Impressora para Placas de Circuito Impresso

Antônio Prado da Silva Júnior Faculdade Gama
Universidade de Brasília
Gama, Distrito Federal
contato.pradojr@gmail.com

Ítalo Barbosa Santos
Faculdade Unb Gama
Universidade de Brasília
Gama, Distrito Federal
Italo.b.s.35@gmail.com

I. RESUMO

A Impressora de Circuitos Impressos é uma ferramenta que confecciona o layout de Placas de Circuitos Impressos (PCI) direto na placa de Fenolite. Devido ao processo de fabricação das PCI ser feito de forma manual, o tempo gasto, os riscos e a praticidade são afetados. O intuito da Impressora de Circuitos Impressos é minimizar todos esses fatores apresentados anteriormente tornando o processo de fabricação mais automatizado.

II. INTRODUÇÃO

Atualmente o processo de fabricação artesanal das PCIs (Placas de Circuito Impresso) é um processo lento e extenso. Neste processo o usuário desenha o Layout diretamente na placa fenolite, para circuitos mais simples, utilizando uma caneta permanente. No outro processo o usuário planeja o layout em algum software em seguida faz a impressão em um papel específico, em seguida é necessário transferir a impressão para a placa de fenolite e isso é feito utilizando um ferro de passar sobre o desenho virado para a placa, após esse processo é necessário retirar o papel por meio de uma imersão em água, o que demanda tempo, terminado o processo de imersão o papel é retirado da placa ao final dos dois processos as placas são imersas em um ácido corrosivo que ao entrar em contato com a placa de fenolite corrói todo o espaço da placa exposta, não corroendo onde existe marcação da caneta ou a impressão do layout.

Apesar dos processos serem simples, são vários os problemas apresentados. No processo onde o usuário desenha diretamente na placa a precisão das trilhas no layout dependem da precisão do próprio usuário com a caneta, o que muita das vezes prejudica o processo, pois a precisão é algo instável devido a movimentos involuntários da mão. No segundo processo os problemas começam a partir da transferência do layout, que até então se encontra no papel, para a placa, esse processo tem que ser repetido várias vezes até que dê certo, pois com o uso do ferro de passar pode acontecer do desenho ficar borrado ou alguma trilha ficar partida fora o perigo de queimadura devido ao aquecimento da placa. Outro problema identificado neste segundo processo é que após a imersão da placa na água, além do tempo gasto, o papel não é solto totalmente sendo necessário que o usuário faça essa correção manualmente retirando os restos que sobraram do papel na placa.

III. OBJETIVOS

Este projeto tem como objetivo a fabricação de uma Impressora de Circuitos Impressos para solucionar os problemas anteriormente citados que une tanto a praticidade de desenhar o layout do circuito com a mão com a precisão do layout obtido através do método com o ferro de passar.

IV. DESCRIÇÃO DO HARDWARE

Para a confecção da Impressora de circuitos impressos foram usados dois motores de passo com driver ULN2003 que controlam a movimentação da plataforma nos eixos X e Y, um micro servo onde fica fixada a caneta que se movimenta no eixo Z e tem a função de gravar a imagem desejada na placa.

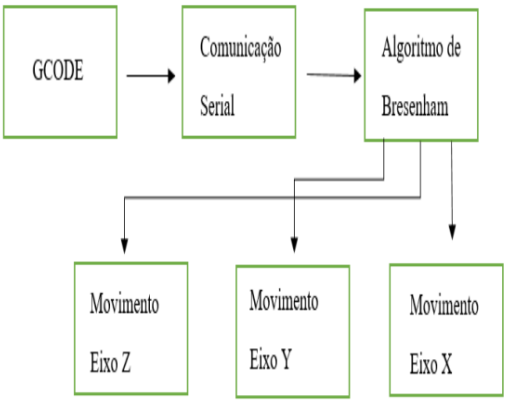

Figura 01 - Diagrama de Blocos do Sistema.

