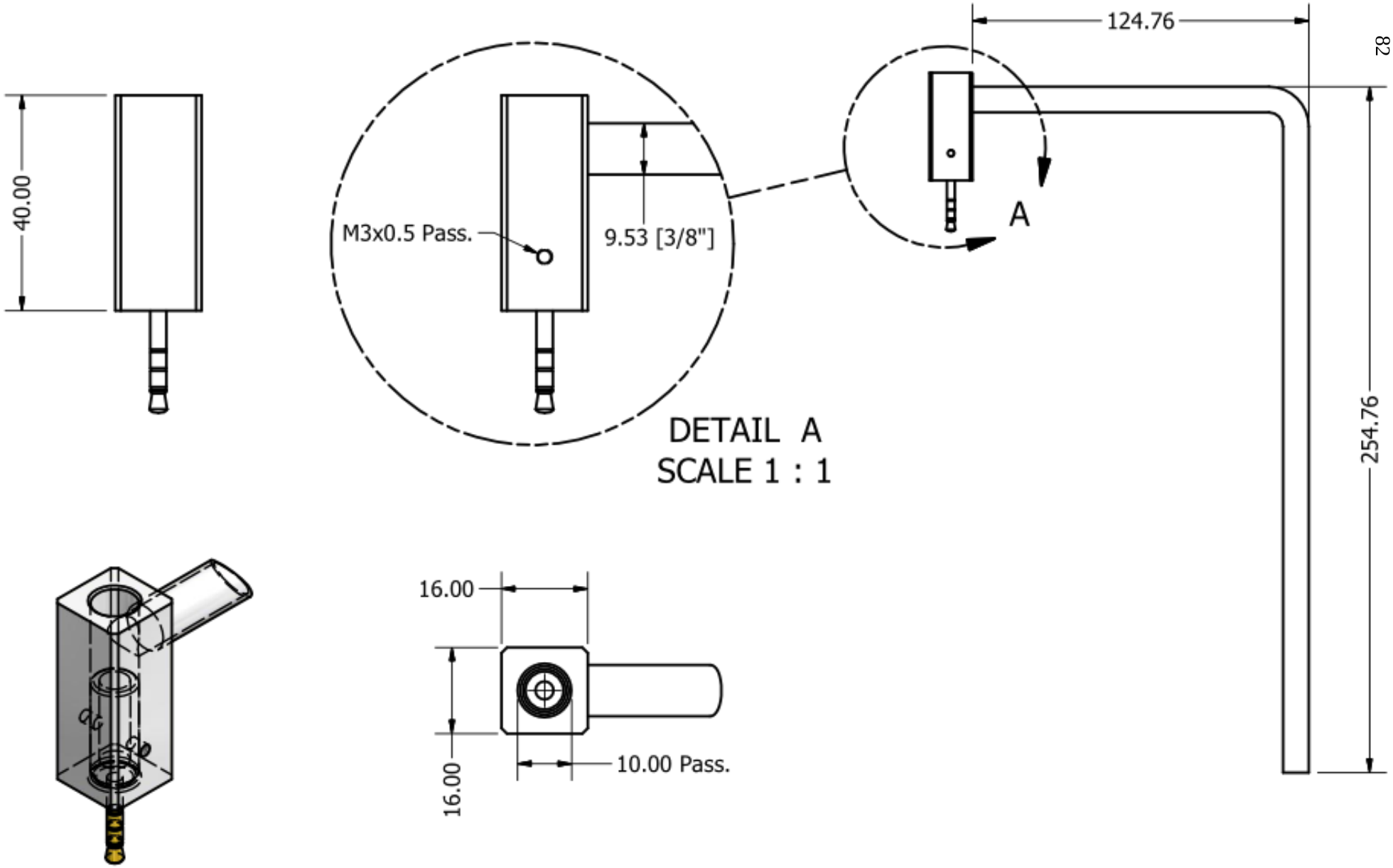


Designed by	Date	Scale	Unit
Igor Guerrero	18/04/12	2:1	mm
	Conector Inferior F mea	Edition 1	Sheet 1 / 1

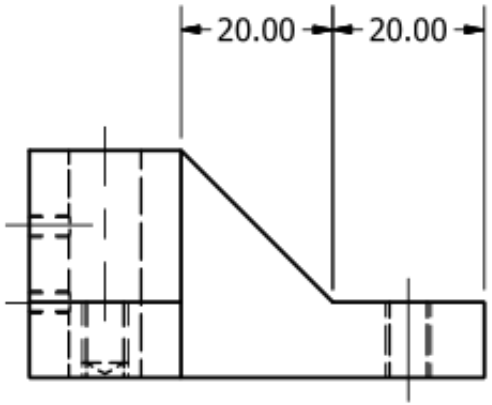


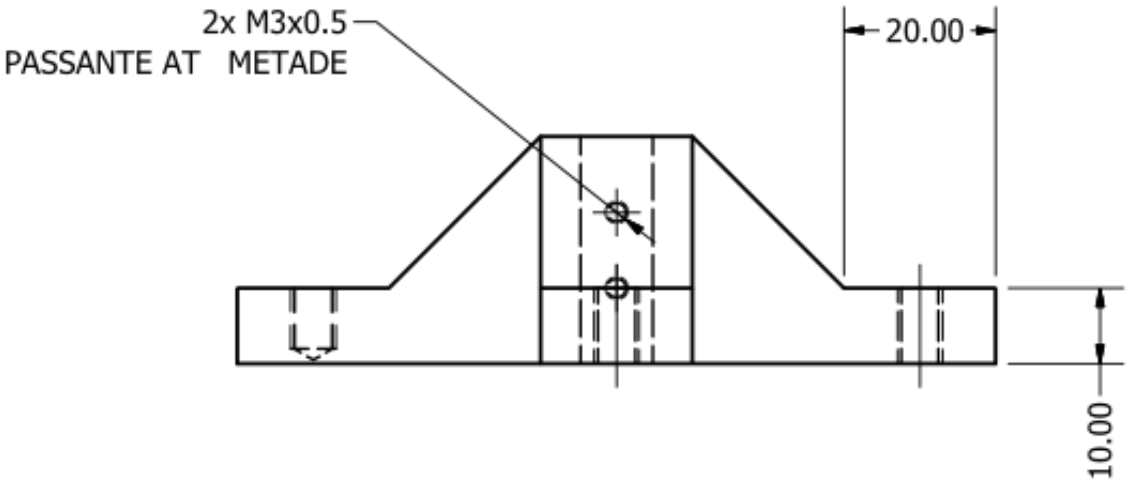


Designed by	Date	Scale	Unit
Igor Guerrero	18/04/12	2:1	mm
	Conector Superior F mea	Edition 1	Sheet 1 / 1

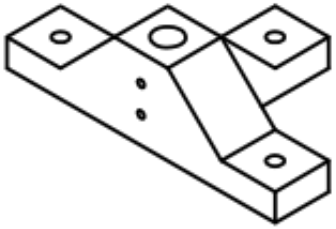


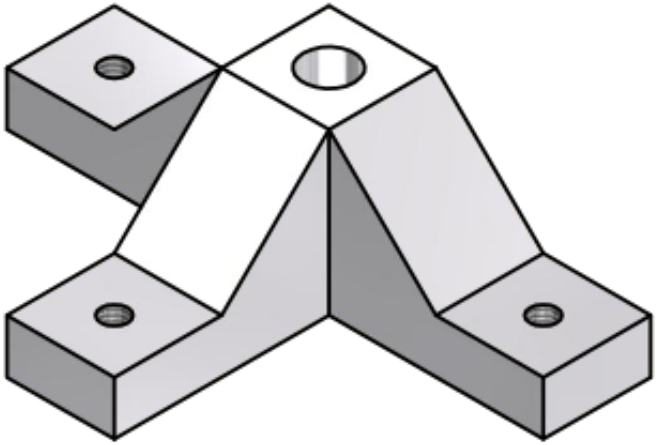
Designed by	Date	Scale	Unit
Igor Guerrero	21/05/12	1/2	mm
	Conector Superior Macho	Edition 1	Sheet 1 / 1

