CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Daniel Henriques Cézar Miranda Soares

EMBARQUE DE JOGO *PONG* PARA MICROCONTROLADOR PIC18F4550 COM *DISPLAY* NOKIA 5110 COM CONTROLADOR PCD8544

Timóteo

2019

Daniel Henriques Cézar Miranda Soares

EMBARQUE DO JOGO PONG PARA MICROCONTROLADOR PIC18F4550 COM DISPLAY NOKIA 5110 COM CONTROLADOR PCD8544

Monografia apresentada ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais para a obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo 17 de junho de 2019:

Prof. Dr. Elder de Oliveira Rodrigues
Orientador

Prof. Me. Adilson Mendes Ricardo
Professor Convidado

Prof. Dr. Rodrigo Gaiba de Oliveira
Professor Convidado

Timóteo
2019

Dedico a meus pais por me apoiarem em meus erros e acertos, que me ajudaram a melhorar em todos os meus aspectos, a meus professores que contribuíram grandemente em meus estudos, sendo com didática ou paciência, e a meus queridos amigos que estiveram sempre presentes quando precisei.

Agradecimentos

Agradeço novamente aos meus pais, que sempre incentivaram meus estudos. Agradeço muito a meu orientador que com paciência e confiança, tem me acompanhado e instruído na construção não apenas deste trabalho, mas de diversos projetos.

”Alguns homens veem as coisas como são, e dizem ‘Por quê?’ Eu sonho com as coisas que nunca foram e digo ‘Por que não?’ ”.

Geroge Bernard Shaw

Resumo

Este trabalho se caracteriza por criar uma versão do conjunto *hardware e software* do jogo *Pong*, com o uso de um *display* LCD (cristal liquido) *nokia* 5110, com controlador PCD8544 e com o jogo embarcado em um microcontrolador PIC18F4550. A comunicação entre o microcontrolador e o *display* é feita com o uso de um protocolo simples, serial, usado para mandar comandos e dados para o *display*, através de uma biblioteca de controle implementada pelo autor. O circuito desenvolvido utiliza um cristal oscilador gerador de *clock*, resistores, capacitores e botões para interface com jogador. O jogo é recriado em cima da nova plataforma, sendo feito uso de técnicas simples de manipulação gráfica, como o desenho *pixel* a *pixel* e re-cálculo de posição de objetos na tela. Como resultado, a tela gráfica e a velocidade da bola proporcionaram o resultado esperado para o ambiente que o jogo exige e sua originalidade, numa versão em menor escala.

Palavras-chave: *Pong*, PIC18F4550, PCD8544, Nokia 5110.

Abstract

This work is characterized by creating a version of the hardware and software set of the Pong game with the use of a 5110 LCD screen, with PCD8544 controller with the game embeded on a PIC18F4550 microcontroller. The communication between the microcontroller and the display is made using a simple serial protocol, used to send commands and data to the display through a control library implemented by the author. The developed circuit uses an oscillating crystal to generate clock, resistors, capacitors and buttons for player interface. The game is recreated on top of the new platform using simple graphic manipulation techniques, such as pixel by pixel drawing and re-calculating the position of objects on the screen. As a result, the graphical display and ball speed provided the expected result for the environment the game demands and its originality in a smaller scale version.

Keywords: Pong, PIC18F4550, PCD8544, Nokia 5110.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Sumário

1 Introdução

"Em algum lugar algo incrível está esperando para ser descoberto.".
Carl Sagan

De acordo com The Centre for Computing History, Cambridge (2010), o *Pong* foi o primeiro jogo de vídeo game comercial da história. Lançado em 1972, para fliperamas, é baseado em uma simples representação bidimensional de uma partida de tênis em uma tela preto e branco. Foi produzido pela então famosa empresa *Atari* e desenvolvido por Nolan Bushnell e Ted Dabney.

Figura 1: Controlador da primeira versão comercial do Pong.

Fonte: The Centre for Computing History, Cambridge (2010).

Ainda de acordo com The Centre for Computing History, Cambridge (2010), a versão de *Pong* que foi lançada no mercado, em 1975, foi desenhada e produzida em um único CI (Componente Integrado) que, por mais que precise de transistores, cristais geradores de frequência e fonte para funcionar, não requer outro CI adicional, assim como a maioria dos Microcontroladores.

Os microcontroladores são computadores em um só chip *(SoC, System on a Chip)*, que contem processador, memória e periféricos de entrada e saída, sendo utilizados em diversas aplicações, de acordo com Nobuo Oki, Suely Cunha Amaro Mantovan (2013). Podem ser usados em sistemas embarcados, automação, controle de processos, e várias outras aplicações por ser um produto muito versátil.

Os *displays* ou monitores são imprescindíveis para o uso ou criação de um jogo de videogame, e para o *Pong* não seria diferente. Por ter sido o primeiro videogame doméstico, e monitores ou *displays* não serem acessíveis ao público doméstico, a escolha da Atari foi lançar o jogo com uma saída de vídeo para televisões com tubo de raios catódicos. Os *displays* surgiram em meados da década de 40 e, inicialmente, eram apenas luzes indicadoras que mudavam de estado (acesa ou apagada), quando o processador acessava uma região da memória ou executava alguma instrução específica. Com o passar dos anos, os monitores foram adaptados para se comunicar com o computador e exibir informações, surgindo assim os primeiros monitores gráficos, que eram chamados na época de terminais. Em 1980 surgiram os primeiros monitores de cristal líquido, com taxas de atualização baixas (quantas vezes a imagem é redesenhada na tela) de acordo com (Benj Edwards, 2010).

Para que seja possível o uso de um microcontrolador, seja ele para controlar um processo ou executar um jogo em um *displays*, ele primeiramente precisa ser programado e gravado. Este processo consistem em codificar, em alguma linguagem de programação compatível com o microcontrolador desejado, e compilar o código em um compilador específico para aquele SoC. De acordo com Ivan L. M. Ricarte (2003), um compilador é um programa ou sistema que traduz outro programa descrito em uma linguagem de alto nível, para uma equivalente de baixo nível ou código de máquina, para um processador executar. Em geral, um compilador não produz diretamente o código de máquina mas sim um programa em linguagem simbólica *Assembly*, semanticamente equivalente ao mesmo em alto nível. O programa em linguagem simbólica é então traduzido para código de máquina através de montadores (*Assembler*).

Diante do que foi exposto, este trabalho busca desenvolver uma estrutura de *hardware* e *software* que se assemelha ao jogo *Pong* em escala reduzida, mantendo a maior fidelidade ao jogo original possível. Para isto, componentes específicos serão estudados e analisados para melhor traçar os objetivos a serem alcançados.

1.1 Justificativa

Os vários trabalhos encontrados na literatura, que fazem uso de *Displays* LCD com microcontrolador PIC18F4550 para criar uma versão do jogo *Pong*, utilizam *Displays* de 16x2 caracteres, sendo que cada caractere possui 8x5 *pixels* de dimensão. Porém, estes *Displays* tem baixa taxa de atualização e devido a sua característica de controle, criam uma dificuldade de manipulação *pixel* a *pixel* para gerar uma imagem, com isto a versão do jogo foge muito a ideia original. Logo, o interessante seria a utilização de *Displays* LCD com maior número de *pixels* e que seja mais fácil a manipulação direta dos *pixels*.

Com a escolha de um monitor LCD de melhor resolução, surge outro problema: a falta de recursos para implementação do mesmo ou o alto valor a ser pago pelos recursos disponíveis. Os compiladores e linguagens mais utilizados para PIC são o C18, XC8 e MikroC, sendo os dois primeiros gratuito e produzidos pela mesma empresa que produz o PIC, porém o C18 está bem consolidado enquanto o XC8 é recém lançado. Já o MikroC é pago, porém contem vários recursos e bibliotecas de compilação e programação embutidos, sendo o preço pesquisado no momento de realização deste trabalho 249 dólares. O MikroC fornece bibliotecas para controle de LCD's de maior resolução, porém é um recurso pago de alto valor, o que torna interessante o uso do compilador C18, já que o mesmo é gratuito, entretanto não dispõe de bibliotecas prontas para controle de monitor LCD e quase todo o controle do *Display* precisa ser implementado.

Neste caso é necessária a implementação de todos os protocolos e padrões de comunicação usados pelo dispositivo escravo, no caso o monitor LCD, o modelo escolhido é o *Display Nokia 5110* de 84 por 48 *pixels*, que tem internamente um controlador PCD8544, que deve comunicar com a biblioteca produzida.

Como exemplo de trabalhos relacionados pode-se citar o trabalho de Rickard Gunee (2003), que trata do jogo *Pong* usando um gerador de sinal analógico e uma televisão de tubo. Outro trabalho, a produção de Dysfunctional Technologies, Inc. (2012) detalha muito bem a implementação e usa uma saída de vídeo RGB(*Red Green Blue*) e outra analógica com uma televisão CRT.

Não existindo uma versão de *Pong* para *Display Nokia 5110* em PIC18F4550, com uso do compilador gratuito C18 para controlar esse tipo de monitor LCD, a justificativa deste trabalho é a construção de uma versão em menor escala do jogo *Pong* com uso de *display* LCD e uma biblioteca com os protocolos de comunicação necessários para que seja possível o uso do *Display* Nokia 5110 com o PIC e com o compilador gratuito C18.

1.2 Objetivos

O objetivo geral proposto é criar uma versão da estrutura *hardware* e *software* do jogo *Pong* com o uso do microcontrolador PIC 18F450, do *Display Nokia 5110* e do compilador C18. Para alcançar este objetivo é necessário primeiro construir uma biblioteca completa para controle do *Display Nokia 5110* com o PIC18F4550. Para teste será feita a interação do jogador através de quatro botões, dois para cada jogador, algo mais simples, ao contrário dos potenciômetros do jogo original. Para que estes objetivos sejam alcançados é necessário:

1. Implementar uma versão da Interface de Periféricos Serial, SPI entre o PIC18F4550 e o PCD8544;
2. Implementar os protocolos necessários para controlar o *display* através da interface SPI criada no item 1;
3. Definir os métodos básicos de desenho em tela e lógicas de jogo a serem utilizados;
4. Implementar o desenho de objetos na tela, pontuação e controle dos jogadores;
5. Criar um protótipo de circuito para testes dos itens anteriores;
6. Testar o protótipo e avaliar resultados.

1.3 Referencial Teórico

1.3.1 Microcontroladores

De acordo com Souza (2008), microcontroladores são SoC's que consistem basicamente de uma CPU (*Central Process Unit*) que executa as instruções, memória de acesso dinâmico para uso do processador, memória programável apagável somente de leitura, também chamada de EPROM (*Erasable Programmable Read-Only Memory*), temporizadores, conversores analógico-digitais e barramentos de E/S (entrada e saída).

Uma CPU é um componente que consegue interpretar instruções binárias e executá-las de forma estruturada, ou seja, seguindo um fluxo de execução. A CPU da maioria dos microcontroladores usa instruções de 16 *bits* e dados de 8, significa que cada pedaço de informação na memória pode ser um número binário de 00000000 até 11111111, que traduz em um número decimal de 0 até 255. No caso do PIC da família 18F, as instruções são divididas em *opcode* (código de operação) e literal, ou *opcode* e registrador ou , ainda, *opcode* e endereço de memória e finalmente pode ser somente o opcode para instruções mais simples. As instruções podem ser operações matemáticas, tomada de decisões (mudança no fluxo de execução), leitura de entrada, escrita em saída, conversões analógico para, digital dentre outras, no total são 76 instruções no 18F4550.

A memória de acesso dinâmico é utilizada pelo processador para armazenar e ler dados durante a execução do programa. É mais rápida que a memória onde o programa está, porém só armazena a informação quando está ligada, perdendo os dados quando o microcontrolador é desligado.

A memória EPROM armazena o programa que é executado pela CPU, não sendo possível sobrescrever durante a execução, somente leitura. Para gravar outro programa é necessário utilizar de gravadores especiais como o *PicKit2*, *PicKit3*, entre outros, que conseguem gravar os microcontroladores da família PIC.

Os temporizadores são contadores especiais que conseguem ser utilizados para contar de 0 a 255, por exemplo um de 8 *bits*. O temporizador leva um valor em milissegundos a cada vez que é incrementado, isto pode ser utilizado para contar o tempo que se passou desde que foi iniciado, e eles funcionam em paralelo a execução do programa principal, o que possibilita boa precisão.

Os conversores analógico-digital conseguem transformar um valor analógico de tensão em um valor binário, por isto seu nome. Os valores de entrada analógica normalmente são entre 0 a 5 Volts e de saída de 0 a 11 1111 1111 ou 0 a 1023 em decimal, que pode ser usado para ler um sensor ou uma resistência elétrica.

Por ultimo, existem, dento do microcontroladores, os barramentos de entrada e saída, sendo que estes consistem em entradas e saídas digitais. Estas portas podem estar em estado alto ou baixo, que traduz para 1 ou 0 em binário ou 5 Volts ou 0 Volts na porta física e são muito úteis para comunicação com o mundo fora do chip. Todos estes componentes descritos acima estão dentro de um único chip.

1.3.2 Monitores

De acordo com Benj Edwards (2010), no início da era da computação as máquinas utilizavam lâmpadas para indicar algum estado da memória ou do CPU e isso era o suficiente para o uso. Após o surgimento de impressoras, foi comum o uso para gerar as mudanças que aconteciam na máquina ou impressão de resultados matemáticos. Depois do surgimento dos primeiros *displays* ou monitores a interação com máquina ficou mais simples e facilitou o uso.

Os primeiros *displays* eram televisões de tubo adaptados para receber um sinal analógico de vídeo, que era gerado por um circuito a parte nas máquinas. Com o passar dos anos e surgimento dos *displays* LCD, a imagem é enviada de forma digital aos monitores que transformam os dados em imagem. Estes *displays* digitais contém embutido um controlador digital, que recebe os dados ou comandos e gera a imagem na tela. A comunicação é feita entre o computador e o controlador embutido ou entre o microcontrolador e o controlador embutido no caso deste trabalho.

1.3.3 Compiladores

De acordo com Ivan L. M. Ricarte (2003), as linguagens de programação são formas de simplificar a escrita de um programa de computador, ou seja, de um código a ser executado. A CPU só consegue executar instruções binárias, o que significa que gerar um programa sem o uso de uma linguagem de programação necessitaria a codificação de instrução por instrução de forma binária, sem mnemônicos. Com a utilização de uma linguagem de alto nível é possível transcrever para baixo nível com o uso de um compilador. O trabalho dele é traduzir um código escrito em alto nível para um em linguagem de máquina.

