Contador usando PIC16F628A

Gabriela G. Santos, Vitor Bruno de O. Barth

IFMT – DAI

gabi.gsantos@hotmail.com, vbob@vbob.com.br

***Resumo* – Este artigo descreve o projeto e a confecção de um contador, feito para um invólucro de pequenas dimensões, baseado no microcontrolador PIC16F628A.**

I. INTRODUÇÃO

Pequenos contadores manuais são utilizados em diversas aplicações: contagem de fluxo pessoas em comércios ou entrada de grupos em locais públicos, contagem de chutes em lutas de *Taekwondo*, *etc.*

Estes contadores, preferencialmente, precisam ter design ergonômico e dimensões pequenas. Além da ergonomia, espera-se que a resposta seja imediata e precisa, sendo ativado ao menor toque e não contando com duplicidade.

O consumo de energia também deve ser levado em consideração: as informações que estão sendo armazenadas não possuem redundância, logo a falta de bateria causaria a perda destas informações.

Sendo assim, este trabalho apresenta um projeto de um contador de pequenas dimensões, que foi projetado sobre o microcontrolador PIC16F628A, buscando baixo custo e alta eficiência energética.

As próximas seções são estruturadas como segue: seção 2 descreve os equipamentos utilizados, enquanto os códigos e o algoritmo do *software* são descritos na seção 3. A seção 4 apresenta a construção do equipamento, o resultado atingido e as dificuldades encontradas, e na seção 5, por fim, são apresentadas as conclusões e comentários acerca do produto.

II. EQUIPAMENTOS

Buscando o menor custo e menores dimensões da placa, foi utilizado o microcontrolador PIC16F628A como base para o projeto.

A interface com o mundo externo ocorreu por meio de dois botões do tipo *push* momentâneo, redondos de 6mm, apresentado na Figura 1.

Figura 1 - Botão Push Utilizado. Fonte: Surven

O encapsulamento do conjunto é feito por um hexaedro de 63mm x 50mm x 27mm, e 1.5mm de espessura da casca, com tampa inferior removível, apresentado na Figura 2.

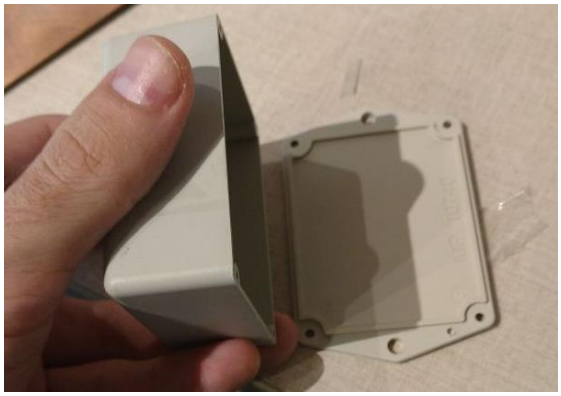

Figura 2 - Encapsulamento. Desenvolvido pelos Autores.

A exibição de dados será feita em um *display Titan Micro Electronics* TM1637, de 7 segmentos e 4 dígitos, com comunicação I²C.

A confecção da placa de circuito impresso utilizou uma peça de cobre de 250mm x 250mm, que foi posteriormente corroída com o circuito desejado, e cortada em dimensões suficientes para ser fixada no invólucro especificado. Mais informações da confecção da placa são apresentadas na seção 4.

A alimentação é feita através de 3 pilhas AAA, e pode ser controlada por meio de um *switch on-off*.

III. SOFTWARE

O *software* foi desenvolvido através do compilador PIC C Compiler v5.15, e gravado por meio do gravador *PicKit 3* v1.00.

A contagem se baseou em um *loop*, que verificava continuamente o estado dos botões. Quando o botão de incremento é pressionado, o valor do contador é atualizado. Quando o botão de *reset* é pressionado, o contador assume o valor 0.

Por não possuir interface I²C nativa, esta foi implementada manualmente no PIC16F628A. O *datasheet* do *display* TM1637 possuí informações acerca do protocolo de comunicação utilizado, do tipo *half-duplex,* com confirmação de recebimento.

Os pulsos de *clock* e de dados são definidos por meio da alteração, em *software*, do estado de pinos genéricos do microcontrolador. O código completo é apresentado no Anexo 01.

IV. CONSTRUÇÃO

Como prova de conceito, foi inicialmente desenvolvido um protótipo em uma matriz de contato, apresentado na Figura 3.

Ao possuir o código funcional, deu-se início ao projeto da placa e posteriormente à corrosão desta, utilizando Percloreto de Ferro. O modelo da placa é apresentado na Figura 4, e o processo de corrosão apresentado nas Figuras 5, 6 e 7.