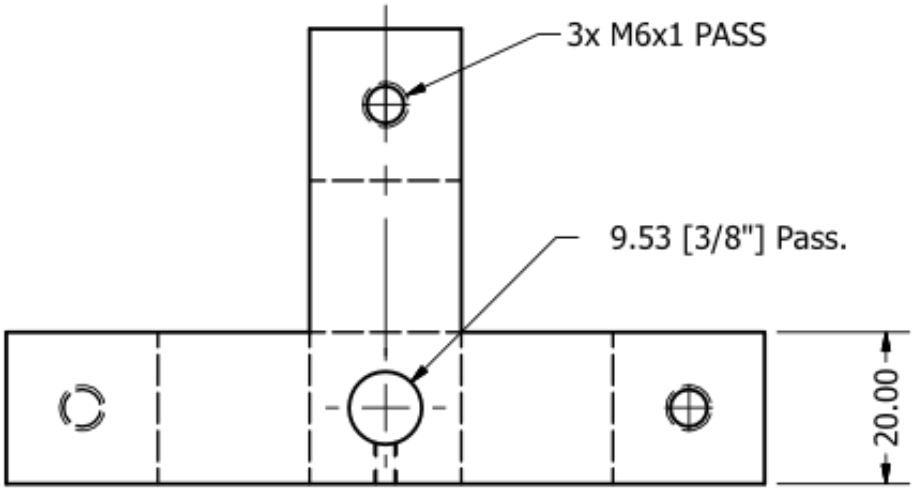




Escala 1:2

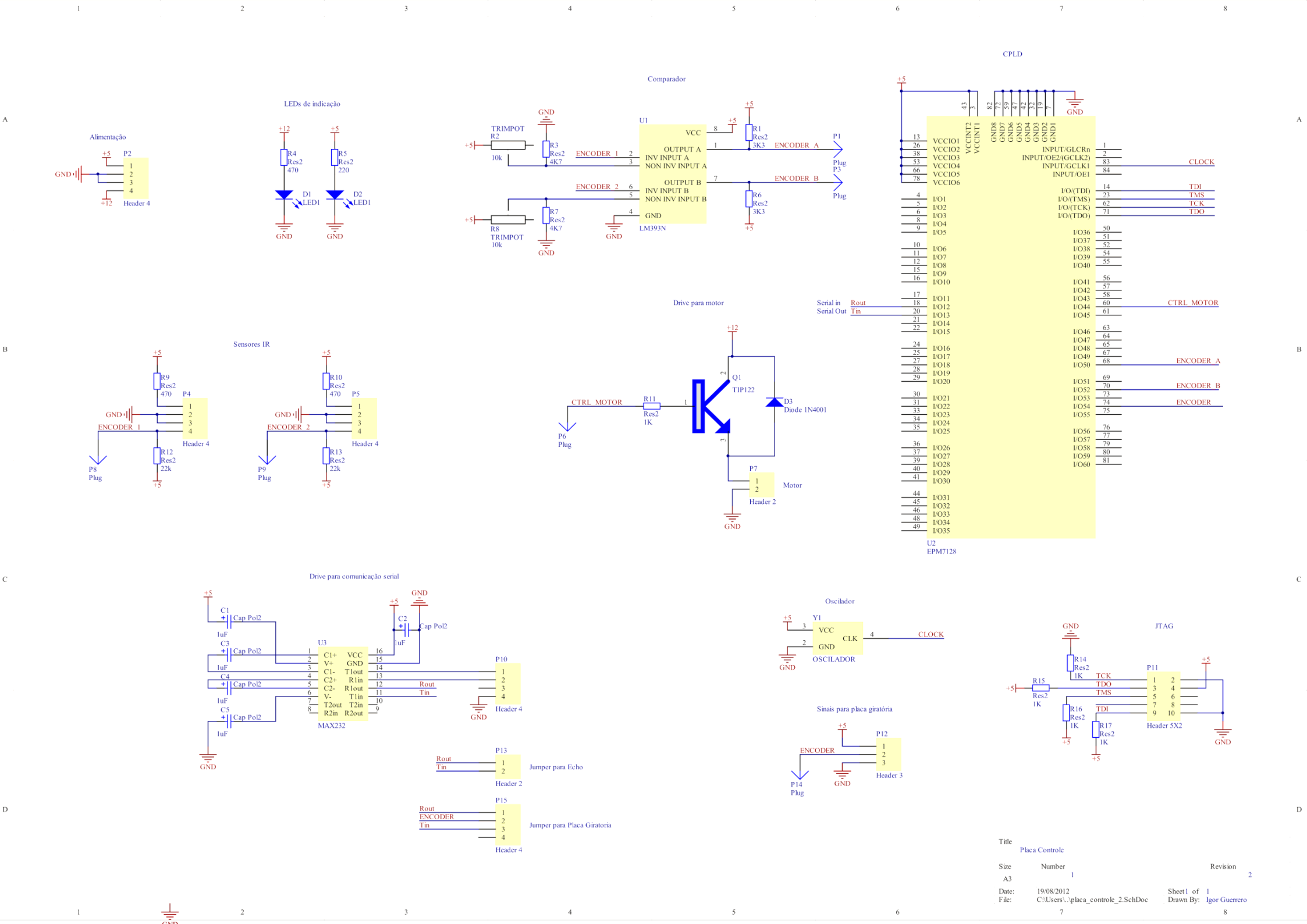


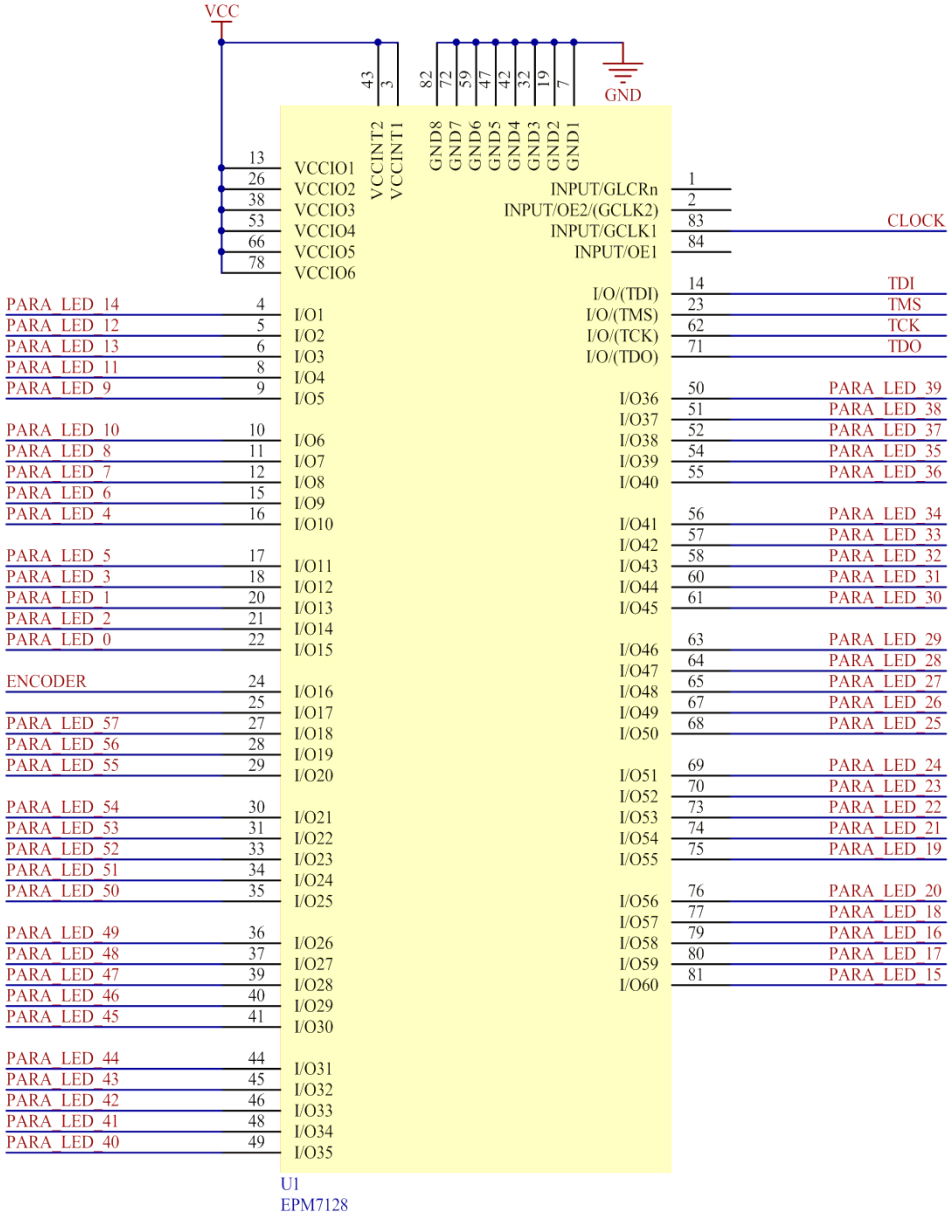


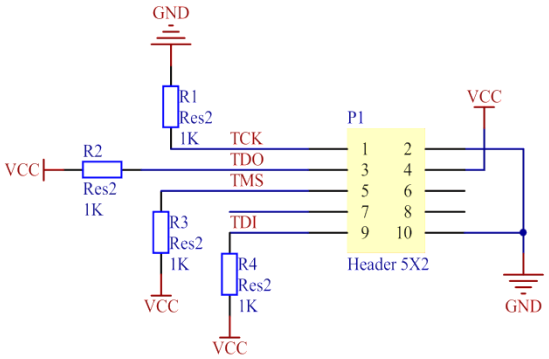


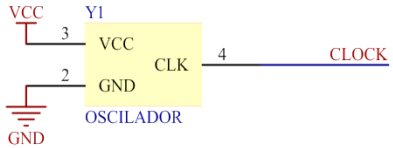
Designed by	Date	Scale	Unit
Igor Guerrero	23/04/12	1:1	mm
	Conector Tubo Caixa	Edition 1	Sheet 1 / 1

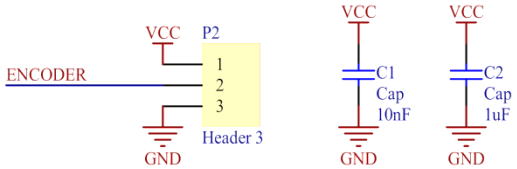
Apêndice B – Esquemas elétricos


Alimentação
P2
Header 4
LEDs de indicação
R4 Res2 470
R5 Res2 220
D1 LED1
D2 LED1
Comparador
TRIMPOT R2 10k
R3 Res2 4K7
R7 Res2 4K7
R8 TRIMPOT 10k
U1
LM393N
ENCODER_1
ENCODER_2
ENCODER_A
ENCODER_B
R1 Res2 3K3
R6 Res2 3K3
P1 Plug
P3 Plug
CPLD
U2 EPM7128
Serial in Rout
Serial Out Tin
CLOCK
TDI
TMS
TCK
TDO
CTRL_MOTOR
ENCODER_A
ENCODER_B
ENCODER
Sensores IR
R9 Res2 470
R10 Res2 470
