Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Display de Persistência de Visão com Varredura Horizontal Mecânica

Autor:

Andrew Buch Sampaio

Orientador:

Prof. Fernando Antônio Pinto Barúqui

Examinador:

Prof. Carlos José Ribas D'ávila

Examinador:

Prof. Joarez Bastos Monteiro

DEL

Agosto de 2009

Aos meus pais Lilian e Fred e aos

meus irmãos Allan e Amanda.

Agradecimentos

- À Deus.
- À minha família pela educação que me deram e pela infra-estrutura que me permitiu mais esta conquista. Além disso, agradeço pelo contínuo incentivo, pelo investimento e pela paciência.
- À minha namorada Cynthia pelo incentivo e apoio constante.
- Às amizades criadas no ambiente universitário pelo companheirismo e espírito de equipe, essenciais para conclusão deste curso.
- Ao Departamento de Eletrônica e de Computação e à Escola Politécnica da UFRJ pelo excelente curso de engenharia oferecido.
- Aos professores, que souberam ensinar e compartilhar suas experiências magistralmente.

Resumo

A maior parte dos *displays* comerciais atuais é composta por matrizes estáticas de *led's* controlados por varredura eletrônica que proporcionam imagens em apenas um plano de visão.

Neste sentido, o presente projeto é uma complementação de estudos anteriores e propõe uma forma diferente de varredura que possibilita a criação de um *display* com trezentos e sessenta graus de visibilidade.

O trabalho irá utilizar o fenômeno da persistência da visão para criar um *display* de varredura horizontal mecânica, capaz de produzir imagens nítidas com apenas uma barra vertical de *led's*.

No projeto do *hardware* foram utilizados um microcontrolador PIC16F877A, um conversor de nível HIN232, um cristal oscilador em 4MHz, conectores, capacitores, resistores e leds RGB. O *firmware* do microcontrolador foi construído em linguagem C segundo o padrão CCS .

Palavras-Chave: Display, Persistência de Visão, Memória Retiniana, Varredura Mecânica, Led, RGB, Microcontrolador.

SIGLAS

UFRJ – Universidade Federal do Rio de Janeiro

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

A maior parte dos *displays* comerciais atuais é composta por matrizes estáticas de *led's* controlados por varredura eletrônica que proporcionam imagens em apenas um plano de visão.

Apesar de sua larga aplicabilidade, as matrizes de *led's*, utilizadas em painéis eletrônicos comerciais, nem sempre são a solução mais eficiente no que tange o raio de visibilidade. Em aplicações voltadas para a área de Propaganda e Marketing, por exemplo, além da estética e conteúdo, existe também a preocupação em transmitir a mensagem ao maior número possível de pessoas. Neste caso, existem duas possibilidades: Aumentar a resolução do painel ou aumentar o ângulo de visão da mensagem.

Aumentar a resolução do painel, ou seja, elevar a quantidade de linhas e colunas de *led's* empregados em sua construção irá fatalmente incrementar de forma exponencial seu custo. Portanto, existe um compromisso custo *versus* alcance a ser avaliado ao confeccionar um painel de propaganda.

Aumentar o ângulo de visão da mensagem implica em construir um painel eletrônico de secção transversal não plana. A solução mais trivial deste problema é construir um painel de matriz de *led's* cilíndrica. Mas devido à redução de área na secção vertical do painel, é necessário, a fim de se obter uma imagem nítida, aumentar sua resolução horizontal, o que eleva o custo do produto. Outra solução é construir uma estrutura mecânica capaz de girar um painel de *led's* de secção transversal plana, mas novamente, seria uma solução de custo elevado, pois o sistema seria composto pelo painel de *led's* e, no mínimo, um motor com torque suficiente para girar a estrutura.

Uma solução mais criativa e eficiente, proposta neste trabalho, é utilizar um modelo híbrido eletro-mecânico. Este modelo, que se vale de certas propriedades do sistema óptico humano, utiliza apenas um motor e uma coluna de *led's* com a desejada resolução vertical para a criação de um display de baixo custo com trezentos e sessenta graus de visibilidade.

1.1 Contribuições do Trabalho

Este trabalho pretende complementar estudos anteriores e propor uma forma diferente de varredura que possibilite a criação de um display de persistência de visão capaz de gerar imagens visíveis a qualquer ângulo relativo.

1.2 Organização do Documento

O presente trabalho está dividido em 9 capítulos. O capítulo 2 (Teoria) aborda os aspectos teóricos relevantes para a compreensão deste trabalho. O capítulo 3 (Planejamento) trata da planificação do projeto, passando pelos requisitos básicos do equipamento até análises matemáticas decisórias na escolha de componentes. O capítulo 4 (Análise) apresenta os possíveis componentes a serem utilizados no projeto, seus requisitos elementares, e os critérios relevantes para sua escolha. O capítulo 5 (Simulação) demonstra a simulação do sistema, apresentando, por meio de ilustrações, o passo a passo de uma simulação bem sucedida. O capítulo 6 (Implementação) aborda a construção física do sistema, detalhando os aspectos mecânicos e eletrônicos do projeto. O capítulo 7 (Conclusão) traz uma análise final do trabalho realizado e apresenta de forma sucinta os problemas e soluções encontradas no decorrer do projeto, bem como mostra quais partes do inicialmente proposto foram realizadas com sucesso. O

capítulo 8 (Melhorias Propostas para o Sistema) traz um conjunto de melhorias que podem ser implementadas ao sistema. O capítulo 9 (Bibliografia) traz a bibliografia recomendada para o aprofundamento teórico dos assuntos abordados neste trabalho. Os apêndices A (Firmware) e B (Esquema Elétrico), respectivamente, trazem o código integral do *firmware* e o esquema elétrico do *hardware* desenvolvidos para este projeto.

2 Teoria

2.1 Percepção Visual Humana

O Sistema Óptico Ocular [1] é formado por um complexo esquema fisiológico o qual permite interpretar não somente a sensação de cor, mas também a profundidade, textura, movimento etc. O mesmo é formado basicamente pelos globos oculares (olhos), nervos ópticos, corpos geniculados laterais e as áreas de visões. Uma das características deste sistema é a sua duplicidade, que permite exibições estéreo-dinâmicas possibilitando a insinuação de profundidade estereoscópica constituída pelas diferenças relativas ou disparidades entre as partes das imagens disponíveis nos dois olhos.

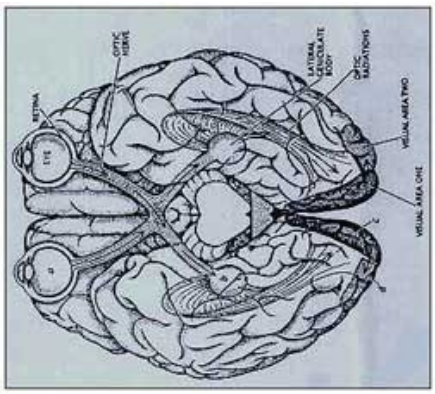


Figura - Sistema Ótico Ocular

A parte deste modelo fisiológico que traduz os raios luminosos em impulsos (sinais químicos) a serem interpretados pelo cérebro é denominada globos oculares (receptores das imagens) os quais são formados pela córnea, íris, retina, etc. A retina, por sua vez, é uma membrana ocular interna, em que estão as células nervosas (bastonetes e cones) que recebem os estímulos luminosos, e onde se projetam as imagens produzidas pelo sistema óptico ocular.

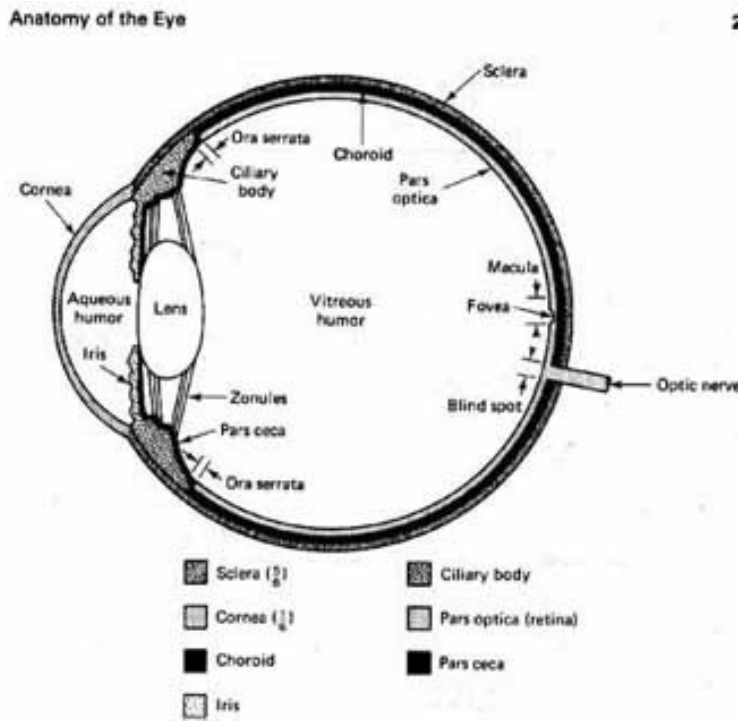


Figura - Globo Ocular

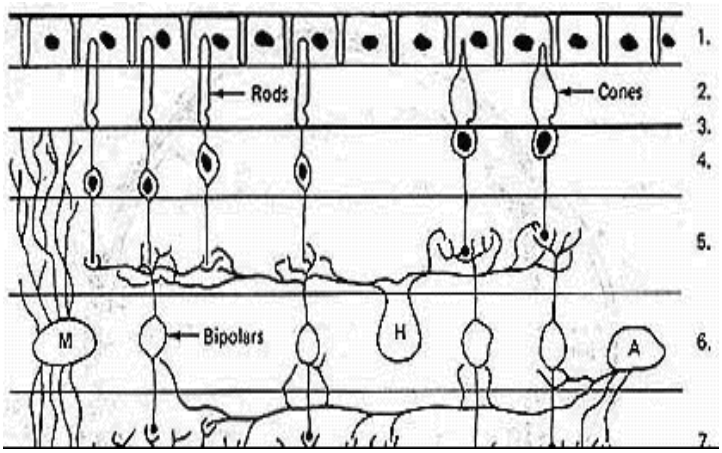


Figura - Bastonetes e Cones

2.1.1 O Funcionamento dos Olhos

Os olhos podem ser comparados a uma máquina fotográfica. Quando os raios de luz encontram a córnea e a penetram, são refratados pela primeira vez. Atravessam então a câmara anterior, que contém o humor aquoso, atingindo o cristalino e o humor

vítreo. O cristalino converge então os raios luminosos que o atravessam. Após ter atravessado o humor vítreo, a luz encontra a retina, que constitui a membrana mais interna das três que formam a parede do globo ocular. As verdadeiras células da visão encontram-se na retina, que se transforma no nervo óptico, e a imagem que surge nesta membrana é levada como um impulso elétrico até ao centro da visão, situado no lobo occipital do cérebro.