No MSP430 são usadas quatro entradas para cada motor de passo, uma entrada para o micro servo e os pinos VCC e GND. A alimentação do MSP430 neste projeto é feita através da porta USB e os motores de são alimentados por uma fonte externa de 5v.

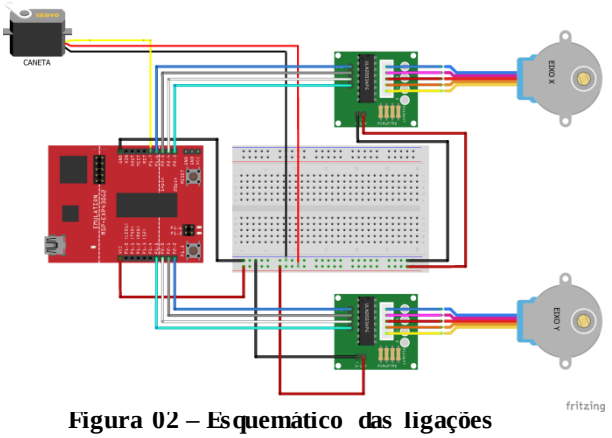

Figura 02 – Esquemático das ligações

A estrutura da Impressora de Circuitos Impressos é composta por uma base feita de madeirite de tamanho 42x32 cm, contornada por cantoneiras para um melhor acabamento, nesta base é fixado o eixo X, os suportes para o eixo Y com o driver do eixo Y, tais suportes são feitos de pedaços de madeira de tamanho 15x7 cm, um encaixe para o MSP430 feito de palitos de picolé e o driver do motor do eixo X. Os eixos X e Y foram reciclados de impressoras velhas e correspondem aos trilhos. Nestes trilhos foram fixados os motores de passo, cada motor foi encaixado na correia do trilho de forma que quando o motor girar a correia puxa o carrinho que está fixado nela.

Figura 03 – Partes utilizadas para a confecção da estrutura

O eixo Z consiste em um Servo Motor que controla a elevação da caneta quando o mesmo é ativado, assim, a caneta pode se movimentar sem que risque por onde passar. Para que isso aconteça um suporte foi adicionado ao carrinho do eixo Y assim a caneta movimenta apenas verticalmente.

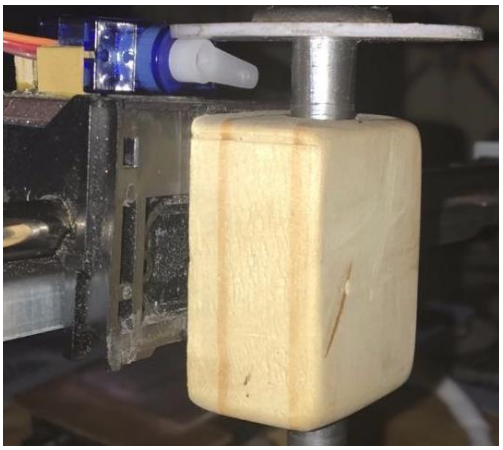

Figura 04 – Suporte para caneta eixo Z

Para a fixação das placas de fenolite no carrinho, foi feito um suporte de madeira e neste suporte é colocado fita dupla face 3M prendendo a placa de fenolite no suporte.

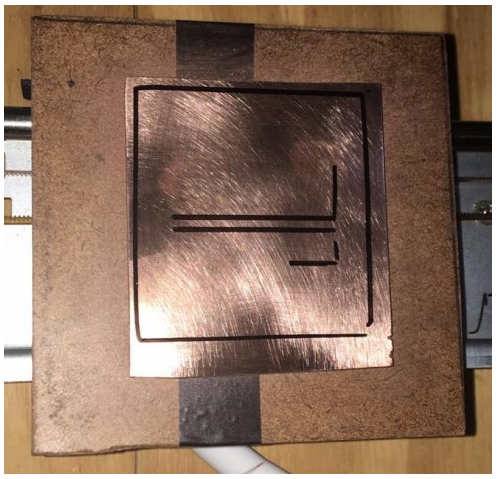

Figura 05 – Suporte para placa de fenolite

Em relação aos custos cada motor de passo é encontrado em média por R$ 20,00, o servo motor R$ 15,00 e o MSP430 R$ 30,00. Os eixos foram reciclados de impressoras quebradas, porém podem ser comprados por R$ 10,00 cada e a caneta custa R$ 5,00. As peças de madeira e cantoneiras foram reutilizadas de móveis quebrados e são fixadas na base com parafusos. A fonte usada é um carregador de celular de 5v.