P4 Header 4
P5 Header 4
R12 Res2 22k
R13 Res2 22k
P8 Plug
P9 Plug
Drive para motor
CTRL_MOTOR
R11 Res2 1K
Q1 TIP122
D3 Diode 1N4001
P6 Plug
P7 Header 2
Motor
Drive para comunicação serial
C1 Cap Pol2 1uF
C2 Cap Pol2 1uF
C3 Cap Pol2 1uF
C4 Cap Pol2 1uF
C5 Cap Pol2 1uF
U3 MAX232
C1+ VCC
V+ GND
C1- T1out
C2+ R1in
C2- R1out
V- T1in
T2out T2in
R2in R2out
P10 Header 4
Rout
Tin
P13 Header 2
Jumper para Echo
P15 Header 4
Jumper para Placa Giratoria
Oscilador
Y1 OSCILADOR
VCC
GND
CLK
CLOCK
Sinais para placa giratória
P12 Header 3
ENCODER
P14 Plug
JTAG
R14 Res2 1K
R15 Res2 1K
R16 Res2 1K
R17 Res2 1K
P11 Header 5X2
TCK
TDO
TMS
TDI
Title
Placa Controle
Size A3
Number 1
Revision 2
Date: 19/08/2012
File: C:\Users\..\placa_controle_2.SchDoc
Sheet 1 of 1
Drawn By: Igor Guerrero