Devido aos sete milhões de células em cones que se encontram na retina, os olhos são capazes de perceber os diferentes comprimentos de onda da luz (as cores) como sensações heterogêneas.

Para a visão de penumbra (visão de crepúsculo), a retina contém ainda cerca de cento e vinte e três milhões de bastonetes, ocupando uma superfície total de apenas cinco centímetros quadrados.

Os olhos não diferenciam apenas as várias intensidades e formas da luz, mas percebem também, separadamente, os raios luminosos enviados de pontos isolados do objeto. Deste modo, é possível apreender visualmente a forma de um objeto. Na membrana sensível que constitui a retina, são projetadas imagens menores e invertidas dos objetos emissores da luz: o que no objeto está à direita encontra-se à esquerda na imagem projetada na retina, e o que no objeto se localiza na parte superior encontra-se na parte inferior da imagem.

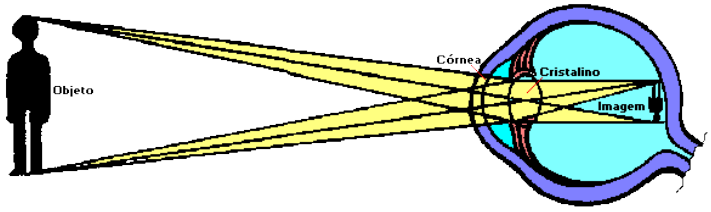


Figura - Formação da Imagem na Retina

Mas por que razão vê-se o objeto 'direito', se este se encontra invertido na imagem da retina? Na realidade, os olhos não vêem, mas reconhecem a imagem. Somente após da transmissão desta ao cérebro, e sua subseqüente inversão, obtém-se uma imagem com orientação correta.

O sistema óptico acusa numerosas imperfeições. Estas, juntamente com os erros de transmissão que se verificam entre os olhos e o cérebro, assim como as falsas interpretações transmitidas por este, explicam a existência de ilusões ópticas.

Como exemplo, pode-se citar o ‘ponto cego’. Neste ponto, zona em que o nervo óptico penetra na retina, não existe células fotossensíveis e, conseqüentemente, não ocorre qualquer sensação de luz. A figura abaixo ilustra essa propriedade: Se o leitor, com o olho direito fechado, fixar o ponto direito com o olho esquerdo e, gradualmente, aproximar-se da figura, perceberá que o ponto da esquerda tornar-se-á invisível.

Figura - Ponto Cego

2.1.2 Ilusões Sensoriais

As ilusões sensoriais verificam-se quando a percepção não se conjuga com a realidade. As ilusões sensoriais normais são, na estrutura e no modo de funcionamento dos respectivos órgãos, fundamentadas do mesmo modo que nos processos psicológicos, por impressões sensoriais associadas apenas a percepções. São, pois, fenômenos concomitantes regulares e não interrompidos de percepções sensoriais.

A figura 6 ilustra uma ilusão óptica. Ao aproximarem-se os olhos do desenho até que o nariz o toque, pode-se perceber a ponte fechar.

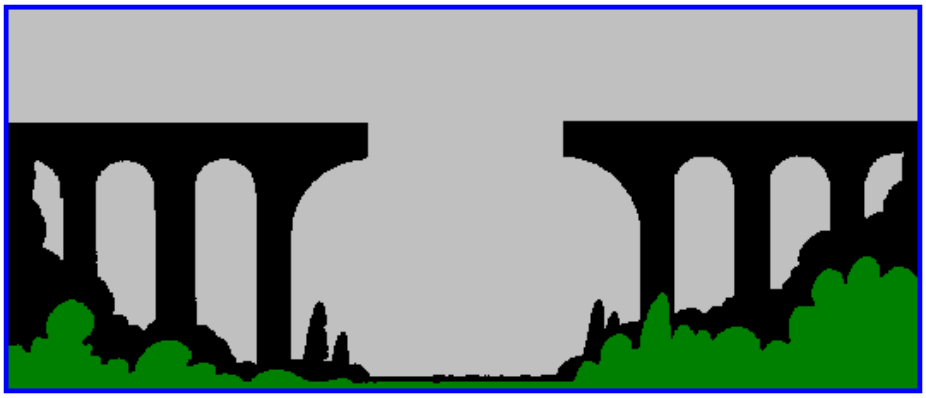


Figura - Ilusão Óptica

Na maioria das ilusões ópticas não se verificam ilusões reais dos sentidos, mas ilusões que surgem da interpretação habitual de um objeto. Neste caso, a impressão do sentido da visão, o registro óptico, é inteiramente correta, verificando-se a ilusão apenas no cérebro.

Existem variados tipos de ilusão óptica, porém, como não é escopo desde trabalho, a descrição detalhada das imperfeições do sistema óptico humano. A seguir, somente a ilusão de interesse será descrita.

2.1.2.1 Ilusões por Persistência de Imagem

As ilusões por persistência de imagem ocorrem devido a uma propriedade da visão humana denominada persistência retiniana. A persistência ou retenção retiniana designa o fenômeno ou a ilusão provocada quando um objeto visto pelo olho humano persiste na retina por uma fração de segundo após a sua percepção. Assim, imagens projetadas a um ritmo superior a 16 quadros por segundo associam-se na retina sem interrupção.

Segundo essa teoria, ao captar uma imagem, o olho humano levaria uma fração de tempo para "esquecê-la". Assim, quando os fotogramas de um filme de cinema são projetados na tela, o olho misturaria os fotogramas anteriores com os seguintes, provocando a ilusão de movimento: um objeto colocado à esquerda num fotograma, aparecendo à direita no fotograma seguinte, cria a ilusão de que o objeto se desloca da esquerda para a direita.

Os filmes ou seqüências de imagens em vídeo mostram o movimento mais suave e menos saltitante. Um filme de celulóide é apresentado a 24 fotogramas por segundo. Hoje, o vídeo digital (ou DV) é gravado a 25 (Europa) ou 29.97 q/s (cerca de 30 *frames* por segundo).

A influência de um estímulo luminoso na retina provoca uma sensação luminosa. Em conseqüência da inércia, decorre um determinado intervalo de tempo até a retina ser impressionada. Por outro lado, a excitação sobrevive ao estímulo provocado durante um breve período. É por essa razão que ao olhar diretamente para o sol durante alguns poucos segundos, e em seguida fechar os olhos, sua imagem permanece visível ainda durante algum tempo (persistência da visão).

Se a impressão luminosa for intensa, a hiper-excitabilidade da retina, decorrente do cansaço, pode reduzir-se de tal maneira que se vê uma mancha negra da forma do objeto visto - o que se designa por persistência de imagem negativa.

Exemplificando o acima exposto, com uma luz clara e durante 30 segundos, olhe para a metade esquerda da figura 7. Logo a seguir fixe o lugar marcado (ponto) da metade direita da mesma figura.

Figura - Ilusão por Persistência de Imagem

Como um segundo exemplo, fixe, durante 30 segundos e com luz clara, o olhar para a imagem da esquerda da figura 8 e logo a seguir olhe para a da direita. Verá uma persistência de imagem colorida: o fundo vermelho aparecerá agora na sua cor complementar, o verde, e a face branca tornar-se-á ligeiramente avermelhada, devido à impressão de contraste.

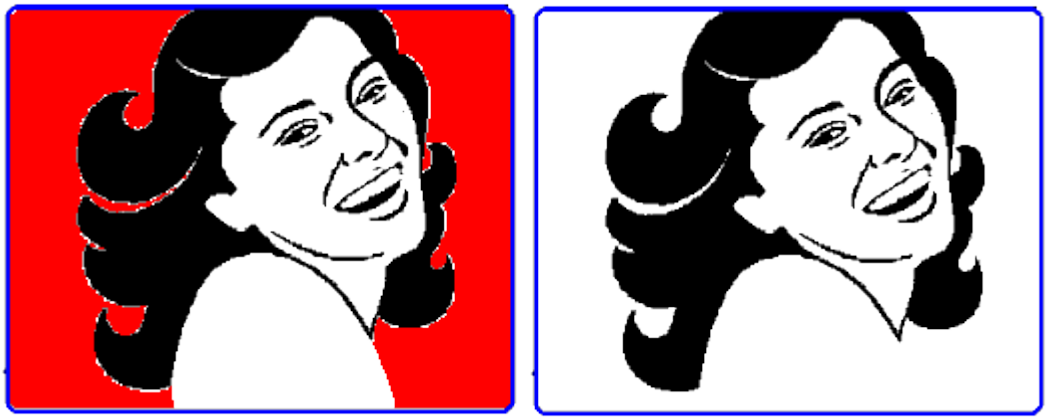


Figura - Ilusão por Persistência Colorida de Imagem

2.1.3 Visão das Cores

O olho humano sente o espectro de cores usando uma combinação da informação vinda de células localizadas no olho, chamadas de cones e bastonetes. Os bastonetes são mais adaptados a situações de pouca luz, mas eles somente detectam a

intensidade da luz, os cones, por outro lado, funcionam melhor com intensidades maiores de luz e são capazes de discernir as cores. Existem três tipos de cones nos olhos, cada um especializado em comprimentos de luz curtos (S), médios (M) ou longos (L). O conjunto de sinais possíveis dos três tipos de cones define a gama de cores que conseguimos ver. As figuras 9 e 10[1] ilustram a sensibilidade relativa de cada um dos tipos de células cone para todo o espectro de luz visível – de ~400 nm a 700 nm – e sua luminosidade.