2 Revisão Bibliográfica

"Jesus faz milagres e dá, o homem não faz milagres e vende.".
Karl Marx

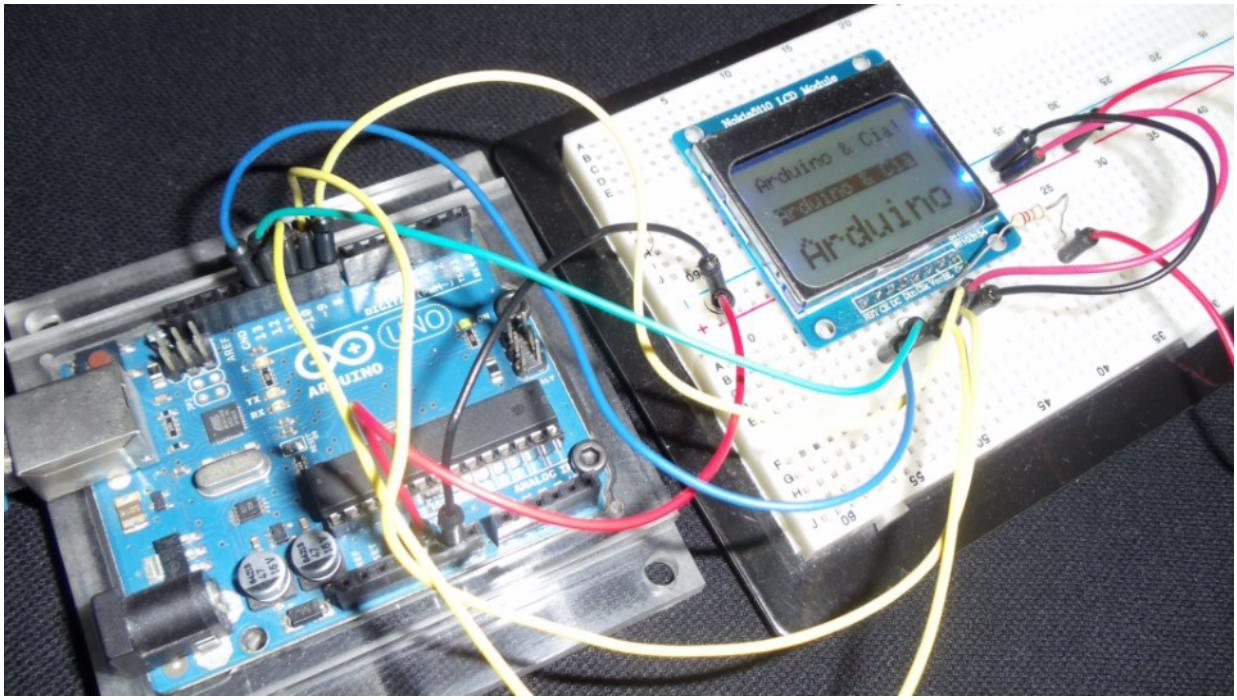
Tendo como base o objetivo de controlar o *Display* Nokia 5110 com o PIC18F4550, e após pesquisas sobre publicações relacionados ao assunto, o trabalho mais interessante e útil é a biblioteca para controle do *display*, porém para o microcontrolador ATmega328, escrita e desenvolvida por Adafruit (2015). Porém, antes de entrar em detalhes sobre o funcionamento da mesma, uma leitura do *datasheet* (Sparkfun (1999)) do PCD8544 se faz necessária.

Esta biblioteca utiliza do controlador SPI interno do ATmega328 para gerar e enviar pacotes binários de 8 *bits* para o PCD8544, sendo que alguns são comandos e outros dados. Os comandos, como visto no *datasheet* do mesmo, são interpretados de forma diferente dos dados e realizam operações no *display*, como setar em qual posição x e y da tela o ponteiro aponta, ou seja, para qual *pixel*. Sendo possível também definir como aceso ou apagado o ponto atual, limpar a tela (definir todos os pontos como apagados). Os dados são enviados byte a byte para sobrescrever diretamente a posição de memória apontada anteriormente.

Rickard Gunee (2003) criou uma biblioteca simplista para um *display* de caracteres com 16x2 caracteres de 8x5 *pixel* cada. O ponto de interesse no trabalho de Gunee é usar o PIC18F4550 de forma bem enxuta e de fácil interpretação, assim como o uso eficiente (poucas instruções utilizadas) do PIC para realizar a comunicação SPI, sendo de grande relevância o seu protocolo de comunicação.

Na parte de computação gráfica, o trabalho de Dysfunctional Technologies, Inc. (2012) é interessante por ser muito fidedigno ao jogo *Pong* original. Por mais que a interface de imagem utilizada tenha sido uma saída analógica do PIC18F4550, para controlar um monitor CRT, a lógica de desenho na tela é organizada e usa a proposta de desenho cíclico, onde a cada interação do código as posições dos objetos a serem desenhados é atualizada e são calculadas colisões e interações entre os objetos. Finalmente os objetos são desenhados um a um em camadas e, caso necessário, é feita a sobreposição. A leitura dos botões passa por tratativas, como somente executar a próxima instrução quando o botão pressionado for liberado, para evitar múltiplos toques nos mesmos e criar uma melhor experiência de jogo.

Figura 2: Uso do *display* 5110 com Arduino.



Fonte: Arduino e Cia (2013).

Vários outros projetos com microcontroladores diferentes do PIC são encontrados, principalmente para a plataforma Arduino, que usa o microcontrolador ATmega328, sendo um exemplo interessante e simples, o uso do *Display LCD 5110* com o Arduino Uno feito por Arduino e Cia (2013) e com um circuito bem simples, apresentado na Figura 2.

De acordo com o *datasheet* da própria Microchip (2006), para o PIC18F4550, na secção 19.1, é descrito como usar os registradores do microcontrolador PIC18F4550 na linguagem *assembly* para fazer o uso do SPI síncrono. É interessante observar que este modo de configuração é muito útil para comunicação onde ambos os lados trocam dados, caso seja preciso uma resposta de recebimento, e é de interesse sua leitura para fim de testes para decisão de qual modo usar, síncrono ou assíncrono.

Figura 3: Kit de estudo PICLAB criado por Yousif I. Al Mashhadany.

Fonte: Yousif Ismaill Al-Mashhadany (2012).

Um outro trabalho interessante e muito complexo é o de Yousif Ismaill Al-Mashhadany (2012), onde é criado um kit de estudo e aprendizado de microcontroladores, que utiliza o PIC18F e contém internamente um gravador de PIC, para que o mesmo possa ser gravado na placa, e que, de acordo com o autor, podem ser feitos 36 experimentos, entre eles analógicos e digitais e alguns de controle de sistemas dinâmicos. Uma imagem da bancada produzida pelo autor pode ser vista na Figura 3. Este trabalho é interessante pois faz uso de displays LCD, porem toda a programação tem que ser feita pelo usuário ou aluno do kit.

Outro kit de estudos para microcontroladores é o da Exsto Tecnologia (2018), como pode ser visto na Figura 4, que é especifico para o microcontrolador PIC18F4550, e é brasileiro, fabricado em Santa Rita do Sapucaí em Minas Gerais. O kit conta com várias aulas práticas sobre microcontroladores, como uso das portas de entrada e saída, temporizadores, conversores analógico-digital, geradores de PWM (*Pulse Width Modulation*), *display* 16x2. Este kit é usado atualmente em vários cursos de computação no Brasil, sendo um deles o do CEFET-MG Campus Timóteo, na disciplina de Microprocessadores e Microcontroladores.

Figura 4: Kit de estudo XM118 criado por Exsto Tecnologia.

Fonte: Exsto Tecnologia (2018).

3 Estado da Arte

”Não se preocupe com a perfeição - você nunca irá consegui-la.”
Salvador Dali

Vários trabalhos acadêmicos serão abordados de forma a verificar as pesquisas feitas quando se trata de miniaturizar o jogo *Pong* ou de acoplar um mini *Display* ao mesmo. A ordem de estudo dos trabalhos será feita do mais simples ao mais complexo analisado.

3.1 Estrutura básica de programação usando um sistema microcontrolado: Jogo de ping-pong

O trabalho de Andrés F. Restrepo-Alvarez (2014) é uma implementação simples do jogo *Pong* com o uso de microcontrolador AT89C52 e uma matriz de LED de 7x5 elementos, como visto na Figura 5.

Figura 5: Circuito final desenvolvido no trabalho de Andrés F. Restrepo-Alvarez (2014).

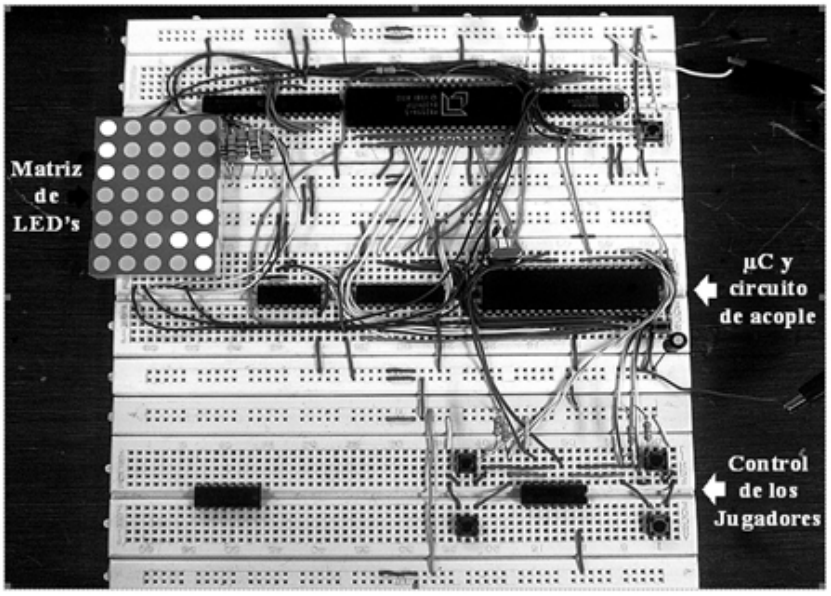


Figura 7. *Implementación física del juego de ping-pong.*

Fonte: Andrés F. Restrepo-Alvarez (2014).

Como o microcontrolador utilizado é de baixo custo e tem poucas portas de entrada/-saída disponíveis, o autor utiliza do *chip Intel* 82C55A para expandir o número de portas e controlar assim a matriz de LEDS. São utilizados dois *chips* extras para ajudar a controlar a matriz, sendo um *buffer* inversor 74LS240 para as colunas e um *buffer* inversor 74LS244 para as linhas.

Por mais que o resultado final do trabalho seja simples, o autor não utiliza nenhum CI complexo. Foi feito uso de estratégias de *hardware* para que o microcontrolador não necessite tratar o controle em *software*. De forma análoga, o que um *chip* de controle de *display*, como o PCD8544, faz é exatamente o que foi feito neste trabalho, porém em um só *chip*.

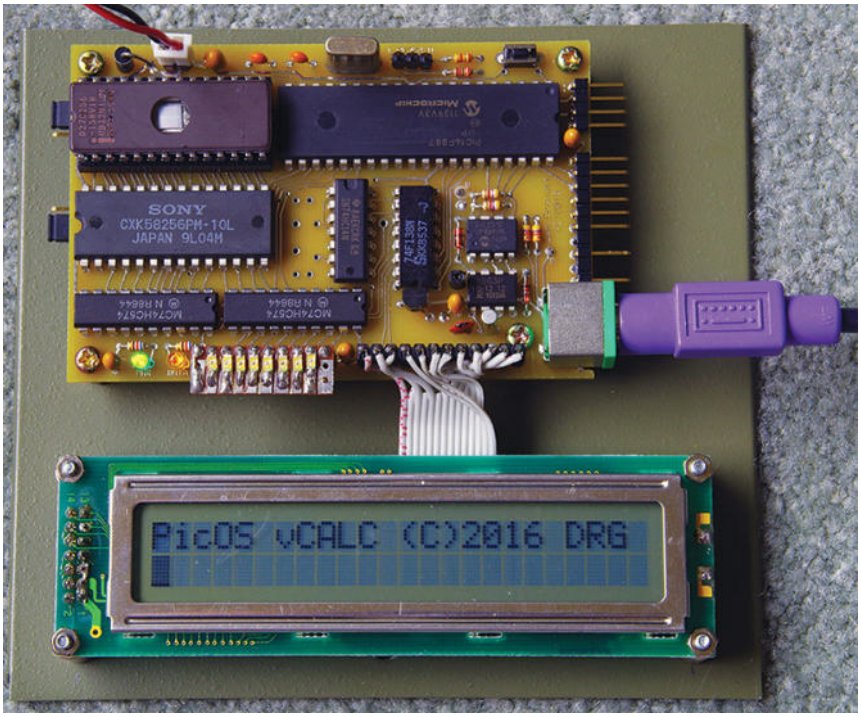
3.2 O computador de placa única retro PIC

Microcontroladores tem inúmeras aplicações, dentre uma delas a recriação de arquiteturas de computador antigas, como por exemplo computadores baseados no Zilog Z80 ou no Motorola 68000 ou ainda no Intel 8088. De acordo com Dan Gravatt (2017), é uma ótima forma de estudar arquitetura e organização de computadores ao recriar computadores do fim da década de 70.

A grande diferença de um processador para um microcontrolador se resume no fato de que o processador precisa de memória RAM, memória EEPROM, um *chip* para realizar a comunicação serial com componentes de entrada e saída, entre outros para funcionar e ser utilizado, já o microcontrolador tem estes componentes embarcados em um único CI.

O autor utiliza de um microcontrolador PIC16F887 no lugar do Zilog Z80, que é um processador de 8 *bits*, para que isso seja possível, ele não utiliza da memória interna do PIC nem da RAM interna. Foram utilizados *chips* de RAM(62256) e EPROM(24LC512) para que o PIC funcionasse como um processador somente. A interface com o usuário é feita a partir de um teclado padrão PS/2 e um display de 16x2 caracteres como podemos ver na Figura 6.

Figura 6: Circuito final desenvolvido no trabalho de Dan Gravatt (2017).

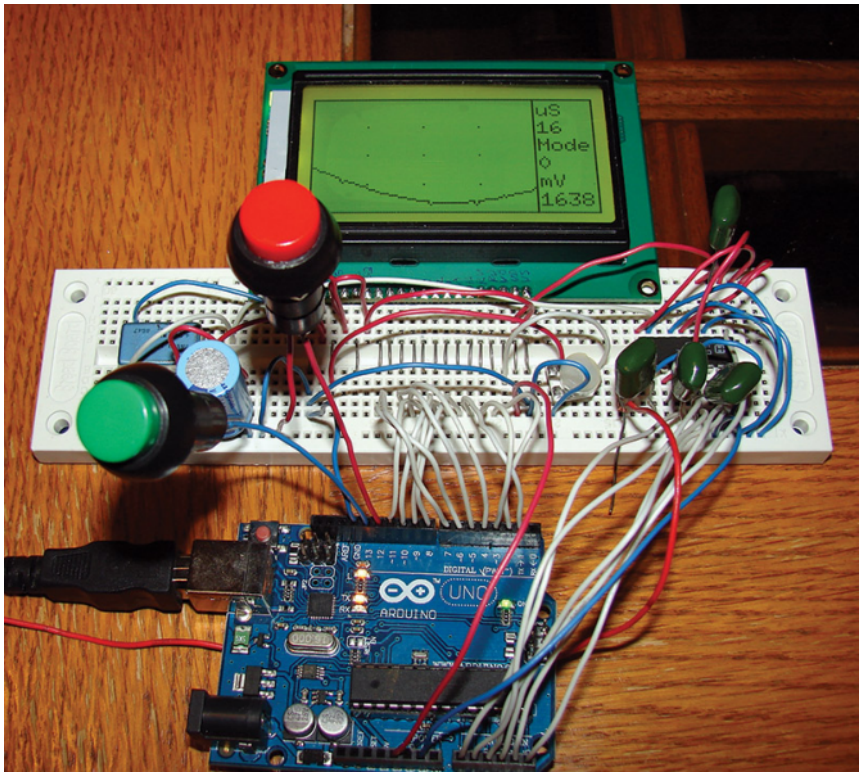


Fonte: Dan Gravatt (2017).