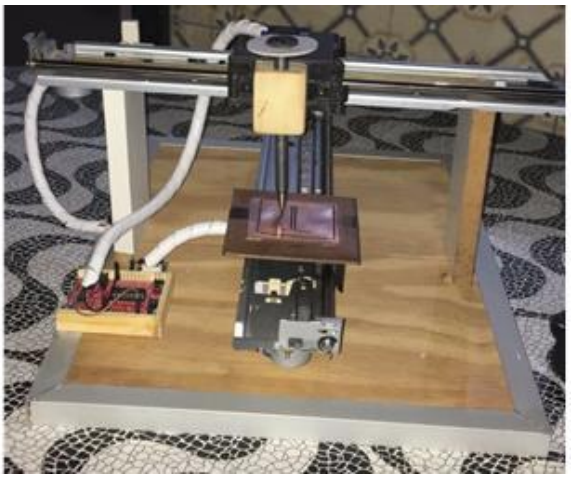

Figura 06 – Impressora de Circuitos Impressos

V. DESCRIÇÃO DO SOFTWARE

A ideia inicial para o funcionamento geral do projeto é importar coordenadas cartesianas e movimentar os eixos de acordo com os valores recebidos, sendo a posição inicial a que o programa deu se início.

O algoritmo criado para os motores se consiste em uma comparação com o valor atual e o valor recebido pelo GCode após verificar se houve ou não alguma mudança do valor daquela variável. Logo em seguida é feita uma nova verificação na grandeza da variável para determinar o sentido de giro dos motores, observa se que a variável correspondente ao eixo Z deve ser testada primeiro enquanto as variáveis dos eixos X e Y não precisam ter uma ordem específica entre si. Todo esse algoritmo foi colocado em uma função que recebe coordenadas em valores inteiros e movimenta os motores até os valores atuais e das coordenadas serem iguais e para movimentar os motores cada um tem uma função própria para isso.

O código desenvolvido para o servo foi o de um controle PWM utilizando o timer A no modo de comparação, para determinar o período de clock do PWM e o tempo necessário em nível alto para determinar os ângulos foi verificado o datasheet do micro servo[3]. O código foi feito utilizando um clock de 16MHz com a frequência dividida por 8. Por isso os valores do TA1CCR0 e TA1CCR1 são correspondentes a um período total de 20ms com duty cycle variado de acordo com os ângulos escolhidos para definir a subida e descida da caneta.

```
void LevantaZ()
{
    TA1CCR0 = 40000-1;
    TA1CCR1 = 2000;
    TA1CCTL1 = OUTMOD_7;
    TA1CTL   = TASSEL_2 + ID_3 + MC_1;
    P2SEL |= BIT2;
    P2OUT |=BIT2;
    TA1CCR1 = 2500;

     }
```

Figura 07 - Controle PWM do servo.

Como observado no código de controle do servo o registrador utilizado para o período timer é o TA1CCR0 isso foi porque o TA0CC0 já estava sendo utilizado em outra função, a do atraso, essa função é de extrema importância para o funcionamento dos motores de passo. A função atraso consiste

em uma contagem até determinado valor ao terminar essa contagem o registrador TAIFG seja ativado para fazer essa função basta criar um 'loop" até o registrador ser ativado.

```
void Atraso(volatile unsigned int t)
{
  TA0CTL    = TASSEL_2 + ID_3 + MC_1;
  TACCR0 = 2000-1;
  while(t--)
  {
    while((TACTL&TAIFG)==0);
    TACTL &= ~TAIFG;
  }
  TACTL = MC_0;
}
```

Figura 08 - Função Atraso.