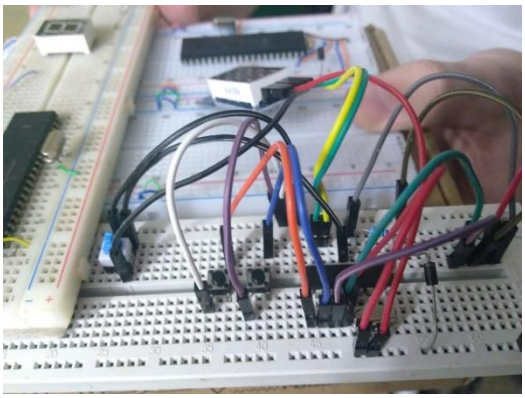

Figura 3 - Protótipo. Desenvolvido pelos Autores.

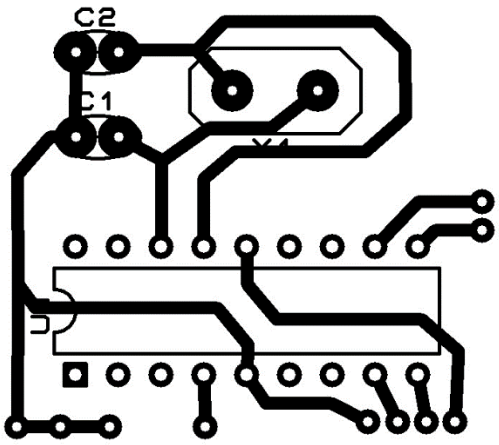

Figura 4 - Modelo da Placa. Desenvolvido pelos Autores.

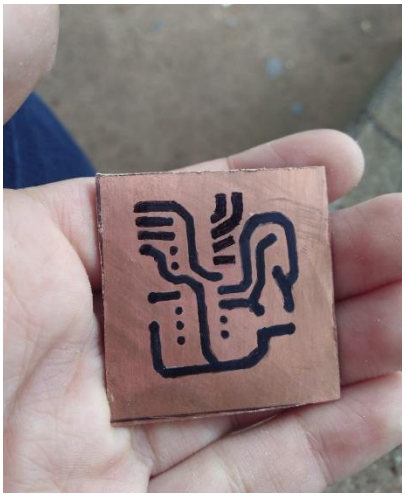

Figura 5 - Placa antes da Corrosão. Desenvolvido pelos Autores.

Figura 6 - Processo de Corrosão. Desenvolvido pelos Autores.

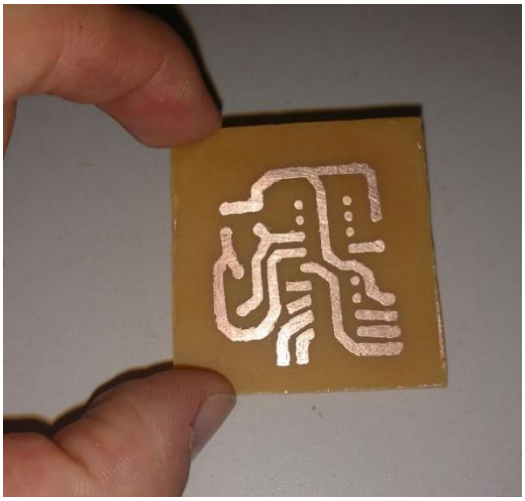

Figura 7 - Placa Corroída. Desenvolvido pelos Autores.

Após o processo de corrosão, a placa foi devidamente furada e os componentes nela soldados para que fosse possível seu funcionamento, apresentada na Figura 8.

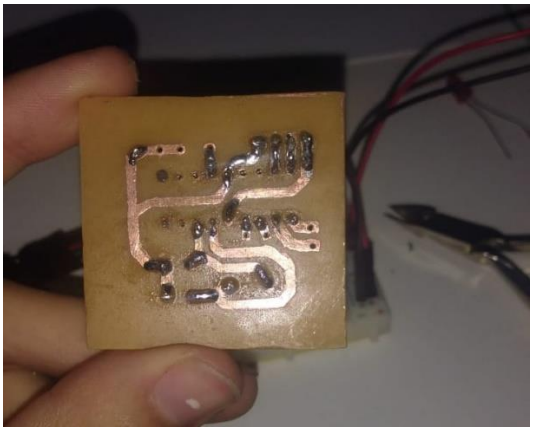

Figura 8 - Placa Soldada. Desenvolvido pelos Autores.

V. CONSIDERAÇÕES FINAIS

Por falha no projeto, o produto final não pode ser encapsulado e devidamente finalizado: como pode ser visto na Figura 4, o pino 4, que é possui função MCLR não possui ligação à alimentação, e deste modo o microcontrolador permanece desligado, reiniciando continuamente.