3.3 Quebrando o limite de velocidade do Arduino

A plataforma Arduino com o microcontrolador ATmega328 é uma das mais utilizadas por iniciantes, mas de acordo com Bob Davis (2014) ainda é uma plataforma poderosíssima em processamento de dados. O Autor utiliza de um *display* de 128 por 64 *pixels*, com o controlador ST7920 paralelo, que utiliza de 8 saídas de dados do microcontrolador e 3 de controle.

Figura 7: Circuito final desenvolvido no trabalho de Bob Davis (2014).



Fonte: Bob Davis (2014).

Ao criar uma versão simplista de um osciloscópio com o ATmega328 e o *display* de 128 por 64 *pixels*, como vemos na Figura 7, o autor demonstra a capacidade desta plataforma, mas ainda foi possível melhorar a precisão de amostragem do osciloscópio de 8.9 mil amostras por segundo para 100 mil. Para que isso seja possível, o autor mudou a configuração interna do microcontrolador, que ao invés de usar 10 *bits* de precisão ao converter um sinal de analógico para digital, use somente 8, que melhora consideravelmente a taxa de amostragem.

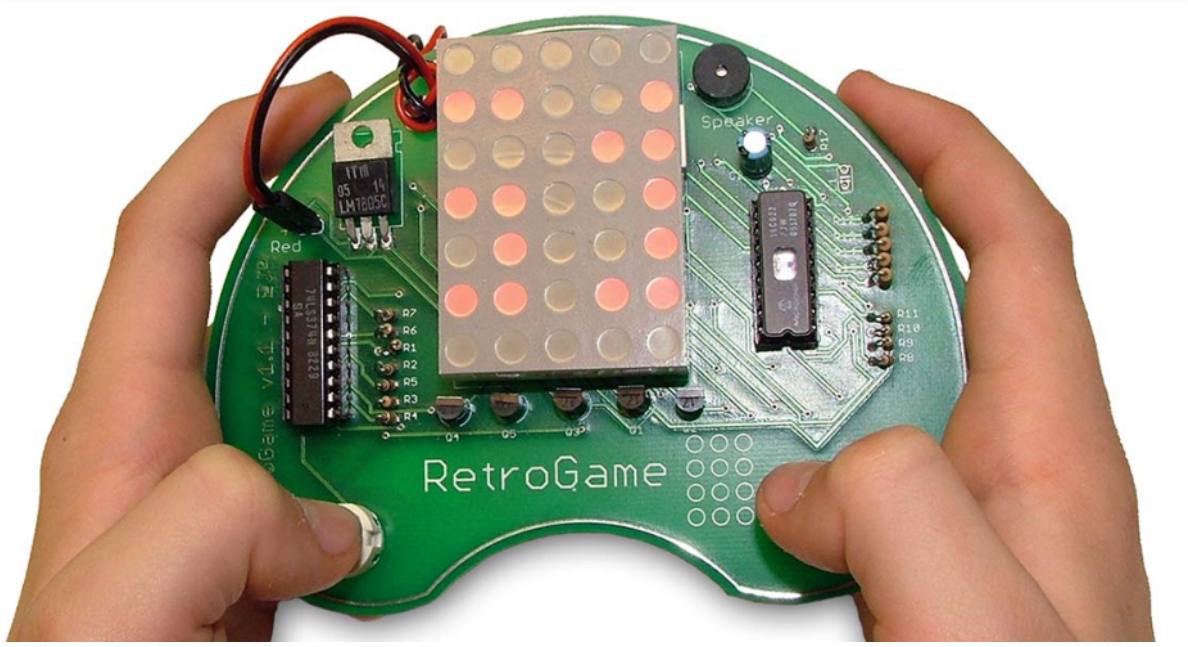
Para melhorar a velocidade, o autor ainda muda a entrada de dados de analógico para digital e analisa os dados em *software* e, para melhorar ainda mais, é usada uma técnica que compiladores avançados utilizam, chamada *loop unrolling*, que consiste em desmembrar trechos de retorno repetitivo no código em várias linhas em sequência, evitando os desvios.

3.4 Construção do jogo retrô

Uma aplicação de microcontroladores para videogame portátil é a descrita por Eric Rothfus (2017). No artigo, o autor mostra o passo a passo para usar o PIC16C622A em conjunto com uma matriz de 5x7 LED e dois botões.

O videogame portátil usa uma bateria de 9V acoplada a um regulador 7805, que fornece os 5V necessários para o PIC e a matriz de LED. O resultado final é extremamente simples e elegante, como pode ser visto na Figura 8.

Figura 8: Circuito final desenvolvido no trabalho de Eric Rothfus (2017).



Fonte: Eric Rothfus (2017).

Com o objetivo de simplificar o circuito e melhorar o controle da matriz de LED, o autor utiliza um circuito multiplexado que consiste de um CI 74LS374, para comandar as linhas e cinco transistores para controlar as colunas. Dessa forma, somente uma linha é ligada de cada vez, mas como o tempo que cada uma fica ligada é muito baixo, para o olho humano parece que estão todas ligadas.

4 Materiais e Métodos

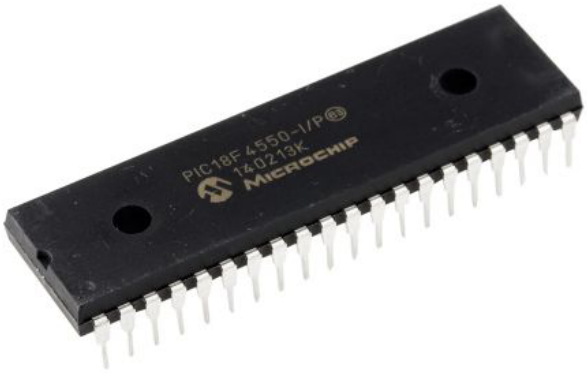
”As crianças devem ser autorizadas a quebrar coisas mais vezes. Isso é uma consequência da exploração. Exploração é o que você faz quando você não sabe o que está fazendo. É o que os cientistas fazem todo dia.”.
Neil deGrasse Tyson

A proposta deste trabalho consiste em criar uma versão do conjunto *hardware* e *software* do jogo *Pong*, em menor escala, com o uso do *Display Nokia 5110* e seu controlador PCD8544 embutido, em comunicação com o microcontrolador PIC18F4550. É necessário para isto elaborar uma biblioteca completa com um protocolo de comunicação serial e funções para controle do *display*. Esta biblioteca será escrita em linguagem C18, que pode ser compilada e usada com o compilador de mesmo nome, sendo de uso gratuito. Com isso esta pesquisa pode ser descrita como qualitativa descritiva com os dados obtidos por observação.

Os procedimentos utilizados neste trabalho subdividem-se então em, desenvolvimento da biblioteca simplista, que usa internamente o protocolo de comunicação serial, baseado no modelo já existente para o microcontrolador *ATmega328* escrito por Adafruit (2015), porém adaptado para as características do PIC18F4550. Com uso da biblioteca anterior elaborar um conjunto de funções para desenho no *display*, e por último elaboração de um circuito para testes com os principais componentes:

- PIC18F4550 (Figura 9a);
- *Display Nokia 5110* com controlador PCD8544 embutido (Figura 9b);
- Cristal gerador de frequência de 20Mhz.
- Botões e reguladores para interação com usuário.

Figura 9: Componentes principais

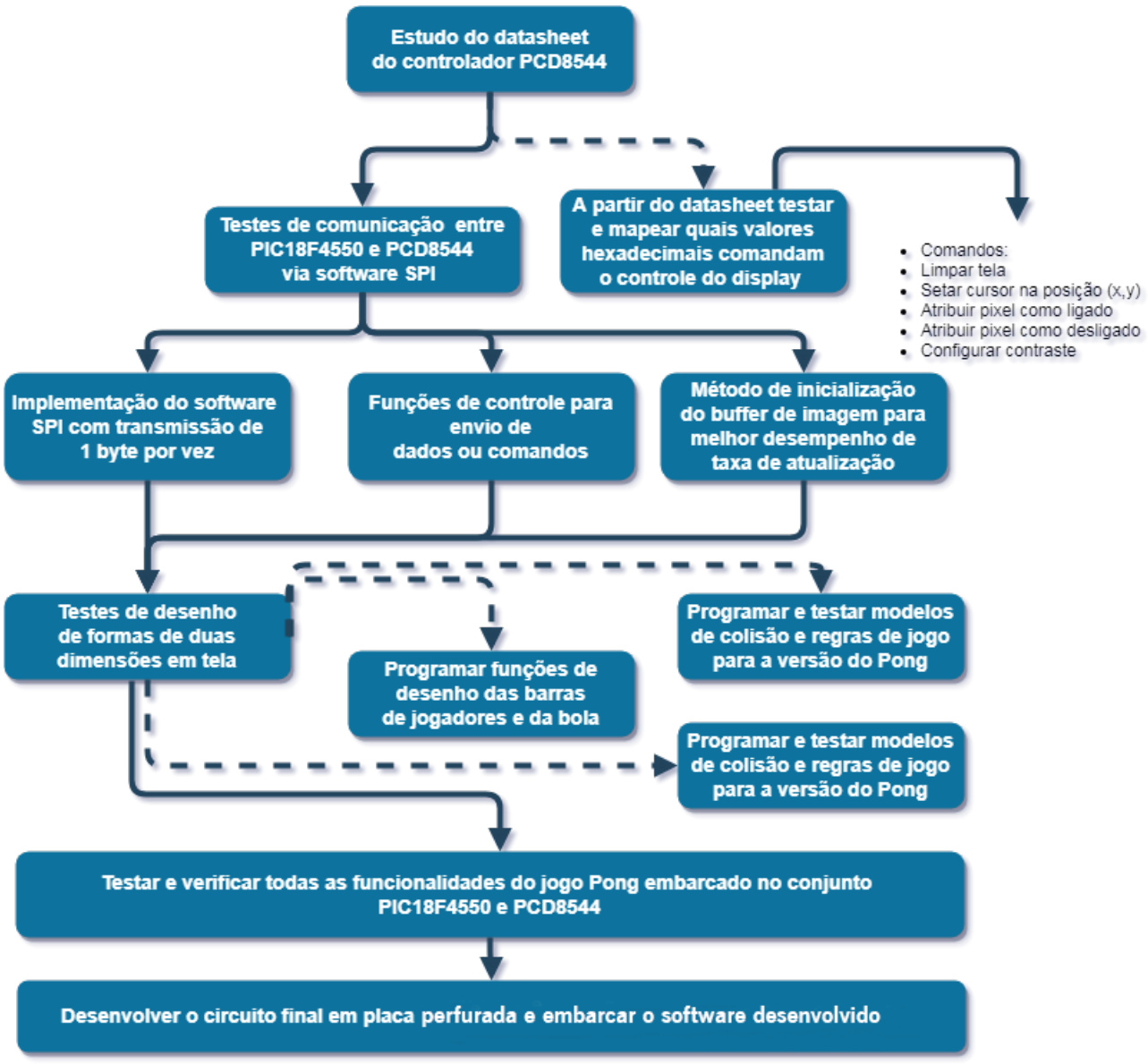


(a) PIC18F4550
Fonte: Microchip (2006).

(b) *Display* Nokia 5110
Fonte: FelipeFlop (2019).

Por fim, o desenvolvimento de uma versão do jogo *Pong* e teste comprobatório de funcionamento adequado do jogo, controles e da biblioteca desenvolvida e embarcada no microcontrolador. O processo é detalhado na Figura 10.

Figura 10: Fluxograma detalhado das etapas do trabalho.



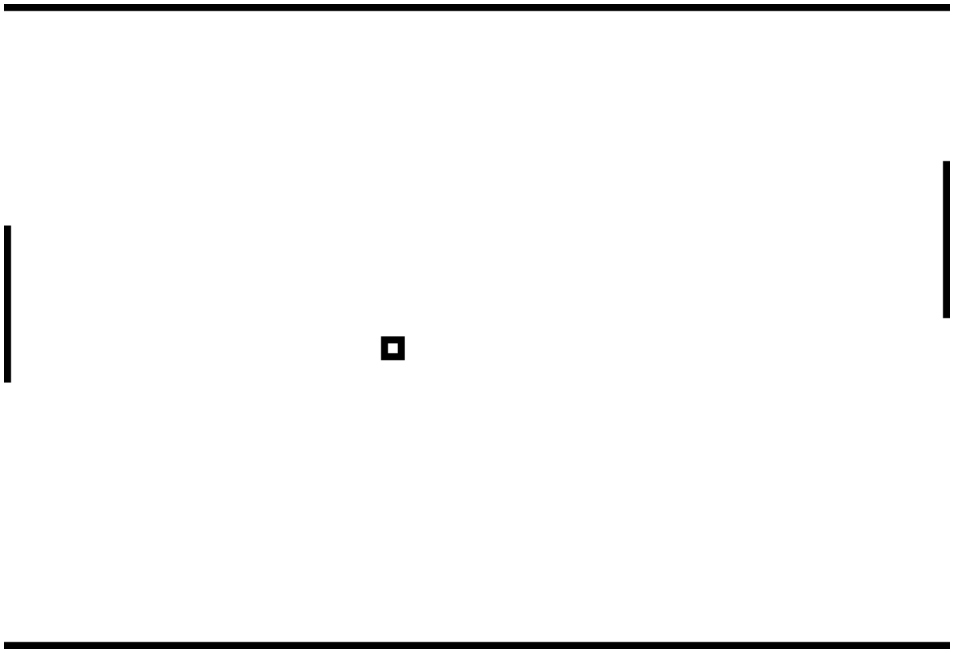
Fonte: O Autor.

5 Construção da estrutura *hardware* e *software* da versão do jogo *Pong*

”Se não fosse imperador, desejaria ser professor. Não conheço missão maior e mais nobre que a de dirigir as inteligências jovens e preparar os homens do futuro.”.
D. Pedro II

O desenvolvimento do trabalho consiste em criar uma versão do jogo pong com o uso do microcontrolador PIC e o *display Nokia 5110*. Ao entorno deste conceito, será necessário adaptar uma parte da biblioteca escrita por Adafruit (2015), para a plataforma Arduino, para a linguagem C18 com o uso do protocolo serial escrito em código pelo autor, tentando ao máximo simplificar a nova biblioteca com objetivo de ter maior desempenho.