Os motores são controlados por uma máquina de estados que vai setando os bits em pares para o acionamento em conjunto de suas bobinas, para isso foi criado um vetor com quatro posições onde cada posição ativa um par de bits do vetor P1 ou P2 dos pinos de I/O do msp430.Para o motores é fisicamente impossível eles realizarem seu movimento na frequência do clock por isso era necessário a função de atraso para que os motores realizassem seu movimento antes de receber o próximo par de bits. Os valores colocados nessa função estavam todos em milissegundos e os valores ideais foram colocados de acordo com testes realizados na estrutura.

```
void eixoY(volatile int Sentido1, volatile int Enable1)
{

    const int horario[4] = {passoy1, passoy2, passoy3, passoy4};
    const int anti_horario[4] = {passoy4, passoy3, passoy2, passoy1};
    int passos = 0;



      if(Enable1 == 1){
          for (passos = 0; passos < 4; passos++)
          {
              if (Sentido1 == 1){
                  P2OUT = horario[passos];
              }

              else if(Sentido1 == 0){
                  P2OUT = anti_horario[passos];
              }
              Atraso(7);
          }
      }
      else if (Enable1 ==0){
          P2OUT &= ~(BIT0+BIT1+BIT3+BIT4);
      }
  }
```

Figura 09 - Controle do motor de passos.

VI. RESULTADOS

O código em si foi testado de maneira individual, cada função foi testada em um projeto separado para otimizar a solução de possíveis problemas. Para o teste dos motores foram feitos "loops" infinitos para ambos sentidos, horário e anti horário, o mesmo foi feito com o servo testando suas três posições.

Após testar cada movimento individualmente, foi realizado um teste para o trabalho em conjunto dos motores. Nos primeiros testes houveram falhas no movimento do eixo Y e do eixo Z,os quais não funcionavam simultaneamente com o eixo X, para o problema do eixo Y foram feitos vários testes de energia para verificar se o problema era a falta de potência para girar os motores até que descobriu se que erro estava em uma condição que impedia o código de verificar a posição Y. Quanto ao eixo Z o problema estava relacionado a má utilização dos valores para as posições abaixar e levanta -1 e 1 respectivamente que faziam o eixo Z movimenta somente na sua inicialização. Após detectar e corrigir esses erros a única parte restante do projeto era a recepção das coordenadas para o movimento dos eixos utilizando a comunicação serial.

Primeiro foi necessário verificar que os valores realmente eram atualizados a cada coordenada nova recebida, para isso foram colocadas coordenadas simples que formavam um quadrado, com isso notou se um novo erro o qual as variáveis não estavam sendo atualizadas e o programa travava até receber os novos valores. Além disso os motores nunca giravam no sentido oposto pois os valores sempre eram maiores que os atuais. Esse problema ocorreu pois a variável "atual" sempre retornava para o valor zero ao chegar a posição final daquela coordenada para solucionar isso bastou transformar ela em uma variável estática.

A comunicação serial foi a única que mesmo com esse método não foi possível realizá-la, a comunicação em si não funcionava como o planejado parte do arquivo das coordenadas do GCode era enviada mas a outra parte ficava estática. Além disso para realizar seu teste era necessário verificar o que estava armazenado no registrador da recepção serial, para isso era feito uma comparação de strings simples caso a comparação funcionasse o LED do msp deveria acender porém mesmo com todos os testes não foi possível validar o funcionamento da comunicação serial então as coordenadas foram colocadas manualmente no código.

Foram feitos vários testes até a conclusão da validade da impressora. Inicialmente uma imagem aleatória foi plotada na placa de fenolite afim de testar a estabilidade e precisão da caneta o esperado era uma figura com retas horizontais, verticais e diagonais sem interrupções ou falhas no decorrer do processo e o resultado pode ser visto na figura abaixo.

Figura 10 - Imagem plotada usada para teste

Foi constatado que a caneta estava em falso e se movimentando muito durante a plotagem então o suporte adaptado e novamente um teste foi feito, porém com uma palavra no lugar de um desenho qualquer, o resultado esperado era a palavra SAMU sem falhas ou interrupções tal resultado está presente na figura abaixo.