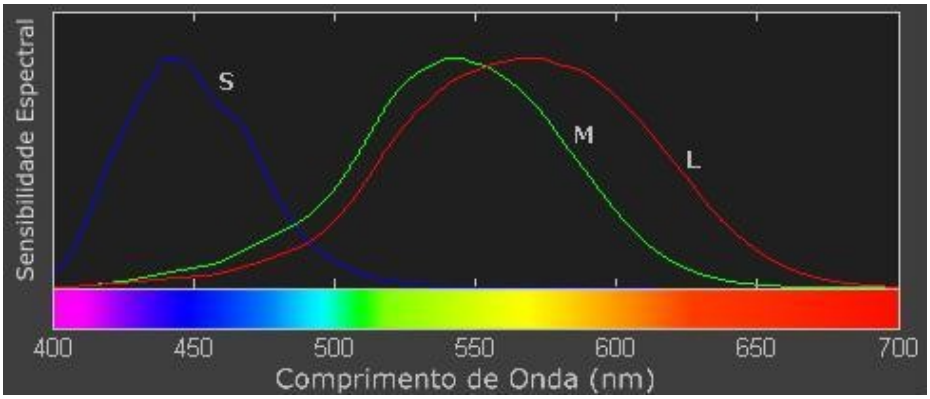


Figura - Sensibilidade Espectral

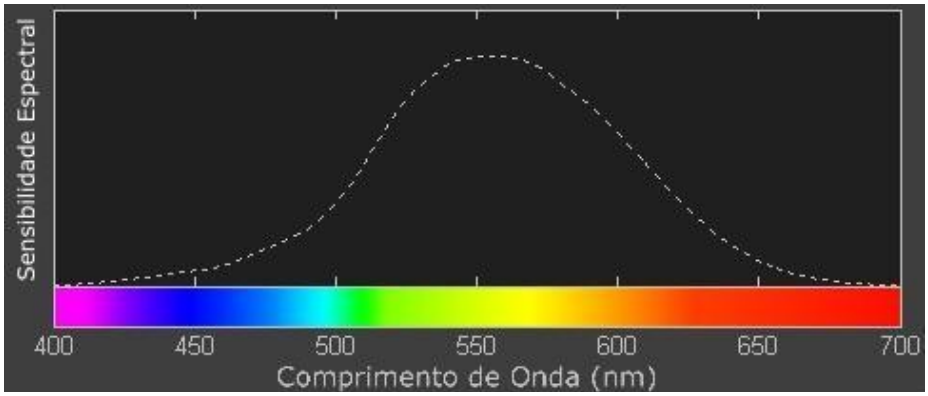


Figura - Intensidade Luminosa

Nota-se que cada tipo de célula não só se especializa em uma cor, mas também tem níveis variáveis de sensibilidade ao longo de uma gama de comprimentos de onda. Conclui-se também que a percepção humana de cores é mais sensível à luz na região amarelo-verde do espectro.

1 Dados cortesia de 'Colour and Vision Research Laboratories' (CVRL), UCL

2.1.3.1 Misturando Cores por Adição e Subtração

Praticamente todas as cores visíveis podem ser produzidas utilizando alguma mistura de cores por combinação aditiva ou subtrativa primárias. Este fato é evidenciado pela figura 11 . O processo aditivo cria cores adicionando luz a um fundo preto, o processo subtrativo usa pigmentos ou tinturas para, seletivamente, bloquear a luz branca. A compreensão de cada um desses processos é a base fundamental para entender a reprodução de cores.

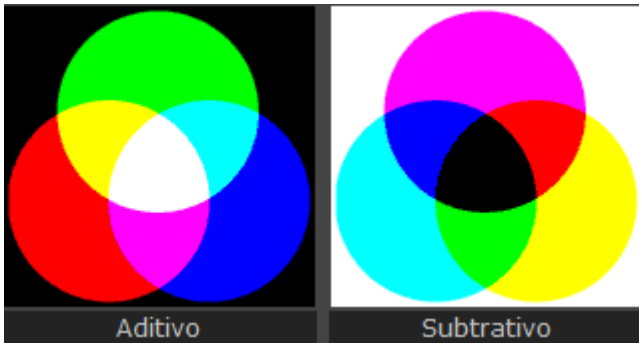


Figura - Métodos de Geração de Cores

As cores nos três círculos exteriores são chamadas de primárias e são diferentes em cada um dos diagramas. Aparatos que se baseiam em cores primárias para representar cores só podem produzir uma gama limitada de cores. Os monitores de computador, por exemplo, emitem luz para produzir cores através do processo aditivo; impressoras, por outro lado, usam tinta, ou pigmento, para absorver a luz, através do processo subtrativo. É por isso que a grande maioria de monitores usa uma combinação de pixels vermelho, verde e azul (o que é comumente chamado de RGB, do inglês *Red, Green and Blue*). As impressoras, por sua vez, usam tintas das cores ciano, magenta e amarelo (o que é chamado de CMY, do inglês *Cyan, Magenta and |Yellow*). Muitas impressoras também utilizam a tinta preta (abreviado, nesse caso, por CMK, onde o K vem de *Black*), já que uma combinação de CMY não é capaz de produzir preto profundo.

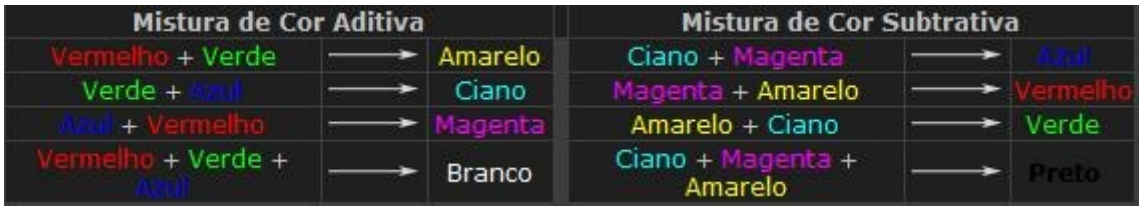


Figura - Cores Primárias

Processos subtrativos são mais suscetíveis as mudanças na luz ambiente já que eles dependem dessa luz para produzir as cores. Cores impressas normalmente necessitam de um tipo específico de luz para reproduzir fielmente as cores que são vistas em uma tela.

3 Planejamento

Antes de começar a desenvolver qualquer projeto, deve-se realizar um estudo prévio do que se pretende construir. Nas seções subseqüentes, está documentado todo o estudo das especificações funcionais e estruturais realizado para o desenvolvimento deste trabalho.

3.1 Descrição Funcional

3.1.1 Requisitos da Aplicação

Na tabela abaixo estão listadas e classificadas as funções desejadas para este projeto.

Tabela - Descritivo Funcional

Descrição	Obrigatório	Recomendável	Prioridade
Mostrar texto	✓		1
Conectividade com computadores pessoais	✓		1
Mostrar mensagens predeterminadas	✓		1
Alimentação do circuito independente de baterias	✓		1
Mostrar imagens		✓	2
Mostrar mensagens personalizadas		✓	2
Software de interface		✓	3
Transferência de dados *wireless*		✓	3

3.1.2 Funções e Propriedades

3.1.2.1 Power Up:

Imediatamente após ser ligado, o *display* deverá exibir uma mensagem de apresentação predeterminada que se repetirá enquanto o usuário não modificá-la.

3.1.2.2 Modificar a Mensagem Apresentada pelo Display:

A operação do equipamento deve ser intuitiva. O usuário não deve encontrar dificuldades para customizar a mensagem mostrada no *display.*

3.1.2.3 Display:

As mensagens devem aparecer nítidas dentro do raio de visão coberto pelo equipamento.

3.2 Descrição Estrutural

A melhor forma de iniciar uma descrição estrutural é utilizar diagrama de blocos. A figura 13 mostra os principais blocos deste projeto.

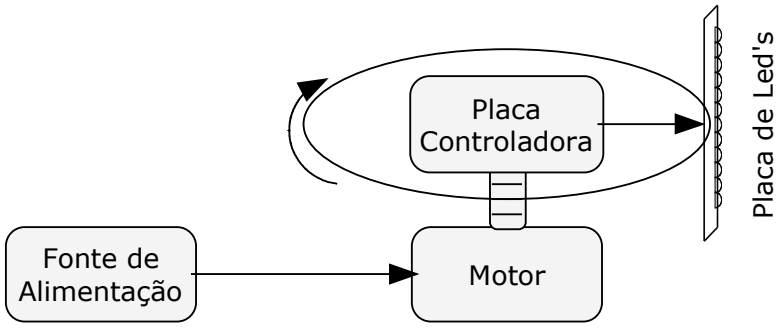


Figura - Diagrama Estrutural de Blocos

3.2.1 Motor

O motor elétrico é uma máquina destinada a transformar energia elétrica em mecânica. É o mais usado de todos os motores, pois combina as vantagens da energia elétrica – baixo custo, facilidade de transporte, limpeza e simplicidade de comando – com sua construção simples, custo reduzido, grande versatilidade de adaptação às cargas dos mais diversos tipos e melhores rendimentos.

Os motores elétricos mais comuns são:

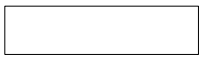
- Motores de corrente contínua:

  São motores de custo elevado e, além disso, precisam de uma fonte de corrente contínua, ou de um dispositivo que converta a corrente alternada comum em contínua. Podem funcionar com velocidade ajustável entre amplos limites e se prestam a controles de grande flexibilidade e precisão.

- Motores de corrente alternada:

  São os mais utilizados, porque a distribuição de energia elétrica é feita normalmente em corrente alternada. Seu princípio de funcionamento é baseado no campo girante, que surge quando um sistema de correntes alternadas trifásico é aplicado em pólos defasados fisicamente de 120º. Dessa forma, como as correntes elétricas são defasadas de 120º, em cada instante, um par de pólos possui o campo de maior intensidade, gerando um vetor que causa o efeito de campo girante.

A escolha de um motor apropriado é determinante para o sucesso deste projeto, pois, como visto anteriormente, necessita-se, a fim de atingir o efeito de persistência de visão, uma taxa de quadros (*frame rate*) de pelo menos 25 quadros por segundo (*fps - frames per second*). Então, para o motor a ser utilizado neste projeto, tem-se como especificação mínima:



Ou seja, será necessário um motor que gire no mínimo a 1500 rpm (rotações por minuto).

3.2.2 Fonte de Alimentação

A fonte de alimentação a ser utilizada depende da escolha do motor. No caso de um motor de corrente alternada, será necessário projetar uma fonte DC para alimentar apenas as placas controladora e de *led's*. No caso de um motor de corrente contínua será necessário projetar uma fonte de alimentação que forneça corrente para o motor e, possivelmente, um regulador de tensão para alimentar as placas.

3.2.3 Mecânica

Antes de iniciar o projeto das placas controladoras de *led's*, realizou-se um estudo dos aspectos mecânicos relevantes para a correta operação do equipamento.