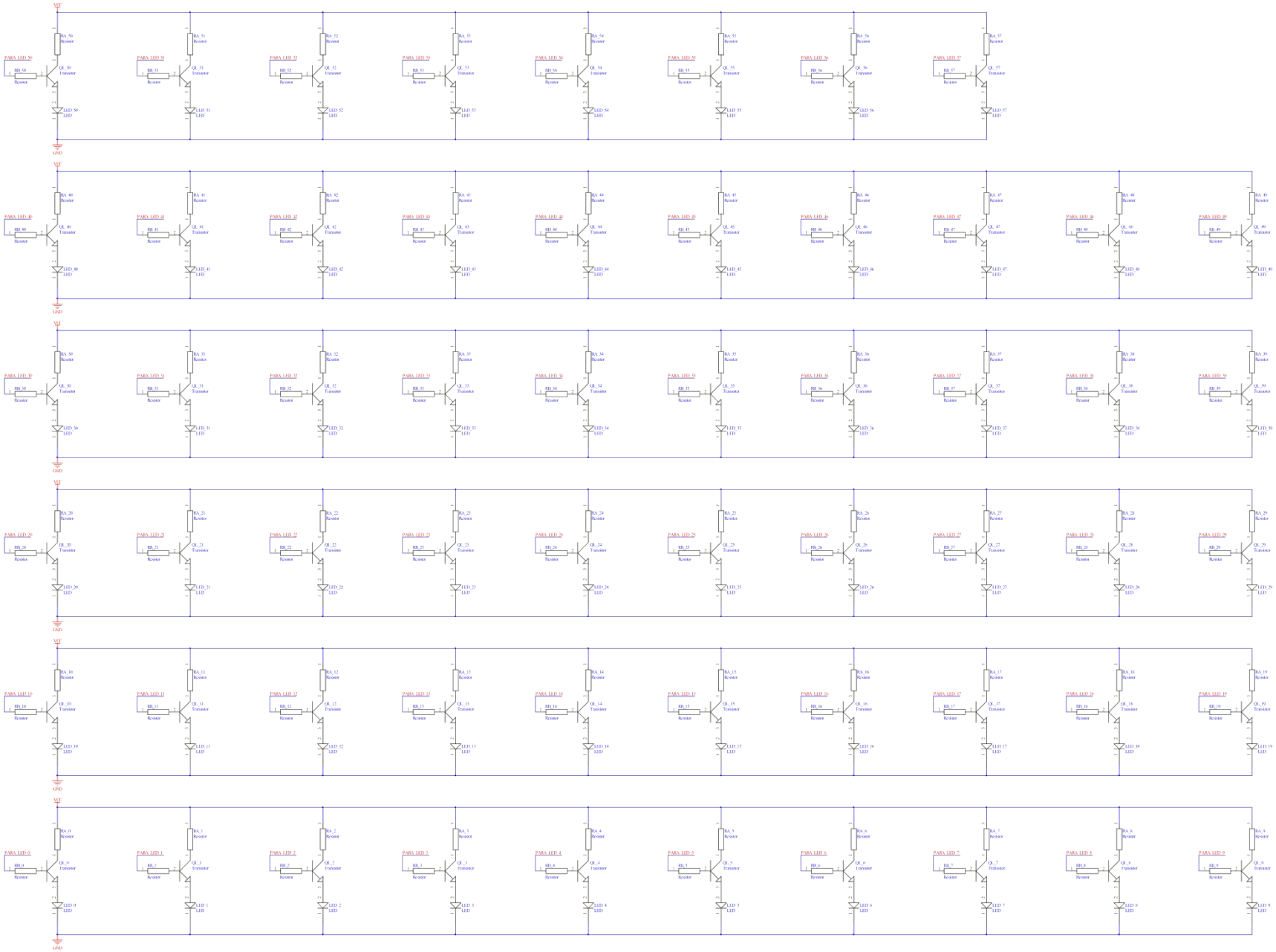


Title
Placa rotativa - CPLD

Size	Number	Revision
A4	1	2

Date: 19/08/2012
File: C:\Users\..\cpld.SchDoc

Sheet 1 of 2
Drawn By: Igor Guerrero



Title: Placa relativa - LEDs

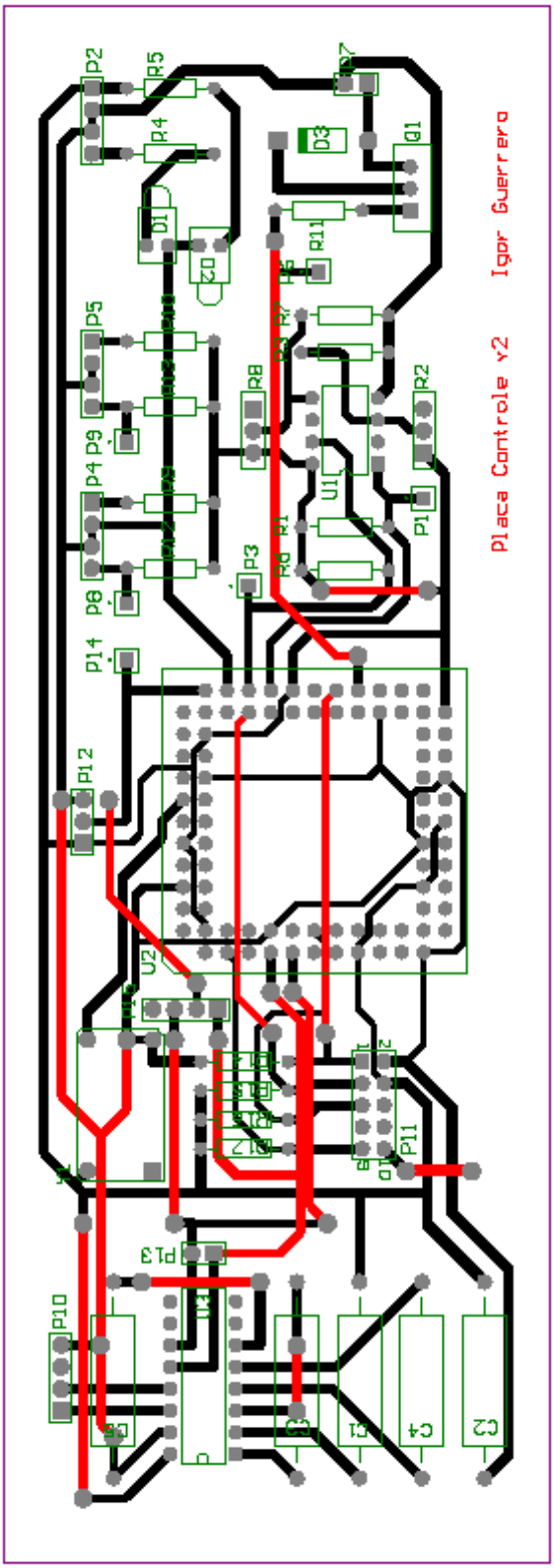
Size: A3 | Number: 1 | Revision: 2

Date: 19/08/2012 | Sheet 2 of 2

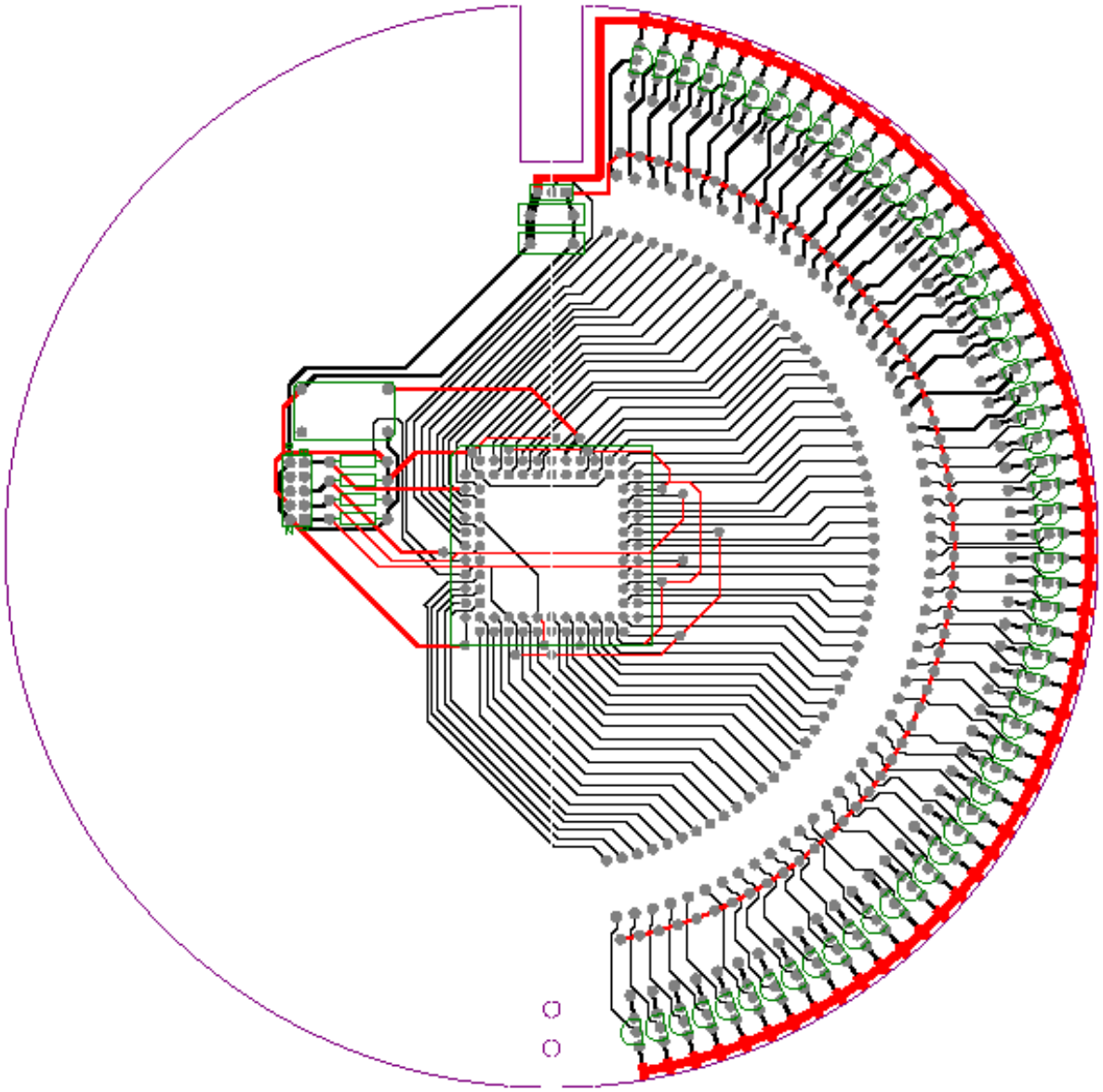
File: C:\Users\...\pic16cmpp_v2_leds.SchDoc | Drawn By: Igor Guerrero