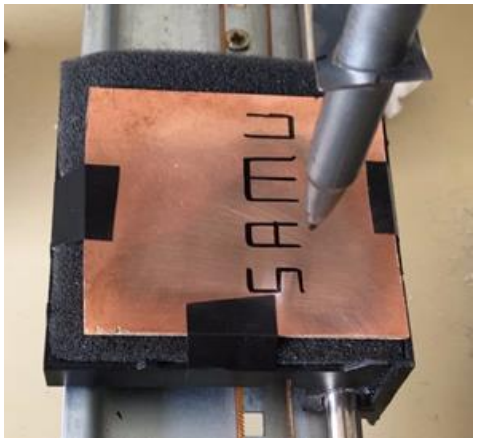

Figura 11 – Imagem plotada para teste palavra SAMU

Com a troca do suporte do eixo Z a estabilidade da caneta foi melhorada e novamente foi feito um teste agora com a simulação de um circuito, devido a falta da comunicação serial o circuito plotado foi o mais simples possível e o resultado está na figura 0X a seguir.

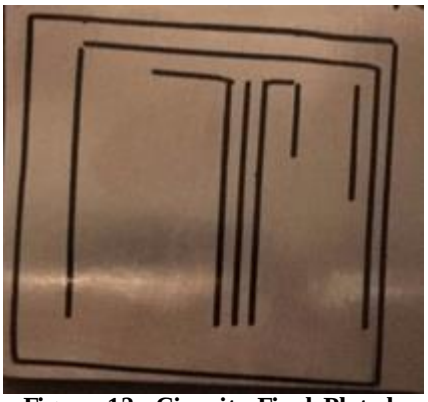

Figura 12 - Circuito Final Plotado

Diante dos resultados apresentados a proposta do projeto foi parcialmente alcançada. Com a falta da comunicação serial a transferência do G-CODE para o MSP430 tornou o processo de plotagem mais demorado devido a inserção das coordenadas da imagem diretamente no código, porém mesmo com essa falta no projeto a praticidade da obtenção do layout na placa ainda é mais viável pela plotter do que pelos outros meios anteriormente citados. Alguns ajustes na estrutura são necessários para deixar cada vez mais precisa a plotagem, mesmo com a troca de do suporte da caneta ela ainda apresenta imperfeições no seu desempenho, tais defeitos são oriundos de inclinações indevidas e movimentos involuntários da caneta no suporte.

VII. CONCLUSÃO

As placas de circuitos impressos são bem presentes em todas as áreas, sua criação, porém é algo em sua maioria cara para ser feito de forma rápida e precisa ou de forma manual que acaba sendo trabalhosa e demorada, com a fabricação da Impressora de Circuitos Impressos esse processo é facilitado não só de forma a visar os custos como ao tempo gasto para a fabricação. Os resultados obtidos com a fabricação da Impressora foram satisfatórios, faltando apenas a comunicação serial para o alcance total dos resultados, porém isso não desvalida de forma alguma o projeto e o que foi obtido até o momento, pelo contrário, com o que foi alcançado ficou clara a diferença entre todos os métodos utilizados.

VIII. REFERÊNCIAS

[1] M. K. K. Prince, M. M. Ansary and A. S. Mondol " Implementation of a Lowcost CNC Plotter Using Spare Parts".

[2] J.Lucca "Plotter de baixo custo para prototipação de placas De circuito impresso".

[3] Akizukidenshi "SG90 micro servo datasheet".