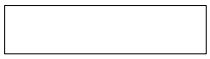
3.2.3.1 Cálculo da Velocidade Tangencial da Placa de Led's:

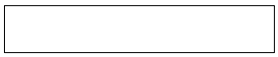
A velocidade de rotação da placa de *led's* impacta diretamente na quantidade de imagens por segundo que podem ser apresentadas. Sabe-se que quanto maior a taxa fps, mais estável se verá a imagem apresentada.

Como a varredura horizontal do display é mecânica, será difícil atingir altas taxas fps. Por este motivo fez-se a escolha de uma taxa de 25 fps o que resulta em um motor de 1500 rpm.

Um fator decisivo para a velocidade tangencial é o valor do raio da circunferência imaginária percorrida pela placa de *led's*. Quanto maior o raio, maior a velocidade tangencial da placa.

Considerando um raio de 12,5 cm a distância horizontal percorrida pela placa e sua velocidade tangencial é:





Provou-se, por meio destes cálculos, que a placa de *led's*, mesmo para taxas fps e raios de circunferências pequenos, girará em velocidades elevadas, o que provocará vibrações mecânicas que devem ser levadas em consideração na construção da estrutura mecânica.

A fim de minimizar os efeitos da vibração mecânica, como a redução de velocidade tangencial – e conseqüente redução na taxa fps – e a degradação estrutural do equipamento, o conjunto formado pelas duas placas (controladora e de *led's*) deverá ser perfeitamente equilibrado.

3.2.4 Placa Controladora

Dependendo da velocidade tangencial da placa de *led's* e da resolução vertical do display, a placa controladora deve ser suficientemente rápida para comutar entre aceso e apagado todos os *led's* existentes. Portanto, deve-se calcular, ainda que

aproximadamente, o tempo de resposta requerido para que a placa controladora atualize de forma satisfatória o conteúdo do display.

Para que se tenha uma imagem limpa e proporcional, as distâncias relativas horizontal e vertical entre *pixels* devem ser iguais. Respeitando esta condição e considerando que os *led's* possuem um comprimento de 5 mm, o número máximo de *pixels* (n) que podem ser representados na horizontal é:

A placa controladora é exigida ao seu máximo quando for necessário comutar os *pixels* em todas as posições espaciais possíveis, ou seja, quando for necessário exibir um padrão de bits igual a 10101010101010101... . Portanto, a placa controladora deve ser capaz de comutar um *pixel* 157 vezes a uma taxa de 25 fps, o que representa uma freqüência de comutação (f) de:

Isto implica em um tempo de reação (*T*) de:

4 Análise

Este capítulo abordará a analise dos dados obtidos na fase de planejamento. Aqui serão tomadas todas as decisões de projeto necessárias para o início da simulação do sistema e sua subseqüente construção física.

4.1 Placa de Led's

A placa de *led's* é decisiva para a qualidade do *display*, pois define sua resolução vertical e gama de cores primárias. Infelizmente, o aumento de complexidade de seu projeto acarreta, inevitavelmente, o aumento de complexidade da placa controladora. Por esse motivo, fez-se um esforço para encontrar um *layout* que proporcione um equilíbrio entre complexidade e qualidade.

Após um minucioso estudo e montagem de dois protótipos, a fim de obter uma imagem de qualidade e aumentar a gama de mensagens do *display*, decidiu-se por uma resolução vertical de 8 *pixels*. Apesar de não ser uma resolução alta para a apresentação de imagens, cumpre perfeitamente com o propósito do projeto que, como visto no capítulo de introdução, é a construção de um display para mensagens de propaganda. Com essa resolução será possível apresentar imagens simples, como, por exemplo, o logo de uma empresa, e textos (utilizando o padrão 7x5 *pixels*).

Outra característica expressiva na funcionalidade do projeto é a opção por um *display* monocromático ou colorido. Apesar do aumento abrupto na complexidade da placa controladora com a opção por um display colorido, esta foi, por motivos óbvios, a alternativa escolhida para este trabalho. No caso de um *display* monocromático, a relação qualidade *versus* custo, para a aplicação em questão, não vale à pena, pois existe uma significante limitação de suas funcionalidades, e, conseqüente, perda de qualidade, para um ganho baixo de custo de produção.

4.1.1 Avaliação dos Led's RGB

Uma vez eleita a opção de construir um *display* colorido, realizou-se um estudo dos possíveis componentes integrantes da placa de *led's*. Nas figuras 14, 15 e 16 encontram-se fotos dos três *led's* candidatos a serem utilizados na construção deste projeto:

Figura – Led Cilíndrico

Figura – Led Plano

Figura - Led SMD

A tabela abaixo apresenta um resumo das mais importantes características dos *led's* avaliados:

Tabela - Avaliação dos Led's Comerciais

Tipo	Cores Emitidas	Tamanho (mm)	Vida Útil (Hr)	Ângulo de Visão	Intensidade Luminosa (mcd)
Led Cilíndrico	RGB	5	100000	25° a 35°	4000(Mín) - 5000(Máx)
Led Plano	RGB	5	100000	120° ± 60°	6000(Mín) - 8000(Máx)
Led SMD	RGB	5	100000	120°	4000(Mín) - 6000(Máx)

Para a aplicação em questão, as características mais impactantes são o ângulo de visão e a intensidade luminosa. Portanto, para a construção da placa de *led's*, decidiu-se utilizar *Led's* Planos.

4.2 Placa Controladora

4.2.1 Requisitos

A placa controladora tem como requisitos mínimos:

- Um microcontrolador com tempo de resposta máximo de 254,7 μs - conforme calculado no sub-tópico 3.2.4;
- Um conversor de nível de tensão para a comunicação serial;

4.2.2 Escolha do Microcontrolador

Devido às características requeridas pela aplicação, o mais natural é a escolha de um microcontrolador que possua muitas portas de entrada e saída, que trabalhe em freqüências altas e que possua bastante memória de programa e RAM. O microcontrolador eleito para este projeto foi o PIC16F877A da Microchip® [2], pois, além de cumprir todas as exigências acima citadas, possui módulos de comunicação serial e I2C por *hardware*, o que agrega velocidade de processamento à placa controladora. Foto do componente e sua estrutura interna, respectivamente, podem ser visualizadas nas figuras 17 e 18:

Figura - PIC16F877A

Figura - Diagrama de Blocos - PIC16F877A

A seguinte tabela reúne as principais características do microcontrolador em questão:

Tabela - Características PIC16F877A

Recurso	Valor
Freqüência de Operação	DC - 20Mhz
Resets (e Delays)	POR,BOR (PWRT,OST)
Memória de Programa (palavras de 14 bits)	8 K
Memória de Dados (bytes)	368
Memória EEPROM (bytes)	256
Interrupções	15

Portas de I/O	Ports A,B,C,D,E
Timers	3
Módulos de Captura/Comparação/PWM	2
Comunicação Serial	MSSP, USART
Comunicação Paralela	PSP
Módulo A/D de 10-Bits	Canais de 8 entradas
Comparadores Analógicos	2
Set de Instruções	35 instruções
Encapsulamentos	PDIP de 40 pinos; PLCC de 44 pinos; TQFP de 44 pinos; e QFN de 44 pinos

4.2.3 Escolha do Conversor de Nível

Foi escolhido o HIN232 da Intersil® [3] para realizar a conversão do nível dos sinais TTL, processados pelo microcontrolador, para RS232, processados pelo computador.

4.3 Alimentação das Placas

A tarefa de alimentar as placas controladora e de *led's* não é simples, uma vez que as mesmas estão em constante movimento de rotação. Dessa forma, visando buscar a estrutura mais eficiente, estudaram-se, utilizando a matriz de decisão representada pela tabela 4, três maneiras distintas de levar a alimentação até as placas de circuito impresso:

- Alimentação por contato deslizante: Uma estrutura cinlíndrica com pelo menos duas faixas condutoras (Terra e Vcc) é anexada ao eixo do motor. A energia pode, então, ser transmitida via carvões ou escovas.
- Alimentação por Gerador: Pode-se aproveitar o movimento de rotação do motor para montar um dínamo.
- Alimentação por Transformador: Montando-se uma bobina no eixo do motor, pode-se aproveitar o campo eletromagnético de suas bobinas primárias para

induzir no secundário – montado no eixo - a corrente elétrica necessária para alimentar o circuito giratório.

Tabela - Matriz de Decisão da Alimentação do Circuito

	Contato Desli-zante	Gerador	Transformador
Critério	Pontos	Pontos	Pontos
Funciona em Movimento Rotativo	10	10	10
Funciona sem a Presença de Movimento	10	0	10
Peso Reduzido	10	2	4
Suporta Correntes Elevadas (>300 mA)	7	5	10
Pouco Sensível à Sujeiras	3	10	10
Possibilidade de Velocidades Elevadas	8	10	10
Baixo Custo	10	4	5
Implementação Física	10	8	7
Soma	68	49	66
Ranking	1	3	2

A melhor opção, segundo a matriz de decisão acima, é utilizar contatos deslizantes para a alimentação do circuito.

4.4 Comunicação com o Computador

A priori, a comunicação com o computador, a exemplo da alimentação do circuito, será realizada por meio de contatos deslizantes. Conforme prioridade recebida na tabela de descritivo funcional do sistema – Tabela 4, em uma segunda instância será projetado um módulo cuja tarefa é executar a comunicação *wireless* com o computador.

4.4.1 Matriz de Decisão do Método de Comunicação *Wireless*

Dois métodos de comunicação *wireless* foram estudados: IrDA (Transferência de dados utilizando *led's* infra-vermelhos) e Radio Freqüência. Na tabela 5, segue a matriz de decisão gerada:

Tabela - Matriz de Decisão dos Métodos de Comunicação Wireless

	IrDA	Rádio Freqüência
Critério	Pontos	Pontos
Link Confiável	3	10
Alcance de Transmissão	4	10
Baixa Potência	5	3
Taxa de Dados Elevada	8	10
Disponibilidade Comercial	10	10
Preço	10	5
Soma	40	48
Ranking	2	1

Optou-se pela rádio-freqüência como método de comunicação *wireless* com o computador.

4.5 Sincronismo Horizontal

Para alcançar o efeito de persistência retiniana, como visto anteriormente, a placa controladora deve ser capaz de comutar a coluna de *led's* exatamente na mesma posição espacial. Qualquer erro de temporização acarretará, possivelmente, em uma imagem borrada ou até mesmo a perda do efeito de persistência de visão. O sincronismo horizontal é, portanto, um ponto crítico deste projeto e deve ser alcançada de forma precisa.