Figura 11: *Exemplo de uma possível versão do Pong.*



Fonte: O Autor.

Com a biblioteca desenvolvida, será criado dois objetos de duas dimensões a serem desenhados na tela, a barra dos jogadores e a bola, como vemos na Figura 11. Os objetos serão desenhados de forma que a cada iteração suas posições sejam levemente alteradas, criando assim noção de movimento. Sendo baseado no jogo original, as paredes laterais contam como marcação de ponto para o adversário, onde o objetivo é rebater a bola e não deixar que ela encoste na parede do lado do jogador.

5.1 Interface do *display*

O *display* utilizado é o mesmo encontrado nos telefones Nokia 3310 e 5110, porém contém um controlador PCD8544, que se comunica com o PIC18F4550 através de uma implementação do protocolo SPI. A tela conta com 84 por 48 *pixels* monocromáticos e uma interface de 8 pinos, como pode ser visto na Figura 9b. Os pinos do *display* são os seguintes:

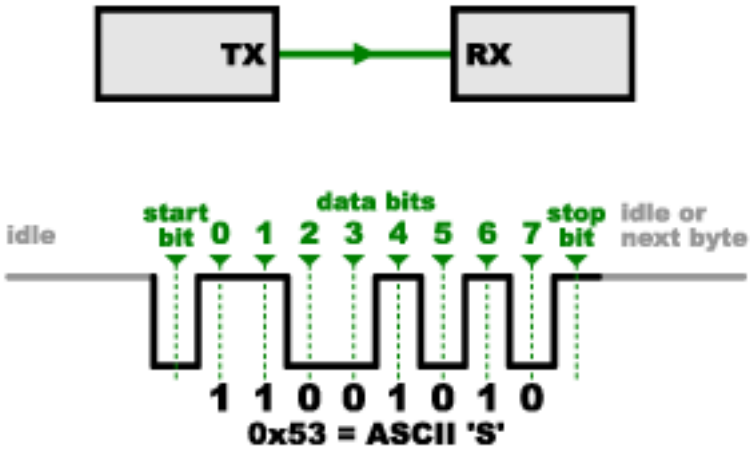
1. RST - *reset*;
2. CE - *chip enable*;
3. DC - *data or command*;
4. DIN - *data in*;
5. CLK - *clock*;
6. VCC - *5V in*;
7. BL - *backlight*;
8. GND - *ground*.

Os pinos estão ligados ao controlador PCD8544 e cada pino por sua vez tem uma função ao comunicar com o *chip*. Os pinos de VCC, GND e BL são pinos simples que fornecem tensão de funcionamento e controlam contraste de iluminação do *display*. O pino CLK é a entrada de clock no controlador, que será controlado via *software*, pois nossa interface SPI será assíncrona. No pino DIN são inseridos os dados ou comandos de forma serial, o pino DC é um pino booleano de controle no qual é selecionado se o pino DIN está recebendo dados ou comandos. Por fim, o pino CE, também booleano, controla se o chip está habilitado para receber informações e ao ser dado um pulso no pino RST é reiniciado o controlador PCD8544.

5.2 Implementação do SPI

De acordo com Sparkfun (2018), que é a produtora do controlador PCD8544, é possível implementar o SPI de duas formas, síncrona e assíncrona, cada qual com suas características e por ser uma implementação simples, aberta a modificações. Ainda de acordo com Sparkfun (2018), o SPI assíncrono é implementado sem uma linha de *clock* e utiliza somente a linha de dados e o CS (*Chip Select*) para enviar os dados do dispositivo mestre. Na Figura 12 é possível observar que existe um *bit* extra antes e um *bit* extra depois dos dados, para informar que começou o envio de informação e depois que terminou, e além dos bits, ambos os lados concordam com uma taxa de transferência padrão (normalmente 9600 bits por segundo) para que o escravo consiga amostrar o valor do pino no momento correto.

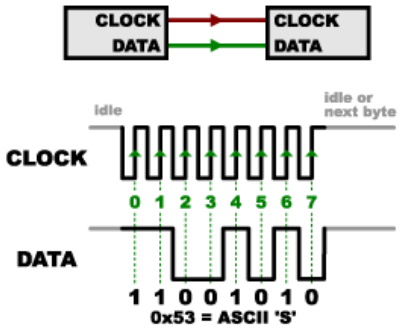
Figura 12: *Possível implementação do SPI assíncrono.*



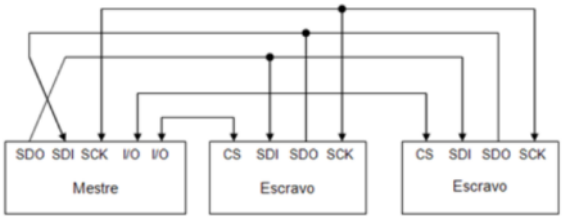
Fonte: (Sparkfun, 2018).

O SPI síncrono funciona de outra maneira, o dispositivo escravo tem uma entrada de *clock* e outra de dados que vem do mestre. O sinal de clock é um sinal que diz exatamente quando o dispositivo escravo deve amostrar a linha de dados, normalmente em mudança de pulso de baixo para alto, o *datasheet* de cada dispositivo indica qual método utilizar. Assim que o escravo detecta o pulso de *clock*, ele amostra o pino de dados, como vemos na Figura 13. Como agora os pulsos são enviados em um barramento diferente dos dados, não é necessário concordar com uma velocidade padrão como no assíncrono, porém, os dispositivos tem uma velocidade limite que eles podem operar, que é encontrada no *datasheet* dos mesmos.

Figura 13: *Possível implementação do SPI síncrono.*



(a) Fonte: Sparkfun (2018).



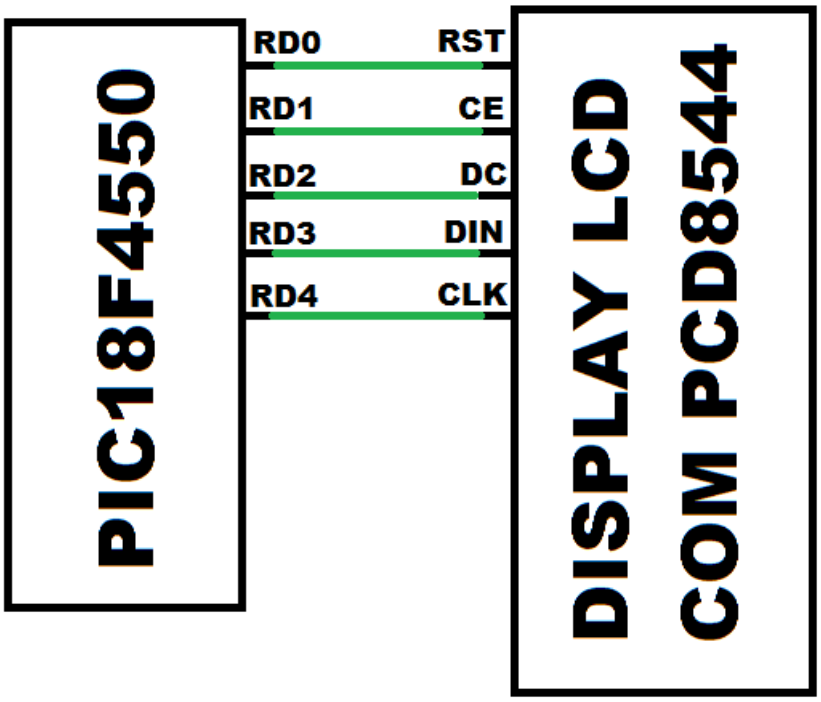
(b) Fonte: Exsto Tecnologia (2018).

O protocolo SPI implementado pelo autor é de autoria própria, pois os dados serão enviados quando o microcontrolador estiver com eles prontos, ou seja, não há a necessidade de manter um fluxo contínuo ou síncrono de dados entre os dois dispositivos, porém será utilizado a linha de *clock* e dados como no modo síncrono, e os pulsos serão gerados somente quando o dado for enviado. Esta escolha também é interessante visto que o *display* com

controlador PC8544 somente recebe dados quando o pino CE ou CS (em outros modelos de *display*) está ativo e temos o pino extra DC para indicar se o *byte* que está sendo recebido é um dado ou comando, então temos uma implementação mais específica, como pode ser visto no diagrama da Figura 14.

O *hardware* SPI existente no PIC18F4550, de acordo com o *datasheet* da Microchip (2006) na secção 19.3, mostra que são necessárias as portas RC7, RB0 e RB1 para implementação do *hardware* SPI. A versão do autor é feita em *software*, no código da biblioteca, possibilitando a modificação do SPI com um controle mais direto, além de ser possível o uso de qualquer porta de Entrada/Saída, de forma a evitar o uso de portas fixas, que podem ser modificadas no código.

Figura 14: Versão modificada do SPI do Autor.



Fonte: O Autor.

Na implementação, para que um *byte* seja transferido do PIC ao controlador do *display* pela porta DIN, de acordo com o *datasheet* do PCD8544 de Sparkfun (1999), é necessário setar o pino DIN com o valor do primeiro *bit*, gerar um pulso de *clock* em seguida o segundo *bit* e assim por diante até completar os 8 *bits* que compõe o *byte*.

Isso é demostrado na função *"escreveSPI"* no código do Apêndice A da seguinte maneira:

1. *loop* ou repetição - a função recebe o *byte* como uma variável sem sinal (*unsigned char*) de nome data e entra no *for*.

2. Deslocando os *bits* - a cada iteração é deslocado a direita de 0 até 7 casas no valor original da variável data. Dessa forma o valor binário que interessa é colocado no *bit* menos significativo da variável temporária d, por exemplo, o *byte* 0b01010000 ao ser deslocado 4 casas, fica com o valor 1 na primeira casa binária, já ao ser deslocado 5 casas, fica com o valor 0.

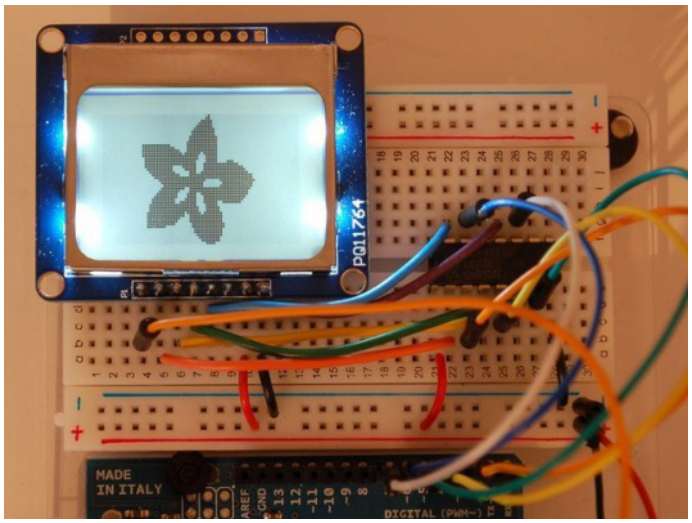
3. Verificação dos *bits* - a cada iteração após o deslocamento é realizada a operação binária E com a variável temporária d e o valor fixo 0b00000001, dessa forma somente o valor binário a ser analisado fica na variável d e na primeira casa binária. Com isso é possível verificar se ele é 1 ou 0 e atribuir esse valor a porta de saída DIN.

4. Pulso de *clock* - é gerado um pulso na porta CLK do controlador para que a informação inserida na porta DIN seja armazenada e a próxima iteração aconteça enviando outro *bit* até que todo o *byte* tenha sido enviado.

5.3 Implementação de uma biblioteca de controle de *display*

Tendo como base a biblioteca de Adafruit (2015), para a plataforma Arduino, que utiliza o microcontrolador ATmega328, será criada uma biblioteca simples, que consiga desenhar ou apagar um *pixel* em qualquer posição da tela e limpar a tela. Dessa forma, será possível mais a frente no trabalho, desenhar no *display* um objeto *pixel* a *pixel*, apagar a tela, e redesenhar os objetos de forma cíclica, com a posição levemente incrementada, gerando assim a noção de movimento.

Figura 15: *Display Nokia 5110 sendo usado com ATmega328 e biblioteca Adafruit.*



Fonte: Adafruit (2015).

Com a função *"escreveSPI"* descrita no código do Apêndice A, é possível implementar mais duas funções simples, para definir se o *byte* enviado é um dado ou um comando.

De acordo com a leitura do *datasheet* do PCD8544 para que dados ou comandos sejam enviados ao controlador, as seguintes etapas devem ser seguidas, como descrito nas funções *"command"* e *"data"* no código do Apêndice A:

1. Setar pino DC - caso a informação a ser enviada sejam dados, o pino deve receber o valor 1, caso sejam comandos o valor 0.
2. Setar pino CE - o pino de *chip enable*, que no código foi chamando do análogo CSchip select, este pino é ativo em baixo, o que significa que deve receber o valor 0 para que o *chip* receba as informações.
3. Chamada de função - a função *"escreveSPI"* é chamada e é passado o valor em *byte*. Esta função já descrita no Apêndice A só pode ser chamada entre tratamento dos pinos DC e CE.
4. Setar pino CE - o pino CE deve receber o valor 1 para que o *chip* agora já com as informações recebidas possa executar o comando ou tratar os dados.

A partir de uma nova leitura do *datasheet* do PCD8544 e as funções de envio de dados e comandos já existentes, é possível criar valores fixos com os comandos utilizados, que podem ser vistos no Apêndice A, logo após os vetores *buffer* e *buffer2* e são identificados sempre como *"define nome valor"*, onde nome é o nome que será usado para substituir o valor no código.

Os valores em hexadecimal dos comandos podem ser encontrados no *datasheet* do PCD8544. O valor BV(x) é um deslocamento do *byte* 0b00000001, x vezes para a esquerda, para que possa ser utilizado na função *"drawPixel"* encontrada no Apêndice A ao ser feito acesso aos *buffers*.

Com os valores dos comandos configurados, foram desenvolvidas rotinas de controle do *display* que serão apresentadas e descritas a seguir.

De acordo com a função *"setContrast"*, encontrada no Apêndice A, e Adafruit (2015), o valor do contraste, que é a intensidade dos *pixels* aceso no *display*, é realizada da seguinte forma:

1. Verificar valor máximo da variável val, que não pode exceder 7F em hexadecimal ou 127 em decimal, e caso aconteça setar no valor máximo de 127.
2. Enviar o comando de instruções estendidas, um modo do controlador PCD8544 para receber comandos complexos.
3. Enviar o comando de setar contraste com o valor do contraste em si.
4. Enviar o comando de instruções comuns, para voltar o controlador para modo de operação padrão.