IX. Apêndice

Código do projeto:

```
#include <msp430.h>

static volatile int atual_x = 0 ;
static volatile int atual_y = 0;
static volatile int atual_z = 0;


volatile int PosGcode_x;
volatile int PosGcode_y;
volatile int PosGcode_z;

//Valores de máximo,  ainda precisam ser testados com a
estrutura
const int xMax = 1000;
const int yMax = 1100;
const int zMax = 10;
int f = 4;

const int Desenha = 0;
const int Levanta = 1;

void Atraso(volatile unsigned int t);
void Inicio();
void EixoX(volatile  int Sentido,volatile int Enable);
void EixoY(volatile  int Sentido1,volatile int Enable1);
void MoverGCode(volatile int eixo_x,volatile  int
eixo_y,volatile int eixo_z );

#define passox1 BIT4|BIT5
#define passox2 BIT5|BIT6
#define passox3 BIT6|BIT7
#define passox4 BIT7|BIT4

#define passoy1 BIT0|BIT1
#define passoy2 BIT1|BIT3
#define passoy3 BIT3|BIT4
#define passoy4 BIT4|BIT0

int main(void)
{
        WDTCTL = WDTPW | WDTHOLD;


        BCSCTL1 = CALBC1_16MHZ;
        DCOCTL = CALDCO_16MHZ;
        TA1CCR0 = 40000-1;
        TA1CCR1 = 2000;
        TA1CCTL1 = OUTMOD_7;
        TA1CTL  = TASSEL_2 + ID_3 + MC_1;

        P1OUT = 0x00;
        P1SEL = 0x00;
        P1DIR = 0xFF;
        P1REN = 0x00;

        P2OUT = 0x00;
        P2SEL = 0x00;
        P2DIR = 0xFF;
        P2REN = 0x00;

        P2DIR |= BIT2;
        P2SEL |= BIT2;

/*
        P1IES |= BIT3;
```

```
        P1IE |= BIT3;
*/

        Quadrado();
        Inicio();
        _BIS_SR(GIE  + LPM4_bits);



        //return 0;
}


void Quadrado(){

        MoverGCode(10*f,274*f,Levanta);
        Atraso(5);
        MoverGCode(274*f,274*f,Desenha);
        Atraso(5);
        MoverGCode(274*f,10*f,Desenha);
        Atraso(5);
        MoverGCode(10*f,10*f,Desenha);
        Atraso(5);
        MoverGCode(10*f,274*f,Desenha);
        Atraso(5);
        MoverGCode(10*f,274*f,Levanta);
        Atraso(5);

        MoverGCode(42*f,204*f,Levanta);
        Atraso(5);
        MoverGCode(96*f,204*f,Desenha);
        Atraso(5);

        MoverGCode(42*f,182*f,Levanta);
        Atraso(5);
        MoverGCode(42*f,205*f,Desenha);
        Atraso(5);

        MoverGCode(43*f,157*f,Levanta);
        Atraso(5);
        MoverGCode(43*f,99*f,Desenha);
        Atraso(5);

        MoverGCode(43*f,157*f,Levanta);
        Atraso(5);
        MoverGCode(217*f,157*f,Desenha);
        Atraso(5);

        MoverGCode(43*f,170*f,Levanta);
        Atraso(5);
        MoverGCode(216*f,170*f,Desenha);
        Atraso(5);

        MoverGCode(42*f,182*f,Levanta);
        Atraso(5);
        MoverGCode(216*f,182*f,Desenha);
        Atraso(5);

        MoverGCode(31*f,45*f,Levanta);
        Atraso(5);
        MoverGCode(222*f,45*f,Desenha);
        Atraso(5);

        MoverGCode(44*f,244*f,Levanta);
        Atraso(5);
        MoverGCode(127*f,244*f,Desenha);
        Atraso(5);

        MoverGCode(31*f,259*f,Levanta);
        Atraso(5);
        MoverGCode(215*f,259*f,Desenha);
        Atraso(5);

        MoverGCode(31*f,259*f,Levanta);
        Atraso(5);
        MoverGCode(31*f,45*f,Desenha);
        Atraso(5);
        MoverGCode(0,  0,Levanta);
        Atraso(100);


}

void Atraso(volatile unsigned int t)
{
 TA0CTL   = TASSEL_2  + ID_3 + MC_1;
 TACCR0  = 2000-1;
 while(t--)
 {
        while((TACTL&TAIFG)==0);
        TACTL &= ~TAIFG;
 }
 TACTL = MC_0;
}

//Posição inicial
void Inicio()
{
        atual_x = 0;
        atual_y = 0;