O método a ser utilizado para a sincronização do *hardware* depende inteiramente do motor escolhido. Caso seja utilizado um motor DC, cuja velocidade pode ser facilmente controlada por meio de modulação de largura de pulsos, PWM (*Pulse Width Modulation*), pode-se realizar o sincronismo por meio dos *timers* internos do microcontrolador. Como neste projeto, devido a ampla disponibilidade no mercado, optou-se por um motor AC, cuja velocidade varia de acordo com a flutuação da rede elétrica, é prudente utilizar um método mais confiável para a sincronização horizontal.

Utilizou-se, para alcançar o sincronismo horizontal, um par emissor-receptor de *led's* infravermelhos. O *led* emissor foi alocado na placa de conexão localizada na

superfície do motor e o *led* receptor foi conectado ao pino RA0 (pino de interrupção externa) do microcontrolador. Assim, o sinal de sincronismo é gerado sempre que os *led's* se alinharem, e, desta forma aplaca conroladora sabe a posição exata do motor. Utilizando este método, o sincronismo horizontal se torna independente de flutuações da rede elétrica, uma vez que o pulso de sincronismo é gerado a partir de uma posição inicial predeterminada do motor.

5 Simulação do Sistema

Antes de prosseguir para a fase de implementação e montagem física, todo sistema deve, a fim de encontrar possíveis erros e identificar pontos críticos para o correto funcionamento físico, passar por um longo processo de simulação.

Para a simulação da aplicação em questão, foi utilizada uma ferramenta da Labcenter Electronics® chamada Proteus – ISIS Professional [4]. Uma imagem de sua interface pode ser vista na figura 19:

Figura - IDE Proteus ISIS Professional

5.1 Montagem do Circuito para Simulação

Num processo de construção e simulação de *firmware*, é recomendável que um sistema complexo seja dividido em subsistemas simples, ou seja, que o projeto e simulação do *firmware* do sistema sejam realizados gradualmente.

Apesar de o circuito ter sido simulado em vários estágios distintos, somente será apresentada a última versão construída no simulador. Esta versão engloba todos os periféricos utilizados na montagem real do circuito e simula, com bastante fidelidade, o que acontece em um teste real. Encontra-se, na figura 20, uma imagem do circuito simulado:

Figura - Circuito a ser Simulado

5.2 Simulação do Firmware

O *firmware* do PIC16F877A foi desenvolvido utilizando a linguagem C no padrão CCS [5]. Depois de compilado o *firmware*, são gerados arquivos simuláveis de extensão ”.cof” e “.hex” que devem ser adicionados ao projeto criado no Proteus. Após configurar corretamente o caminho do arquivo hexadecimal do *firmware* desenvolvido nas opções do microcontrolador, a primeira tela apresentada pelo programa depois de iniciada a simulação é apresentada na figura 21:

Figura - Simulação Iniciada

Na parte inferior esquerda da imagem encontra-se o terminal de comunicação serial. Como o microcontrolador encontra-se em modo DEBUG, ou seja, em modo de depuração, o mesmo, imediatamente, escreve na porta serial o seguinte texto: ”Digite o texto desejado:”. Neste ponto o usuário deve inserir o texto desejado e o *hardware* deve

salvá-los na memória de programa e, posteriormente, deve detectar a o chaveamento do sensor e exibir o padrão na coluna de leds. Este comportamento é evidenciado pela figura abaixo:

Figura - Entrada de dados

Pode-se verificar, a partir da figura 22, o correto funcionamento do protocolo de comunicação serial. O sucesso na gravação da memória de programa pode ser visualizado na figura 23:

Figura - Memória Corretamente Gravada

Após a correta gravação da memória, o hardware espera pelo chaveamento do sensor de posição acontecer (simulado neste circuito pelo botão à esquerda do microcontrolador). A figura 24 mostra o comportamento do circuito caso o botão seja pressionado: O processador identifica o pulso de sincronismo (ao ser pressionado, o botão chavea Vcc no pino RA0) e imediatamente inicia a sequência de amostragem dos pixels correnpondentes à letra que deve ser apresentada. Na figura 24 pode-se perceber o padrão de pixels de uma coluna da letra “A”, a qual, de acordo com a figura 23, deve ser apresentada primeiro.

Figura - Correto Padrão de Pixels Amostrado

6 Implementação

Este capítulo tratará da implementação real do sistema, cobrindo em detalhes todos os aspectos relevantes à montagem do sistema final.

6.1 Firmware Desenvolvido

6.1.1 Alfabeto do Display

O alfabeto do *display* é embarcado no microcontrolador. O padrão de *pixels* de cada letra foi gravado na memória de programa do microcontrolador em forma de matrizes de bytes. Ou seja, a função de amostragem dos *pixels* deve, afim de escrever uma letra qualquer no *display*, jogar para a respectiva porta do microcontrolador a linha correspondente a referida letra. Por exemplo, caso seja necessário apresentar o símbolo "!", a função deve apresentar byte a byte a linha 1 da matriz font[252]. Para facilitar a função que consulta o alfabeto, as letras foram organizadas na mesma seqüência da tabela ASCII. Abaixo pode-se ver um exemplo de matriz de letras:

```
/
**********************************************************************
*                          Alfabeto do display                       *
**********************************************************************/

const                                char                                font[252]
={          0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b000
00000,                          //'                '                          0
0b00000000,0b00000000,0b00000000,0b11110010,0b00000000,0b00000000,   //'!'   1
0b00000000,0b00000000,0b11100000,0b00000000,0b11100000,0b00000000,   //'"'   2
0b00000000,0b00101000,0b11111110,0b00101000,0b11111110,0b00101000,   //'#'   3
0b00000000,0b01001000,0b01010100,0b11111110,0b01010100,0b00100100,   //'$'   4
0b00000000,0b01000110,0b00100110,0b00010000,0b11001000,0b11000100,   //'%'   5
0b00000000,0b00001010,0b01000100,0b10101010,0b10010010,0b01101100,   //'&'   6
0b00000000,0b00000000,0b00000000,0b11000000,0b10100000,0b00000000,   //'''   7
0b00000000,0b00000000,0b10000010,0b01000100,0b00111000,0b00000000,   //'('   8
0b00000000,0b00000000,0b00111000,0b01000100,0b10000010,0b00000000,   //')'   9
0b00000000,0b00101000,0b00010000,0b01111100,0b00010000,0b00101000,   //'*'   10
0b00000000,0b00010000,0b00010000,0b01111100,0b00010000,0b00010000,   //'+'   11
0b00000000,0b00000000,0b00000000,0b00001100,0b00001010,0b00000000,   //','   12
0b00000000,0b00010000,0b00010000,0b00010000,0b00010000,0b00010000,   //'-'   13
0b00000000,0b00000000,0b00000110,0b00000110,0b00000000,0b00000000,   //'.'   14
0b00000000,0b01000000,0b00100000,0b00010000,0b00001000,0b00000100,   //'/'   15
```

```
0b00000000,0b01111100,0b10100010,0b10010010,0b10001010,0b01111100,   //'0'   16
0b00000000,0b00000000,0b00000010,0b11111110,0b01000010,0b00000000,   //'1'   17
0b00000000,0b01100010,0b10010010,0b10001010,0b10000110,0b01000010,   //'2'   18
0b00000000,0b10001100,0b11010010,0b10100010,0b10000010,0b10000100,   //'3'   19
0b00000000,0b00001000,0b11111110,0b01001000,0b00101000,0b00011000,   //'4'   20
0b00000000,0b10011100,0b10100010,0b10100010,0b10100010,0b11100100,   //'5'   21
0b00000000,0b00001100,0b10010010,0b10010010,0b01010010,0b00111100,   //'6'   22
0b00000000,0b11000000,0b10100000,0b10010000,0b10001110,0b10000000,   //'7'   23
0b00000000,0b01101100,0b10010010,0b10010010,0b10010010,0b01101100,   //'8'   24
0b00000000,0b01111000,0b10010100,0b10010010,0b10010010,0b01100000,   //'9'   25
0b00000000,0b00000000,0b01101100,0b01101100,0b00000000,0b00000000,   //':'   26
0b00000000,0b00000000,0b01101100,0b01101010,0b00000000,0b00000000,   //';'   27
0b00000000,0b00000000,0b10000010,0b01000100,0b00101000,0b00010000,   //'<'   28
0b00000000,0b00101000,0b00101000,0b00101000,0b00101000,0b00101000,   //'='   29
0b00000000,0b00010000,0b00101000,0b01000100,0b10000010,0b00000000,   //'>'   30
0b00000000,0b01100000,0b10010000,0b10001010,0b10000000,0b01000000,   //'?'   31
0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,   //'@'   32
0b00000000,0b01111110,0b10001000,0b10001000,0b10001000,0b01111110,   //'A'   33
0b00000000,0b01101100,0b10010010,0b10010010,0b10010010,0b11111110,   //'B'   34
0b00000000,0b01000100,0b10000010,0b10000010,0b10000010,0b01111100,   //'C'   35
0b00000000,0b00111000,0b01000100,0b10000010,0b10000010,0b11111110,   //'D'   36
0b00000000,0b10000010,0b10010010,0b10010010,0b10010010,0b11111110,   //'E'   37
0b00000000,0b10000000,0b10100000,0b10100000,0b10100000,0b11111110,   //'F'   38
0b00000000,0b01011110,0b10010010,0b10010010,0b10000010,0b01111100,   //'G'   39
0b00000000,0b11111110,0b00100000,0b00100000,0b00100000,0b11111110,   //'H'   40
0b00000000,0b00000000,0b10000010,0b11111110,0b10000010,0b00000000}; //'I' 41
```

6.1.2 Rotina de Recepção Serial de Dados

Encontra-se, abaixo, a função que realiza a recepção dos dados e os salva no buffer interno de noventa bytes:

```
/*****************************************************************
*              void recebeUsart() - Funçao responsavel por escrever na          *
*         memoria de programa  os dados recebidos na porta serial          *
*****************************************************************/
void recebeUsart(){
  char dummy,i;

  do{
    dummy = getc();
    if(dummy == '\r'){
      frase_inicial = TRUE;
      frase_completa = TRUE;
      wait = TRUE;
      if(indice_usart<20){
        for(i=indice_usart;i!=20;i++)text[i]=' ';
```

```
        indice_usart=i;
      }
      text[indice_usart] = 255;
      tamanho_string = indice_usart;
      indice_usart = 0;

    }
    else{
    text[indice_usart] = dummy;
    indice_usart++;

    #ifdef DEBUG
      printf("\r\nO seguinte caracter esta sendo gravado na memoria de programa: ");
    #endif
    putc(dummy);
    }
  }
  while((dummy != '\r') && (indice_usart < 90));
}
```

6.1.3 Sincronização Horizontal e *Scroll* de Texto

De acordo com o exposto no capítulo 4 sub-tópico 4.5, utilizou-se um par emissor-receptor de *led's* infravermelho para a sincronização horizontal do *display*.