Apêndice C – Placas de circuito impresso

Placa Controle

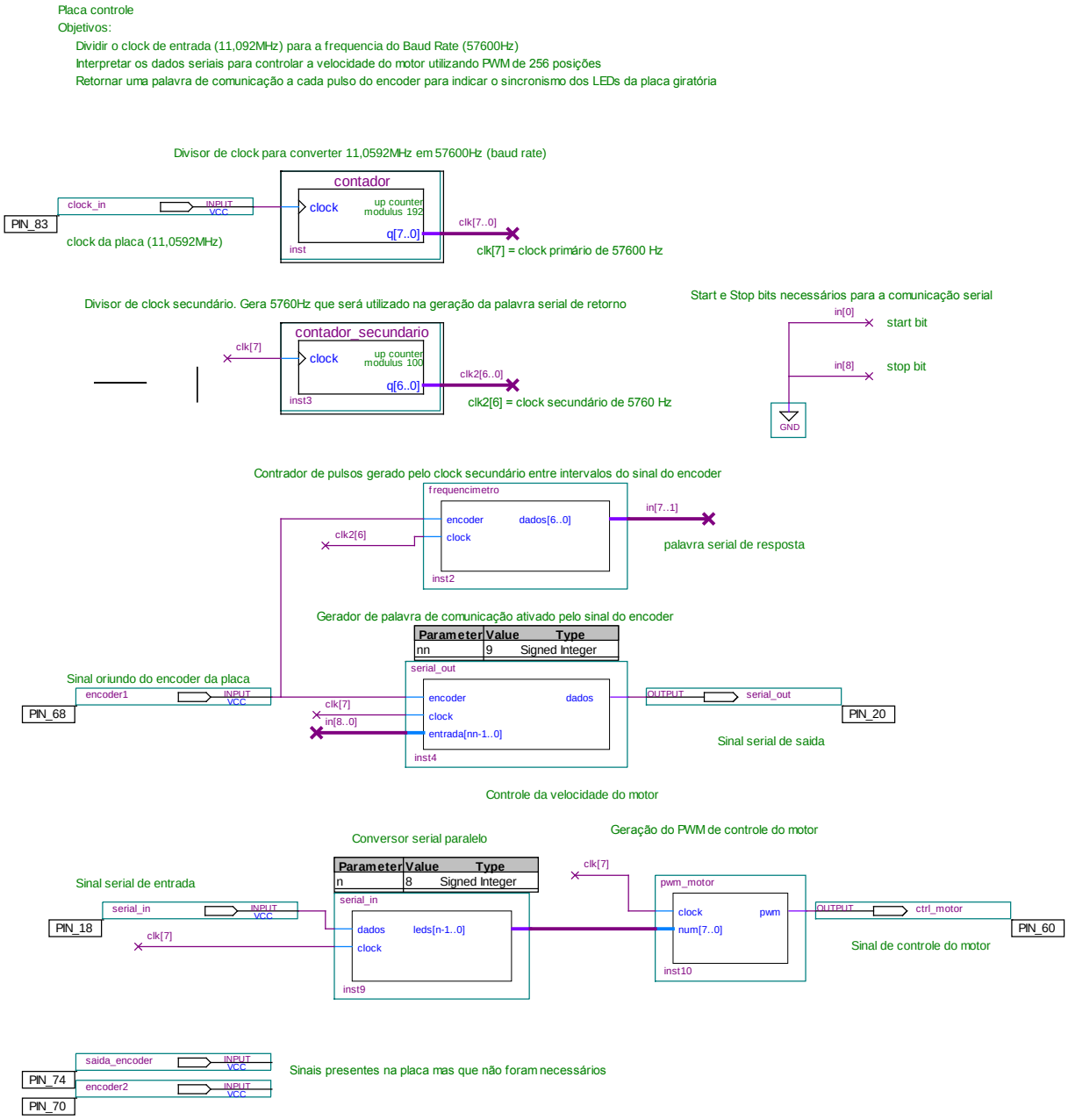


Placa Rotativa



Apêndice D – Descrições de hardware: Diagrama em blocos

Placa Controle

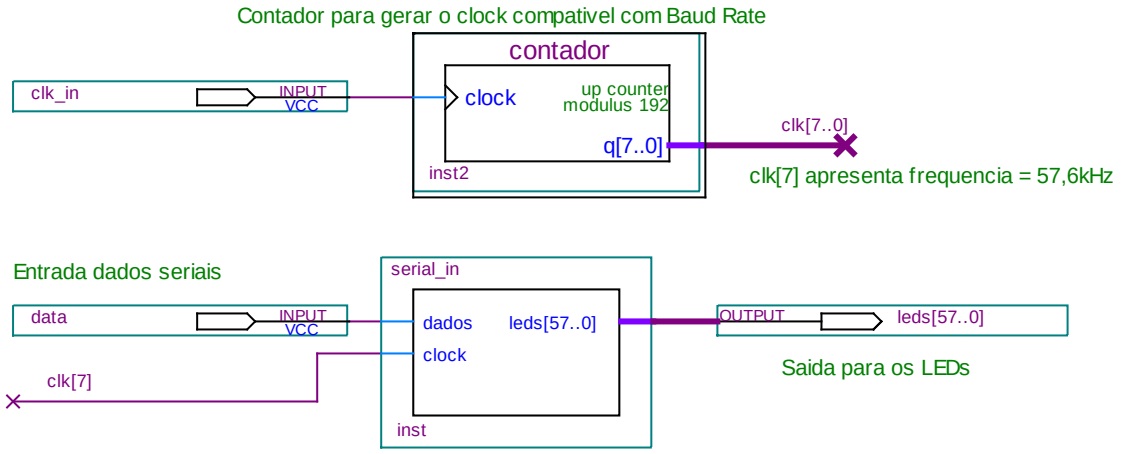


Placa Rotativa

Diagrama em blocos
Placa rotativa

Objetivos:

Gerar clock compativel com Baud Rate da comunicação serial (57600 Hz)
Interpretar dados seriais para acender os LEDs