De acordo com o *datasheet* do PCD8544 e a biblioteca desenvolvida para a plataforma Arduino por Adafruit (2015), o PCD8544 usa uma memória RAM de 504 *bytes write only*, somente escrita, para mapear todos *pixels* do *display*, sendo que estes pontos são subdivididos em 6 bancos de 8 linhas cada, com 84 colunas cada, com isso temos:

6 bancos x 8 linhas x 84 colunas = 4032 *bits* = 504 *bytes*

Cada banco tem oito linhas, e cada linha na memória é representada por um *byte*, como pode ser visto na Figura 16, retirada do *datasheet* do PCD8544. Por isso e por não ser possível ler da memória, é necessário ter no microcontrolador uma cópia da RAM do PCD8544, pois não se altera um *bit* de cada vez, somente um *byte* pode ser escrito na memória. Logo ao alterar um *pixel* mudamos ele no *byte* que se localiza na cópia, para depois enviar a mudança para a memória do controlador do *display*.

Figura 16: Seção 7.7.1 do *datasheet* do PCD8544

7.7.1 DATA STRUCTURE

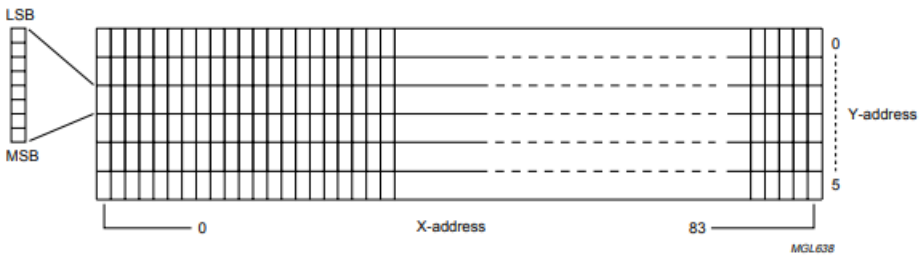


Fig.4 RAM format, addressing.

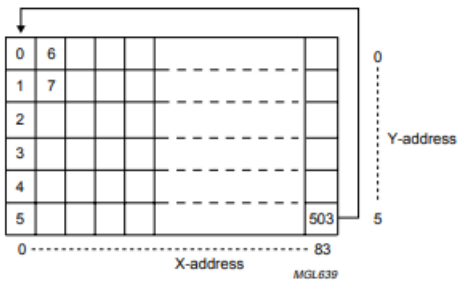


Fig.5 Sequence of writing data bytes into RAM with vertical addressing (V = 1).

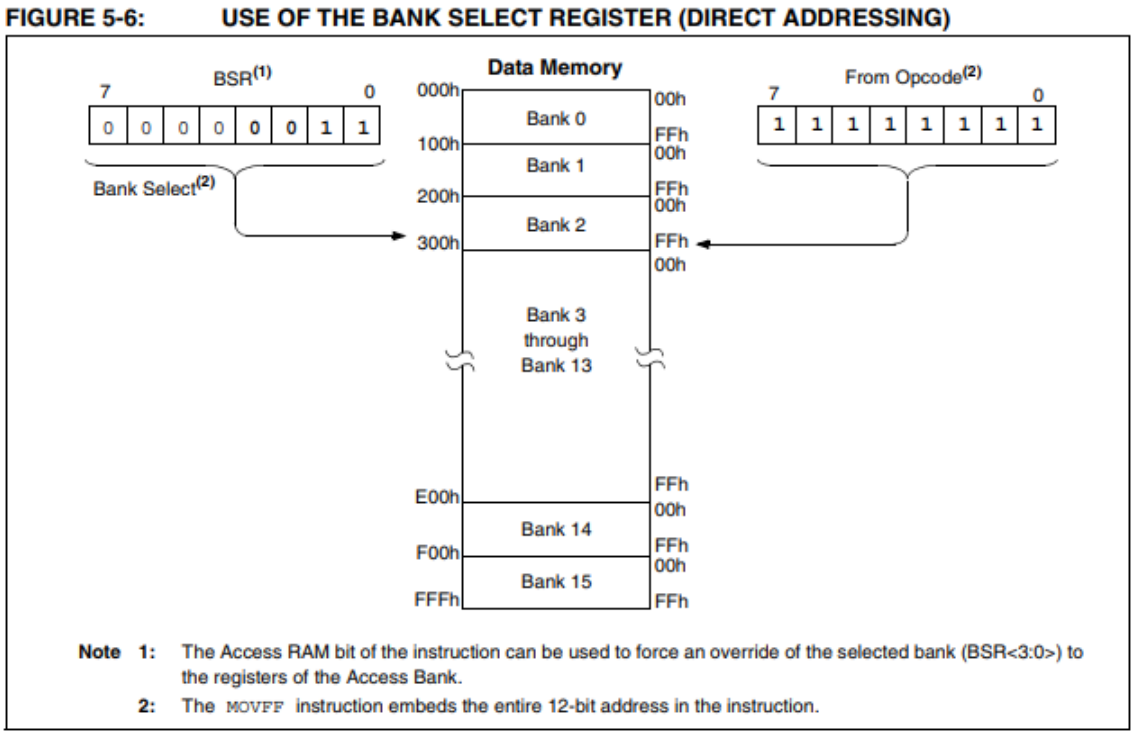
Fonte: Sparkfun (1999).

A biblioteca desenvolvida por Adafruit (2015) armazena a informação da RAM gráfica em um vetor dentro da memória do ATmega328, pois só é possível alterar um *byte* de cada

vez. De acordo com Microchip (2007) no *datasheet* do PIC16F628A, sua memória para uso geral é de 224 bytes, logo impossibilita o uso desse microcontrolador, ainda de acordo com Microchip (2003) no *datasheet* do PIC16F877A, sua memória para uso geral é de 368 bytes, também impossibilita o uso desse microcontrolador, pulando para a família 18F, de acordo com Microchip (2006) no *datasheet* do PIC18F4550, sua memória para uso geral é de 2048 bytes, porém a memória de uso geral disponível no PIC18F4550 é dividida em 16 bancos de 256 *bytes* cada e pode ser vista na Figura 17, retirada do *datasheet* do PIC18F4550, porém somente é possível acessar dos bancos 0 ao 7, visto na figura 5.5 do *datasheet*. Logo, não é possível armazenar um vetor de 504 *bytes* linearmente, a solução mais simples é dividir em dois vetores, um de 256 *bytes* no banco 1 e outro de 248 *bytes* no banco 2, como visto no Apêndice A os vetores *buffer* e *buffer2*.

De acordo com o *datasheet* do PCD8544 e da biblioteca de Adafruit (2015), foram desenvolvidos métodos para melhorar o desempenho de atualização da RAM gráfica a partir dos vetores *buffer* e *buffer2*. As funções *"updateBoundingBox"* e *"display"* encontradas no Apêndice A mostram a atualização da RAM gráfica do controlador, somente nas partes que mudaram com a função *"display"*, as porções da memória que mudaram são atualizadas com o uso da função *"updateBoundingBox"*, que recebe os valores de x e y mínimos e máximos das áreas que foram alteradas e salvas em variáveis globais, que são usadas pela função *"display"*.

Figura 17: Figura 5.6 do *datasheet* do PIC18F4550



Fonte: (Microchip, 2006).

A função *"display"* aplica as mudanças realizadas nos vetores *buffer* e *buffer2* desde a última atualização. Para saber qual região do *display* foi alterada são consultadas as variáveis globais *xUpdateMin*, *xUpdateMax*, *yUpdateMin* e *yUpdateMax* logo no início da função. Como a memória do PCD8544 é divida em 6 bancos, é verificado se ouve alteração no banco 'p' que também será nosso y do *display* e são definidas as colunas inicial e final, 'col' e 'maxcol' respectivamente, com uso das variáveis globais, que pode ser visto no código 1, retirado do Apêndice A:

```
1 command(PCD8544_SETYADDR | p);
2
3 col = xUpdateMin;
4 maxcol = xUpdateMax;
5
6 command(PCD8544_SETXADDR | col);
```

código 1: Trecho da função display parte 1

Assim que os valores de x e y são passados em forma de comando para o PCD8544, o próximo dado a ser enviado irá sobrescrever aquela posição da memória interna. Com isso, agora é possível o envio de dados que são buscados da cópia da memória, que está contida nos vetores *buffer* e *buffer2*, como pode ser visto no código 2 retirado do Apêndice A.

```
1 _dc = 1;
2 _cs = 0;
3 for(; col <= maxcol; col++) {
4         if(((LCDWIDTH*p)+col)<256)
5         escreveSPI(buffer[(LCDWIDTH*p)+col]);
6         else
7         escreveSPI(buffer2[((LCDWIDTH*p)+col)-256]);
8 }
9 _cs = 1;
```

código 2: Trecho da função display parte 2

Logo após todas as iterações dos blocos de memória, que são seis, os valores das variáveis globais são reiniciados e o ponteiro de y do *display* é reiniciado também. A função mais importante e que utiliza dos conceitos de várias funções já apresentadas é a *"drawPixel"* vista no Apêndice A, que consiste em:

1. Verificar se valor de x e y passados não estouram valores mínimos e máximos.
2. Alterar no *buffer* correto se o *pixel* estará ligado ou não.
3. Chamar função *"updateBoundingBox"* com os valores x e y alterados para que seja feita a mudança na próxima atualização de *display*.

5.4 Manipulação gráfica do *display*

O jogo *pong* é composto de movimentos, colisões e regras de jogo. A partir das funções e códigos implementados na biblioteca, principalmente as funções *"drawPixel"* e *"display"* que são usadas de forma recorrente, é possível criar todos os movimentos, colisões e regras com as mesmas, sendo necessário a cada iteração apagar a tela com a função *"clearDisplay"*.

As regras de jogo implementadas foram as seguintes:

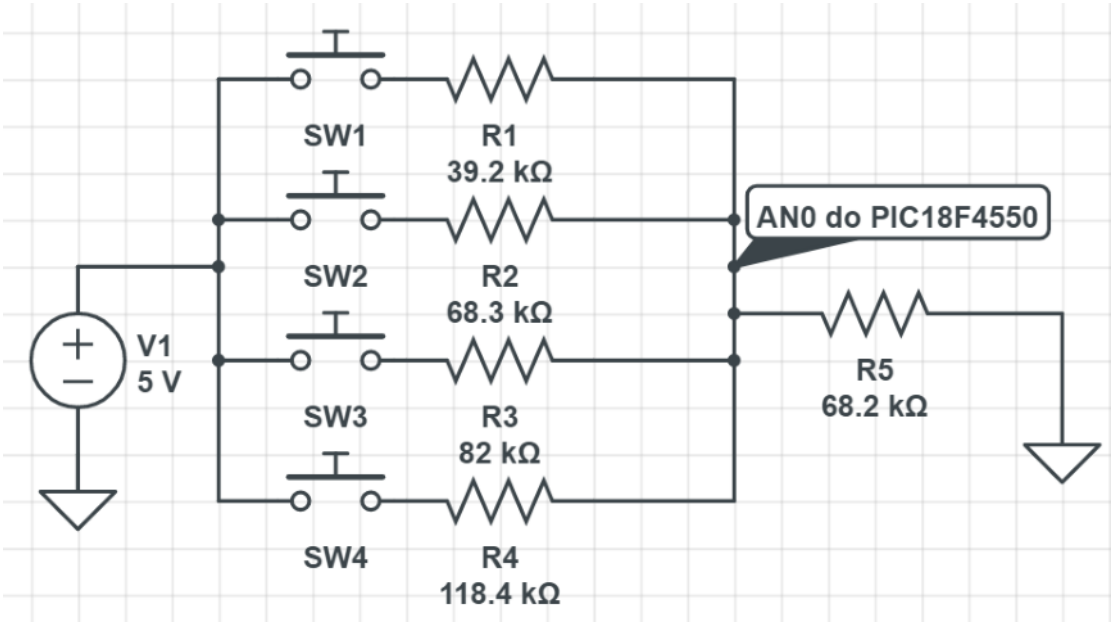
1. Ganha um ponto o jogador que consegue fazer com que a bola colida com a parede lateral do outro jogador;
2. Colisões com as paredes superior e inferior ou com as barras dos jogadores somente resultam em mudança de sentido de movimento da bola;
3. Cada vez que um ponto é realizado é exibido o placar e a bola volta para o centro da tela em direção ao jogador que não fez o ponto;
4. Ganha o jogo o jogador que fizer três pontos primeiro;
5. Para recomeçar o jogo, pressionar o botão *reset*.

As regras implementas acima estão localizadas no trecho *while(1)* da função *main* do código principal, localizado no Apêndice A.

5.5 Interação com o jogador

A entrada de usuário, inicialmente, era feita com quatro botões ligados a quatro resistores que então eram conectados a um último resistor ligado ao terra. A leitura era feita entre os resistores, como um divisor de tensão, conectada a porta AN0 do PIC18F4550, para realizar a conversão analógica/digital, como pode ser visto na Figura 18.

Figura 18: Primeiro modelo de entrada de usuário.



Fonte: O Autor.

A leitura analógica do microcontrolador PIC18F4550 converte um valor contínuo de tensão entre zero e cinco *volts* para um valor discreto entre zero e mil e vinte três. A Tabela 1 contem os valores respectivos de leitura para as situações possíveis de botões apertados.

Tabela 1: Leitura analógica dos botões

Botão	Resistência equivalente	Tensão	Leitura analógica
SW1	39,2kΩ	3,17V	649
SW2	68,3kΩ	2,5V	512
SW3	82kΩ	2,27V	465
SW4	118,4kΩ	1,82V	373
SW1 e SW3	26,52kΩ	3,6V	736
SW1 e SW4	29,44kΩ	3,49V	714
SW2 e SW3	37,26kΩ	3,23V	661
SW2 e SW4	43,31kΩ	3,05V	624
SW1 e SW2 e SW3	19,09kΩ	3,9V	797
SW1 e SW2 e SW4	20,56kΩ	3,84V	785
SW1 e SW3 e SW4	21,66kΩ	3,79V	775
SW2 e SW3 e SW4	28,32kΩ	3,53V	722

Fonte: O Autor

Mapeados os valores possíveis de leitura analógica, foi desenvolvida uma função para ler a entrada dos usuários e armazenar em variáveis globais, se a barra estaria subindo ou

descendo, caso nenhuma variável esteja ativa, ela se mantém parada, como pode ser visto no código 4 localizado no Apêndice A.