Abaixo encontra-se uma ilustração do circuito emissor e receptor infravermelho (Sensor de Posição):

Figura - Sensor de sincronismo

O circuito tem o seguinte princípio de funcionamento: quando o emissor, localizado na parte estática do *display*, se alinhar com o receptor, localizado na parte rotativa do *display*, será iniciado um trecho de código no *firmware* responsável por desencadear a amostragem da mensagem. Pode-se ver, abaixo, o trecho acima mencionado:

```
/*****************************************************************
*                 trecho de código responsavel pelo alinhamento                 *
*                 horizontal das palavras  e scroll de mensagem                 *
*****************************************************************/
```

```
if (wait)wait_vb();
if(amostrar){
  // scrolling delay
  for (coluna=20;coluna>delta_janela;coluna--) delay(80);

  // desenho os caracteres na janela
  for (coluna=19;coluna>=0;coluna--) draw_char(text[coluna+delta_scroll],cor);

  // incremento k
  delta_janela--;
 output_b(0);
 output_c(0);
 output_d(0);
  // se chegou a 10 reinicio e caminho com a janela
  if (delta_janela == 0){
        delta_janela=20;
        delta_scroll++;
        if (text[18+delta_scroll] == 255)
        {
           delta_scroll = 0;
           cor++;
           if (cor==7) {
            cor=0;
            wait = FALSE;
           }
        }
  }
  amostrar = FALSE;
}
```

A função acima, conforme descrito em seu comentário inicial, também é responsável pelo *scroll* do texto. Como o motor gira no sentido horário, a placa controladora deve amostrar a imagem da direita para a esquerda. Caso o sentido esteja equivocado, a mensagem aparecerá espelhada. A figura 26 ilustra este conceito:

Figura - Varredura dos *led's*

Os caracteres são desenhados em uma janela de tamanho fixo. O tamanho da janela deve ser ajustado experimentalmente de modo a exibir o maior número de

colunas possível. Após a amostragem das colunas, caminha-se com a janela e obtém-se o efeito de *scroll* desejado. No caso do *firmware* desenvolvido neste projeto o tamanho da janela é armazenado pela variável delta_janela. Cada vez que o valor dessa variável atingir zero (delta_janela é inicializado com 20), a função draw_char() recebe um incremento em seu parâmetro (coluna+**delta_scroll**), o que proporciona o efeito de texto rolante. Abaixo segue uma ilustração deste procedimento.

Figura - *Scroll* de Texto

6.1.4 Amostragem dos Pixels

A função de amostragem dos *pixels*, conforme visto acima, é a draw_char(), cuja tarefa é carregar a coluna de *pixels* com as cores corretas, consultando o padrão correto mas matrizes de alfabeto (font[], font1[], font2[]), e esperar o tempo desejado para cada *pixel*. Abaixo segue seu código[2]:

```
/*****************************************************************
*                  void draw_char(char index, char cor) - Escreve uma letra no display     *
*****************************************************************/
void draw_char(char index,char cor){
    signed j;
    int16 k=0;
    int16 i=0;
    //putc(index);
    i=((int16)index-32)*6 ;
    k=i;
    //printf("i = %Lu\r\n",i);
    if((i>251)&&(i<503)) i -= 252;
    else if(i>503) i -= 504;
    //printf("i2 = %Lu\r\n",i);
    switch (cor)
    {
       case 0: // joga o padrao de bits na porta com a cor correta
             for (j=0;j<6;j++){
                if(k<251)output_d(font[j+i]);
```

2 Para maiores detalhes sobre as funções nela contidas, consultar o Apêndice A - Firmware

```
            else if((k>251)&&(k<503))output_d(font1[j+i]);
            else if(k>503)output_d(font2[j+i]);
            delay_ms(500);
            //delay(100);
            }
        break;
case 1:// joga o padrao de bits na porta com a cor correta
        for (j=0;j<6;j++){
            if(k<251)output_c(font[j+i]);
            else if((k>251)&&(k<503))output_c(font1[j+i]);
            else if(k>503)output_c(font2[j+i]);
            delay(100);
            }
        break;
case 2:// joga o padrao de bits na porta com a cor correta
        for (j=0;j<6;j++){
            if(k<251)output_b(font[j+i]);
            else if((k>251)&&(k<503))output_b(font1[j+i]);
            else if(k>503)output_b(font2[j+i]);
            delay(100);
            }
        break;
case 3:// joga o padrao de bits na porta com a cor correta
        for (j=0;j<6;j++){
            if(k<251){
                output_d(font[j+i]);
                output_c(font[i+j]);
            }
            else if((k>251)&&(k<503)){
                output_d(font1[j+i]);
                output_c(font1[i+j]);
            }
            else if(k>503){
                output_d(font2[j+i]);
                output_c(font2[i+j]);
            }
            delay(100);
            }
        break;
case 4:// joga o padrao de bits na porta com a cor correta
        for (j=0;j<6;j++){
            if(k<251){
                output_d(font[j+i]);
                output_b(font[i+j]);
            }
            else if((k>251)&&(k<503)){
                output_d(font1[j+i]);
                output_b(font1[i+j]);
            }
```

```
            else if(k>503){
               output_d(font2[j+i]);
               output_b(font2[i+j]);
            }
            delay(100);
            }
         break;
      case 5:// joga o padrao de bits na porta com a cor correta
         for (j=0;j<6;j++){
            if(k<251){
               output_b(font[j+i]);
               output_c(font[i+j]);
            }
            else if((k>251)&&(k<503)){
               output_b(font1[j+i]);
               output_c(font1[i+j]);
            }
            else if(k>503){
               output_b(font2[j+i]);
               output_c(font2[i+j]);
            }
            delay(100);
            }
         break;
      case 6:// joga o padrao de bits na porta
         for (j=0;j<6;j++/*j=5;j>=0;j--*/){
            if(k<251){
               output_d(font[j+i]);
               output_c(font[i+j]);
               output_b(font[i+j]);
            }
            else if((k>251)&&(k<503)){
               output_d(font1[j+i]);
               output_c(font1[i+j]);
               output_b(font1[i+j]);
            }
            else if(k>503){
               output_d(font2[j+i]);
               output_c(font2[i+j]);
               output_b(font2[i+j]);
            }
            delay(100);

               delay_ms(500);

            }
         break;
      }
}
```

6.1.5 Função Main

A função principal do firmware gravado no microcontrolador apenas percorre uma simples máquina de estados. Abaixo segue sua implementação:

```
/************************************************************************ *
*                                                                       *
*                                  void main()                          *
*                                                                       *
************************************************************************/
main(){

  char cor=6;//0;

  output_b(0);
  output_c(0);
  output_d(0);
  output_high(pin_d0);
  delay_ms(500);
  output_low(pin_d0);

  while( 1 )
  {
   if(frase_inicial){
     frase_inicial = FALSE;
     printf ("\r\nDigite o texto desejado:\r\n");
   }
   if(kbhit())recebeUsart();


   if (wait)wait_vb();
   if(amostrar){
      // scrolling delay
      for (coluna=20;coluna>delta_janela;coluna--) delay(80);

      // desenho os caracteres na janela
      for (coluna=19;coluna>=0;coluna--) draw_char(text[coluna+delta_scroll],cor);

      // incremento k
      delta_janela--;
     output_b(0);
     output_c(0);
     output_d(0);
      // se chegou a 10 reinicio e caminho com a janela
      if (delta_janela == 0){
            delta_janela=20;
```

```
            delta_scroll++;
            if (text[18+delta_scroll] == 255)
            {
               delta_scroll = 0;
               cor++;
               if (cor==7) {
                 cor=0;
                 wait = FALSE;
               }
            }
       }
       amostrar = FALSE;
    }


   }
}
```

6.2 Hardware Desenvolvido

A seguir serão apresentadas as placas controladora e de Alimentação e comunicação serial. Para maiores detalhes do hardware desenvolvido, consultar o Apêndice B – Esquema Elétrico.

6.2.1 Placa Controladora

A placa controladora consiste basicamente em um PIC16F877A, um capacitor de 1000 μF para filtrar os ruídos da alimentação por contatos deslizantes, 24 resistores – um para cada Led – e o conversor de nível HIN232 para a comunicação serial. Uma foto do circuito real pode se vista na figura 28:

Figura - Placa Controladora

6.2.2 Placa de Alimentação e Comunicação Serial

Foi desenvolvida uma pequena placa responsável pelas conexões da alimentação DC e cabo serial. Na figura 29 é apresentada uma foto real do circuito:

Figura - Placa de Alimentação e Comunicação

6.3 Sistema Mecânico Desenvolvido

6.3.1 Contato Deslizante

A alimentação do circuito, de acordo com o decidido no sub-tópico 4.3 – Tabela 4, foi realizada por meio de contatos deslizantes. Para este fim foram utilizados:

- Uma fita de cobre com cola condutora;
- Dois pares de carvão para os contatos necessários;
- Dois pares de molas;
- Um suporte físico para os carvões e molas;
- Uma estrutura de plástico de encaixe perfeito com o eixo do motor.

Figura - Fita de Cobre (Plano Vertical)

Figura - Peça Construída para o Contato Deslizante (Plano Vertical)

Figura - Peça com Cabos de Alimentação e Comunicação (Plano Vertical)

Figura - Peça com Cabos de Alimentação e Comunicação (Plano Horizontal)

Figura - Peça Revestida de Cobre (Plano Vertical)

Figura - Peça do Contato Deslizante (Montagem Final)

Figura - Contato Deslizante Montado com os Carvões

6.3.2 Suporte Mecânico

Para melhorar a estabilidade do sistema, foi projetado um suporte mecânico de forma a aumentar a área de contato entre o sistema e a superfície onde está localizado. Assim, as vibrações mecânicas são uniformemente distribuídas e, conseqüentemente, o a estabilidade do sistema é melhorada.

Para reduzir as vibrações mecânicas foram inseridos batentes. Estes são responsáveis por centralizar e balizar o motor. Abaixo, seguem fotos do sistema final.