Apêndice E – Descrição de hardware: VHDL

Placa Controle - Bloco Serial in

```
-- Conversor serial paralelo
-- TCC - Placa controle
-- Igor Guerrero
-- Objetivos:
-- Converter os bits adquiridos da comunicação serial para bits em paralelo

-- Declaração da entidade
ENTITY serial_in IS
    -- Generico que indica o número de bits que formam uma palavra
    GENERIC( n          :   INTEGER := 8    );

    PORT    (   dados :   IN  BIT; -- Bits seriais
                clock :   IN  BIT; -- Clock compativel com Baud Rate
                leds  :   OUT BIT_VECTOR ( 0 TO n-1 ) -- Vetor de saida com os 19 bits de
entrada em paralelo
            );
END serial_in;
-- Fim da entidade

-- Inicio da arquitetura
ARCHITECTURE UAR OF serial_in IS
BEGIN

-- Processamento sincrono com o Baud Rate da comunicação serial
    PROCESS(clock)
        VARIABLE    head        :   BIT_VECTOR ( 2 DOWNTO 0 )   :=  "000"; -- Verificador de
head do sinal
        VARIABLE    inicio      :   BIT :=  '0'; -- Flag de start bit
        VARIABLE    i           :   INTEGER RANGE 0 TO 127 := 0; -- Contador
        VARIABLE    leds_aux    :   BIT_VECTOR (n-1 DOWNTO 0 ); -- Varivel auxiliar
        VARIABLE    leds_in     :   BIT_VECTOR (n-1 DOWNTO 0 ) := (OTHERS => '0'); -- Variavel
de realimentação
        VARIABLE    paridade    :   BIT := '0'; -- Bit que verifica paridade
        VARIABLE    pad         :   BIT_VECTOR (8 DOWNTO 0) := "000000000"; -- Bits para
verificação dos pads entre bytes
    BEGIN

    -- condição 1 - Ativo na subida do clock
        IF ( clock'EVENT AND clock = '1' ) THEN

     -- condição 2 - Verifica se os dados recebidos atualmente pertencem a uma serie já
iniciada
            IF ( inicio = '1' ) THEN

     -- Utilizando o contador e verificando "head" da palavra serial muda os valores dos leds
                CASE i IS
          WHEN 1     =>    head(0)  := dados;
          WHEN 2     =>    head(1)  := dados;
          WHEN 3     =>    head(2)  := dados;
          WHEN 4     =>    IF head = "001" THEN leds_aux(0)  := dados; ELSE i := 30; END IF;
          WHEN 5     =>    IF head = "001" THEN leds_aux(1)  := dados; ELSE i := 30; END IF;
          WHEN 6     =>    IF head = "001" THEN leds_aux(2)  := dados; ELSE i := 30; END IF;
          WHEN 7     =>    IF head = "001" THEN leds_aux(3)  := dados; ELSE i := 30; END IF;
          WHEN 8     =>    pad(2) := dados;
          WHEN 9     =>    pad(3) := dados;
          WHEN 10    =>    pad(4) := dados;
          WHEN 11    =>    IF head = "001" THEN leds_aux(4)  := dados; ELSE i := 30; END IF;
          WHEN 12    =>    IF head = "001" THEN leds_aux(5)  := dados; ELSE i := 30; END IF;
          WHEN 13    =>    IF head = "001" THEN leds_aux(6)  := dados; ELSE i := 30; END IF;
          WHEN 14    =>    IF head = "001" THEN leds_aux(7)  := dados; ELSE i := 30; END IF;
          WHEN 15    =>    paridade := dados;
          WHEN 16    =>    pad(0) := dados;
          WHEN 17    =>    pad(1) := dados;
          WHEN 25    =>    IF( paridade = (leds_aux(0) XOR leds_aux(1) XOR leds_aux(2) XOR
leds_aux(3) XOR leds_aux(4) XOR leds_aux(5) XOR leds_aux(6) XOR leds_aux(7)) AND pad(4 DOWNTO
0) = "01011") THEN inicio := '0'; leds <= leds_aux; leds_in := leds_aux; ELSE leds <= leds_in;
inicio := '0'; END IF; pad(4 DOWNTO 0) := "00000";
          WHEN 26    =>    inicio := '0';
          WHEN 126   =>    inicio := '0';
          WHEN 127   =>    inicio := '0';
```

```
                WHEN OTHERS =>    NULL;
                END CASE;

                -- Caso serie de dados tenha sido iniciada, incrementa contador a cada loop.
                i := i + 1;

            ELSE    -- condição 2

                i := 1; -- Reinicializa contador

                -- condição 3 - Caso canal esteja em Idle, verifica Start Bit
                IF (dados = '0') THEN   -- Start Bit = '0'
                    inicio := '1';
                END IF; -- condição 3

            END IF; -- condição 2
        END IF; -- condição 1
    END PROCESS; -- Finaliza processo sincrono ao clock
END UAR;
-- Fim da arquitetura
```

Placa Controle - Bloco PWM Motor

```
-- Conversor paralelo em PWM
-- TCC - Placa controle
-- Igor Guerrero
-- Objetivos:
-- Converter uma entrada (inteiro) em um sinal de PWM

-- Declaração da entidade
ENTITY pwm_motor IS
    PORT    (   clock   :   IN      BIT;                    -- Clock base
                pwm     :   OUT     BIT;                    -- Saida
                num     :   IN      INTEGER RANGE 0 TO 255  -- Entrada
            );
END pwm_motor;
-- Fim da entidade

-- Inicio da arquitetura
ARCHITECTURE motor OF pwm_motor IS
    SIGNAL  pwm_aux     :   BIT :=  '0';    -- Variavel auxiliar
    SIGNAL  num_aux     :   INTEGER RANGE 0 TO 255  := 127;
BEGIN

-- Processamento sincrono com o clock base
PROCESS(clock)
    VARIABLE    aux :   INTEGER RANGE 0 TO 255;

    BEGIN
        -- Na subida de borda incrementa um contador e compara com o número de entrada
        IF ( clock'EVENT AND clock = '1' ) THEN

            aux := aux + 1;

            IF ( aux <= num_aux ) THEN
                pwm_aux <= '1';
            ELSE
                pwm_aux <= '0';
            END IF;

            IF ( aux = 255 ) THEN
                aux := 0;
                num_aux <= num;
            END IF;

        END IF;
    END PROCESS;

    pwm <= pwm_aux;

END motor;
-- Fim da arquitetura
```