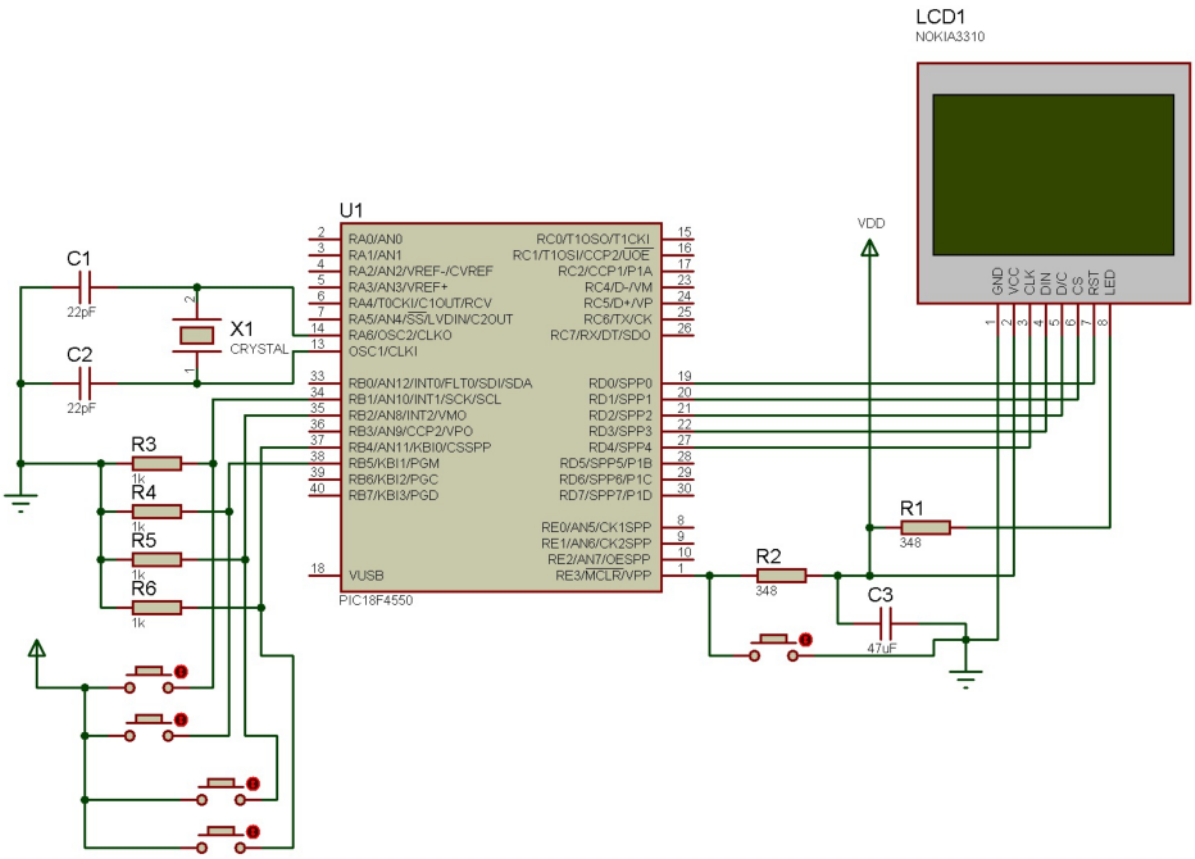
Outra possível abordagem é realizar a leitura da entrada dos usuários através de entradas digitais ao invés de analógicas. Dessa forma, a leitura de cada botão é independente do ouro, porém é necessário utilizar parte de uma porta para os botões, como pode ser visto na Figura 19. A porta mais interessante é a PORTB, pois ela contem resistores internos que auxiliam na leitura de botões, que podem ser ativados via comando no código, como pode ser visto na secção de configuração do código completo do Apêndice A, entre as linhas 478 e 508 do código.

6 Experimentos e Resultados

"Xekinóntas eínai to ímisy tis drásis: Começar ja é metade de toda a ação."
Provérbio Grego

Com toda a parte teórica desenvolvida anteriormente, foi criada uma simulação no *software Protheus 8.6 Professional* do circuito completo, como visto na Figura 19 e testado o funcionamento adequado da biblioteca e jogo, porém a simulação rodou de forma lenta, a próxima etapa é a criação do circuito real.

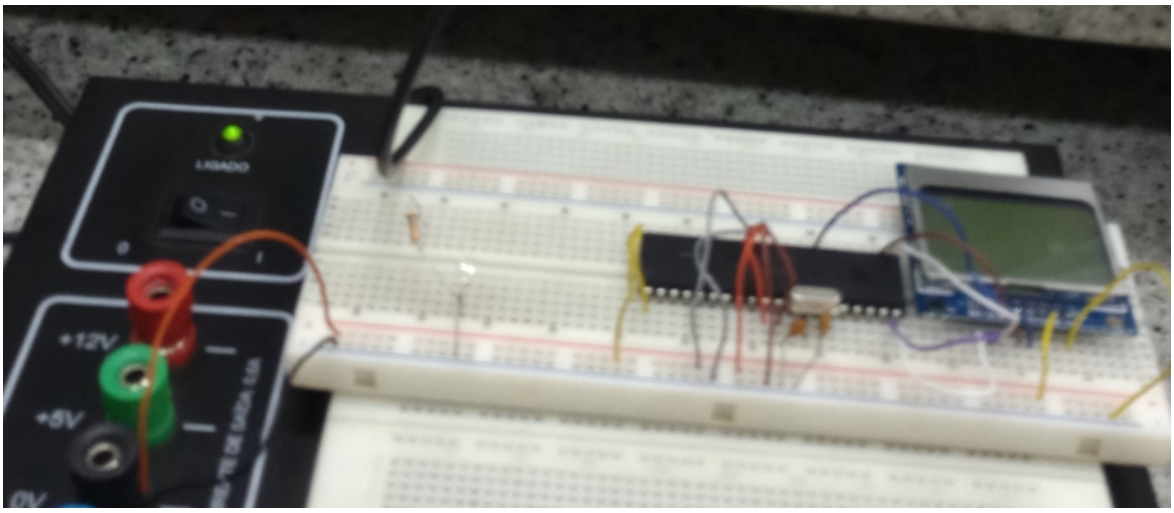
Figura 19: Simulação do circuito no *software Protheus 8.6 Professional*.



Fonte: O Autor.

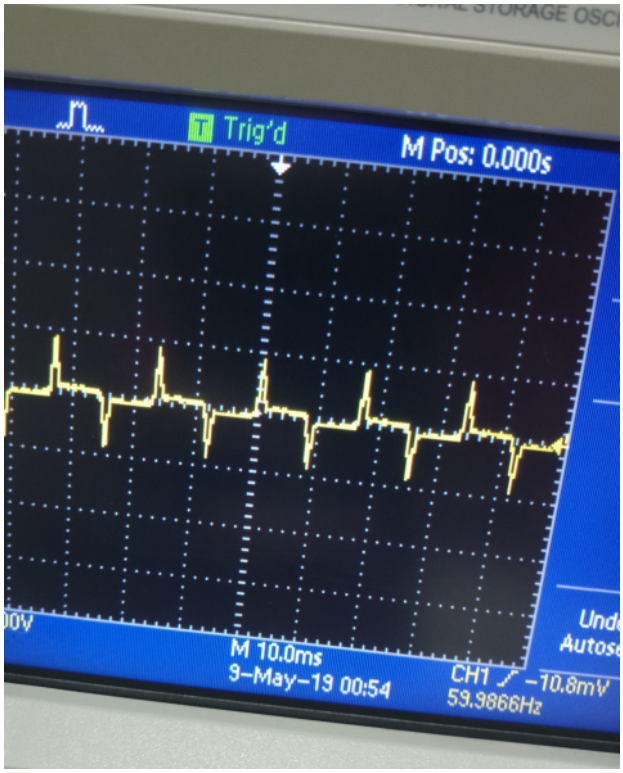
Para os testes reais os circuitos foram implementados em uma *protoboard* com alimentação de 5V própria, como visto na Figura 20.

Figura 20: Testes iniciais em *protoboard*.



Fonte: O Autor.

Figura 21: Testes iniciais em *protoboard* com uso do osciloscópio.

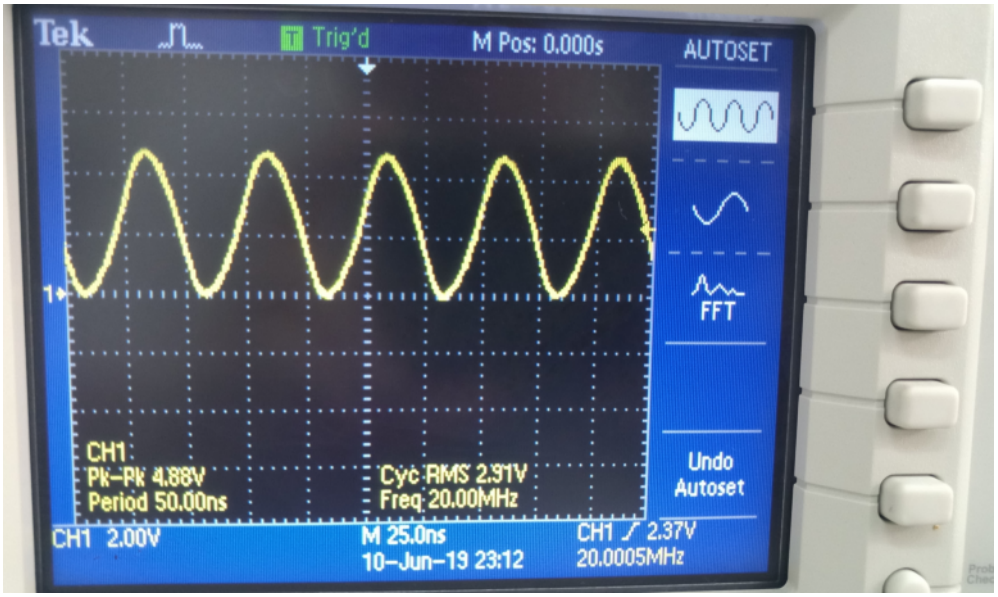


Fonte: O Autor.

Porém, devido a interferência causada pela mesma em relação ao cristal oscilador, que gerava uma frequência de somente 60 hertz, como visto na Figura 21, foi necessário soldar os componentes em uma placa de circuito ponteada, que resultou em funcionamento correto,

como pode ser visto em teste de bancada com osciloscópio na Figura 22. Os resultados foram obtidos por observação de funcionamento correto, tanto na simulação quanto em mundo real e o uso de osciloscópio e multímetro.

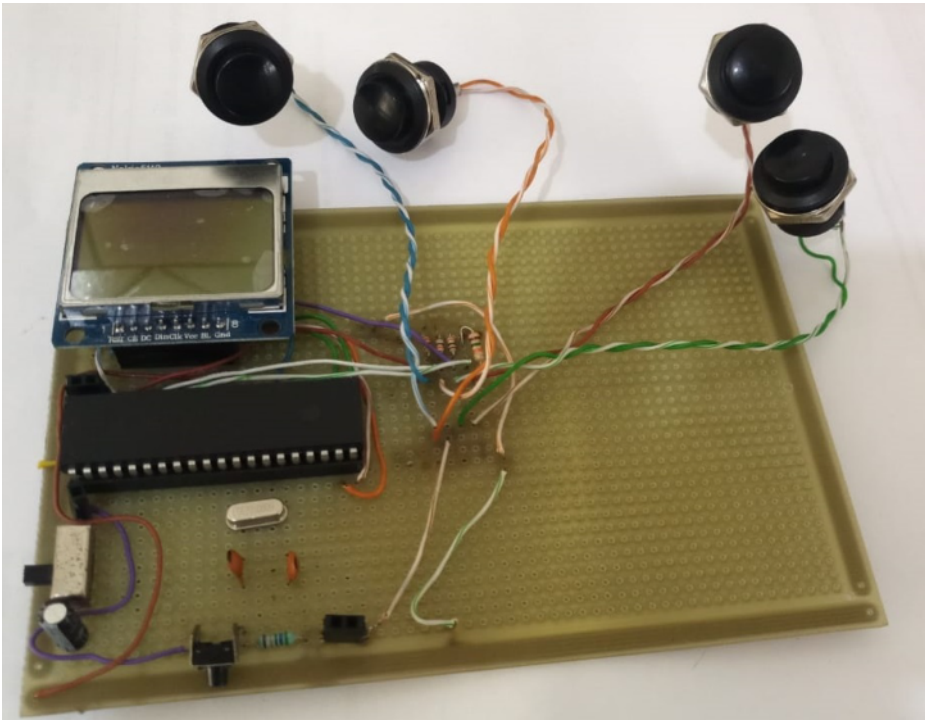
Figura 22: Teste em placa ponteada com uso do osciloscópio.



Fonte: O Autor.

O circuito de testes criado foi modificado uma vez para testar as duas possibilidades de entrada de usuário, já discutidas no desenvolvimento, e a opção mais simples, com o uso de 4 portas de E/S foi utilizada como pode ser visto na Figura 23.

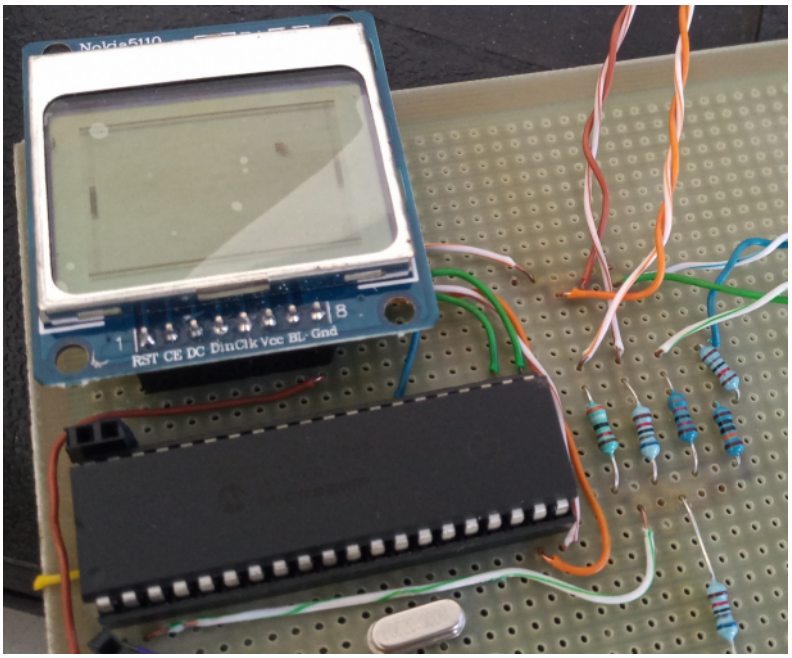
Figura 23: Protótipo de testes.



Fonte: O Autor.

A Figura 24 e a Figura 25 exibem a tela do jogo quando em andamento e com o placar.