Figura - Superfície estabilizadora

Figura - Batentes de centralização

7 Conclusão

A seguir se fará uma análise dos requisitos de projeto alcançados pela versão atual do sistema.

7.1 Matriz de Requisitos Contemplados

Abaixo se encontra a matriz de requisitos contemplados. Os itens que estão em verde foram contemplados satisfatoriamente por esta versão de sistema, os itens em vermelho são contemplados somente pela atualização do sistema proposta no capítulo seguinte.

Tabela - Matriz de Requisitos Contemplados

Descrição	Obrigatório	Recomendável	Prioridade
Mostrar texto	✓		1
Interconectividade com computadores pessoais	✓		1
Mostrar mensagens predeterminadas	✓		1
Alimentação do circuito independente de baterias	✓		1
Mostrar imagens		✓	2
Mostrar mensagens personalizadas		✓	2
Software de interface		✓	3
Transferencia de dados *wireless*		✓	3

7.2 Conclusões

A maior barreira enfrentada na realização do projeto foi a construção de um sistema mecânico estável capaz de transmitir dados e energia à placa controladora. Após chegar à estrutura mecânica final, constatou-se que se trata de um modelo que pode ser utilizado na construção de displays de quaisquer medidas.

O sistema cumpre com todos os requisitos básicos – Prioridade 1- e intermediários – Prioridade 2 – esperados para um *display* de propaganda, mas peca no que tange resolução vertical. Como se trata de um *display* voltado para a apresentação de mensagens, o protótipo cumpre, satisfatoriamente, com o seu propósito.

Nas figuras 39 e 40 são apresentadas fotos do *display* funcionando:

Figura - Display de Persistência de Visão 1

Figura - Display de Persistência de Visão 2

Visando obter um *display* voltado para a apresentação de imagens e textos, foi desenvolvida uma atualização para o sistema descrita no capítulo seguinte.

8 Melhorias do Sistema

Atualmente, o sistema está passando por um processo de atualização. Após verificado o sucesso na construção do *display* com 8 leds de resolução vertical, decidiu-se pela implementação de um *display* com resolução vertical de 32 leds.

Para este propósito, são necessários novos *layouts* de *hardware* bem como um *software* para a comunicação com o display.

A seguir, serão explicados, de forma sucinta, as melhorias propostas para o sistema.

8.1 Projeto do Hardware

Para armazenar as informações enviadas pelo *software*, pode-se utilizar uma memória EEPROM externa.

Para aumentar a resolução vertical do *display*, pode-se utilizar registradores de deslocamento. Dessa forma, pode-se aumentar a resolução sem a preocupação da limitação de pinos do microcontrolador, pois os registradores de deslocamento são conectados em cascata e carregados serialmente.

8.1.1 Escolha dos Registradores de Deslocamento

Como se optou pelo método aditivo de geração de cor[3] e por uma resolução vertical de 32 led's RGB, seriam necessárias 97 portas digitais para o controle da placa de *led's* (cada *led* possui 4 pinos - 3 RGB e o comum). Obviamente, nenhum

3 Para maiores informações ver capítulo 2 - sub-tópico 2.1.3.1

microcontrolador comercial possui este número de entradas e saídas digitais, então a solução proposta é utilizar registradores de deslocamento para carregar, de forma serializada, a coluna de *pixels*. Dessa forma o controle de cor será do tipo *on-off*, ou seja, obter-se-ão, para cada *pixel*, apenas 8 cores possíveis: Vermelho, Verde, Azul, Violeta, Amarelo, Azul Claro, Branco e Preto.

Para realizar esta tarefa, pode-se optar pelo registrador de deslocamento fabricado pela Fairchild Semiconductor TM ®, o MM74HC595 [6]. Este componente foi escolhido por três principais motivos:

- Possui *latches* de saída. Esta característica é muito útil para este projeto, pois desta forma não é necessário atualizar a informação de um *pixel* que não será alterado.
- É um registrador de deslocamento de 8 bits. Isto ajuda, pois necessita-se de um número inteiro de componentes (12) para cobrir os 32 *led's* de saída.
- Freqüência de chaveamento de até 30 MHz.

8.1.2 Escolha da Memória Externa

Como aplicação em questão não necessita de capacidades excessivas de memória de dados, pode-se optar por uma memória de fácil obtenção e operação, preocupando-se, unicamente, com protocolo utilizado para a comunicação com mesma.

A memória escolhida foi a 24C256 [7] da Fairchild Semiconductor TM ®, que possui 256K bits de memória e utiliza o protocolo I2C para comunicação.

8.2 Projeto do Firmware

Visando aperfeiçoar o sistema, pode-se desenvolver um *firmware* baseado em interrupções para o microcontrolador. Desa forma, evita-se a perda de processamento ao esperar o pulso de sincronismo (sensor) ou ao receber dados da porta serial.

Para a comunicação entre o *software* e o *hardware*, deve-se desenvolver um protocolo de comunicação, baseado em máquinas de estado, por exemplo, capaz de otimizar a trasferência de informações.

8.3 Sistema Mecânico

Para melhorar a estabilidade do sistema, o cicuito pode ser montado sobre uma estrutura fisicamente separada do eixo do motor. A transmissão da rotação pode ser realizada via polias, possibilitando, inclusive, o uso de reduções para aumentar sua velocidade angular. Este sistema possibilita, também, o uso de amortecedores mecânicos mais eficazes, reduzindo assim as vibrações mecânicas geradas pelo sistema.