Placa Controle – Bloco Serial out

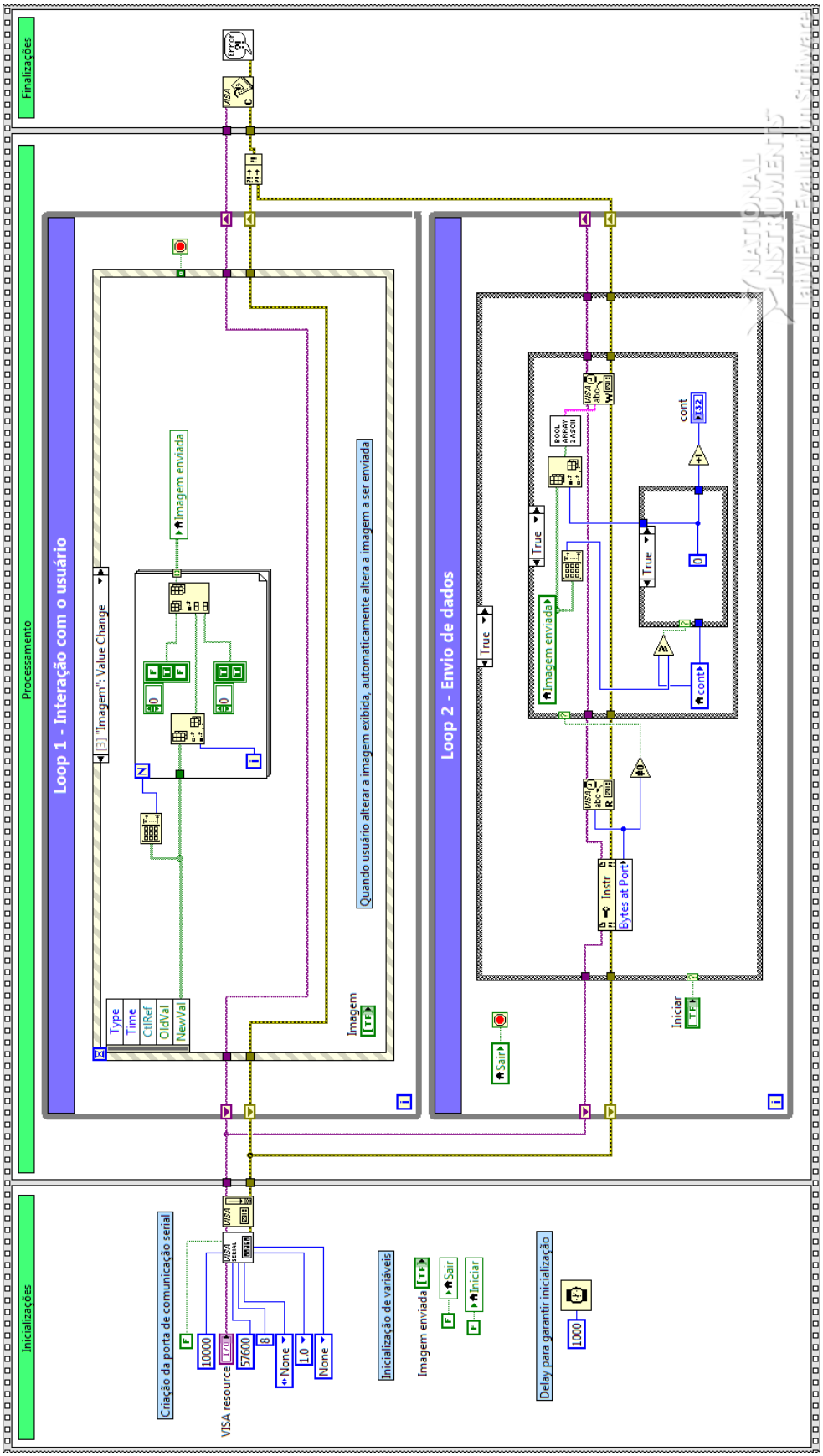
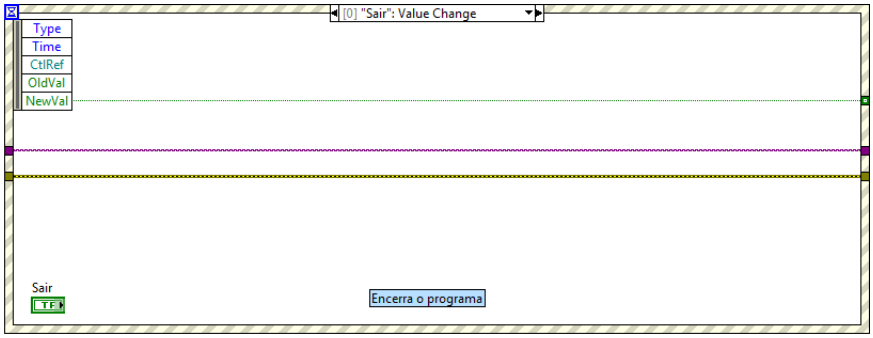
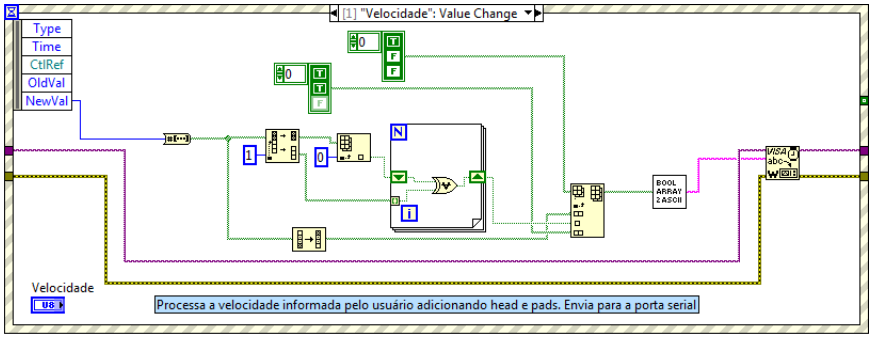
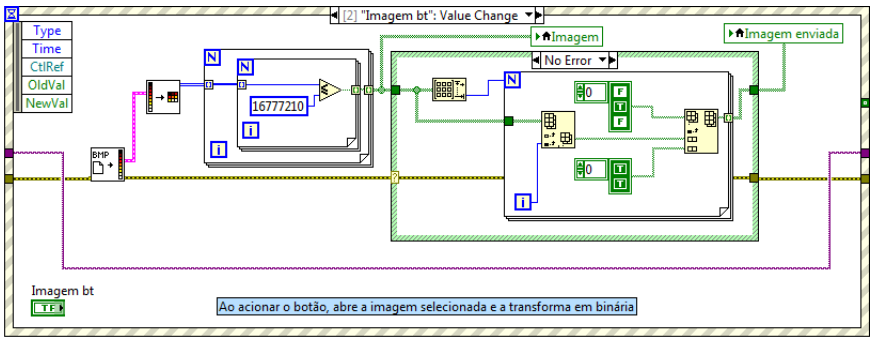
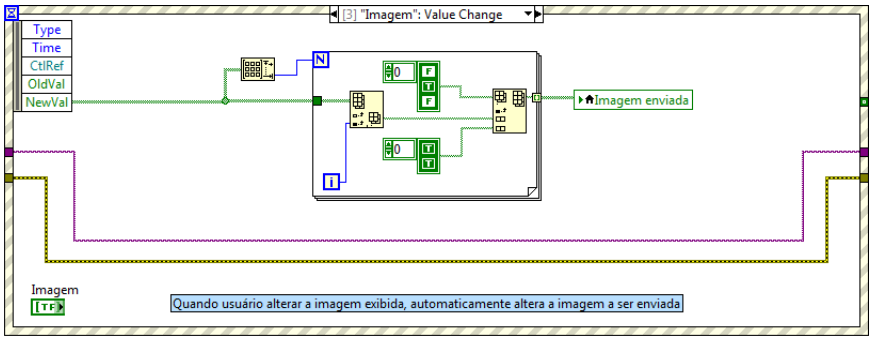
```
-- Conversor paralelo serial
-- TCC - Placa controle
-- Igor Guerrero
-- Objetivos:
-- Converter dados em paralelo para serem utilizados na comunicação serial

-- Declaração da entidade
ENTITY serial_out IS
    GENERIC( nn      :   INTEGER := 9     );

    PORT     (   dados   : OUT   BIT; -- Dados seriais de saida
                 encoder : IN    BIT; -- Trigger de envio
                 clock   : IN    BIT; -- Baud Rate
                 entrada : IN    BIT_VECTOR ( nn-1 DOWNTO 0 ) -- Dados a serem enviados
             );
END serial_out;
-- Fim da entidade

-- Inicio da arquitetura
ARCHITECTURE UAR_out OF serial_out IS
    SIGNAL  aux     :   BIT :=  '0'; --Variáveis auxiliares
    SIGNAL  aux2    :   BIT :=  '0';
    SIGNAL  i       :   INTEGER RANGE 0 TO nn+1 :=  0;
BEGIN

    PROCESS(clock)
    BEGIN

        IF( clock'EVENT AND clock = '1') THEN

            IF( aux = '1' AND aux2 = '0') THEN
                i <= 0;
            END IF;

            CASE i IS
                WHEN 0  =>  dados <= entrada(0);    i <= i+1;   aux2 <= '1';
                WHEN 1  =>  dados <= entrada(1);    i <= i+1;   aux2 <= '1';
                WHEN 2  =>  dados <= entrada(2);    i <= i+1;   aux2 <= '1';
                WHEN 3  =>  dados <= entrada(3);    i <= i+1;   aux2 <= '1';
                WHEN 4  =>  dados <= entrada(4);    i <= i+1;   aux2 <= '1';
                WHEN 5  =>  dados <= entrada(5);    i <= i+1;   aux2 <= '1';
                WHEN 6  =>  dados <= entrada(6);    i <= i+1;   aux2 <= '1';
                WHEN 7  =>  dados <= entrada(7);    i <= i+1;   aux2 <= '1';
                WHEN 8  =>  dados <= entrada(8);    i <= i+1;   aux2 <= '0';
                WHEN OTHERS =>  dados <= '1';       aux2 <= '0';
            END CASE;

        END IF;
    END PROCESS;

    PROCESS(encoder, aux2)
    BEGIN

        IF( encoder'EVENT AND encoder = '1') THEN
            aux <= '1';
        END IF;

        IF( aux2 = '1' ) THEN
            aux <= '0';
        END IF;

    END PROCESS;

END UAR_out;
-- Fim da arquitetura
```