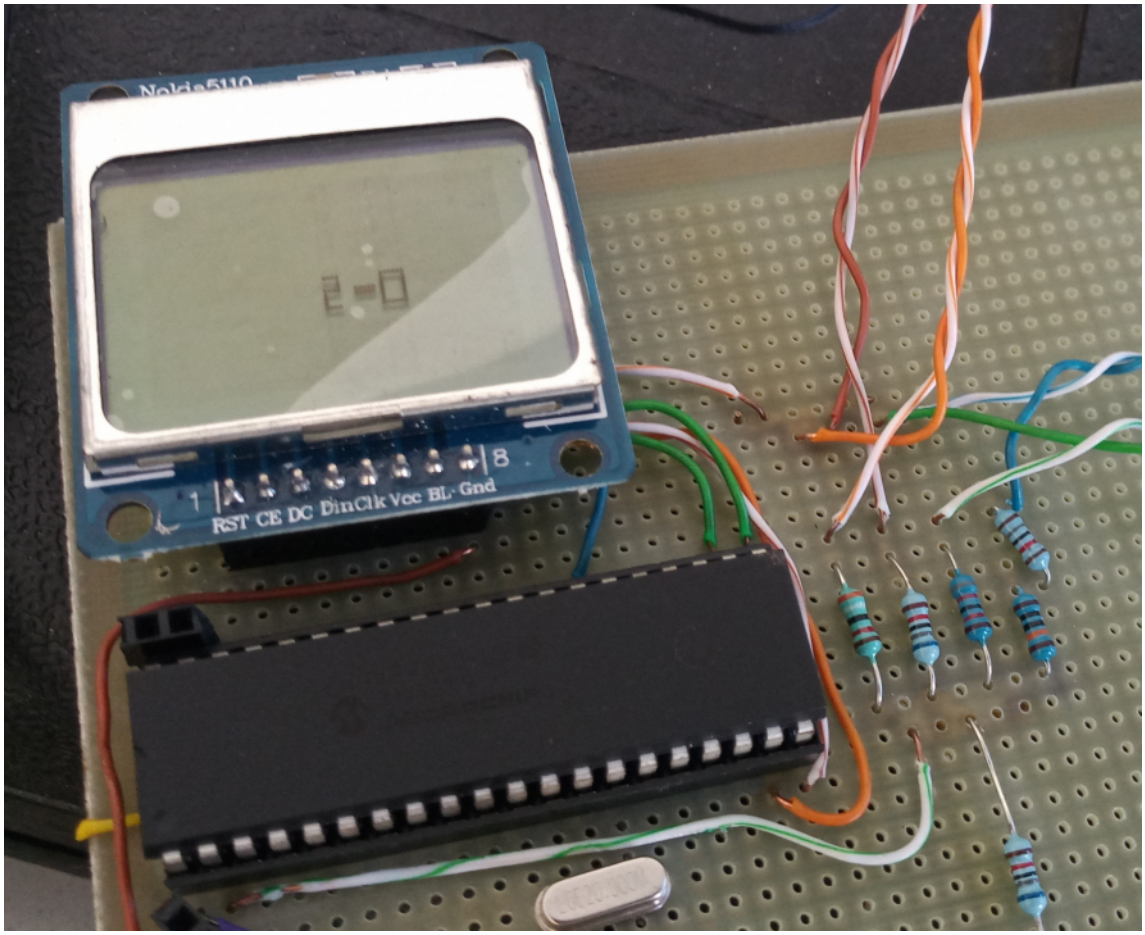
Figura 24: Protótipo de testes durante andamento do jogo.



Fonte: O Autor.

Os pares de fios branco e marrom, branco e laranja, branco e verde e branco e azul vistos nas figuras, são as saídas para os botões que os jogadores utilizaram para controlar suas barras.

Figura 25: Protótipo de testes exibindo placar.



Fonte: O Autor.

7 Conclusão e Trabalhos Futuros

"Uma vida não questionada não merece ser vivida."
Platão

O objetivo proposto neste trabalho foi a construção de uma estrutura *hardware* e *software* que se assemelhe ao jogo *Pong*, em escala reduzida, mas mantendo a maior fidelidade ao jogo original possível. Foi utilizado o microcontrolador PIC18F4550 e o *display* 5110 com controlador PCD8544, o compilador utilizado foi o C18.

O jogo *pong* foi implementado sem falhas, respeitando as regras de colisão com parede e barra do jogador. A biblioteca desenvolvida pelo autor, para controle da tela, funcionou adequadamente. Entretanto ocorreu a necessidade, por conta do microcontrolador PIC18F4550, de dividir o vetor do *display* originalmente de 504 *bytes* em dois de 256 e 248 *bytes* respectivamente, o que não atrapalhou o funcionamento do conjunto. Todos os botões e interação com usuário respondem rapidamente e são fáceis de utilizar para controle das barras do jogador.

Com todas as partes de *hardware* e *software* implementadas e testadas, verificamos que a proposta feita no início deste trabalho foi alcançada.

7.1 Trabalhos Futuros

- Uso de bateria e circuito de carga para tornar portátil;
- Uso de um *display* OLED (*Organic Light Emissor Diode*) para melhor qualidade gráfica;
- Expandir a biblioteca para gerar imagens a partir de arquivos;
- Uso de outro microcontrolador que não seja necessário dividir o vetor;
- Uso de outro compilador para o mesmo microcontrolador para melhor velocidade do jogo;
- Utilizar a biblioteca com o *display* para outras aplicações;

Referências

Adafruit. *Adafruit PCD8544 Nokia 5110 LCD library*. 2015. Disponível em: <https://www.arduinolibraries.info/libraries/adafruit-pcd8544-nokia-5110-lcd-library>. Citado nas páginas 17, 25, 28, 32, 33, 34 e 35.

Andrés F. Restrepo-Alvarez. *Estructura básica de programación usando un sistema microcontrolado: Juego de Ping-Pong*. 2014. Disponível em: <http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-30332014000200004&lng=en&tlng=en>. Citado nas páginas 7 e 21.

Arduino e Cia. *Display LCD Nokia 5110*. 2013. Disponível em: <https://www.arduinoecia.com.br/2013/05/display-lcd-nokia-5110.html>. Citado na página 18.

Benj Edwards. *History of Displays*. 2010. Disponível em: <https://www.pcworld.com/article/209224/displays/historic-monitors-slideshow.html#slide1>. Citado nas páginas 12 e 16.

Bob Davis. *BREAKING THE ARDUINO SPEED LIMIT*. 2014. Disponível em: <https://www.nutsvolts.com/magazine/article/March2014_Davis>. Citado nas páginas 7 e 23.

Dan Gravatt. *THE RETRO PIC SINGLE-BOARD COMPUTER*. 2017. Disponível em: <https://www.nutsvolts.com/magazine/article/January2017_Retro-PIC-Single-Board-Computer>. Citado nas páginas 7 e 22.

Dysfunctional Technologies, Inc. *Pong-PIC12f1840*. 2012. Disponível em: <http://dysfunctionaltechnologies.blogspot.com/2012/12/pong-pic12f1840.html>. Citado nas páginas 13 e 17.

Eric Rothfus. *BUILD THE RETROGAME*. 2017. Disponível em: <https://www.nutsvolts.com/magazine/article/build_the_retrogame>. Citado nas páginas 7 e 24.

Exsto Tecnologia. *XM118 - Banco de Ensaios para Microcontroladores PIC18F4550*. 2018. Disponível em: <http://exsto.com.br/img/produto_download/10590510e38974f57e7efc2c3cacb552.pdf>. Citado nas páginas 20 e 30.

FelipeFlop. *Display LCD Nokia 5110 Backlight Azul*. 2019. Disponível em: <https://www.filipeflop.com/produto/display-lcd-nokia-5110-backlight-azul/>. Citado na página 26.

Ivan L. M. Ricarte. *Compiladores*. 2003. Disponível em: <http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node37.html>. Citado nas páginas 12 e 16.

Microchip. *PIC16F628A*. 2003. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf>. Citado na página 35.

Microchip. *PIC18F4550*. 2006. Disponível em: <https://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf>. Citado nas páginas 18, 26, 31 e 35.

Microchip. *PIC16F628A*. 2007. Disponível em: <http://web.mit.edu/6.115/www/document/16f628.pdf>. Citado na página 35.

Nobuo Oki, Suely Cunha Amaro Mantovan. *Microcontroladores PIC*. 2013. Disponível em: <http://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/microcontroladores-_pic-1.pdf>. Citado na página 11.

Rickard Gunee. *PIC-Pong*. 2003. Disponível em: <http://www.rickard.gunee.com/projects/video/pic/pong.php>. Citado nas páginas 13 e 17.

SOUZA, D. J. d. Desbravando o pic. In: *Desbravando o PIC: Ampliado e atualizado para PIC16F628A*. Tatuapé, SP, Brasil: Editora Érica, 2008. p. 21–23. ISBN 978-85-7194-867-9. Citado na página 15.

Sparkfun. *Philips PCD8544 Datasheet*. 1999. Disponível em: <https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf>. Citado nas páginas 17, 31 e 34.

Sparkfun. *Serial Peripheral Interface (SPI)*. 2018. Disponível em: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all>. Citado nas páginas 29 e 30.

The Centre for Computing History, Cambridge. *Atari PONG*. 2010. Disponível em: <http://www.computinghistory.org.uk/det/4007/Atari-PONG/>. Citado na página 11.

Yousif Ismaill Al-Mashhadany. *Design and Implementation of Electronic Control Trainer with PIC Microcontroller*. 2012. Disponível em: <https://www.researchgate.net/publication/272673295_Design_and_Implementation_of_Electronic_Control_Trainer_with_PIC_Microcontroller>. Citado na página 19.

8 Apêndices

8.1 APÊNDICE A

```c
#include <p18f4550.h>
#include <delays.h>
#include <sw_spi.h>
#include <adc.h>
#include <timers.h>

//configuracoes
#pragma config PLLDIV = 5 //PLL para 20MHZ
#pragma config CPUDIV = OSC1_PLL2 // USB 48mhz
#pragma config FOSC = HS // Fosc = 20mhz; Tcy = 200ns;
#pragma config FCMEN = OFF
#pragma config IESO = OFF

#pragma config WDT = OFF //Watchdog desligado
#pragma config PBADEN = OFF //pinos do portb digitais
#pragma config LVP = OFF // desabilita gravacao em baixa tencao
#pragma config XINST = OFF

#pragma idata ms1
unsigned char buffer[256] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

#pragma idata ms2
unsigned char buffer2[248] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};