Apêndice A – Firmware

```
/
*********************************************************************
*                                Firmware – Listagem Completa                                *
*********************************************************************/

#include <16f877.h>
#use delay(clock = 10000000)
#fuses HS, NOWDT, PUT, NOLVP
#use rs232 (baud = 9600, xmit = PIN_E2, rcv = PIN_E1)

#include <string.h>

/*#use fast_io(A)
#use fast_io(B)
#use fast_io(C)
#use fast_io(D)*/
// v1.0
//#define DEBUG
const char font[252]
={                0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b000
00000, //' ' 0
0b00000000,0b00000000,0b00000000,0b11110010,0b00000000,0b00000000, //'!' 1
0b00000000,0b00000000,0b11100000,0b00000000,0b11100000,0b00000000, //'"' 2
0b00000000,0b00101000,0b11111110,0b00101000,0b11111110,0b00101000, //'#' 3
0b00000000,0b01001000,0b01010100,0b11111110,0b01010100,0b00100100, //'$' 4
0b00000000,0b01000110,0b00100110,0b00010000,0b11001000,0b11000100, //'%' 5
0b00000000,0b00001010,0b01000100,0b10101010,0b10010010,0b01101100, //'&' 6
0b00000000,0b00000000,0b00000000,0b11000000,0b10100000,0b00000000, //''' 7
0b00000000,0b00000000,0b10000010,0b01000100,0b00111000,0b00000000, //'(' 8
0b00000000,0b00000000,0b00111000,0b01000100,0b10000010,0b00000000, //')' 9
0b00000000,0b00101000,0b00010000,0b01111100,0b00010000,0b00101000, //'*' 10
0b00000000,0b00010000,0b00010000,0b01111100,0b00010000,0b00010000, //'+' 11
```

```
0b00000000,0b00000000,0b00000000,0b00001100,0b00001010,0b00000000, //',' 12
0b00000000,0b00010000,0b00010000,0b00010000,0b00010000,0b00010000, //'-' 13
0b00000000,0b00000000,0b00000110,0b00000110,0b00000000,0b00000000, //'.' 14
0b00000000,0b01000000,0b00100000,0b00010000,0b00001000,0b00000100, //'/' 15
0b00000000,0b01111100,0b10100010,0b10010010,0b10001010,0b01111100, //'0' 16
0b00000000,0b00000000,0b00000010,0b11111110,0b01000010,0b00000000, //'1' 17
0b00000000,0b01100010,0b10010010,0b10001010,0b10000110,0b01000010, //'2' 18
0b00000000,0b10001100,0b11010010,0b10100010,0b10000010,0b10000100, //'3' 19
0b00000000,0b00001000,0b11111110,0b01001000,0b00101000,0b00011000, //'4' 20
0b00000000,0b10011100,0b10100010,0b10100010,0b10100010,0b11100100, //'5' 21
0b00000000,0b00001100,0b10010010,0b10010010,0b01010010,0b00111100, //'6' 22
0b00000000,0b11000000,0b10100000,0b10010000,0b10001110,0b10000000, //'7' 23
0b00000000,0b01101100,0b10010010,0b10010010,0b10010010,0b01101100, //'8' 24
0b00000000,0b01111000,0b10010100,0b10010010,0b10010010,0b01100000, //'9' 25
0b00000000,0b00000000,0b01101100,0b01101100,0b00000000,0b00000000, //':' 26
0b00000000,0b00000000,0b01101100,0b01101010,0b00000000,0b00000000, //';' 27
0b00000000,0b00000000,0b10000010,0b01000100,0b00101000,0b00010000, //'<' 28
0b00000000,0b00101000,0b00101000,0b00101000,0b00101000,0b00101000, //'=' 29
0b00000000,0b00010000,0b00101000,0b01000100,0b10000010,0b00000000, //'>' 30
0b00000000,0b01100000,0b10010000,0b10001010,0b10000000,0b01000000, //'?' 31
0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, //'@' 32
0b00000000,0b01111110,0b10001000,0b10001000,0b10001000,0b01111110, //'A' 33
0b00000000,0b01101100,0b10010010,0b10010010,0b10010010,0b11111110, //'B' 34
0b00000000,0b01000100,0b10000010,0b10000010,0b10000010,0b01111100, //'C' 35
0b00000000,0b00111000,0b01000100,0b10000010,0b10000010,0b11111110, //'D' 36
0b00000000,0b10000010,0b10010010,0b10010010,0b10010010,0b11111110, //'E' 37
0b00000000,0b10000000,0b10100000,0b10100000,0b10100000,0b11111110, //'F' 38
0b00000000,0b01011110,0b10010010,0b10010010,0b10000010,0b01111100, //'G' 39
0b00000000,0b11111110,0b00100000,0b00100000,0b00100000,0b11111110, //'H' 40
0b00000000,0b00000000,0b10000010,0b11111110,0b10000010,0b00000000}; //'I' 41

const char font1[252]
={                  0b00000000,0b10000000,0b11111100,0b10000010,0b00000010,0b0000
0100, //'J' 42
0b00000000,0b10000010,0b01000100,0b00101000,0b00010010,0b11111110, //'K' 43
0b00000000,0b00000010,0b00000010,0b00000010,0b00000010,0b11111110, //'L' 44
0b00000000,0b11111110,0b01000000,0b00100000,0b01000000,0b11111110, //'M' 45
0b00000000,0b11111110,0b00001000,0b00010000,0b00100000,0b11111110, //'N' 46
0b00000000,0b01111100,0b10000010,0b10000010,0b10000010,0b01111100, //'O' 47
0b00000000,0b01100000,0b10010000,0b10010000,0b10010000,0b11111110, //'P' 48
0b00000000,0b01111010,0b10000100,0b10001010,0b10000010,0b01111100, //'Q' 49
0b00000000,0b01100010,0b10010100,0b10011000,0b10010000,0b11111110, //'R' 50
0b00000000,0b10001100,0b10010010,0b10010010,0b10010010,0b01100010, //'S' 51
0b00000000,0b10000000,0b10000000,0b11111110,0b10000000,0b10000000, //'T' 52
0b00000000,0b11111100,0b00000010,0b00000010,0b00000010,0b11111100, //'U' 53
0b00000000,0b11111000,0b00000100,0b00000010,0b00000100,0b11111000, //'V' 54
0b00000000,0b11111100,0b00000010,0b00011100,0b00000010,0b11111100, //'W' 55
0b00000000,0b11000110,0b00101000,0b00010000,0b00101000,0b11000110, //'X' 56
0b00000000,0b11100000,0b00010000,0b00001110,0b00010000,0b11100000, //'Y' 57
0b00000000,0b11000010,0b10100010,0b10010010,0b10001010,0b10000110, //'Z' 58
0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, //'[' 59
0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, //'\' 60
```

```
0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, //']' 61
0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, //'^' 62
0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, //'_' 63
0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000, //'`' 64
0b00000000,0b00011110,0b00101010,0b00101010,0b00101010,0b00000100, //'a' 65
0b00000000,0b00011100,0b00100010,0b00100010,0b00010010,0b11111110, //'b' 66
0b00000000,0b00000100,0b00100010,0b00100010,0b00100010,0b00011100, //'c' 67
0b00000000,0b11111110,0b00010010,0b00100010,0b00100010,0b00011100, //'d' 68
0b00000000,0b00011000,0b00101010,0b00101010,0b00101010,0b00011100, //'e' 69
0b00000000,0b01000000,0b10000000,0b10010000,0b01111110,0b00010000, //'f' 70
0b00000000,0b00111110,0b00100101,0b00100101,0b00100101,0b00011000, //'g' 71
0b00000000,0b00011110,0b00100000,0b00100000,0b00010000,0b11111110, //'h' 72
0b00000000,0b00000000,0b00000010,0b10111110,0b00100010,0b00000000, //'i' 73
0b00000000,0b00000000,0b01011110,0b00010001,0b00000001,0b00000010, //'j' 74
0b00000000,0b00100010,0b00010100,0b00001000,0b11111110,0b00000000, //'k' 75
0b00000000,0b00000000,0b00000010,0b11111110,0b10000010,0b00000000, //'l' 76
0b00000000,0b00011110,0b00100000,0b00011000,0b00100000,0b00111110, //'m' 77
0b00000000,0b00011110,0b00100000,0b00100000,0b00010000,0b00111110, //'n' 78
0b00000000,0b00011100,0b00100010,0b00100010,0b00100010,0b00011100, //'o' 79
0b00000000,0b00011000,0b00100100,0b00100100,0b00100100,0b00111111, //'p' 80
0b00000000,0b00111111,0b00010100,0b00100100,0b00100100,0b00011000, //'q' 81
0b00000000,0b00010000,0b00100000,0b00100000,0b00010000,0b00111110, //'r' 82
0b00000000,0b00000100,0b00101010,0b00101010,0b00101010,0b00010010}; //'s' 83

const char font2[42]
={                  0b00000000,0b00000100,0b00000010,0b00100010,0b11111100,0b0010
0000, //'t' 84
0b00000000,0b00111110,0b00000100,0b00000010,0b00000010,0b00111100, //'u' 85
0b00000000,0b00111000,0b00000100,0b00000010,0b00000100,0b00111000, //'v' 86
0b00000000,0b00111100,0b00000010,0b00001100,0b00000010,0b00111100, //'w' 87
0b00000000,0b00100010,0b00010100,0b00001000,0b00010100,0b00100010, //'x' 88
0b00000000,0b00111110,0b00000101,0b00000101,0b00000101,0b00111000, //'y' 89
0b00000000,0b00100010,0b00110010,0b00101010,0b00100110,0b00100010};//'z' 90

int indice_usart = 0;
signed tamanho_string;
//char text[90];
boolean amostrar = FALSE;
boolean frase_inicial = TRUE;
boolean frase_completa = TRUE;//FALSE;
boolean wait = TRUE;//FALSE;
 signed delta_janela=20;
signed coluna=0;
 signed delta_scroll=0;

void recebeUsart(){
   char dummy,i;

   do{
      dummy = getc();
      if(dummy == '\r'){
         frase_inicial = TRUE;
```

```
         frase_completa = TRUE;
         wait = TRUE;
         if(indice_usart<20){
            for(i=indice_usart;i!=20;i++)text[i]=' ';
            indice_usart=i;
         }
         text[indice_usart] = 255;
         tamanho_string = indice_usart;
         indice_usart = 0;

      }
      else{
      text[indice_usart] = dummy;
      indice_usart++;

      #ifdef DEBUG
         printf("\r\nO seguinte caracter esta sendo gravado na memoria de programa: ");
      #endif
      putc(dummy);
      }
   }
   while((dummy != '\r') && (indice_usart < 90));
}


void delay( char i )
{
    char j;
    for (j=0;j<i;j++);


}


void wait_vb(){
        char i;
        if ((input(pin_a4))){
         i = (input(pin_a4));
         while (i == (input(pin_a4)));
         if(frase_completa)
            amostrar = TRUE;
        }

}

void draw_char(char index,char cor){
        signed j;
        int16 k=0;
        int16 i=0;
        //putc(index);
        i=((int16)index-32)*6 ;
        k=i;
```

```
//printf("i = %Lu\r\n",i);
if((i>251)&&(i<503)) i -= 252;
else if(i>503) i -= 504;
//printf("i2 = %Lu\r\n",i);
switch (cor)
{
    case 0: // joga o padrao de bits na porta com a cor correta
            for (j=0;j<6;j++){
               if(k<251)output_d(font[j+i]);
               else if((k>251)&&(k<503))output_d(font1[j+i]);
               else if(k>503)output_d(font2[j+i]);
               delay_ms(500);
               //delay(100);
               }
            break;
    case 1:// joga o padrao de bits na porta com a cor correta
            for (j=0;j<6;j++){
               if(k<251)output_c(font[j+i]);
               else if((k>251)&&(k<503))output_c(font1[j+i]);
               else if(k>503)output_c(font2[j+i]);
               delay(100);
               }
            break;
    case 2:// joga o padrao de bits na porta com a cor correta
            for (j=0;j<6;j++){
               if(k<251)output_b(font[j+i]);
               else if((k>251)&&(k<503))output_b(font1[j+i]);
               else if(k>503)output_b(font2[j+i]);
               delay(100);
               }
            break;
    case 3:// joga o padrao de bits na porta com a cor correta
            for (j=0;j<6;j++){
               if(k<251){
                  output_d(font[j+i]);
                  output_c(font[i+j]);
               }
               else if((k>251)&&(k<503)){
                  output_d(font1[j+i]);
                  output_c(font1[i+j]);
               }
               else if(k>503){
                  output_d(font2[j+i]);
                  output_c(font2[i+j]);
               }
               delay(100);
               }
            break;
    case 4:// joga o padrao de bits na porta com a cor correta
            for (j=0;j<6;j++){
               if(k<251){
                  output_d(font[j+i]);
```

```
            output_b(font[i+j]);
          }
          else if((k>251)&&(k<503)){
            output_d(font1[j+i]);
            output_b(font1[i+j]);
          }
          else if(k>503){
            output_d(font2[j+i]);
            output_b(font2[i+j]);
          }
          delay(100);
          }
        break;
case 5:// joga o padrao de bits na porta com a cor correta
        for (j=0;j<6;j++){
          if(k<251){
            output_b(font[j+i]);
            output_c(font[i+j]);
          }
          else if((k>251)&&(k<503)){
            output_b(font1[j+i]);
            output_c(font1[i+j]);
          }
          else if(k>503){
            output_b(font2[j+i]);
            output_c(font2[i+j]);
          }
          delay(100);
          }
        break;
case 6:// joga o padrao de bits na porta
        for (j=0;j<6;j++/*j=5;j>=0;j--*/){
          if(k<251){
            output_d(font[j+i]);
            output_c(font[i+j]);
            output_b(font[i+j]);
          }
          else if((k>251)&&(k<503)){
            output_d(font1[j+i]);
            output_c(font1[i+j]);
            output_b(font1[i+j]);
          }
          else if(k>503){
            output_d(font2[j+i]);
            output_c(font2[i+j]);
            output_b(font2[i+j]);
          }
          delay(100);

            delay_ms(500);

          }
```

```
                break;
            }
            // apaga os leds
            /*output_b(0b00000000);
            output_c(0b00000000);
            output_d(0b00000000);*/
}

main(){

  char cor=6;//0;

  output_b(0);
  output_c(0);
  output_d(0);
  output_high(pin_d0);
  delay_ms(500);
  output_low(pin_d0);

  while( 1 )
  {
   if(frase_inicial){
     frase_inicial = FALSE;
     printf ("\r\nDigite o texto desejado:\r\n");
   }
   if(kbhit())recebeUsart();


   if (wait)wait_vb();
   if(amostrar){
      // scrolling delay
      for (coluna=20;coluna>delta_janela;coluna--) delay(80);

      // desenho os caracteres na janela
      for (coluna=19;coluna>=0;coluna--) draw_char(text[coluna+delta_scroll],cor);
      // incremento k
      delta_janela--;
     output_b(0);
     output_c(0);
     output_d(0);
      // se chegou a 10 reinicio e caminho com a janela
      if (delta_janela == 0){
            delta_janela=20;
            delta_scroll++;
            if (text[18+delta_scroll] == 255)
            {
               delta_scroll = 0;
               cor++;
               if (cor==7) {
                 cor=0;
                 wait = FALSE;
               }
```

```
            }
        }
        amostrar = FALSE;
    }
  }
}
```

Apêndice B – Esquema Elétrico