Tal função pode ser desativada em *software*, entretanto isso não foi feito, como é mostrado no código do Anexo 1.

Ainda que não tenha sido finalizado, este projeto foi, sem dúvidas, desafiador: a implementação de I²C em *software*, e posteriormente o planejamento e confecção da placa de circuito impresso foram uma experiência de aprendizado que não poderia ter sido atingida em sala, e as falhas que ocorreram no desenvolvimento também servem de aprendizado.

ANEXO I - SOFTWARE DO MICROCONTROLADOR

```
#include <16F628A.h>

#FUSES NOWDT
#FUSES NOLVP

#use delay(crystal=20000000)
#use FIXED_IO( B_outputs=PIN_B2,PIN_B3)

#include <float.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <stdio.h>

uint8_t digitToSegment[] = {
  // XGFEDCBA
  0b00111111, // 0 0x3f
  0b00000110, // 1 0x06
  0b01011011, // 2 0x5b
  0b01001111, // 3 0x4f
  0b01100110, // 4 0x66
  0b01101101, // 5 0x6d
  0b01111101, // 6 0x7d
  0b00000111, // 7 0x07
  0b01111111, // 8 0x7f
  0b01101111, // 9 0x6f
  0b01110111, // A 0x77
  0b01111100, // b 0x7c
  0b00111001, // C 0x39
  0b01011110, // d 0x5e
  0b01111001, // E 0x79
  0b01110001 // F 0x71
};

#define ADDR_AUTO 0x40 // 0b01000000
#define STARTADDR 0xc0 // 0b11000000
#define BRIGHTADD 0x80 // 0b10000000

#define pinClk PIN_B2
#define pinDIO PIN_B3
#define pinAdd PIN_B5
#define pinRst PIN_B4

void start();
void bitDelay();
void writeBit(uint8_t value);
uint8_t readAck();
uint8_t writeByte(uint8_t value);
void stop();
uint8_t encodeDigit(uint8_t digit);
void setSegments(uint8_t segments[], uint8_t length,
uint8_t pos);
```

```
void writeDigits(uint32_t value);

uint16_t counter = 0;

void main()
{
  port_B_pullups(0xFF);
  uint8_t data[] = {
    0b00111111,
    0b00111111,
    0b00111111,
    0b00111111
  };

  setSegments(data, 4, 0);

  while(TRUE)
  {
    if (input(pinAdd) == 0) {
      counter++;
      writeDigits(counter);
      while(input(pinAdd) == 0);
    }

    if (input(pinRst) == 0) {
      counter = 0;
      writeDigits(counter);
      while(input(pinRst) == 0);
    }
  }
}

void writeDigits(uint32_t value) {
  int units = value%10;
  int tenths = ((value-units)%100)/10;
  int thousands = value/1000;
  int hundreds = (value/100) - (thousands*10);


   uint8_t data[] = {
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000
  };

  data[0] = digitToSegment[thousands];
  data[1] = digitToSegment[hundreds];
  data[2] =  digitToSegment[tenths];
  data[3] = digitToSegment[units];
  setSegments(data, 4, 0);
}

void setSegments(uint8_t segments[], uint8_t length,
uint8_t pos) {
  start();
  writeByte(ADDR_AUTO);
  stop();

  // Write COMM2 + first digit address
  start();
  writeByte(STARTADDR + (pos & 0x03));

  // Write the data bytes
  for (uint8_t k=0; k < length; k++)
    writeByte(segments[k]);

  stop();

  // Write COMM3 + brightness
  start();
  writeByte(0b10001111);
  stop();
}

void start() {
  output_low(pinDIO);
  bitDelay();
}

void bitDelay() {
  delay_us(100);
}

void writeBit(uint8_t value) {
  output_low(pinClk);

  if (value) {
    output_high(pinDIO);
  } else {
    output_low(pinDIO);
  }

  bitDelay();
  output_high(pinClk);
  bitDelay();
}

uint8_t readAck() {
  output_low(pinClk);
  uint8_t readPro = input(pinDIO);
  output_high(pinClk);
  bitDelay();
  output_low(pinClk);
  return readPro;
}
```

```
uint8_t writeByte(uint8_t value) {
  uint8_t b = value;

  for (uint8_t i = 0; i< 8; i++) {
    writeBit(b & 0x01);
    b = b >> 1;
  }

  return readAck();
}

void stop() {
  output_low(pinDIO);
  bitDelay();
  output_high(pinClk);
  bitDelay();
  output_high(pinDIO);
  bitDelay();
}

uint8_t encodeDigit(uint8_t digit)
{
  return digitToSegment[digit & 0x0f];
}
```