```

```
59
60 #define BLACK 1
61 #define WHITE 0
62
63 #define LCDWIDTH 84
64 #define LCDHEIGHT 48
65
66 #define PCD8544_POWERDOWN 0x04
67 #define PCD8544_ENTRYMODE 0x02
68 #define PCD8544_EXTENDEDINSTRUCTION 0x01
69
70 #define PCD8544_DISPLAYBLANK 0x0
71 #define PCD8544_DISPLAYNORMAL 0x4
72 #define PCD8544_DISPLAYALLON 0x1
73 #define PCD8544_DISPLAYINVERTED 0x5
74
75 // H = 0
76 #define PCD8544_FUNCTIONSET 0x20
77 #define PCD8544_DISPLAYCONTROL 0x08
78 #define PCD8544_SETYADDR 0x40
79 #define PCD8544_SETXADDR 0x80
80
81 // H = 1
82 #define PCD8544_SETTEMP 0x04
83 #define PCD8544_SETBIAS 0x10
84 #define PCD8544_SETVOP 0x80
85
86 #define _BV(x) (1 << (x))
87
88
89 #define _rst PORTDbits.RD0
90 #define _cs PORTDbits.RD1
91 #define _dc PORTDbits.RD2
92 #define _din PORTDbits.RD3
93 #define _sclk PORTDbits.RD4
94 #define _bot1 PORTBbits.RB1
95 #define _bot3 PORTBbits.RB2
96 #define _bot2 PORTBbits.RB5
97 #define _bot4 PORTBbits.RB4
98
99 unsigned char dado,d,adress,pontos1,pontos2;
100 char i,cubo,x,y,dx,dy,xx,yy,yp1,yp2,dyp1,dyp2;
101
102 void escreveSPI(unsigned char data){
103
104         for(i = 7;i>=0;i--){
105                 d = data >> i;       //shift left na informacao i vezes;
106                 d = d & 0b00000001;  //faz operacao logica E para saber
```

```
107                 if(d)                   //se bit e 1 ou 0;
108                         _din = 1;               //se for 1 seta a porta de
                               entrada
109                 else                      //de dados como 1, se nao como 0;
110                         _din = 0;
111
112                 _sclk = 1;              //pulsa o clock para passar o bit
113                 _sclk = 0;              //de forma serial, repete a
                       operacao
114                                                             //8 vezes para
                                                                   passar 1
                                                                   byte
115         }
116 }
117
118
119 int xUpdateMin, xUpdateMax, yUpdateMin, yUpdateMax;
120
121
122
123 void updateBoundingBox(int xmin, int ymin, int xmax, int ymax) {
124
125 if (xmin < xUpdateMin) xUpdateMin = xmin;
126 if (xmax > xUpdateMax) xUpdateMax = xmax;
127 if (ymin < yUpdateMin) yUpdateMin = ymin;
128 if (ymax > yUpdateMax) yUpdateMax = ymax;
129
130 }
131
132 void command(unsigned char c) {
133 _dc = 0;          //Seta pino DC em 0;
134 _cs = 0;        //Seta pino CS em 0;
135 escreveSPI(c);  //Envia comando por SPI;
136 _cs = 1;        //Seta pino CS em 1;
137 }
138
139 void data(unsigned char c) {
140 _dc = 1;          //Seta pino DC em 1;
141 _cs = 0;        //Seta pino CS em 0;
142 escreveSPI(c);  //Envia o dado por SPI;
143 _cs = 1;        //Seta pino CS em 1;
144 }
145
146 void setContrast(char val) {
147         if (val > 0x7f) {
148                 val = 0x7f;
149         }
150 command(PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION );
```

```
151 command( PCD8544_SETVOP | val);
152 command(PCD8544_FUNCTIONSET);
153 }
154
155 void display(void) {
156 int col, maxcol, p;
157
158         for(p = 0; p < 6; p++) {
159         // checa se esse bloco faz parte da atualizacao
160                 if ( yUpdateMin >= ((p+1)*8) ) {
161                         continue;   // se nao, pula
162                 }
163                 if (yUpdateMax < p*8) {
164                         break;
165                 }
166
167                 command(PCD8544_SETYADDR | p);
168
169                 col = xUpdateMin;
170                 maxcol = xUpdateMax;
171
172                 command(PCD8544_SETXADDR | col);
173
174                 _dc = 1;
175                 _cs = 0;
176                 for(; col <= maxcol; col++) {
177                         if(((LCDWIDTH*p)+col)<256)
178                                 escreveSPI(buffer[(LCDWIDTH*p)+col]);
179                         else
180                                 escreveSPI(buffer2[((LCDWIDTH*p)+col)
                                    -256]);
181                 }
182                 _cs = 1;
183         }
184 command(PCD8544_SETYADDR);
185 xUpdateMin = LCDWIDTH - 1;
186 xUpdateMax = 0;
187 yUpdateMin = LCDHEIGHT-1;
188 yUpdateMax = 0;
189 }
190
191 void clearDisplay(){
192 int i;
193         for(i = 0; i < 256; i++)
194                 buffer[i] = 0x00;
195         for(i = 0; i < 248; i++)
196                 buffer2[i] = 0x00;
197
```

```
198 updateBoundingBox(0, 0, LCDWIDTH-1, LCDHEIGHT-1);
199
200 }
201
202
203
204
205 void begin(int contrast, int bias) {
206
207         if (_rst > 0) {
208                 _rst = 0;
209                 Delay10KTCYx(250);
210                 _rst = 1;
211         }
212
213 // Entra em modo extendido
214 command(PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION );
215
216 // seta LCD bias
217 command(PCD8544_SETBIAS | bias);
218
219 // seta contraste maximo
220 if (contrast > 0x7f)
221 contrast = 0x7f;
222
223 command( PCD8544_SETVOP | contrast);
224 // Determinado Experimentalmente
225
226 // Modo normal
227 command(PCD8544_FUNCTIONSET);
228
229 // Seta em Normal
230 command(PCD8544_DISPLAYCONTROL | PCD8544_DISPLAYNORMAL);
231
232 updateBoundingBox(0, 0, LCDWIDTH-1, LCDHEIGHT-1);
233
234 display();
235 }
236
237 void drawPixel(int x, int y, int color) {
238
239         if ((x < 0) || (x >= LCDWIDTH) || (y < 0) || (y >= LCDHEIGHT))
240                 return;
241
242 // x coluna
243         if((x+ (y/8)*LCDWIDTH)<256){
244                 if (color)
245                         buffer[x+(y/8)*LCDWIDTH] |= _BV(y%8);
```

```
246                     else
247                             buffer[x+(y/8)*LCDWIDTH] &= ~_BV(y%8);
248             }else{
249                     if (color)
250                             buffer2[(x+(y/8)*LCDWIDTH)-256] |= _BV(y%8);
251                     else
252                             buffer2[(x+(y/8)*LCDWIDTH)-256] &= ~_BV(y%8);
253             }
254 updateBoundingBox(x,y,x,y);
255 }
256
257 void imprimeCubo(int x,int y){
258             if(cubo){
259                     drawPixel(y,x,BLACK);
260                     drawPixel(y,x+1,BLACK);
261                     drawPixel(y,x+2,BLACK);
262                     drawPixel(y+1,x,BLACK);
263                     drawPixel(y+1,x+1,BLACK);
264                     drawPixel(y+1,x+2,BLACK);
265                     drawPixel(y+2,x,BLACK);
266                     drawPixel(y+2,x+1,BLACK);
267                     drawPixel(y+2,x+2,BLACK);
268             }else{
269                     drawPixel(y,x,BLACK);
270                     drawPixel(y,x+1,BLACK);
271                     drawPixel(y,x+2,BLACK);
272                     drawPixel(y+1,x,BLACK);
273                     drawPixel(y+1,x+2,BLACK);
274                     drawPixel(y+2,x,BLACK);
275                     drawPixel(y+2,x+1,BLACK);
276                     drawPixel(y+2,x+2,BLACK);
277             }
278 cubo = !cubo;
279 }
280
281 void imprimeBorda(void){
282             for(yy = 0;yy<84;yy++){
283                     drawPixel(yy,0,BLACK);
284                     drawPixel(yy,47,BLACK);
285             }
286 }
287
288 void imprimeP1(int x){
289             for(i = 0;i<9;i++){
290                     drawPixel(2,x+i,BLACK);
291                     drawPixel(3,x+i,BLACK);
292             }
293 }
```

```
294
295 void imprimeP2(int x){
296         for(i = 0;i<9;i++){
297                 drawPixel(80,x+i,BLACK);
298                 drawPixel(81,x+i,BLACK);
299         }
300 }
301
302 void zero(int x,int y){
303         for(i=0;i<8;i++){
304                 drawPixel(y,x+i,BLACK);
305                 drawPixel(y+5,x+i,BLACK);
306         }
307         for(i=0;i<4;i++){
308                 drawPixel(y+1+i,x,BLACK);
309                 drawPixel(y+1+i,x+7,BLACK);
310         }
311 }
312
313 void um(int x,int y){
314         for(i=0;i<8;i++)
315                 drawPixel(y+3,x+i,BLACK);
316 drawPixel(y+2,x,BLACK);
317 drawPixel(y+2,x+1,BLACK);
318 drawPixel(y+1,x+1,BLACK);
319 drawPixel(y+1,x+2,BLACK);
320 drawPixel(y,x+2,BLACK);
321 }
322
323 void dois(int x,int y){
324         for(i=0;i<4;i++)
325                 drawPixel(y+i,x,BLACK);
326         for(i=0;i<4;i++)
327                 drawPixel(y+i,x+3,BLACK);
328         for(i=0;i<4;i++)
329                 drawPixel(y+i,x+7,BLACK);
330 drawPixel(y+3,x+1,BLACK);
331 drawPixel(y+3,x+2,BLACK);
332 drawPixel(y,x+4,BLACK);
333 drawPixel(y,x+5,BLACK);
334 drawPixel(y,x+6,BLACK);
335 }
336
337 void tres(int x,int y){
338         for(i=0;i<4;i++){
339                 drawPixel(y+i,x,BLACK);
340                 drawPixel(y+i,x+3,BLACK);
341                 drawPixel(y+i,x+7,BLACK);
```

```
342         }
343         for(i=0;i<8;i++)
344                 drawPixel(y+4,x+i,BLACK);
345 }
346
347 void pontuacao(int x,int y){
348 clearDisplay();
349         if(pontos1 == 0){
350                 zero(x,y);
351         }else if(pontos1 == 1){
352                 um(x,y);
353         }else if(pontos1 == 2){
354                 dois(x,y);
355         }else if(pontos1 == 3){
356                 tres(x,y);
357         }
358         for(i = 0;i<5;i++){
359                 drawPixel(y+7+i,x+3,BLACK);
360                 drawPixel(y+7+i,x+4,BLACK);
361         }
362
363         if(pontos2 == 0){
364                 zero(x,y+13);
365         }else if(pontos2 == 1){
366                 um(x,y+13);
367         }else if(pontos2 == 2){
368                 dois(x,y+13);
369         }else if(pontos2 == 3){
370                 tres(x,y+13);
371         }
372 display();
373 x = 20;
374 y = 40;
375 Delay10KTCYx(500);
376 Delay10KTCYx(500);
377 Delay10KTCYx(500);
378 }
379
380 void imprimeF1(void){
381 clearDisplay();
382 x = y = 20;
383         for(i=0;i<8;i++)
384                 drawPixel(y,x+i,BLACK);
385 drawPixel(y+1,x,BLACK);
386 drawPixel(y+2,x,BLACK);
387 drawPixel(y+1,x+4,BLACK);
388 drawPixel(y+2,x+4,BLACK);
389 drawPixel(y+3,x+1,BLACK);
```

```
390 drawPixel(y+3,x+2,BLACK);
391 drawPixel(y+3,x+3,BLACK);
392
393         for(i=0;i<8;i++)
394                 drawPixel(y+9,x+i,BLACK);
395 drawPixel(y+8,x,BLACK);
396 drawPixel(y+8,x+1,BLACK);
397 drawPixel(y+7,x+1,BLACK);
398 drawPixel(y+7,x+2,BLACK);
399 drawPixel(y+6,x+2,BLACK);
400
401 drawPixel(y+11,x,BLACK);
402 drawPixel(y+11,x+1,BLACK);
403 drawPixel(y+11,x+2,BLACK);
404 drawPixel(y+18,x,BLACK);
405 drawPixel(y+18,x+1,BLACK);
406 drawPixel(y+18,x+2,BLACK);
407 drawPixel(y+12,x+2,BLACK);
408 drawPixel(y+12,x+3,BLACK);
409 drawPixel(y+12,x+4,BLACK);
410 drawPixel(y+17,x+2,BLACK);
411 drawPixel(y+17,x+3,BLACK);
412 drawPixel(y+17,x+4,BLACK);
413 drawPixel(y+13,x+4,BLACK);
414 drawPixel(y+13,x+5,BLACK);
415 drawPixel(y+13,x+6,BLACK);
416 drawPixel(y+16,x+4,BLACK);
417 drawPixel(y+16,x+5,BLACK);
418 drawPixel(y+16,x+6,BLACK);
419 drawPixel(y+14,x+6,BLACK);
420 drawPixel(y+14,x+7,BLACK);
421 drawPixel(y+15,x+6,BLACK);
422 drawPixel(y+15,x+7,BLACK);
423 display();
424 while(1);
425 }
426
427 void imprimeF2(void){
428 clearDisplay();
429 x = y = 20;
430         for(i=0;i<8;i++)
431                 drawPixel(y,x+i,BLACK);
432 drawPixel(y+1,x,BLACK);
433 drawPixel(y+2,x,BLACK);
434 drawPixel(y+1,x+4,BLACK);
435 drawPixel(y+2,x+4,BLACK);
436 drawPixel(y+3,x+1,BLACK);
437 drawPixel(y+3,x+2,BLACK);
```

```
438 drawPixel(y+3,x+3,BLACK);
439
440         for(i=0;i<4;i++)
441                 drawPixel(y+5+i,x,BLACK);
442         for(i=0;i<4;i++)
443                 drawPixel(y+5+i,x+3,BLACK);
444         for(i=0;i<4;i++)
445                 drawPixel(y+5+i,x+7,BLACK);
446 drawPixel(y+8,x+1,BLACK);
447 drawPixel(y+8,x+2,BLACK);
448 drawPixel(y+5,x+4,BLACK);
449 drawPixel(y+5,x+5,BLACK);
450 drawPixel(y+5,x+6,BLACK);
451
452 drawPixel(y+11,x,BLACK);
453 drawPixel(y+11,x+1,BLACK);
454 drawPixel(y+11,x+2,BLACK);
455 drawPixel(y+18,x,BLACK);
456 drawPixel(y+18,x+1,BLACK);
457 drawPixel(y+18,x+2,BLACK);
458 drawPixel(y+12,x+2,BLACK);
459 drawPixel(y+12,x+3,BLACK);
460 drawPixel(y+12,x+4,BLACK);
461 drawPixel(y+17,x+2,BLACK);
462 drawPixel(y+17,x+3,BLACK);
463 drawPixel(y+17,x+4,BLACK);
464 drawPixel(y+13,x+4,BLACK);
465 drawPixel(y+13,x+5,BLACK);
466 drawPixel(y+13,x+6,BLACK);
467 drawPixel(y+16,x+4,BLACK);
468 drawPixel(y+16,x+5,BLACK);
469 drawPixel(y+16,x+6,BLACK);
470 drawPixel(y+14,x+6,BLACK);
471 drawPixel(y+14,x+7,BLACK);
472 drawPixel(y+15,x+6,BLACK);
473 drawPixel(y+15,x+7,BLACK);
474 display();
475 while(1);
476 }
477
478 void main(void){
479 TRISB = 0x00;
480 TRISD = 0x00;
481 INTCON2bits.NOT_RBPU = 0;//resistores internos da PortB ativos
482 _rst = 0;
483 Delay10KTCYx(250);
484 _rst = 1;
485 begin(46,4);
```

```
display();
yp1 = yp2 = x = 20;
y = 42;
dyp1 = dyp2 = pontos1 = pontos2 = 0;
dx = dy = 1;
Delay10KTCYx(500);

        while(1){
                if(_bot1){
                        dyp1 = 2;
                }else if(_bot2){
                        dyp1 = -2;
                }else{
                        dyp1 = 0;
                }

                if(_bot3){
                        dyp2 = 2;
                }else if(_bot4){
                        dyp2 = -2;
                }else{
                        dyp2 = 0;
                }

        clearDisplay();
        imprimeBorda();
        imprimeP1(yp1);
        imprimeP2(yp2);
        imprimeCubo(x,y);
        display();
        Delay10KTCYx(3);

                if(x>43){
                        dx=-dx;
                }else if(x<2){
                        dx=-dx;
                }

                if(y>79){
                        dy=-dy;
                }else if(y<2){
                        dy=-dy;
                }

                if(pontos1==3)
                        imprimeF1();
                if(pontos2==3)
                        imprimeF2();
```

```
534
535                 if(y<=3){//P1
536                         if(x>=yp1 && x<=yp1+8){
537                                 dy = -dy;
538                         }else{
539                                 pontos2++;
540                                 pontuacao(23,40);
541                                 x=23;y=40;
542                         }
543                 }
544
545                 if(y+3>=80){//P2
546                         if(x>=yp2 && x<=yp2+8){
547                                 dy = -dy;
548                         }else{
549                                 pontos1++;
550                                 pontuacao(23,40);
551                                 x=23;y=40;
552                         }
553                 }
554
555         x+=dx;
556         y+=dy;
557         yp1 += dyp1;
558         yp2 += dyp2;
559                 if(yp1<=0) yp1 = 0;
560                 if(yp1>=38) yp1 = 38;
561                 if(yp2<=0) yp2 = 0;
562                 if(yp2>=38) yp2 = 38;
563
564                 if(x < 0 || x >48)
565                         x=23;
566                 if(y < 0 || y > 83)
567                         y = 40;
568         }
569 }
```

código 3: Código completo

```
1  void lerEntradas(void){
2          SetChanADC(ADC_CH0);
3          ConvertADC();
4          while(BusyADC());
5          temp = ReadADC();
6          if(temp >=636 && temp <=656){//1
7                  up1 = 1;up2 = down1 = down2 = 0;
```

```
8           }else if(temp >=490 && temp <=550){//2
9                   down1 = 1;up1 = up2 = down2 = 0;
10          }else if(temp >=410 && temp <=489){//3
11                  up2 = 1;up1 = down1 = down2 = 0;
12          }else if(temp >=310 && temp <=409){//4
13                  down2 = 1;up2 = down1 = up1 = 0;
14          }else if(temp >=610 && temp <=635){//2 e 4
15                  down1 = down2 = 1;up2 = up1 = 0;
16          }else if(temp >=657 && temp <=675){//2 e 3
17                  down1 = up2 = 1;down2 = up1 = 0;
18          }else if(temp >=705 && temp <=717){//1 e 4
19                  up1 = down2 = 1;down2 = up1 = 0;
20          }else if(temp >=718 && temp <=729){//2 e 3 e 4 == 2
21                  down1 = 1;up1 = up2 = down2 = 0;
22          }else if(temp >=730 && temp <=745){//1 e 3
23                  down1 = down2 = 0;up2 = up1 = 1;
24          }else if(temp >=760 && temp <=778){//1 e 3 e 4 == 1
25                  up1 = 1;up2 = down1 = down2 = 0;
26          }else if(temp >=779 && temp <=791){//1 e 2 e 4 == 4
27                  down2 = 1;up2 = down1 = up1 = 0;
28          }else if(temp >=792 && temp <=830){//1 e 2 e 3 == 3
29                  up2 = 1;up1 = down1 = down2 = 0;
30          }else{
31                  down2 = up2 = down1 = up1 = 0;
32          }
33  }
```

código 4: Código leitura analógica