Placa Rotativa – Bloco Serial in

```
-- TCC
-- Código para a placa rotativa
-- Igor Guerrero

-- Objetivos:
-- Utilizando somente um pino, decodificar o sinal serial
-- recebido e acender corretamente os LEDs.

-- Declaração da entidade principal
ENTITY serial_in IS

    PORT    (   dados   :   IN      BIT; -- Sinal serial
                clock   :   IN      BIT; -- Clock com a frequencia do Baud Rate
                leds    :   OUT     BIT_VECTOR ( 57 DOWNTO 0 ) -- Vetor de controle de todos
os leds da placa
            );
END serial_in;
-- Fim da entidade

-- Declaração da arquitetura
ARCHITECTURE UAR OF serial_in IS
    SIGNAL  head        :   BIT_VECTOR ( 2 DOWNTO 0 )   := (OTHERS => '0'); -- Verificador de
head do sinalo de dados
    SIGNAL  leds_aux    :   BIT_VECTOR ( 10 DOWNTO 0 )  := (OTHERS => '0');
BEGIN

    -- Processamento síncrono com o Baud Rate da comunicação serial
    PROCESS(clock)

        VARIABLE    i       :   INTEGER RANGE 0 TO 127      := 1; -- Contador
        VARIABLE    pad     :   BIT_VECTOR ( 4 DOWNTO 0 )   := (OTHERS => '0');
        VARIABLE    inicio  :   BIT := '0'; -- Flag de indicação para inicio>

    BEGIN

        -- condição 1 - Ativo na subida do clock
        IF ( clock'EVENT AND clock = '1' ) THEN

     -- condição 2 - Verifica se os dados recebidos atualmente pertecem a uma série já
iniciada
            IF ( inicio = '1' ) THEN

     -- Utilizando o contador e verificando "head" da palavra serial muda os valores dos leds
                CASE i IS
                    WHEN    1       =>  head(0)  <= dados;
                    WHEN    2       =>  head(1)  <= dados;
                    WHEN    3       =>  head(2)  <= dados;
                    WHEN    4       =>  leds_aux(0)  <= dados; IF (head /= "010") THEN i :=
108; END IF;
                    WHEN    5       =>  leds_aux(1)  <= dados;
                    WHEN    6       =>  leds_aux(2)  <= dados;
                    WHEN    7       =>  leds_aux(3)  <= dados;
                    WHEN    8       =>  pad(0)   := dados;
                    WHEN    9       =>  pad(1)   := dados;
                    WHEN    10      =>  pad(2)   := dados;
                    WHEN    11      =>  leds_aux(4)  <= dados; IF ( pad(2 DOWNTO 0) /= "010" )
THEN i := 86; END IF;
                    WHEN    12      =>  leds_aux(5)  <= dados;
                    WHEN    13      =>  leds_aux(6)  <= dados;
                    WHEN    14      =>  leds_aux(7)  <= dados;
                    WHEN    15      =>  leds_aux(8)  <= dados;
                    WHEN    16      =>  leds_aux(9)  <= dados;
                    WHEN    17      =>  leds_aux(10) <= dados;
                    WHEN    18      =>  pad(3)   := dados;
                    WHEN    19      =>  pad(4)   := dados;
                    WHEN    20      =>  IF ( pad(4 DOWNTO 3) /= "10" ) THEN i := 86; END IF;
                    WHEN    21      =>  leds(11) <= dados; leds( 10 DOWNTO 0 ) <= leds_aux;
                    WHEN    22      =>  leds(12) <= dados; leds_aux <= (OTHERS => '0');
                    WHEN    23      =>  leds(13) <= dados;
                    WHEN    24      =>  leds(14) <= dados;
                    WHEN    25      =>  leds(15) <= dados;
                    WHEN    26      =>  leds(16) <= dados;
                    WHEN    27      =>  leds(17) <= dados;
                    WHEN    31      =>  leds(18) <= dados;
                    WHEN    32      =>  leds(19) <= dados;
```

```
                WHEN    33      =>  leds(20) <= dados;
                WHEN    34      =>  leds(21) <= dados;
                WHEN    35      =>  leds(22) <= dados;
                WHEN    36      =>  leds(23) <= dados;
                WHEN    37      =>  leds(24) <= dados;
                WHEN    41      =>  leds(25) <= dados;
                WHEN    42      =>  leds(26) <= dados;
                WHEN    43      =>  leds(27) <= dados;
                WHEN    44      =>  leds(28) <= dados;
                WHEN    45      =>  leds(29) <= dados;
                WHEN    46      =>  leds(30) <= dados;
                WHEN    47      =>  leds(31) <= dados;
                WHEN    51      =>  leds(32) <= dados;
                WHEN    52      =>  leds(33) <= dados;
                WHEN    53      =>  leds(34) <= dados;
                WHEN    54      =>  leds(35) <= dados;
                WHEN    55      =>  leds(36) <= dados;
                WHEN    56      =>  leds(37) <= dados;
                WHEN    57      =>  leds(38) <= dados;
                WHEN    61      =>  leds(39) <= dados;
                WHEN    62      =>  leds(40) <= dados;
                WHEN    63      =>  leds(41) <= dados;
                WHEN    64      =>  leds(42) <= dados;
                WHEN    65      =>  leds(43) <= dados;
                WHEN    66      =>  leds(44) <= dados;
                WHEN    67      =>  leds(45) <= dados;
                WHEN    71      =>  leds(46) <= dados;
                WHEN    72      =>  leds(47) <= dados;
                WHEN    73      =>  leds(48) <= dados;
                WHEN    74      =>  leds(49) <= dados;
                WHEN    75      =>  leds(50) <= dados;
                WHEN    76      =>  leds(51) <= dados;
                WHEN    77      =>  leds(52) <= dados;
                WHEN    81      =>  leds(53) <= dados;
                WHEN    82      =>  leds(54) <= dados;
                WHEN    83      =>  leds(55) <= dados;
                WHEN    84      =>  leds(56) <= dados;
                WHEN    85      =>  leds(57) <= dados;  inicio := '0';
                WHEN    127     =>  inicio := '0';
                WHEN    OTHERS  =>  NULL;
            END CASE;

            i := i + 1;     -- Caso série de dados tenha sido iniciada, incrementa
contador a cada loop.

        ELSE    -- condição 2

            -- condição 3 - Caso canal esteja em Idle, verifica Start Bit
            IF (dados = '0') THEN -- Start Bit = '0'
                inicio := '1';
            ELSE
                inicio := '0';
            END IF; -- condição 3

            i := 1;

        END IF; -- condição 2

    END IF; -- condição 1

  END PROCESS; -- Finaliza processo sincrono ao clock

END UAR; -- Finaliza arquitetura
```

Apêndice F – Softwares em LabVIEW



Eventos contidos no Loop 1