        atual_z = 0;


        MoverGCode(0,0,Levanta);
}

char Fim()
{

        int ret = 0;
        if (atual_x == PosGcode_x && atual_y ==
PosGcode_y && atual_z == PosGcode_z)
        {
        ret = 1;
        }
        return ret;


        /*__asm__("clr.b R12 \n"
        "cmp.w &atual_x, &PosGcode_x \n"
        "jne  fim \n"
        "cmp.w &atual_y, &PosGcode_y \n"
        "jne  fim \n"
        "cmp.w &atual_z, &PosGcode_z \n"
        "jne  fim \n"
        "mov.b #0x1,  R12 \n"
        "fim : ret");
*/
}
```

```
void EixoX(volatile   int Sentido,volatile int Enable1)
{

        const int horario[4] = {passox1, passox2, passox3,
passox4};
        const int anti_horario[4] = {passox4, passox3,
passox2, passox1};
        int passos = 0;


        if(Enable1  == 1){
        for (passos = 0; passos < 4; passos++)
        {
        if (Sentido == 1){
                P1OUT  = horario[passos];

        }
        else if(Sentido  == 0){
                P1OUT  = anti_horario[passos];
        }
        Atraso(7);
        }
        }
        else if (Enable1 == 0){
        P1OUT  &= ~(BIT4+BIT5+BIT6+BIT7);
        }


}
void EixoY(volatile   int Sentido1, volatile int Enable1)
{

        const int horario[4] = {passoy1, passoy2, passoy3,
passoy4};
        const int anti_horario[4] = {passoy4, passoy3,
passoy2, passoy1};
        int passos = 0;




        if(Enable1  == 1){
        for (passos = 0; passos < 4; passos++)
        {
        if (Sentido1 == 1){
                P2OUT  = horario[passos];
        }

        else if(Sentido1 == 0){
                P2OUT  = anti_horario[passos];
        }
        Atraso(7);
        }
        }
        else if (Enable1 ==0){
        P2OUT  &= ~(BIT0+BIT1+BIT3+BIT4);
        }
        }


void LevantaZ()
{

        P2SEL  |= BIT2;
        P2OUT  |=BIT2;
        TA1CCR1  = 2500;

        }

void AbaixaZ()
{
        P2SEL  |= BIT2;
        P2OUT  |=BIT2;
        TA1CCR1  = 1700;

        }


void MoverGCode(volatile  int eixo_x,volatile  int
eixo_y,volatile  int eixo_z){
        int x;
        int y;
        int z;

        x=  atual_x;
        y= atual_y;
        z= atual_z;




        if(eixo_x  >= 0)
        x = eixo_x;
        if(eixo_y  >= 0)
        y = eixo_y;
        if(eixo_z  >= 0)
        z = eixo_z;


        if(x > xMax)
        PosGcode_x = xMax;
        else
        PosGcode_x = x;

        if(y > yMax)
        PosGcode_y = yMax;
        else
        PosGcode_y = y;

        if(z > zMax)
        PosGcode_z = zMax;
        else
        PosGcode_z = z;

        while(!Fim())  //Loop até chegar na posição do
GCode
        {
        if (PosGcode_z < atual_z){
        AbaixaZ();
        atual_z --;
        }

        else if(PosGcode_z > atual_z){
        LevantaZ();
        atual_z++;
        }
        Atraso(10);
        if(PosGcode_x > atual_x)
        {
        EixoX(1,1);
```

```
            atual_x++;
            }
            else if (PosGcode_x < atual_x)
            {
            EixoX(0,1);
            atual_x--;
            }
            if(PosGcode_y > atual_y)
            {
            EixoY(1,1);
            atual_y++;
            }
            else if (PosGcode_y < atual_y)
            {
            EixoY(0,1);
            atual_y--;
            }
            }

            Atraso(10);
}
/*
#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)

{
  if((P1IN&BIT3)==0)
            Quadrado();
}
*/
```