UTILIZAÇÃO DE UM MICROCONTROLADOR PARA ACIONAR UM MOTOR DE PASSO

Edson Mancuzo[1]

edson.mancuzo@etec.sp.gov.br

Faculdade de Tecnologia de Garça – Tecnólogo em Mecatrônica

Abstract – *This paper describes the methodology to drive unipolar stepper motors using microcontroller MC9S08SH8 manufacturing Freescale ®, an electronic circuit and its firmware. The platform used is Code Warrior from Metrowerks IDE 6.3, and the firmware was developed in C language The purpose for writing this paper is to present a solution relatively simple and inexpensive to use unipolar stepper motors.*

Resumo - Este trabalho descreve a metodologia de acionamento de motores de passo unipolar, utilizando o microcontrolador MC9S08SH8 de fabricação da Freescale®, apresentando um circuito eletrônico e seu firmware. A plataforma utilizada é o Code Warrior IDE 6.3 da Metrowerks, e o firmware foi desenvolvido em linguagem C. O propósito para a elaboração deste trabalho é apresentar uma solução relativamente simples e de baixo custo para utilização de motores de passo unipolar.

Palavras-chave: *Microcontrolador, Motor de Passo, Linguagem C, Eletrônica Industrial*

[1]Professor de Eletrônica Industrial do curso de Mecatrônica Industrial da Faculdade de Tecnologia de Garça.

1.DEFINIÇÕES E APLICAÇÕES

O Motor de passo pode ser utilizado em diversas aplicações dentro da Automação de processos. Na indústria automotiva, braços robóticos soldam e parafusam com precisão, máquinas de produção injetam a quantidade quase exata de determinado produto nas embalagens, cirurgias são feitas a distância com o auxílio de equipamentos robóticos sofisticados (CARVALHO,2011). Hoje ele é muito comum em impressoras, braços robóticos, esteiras, máquinas de soldas automatizadas, etc. Praticamente ele faz parte de nosso dia a dia.

Um motor de passo, assim como os outros motores, transforma energia elétrica em energia mecânica (movimento), porém ele não gira se apenas ligá-lo a fonte de alimentação. Há necessidade de acionar seqüencialmente suas bobinas.

O microcontrolador é um componente eletrônico que pode ser comparado a um microcomputador, pois possui memória, dispositivos de entrada e saída, CPU e outros periféricos comuns a ambos. Assim como nos microcomputadores, ele também necessita de um software, que devido a se destinar a um sistema embarcado, chamamos de firmware.

Em nosso dia, estamos em contato com muitos produtos que possuem microcontroladores, como por exemplo, TV, celular, geladeira, máquina de lavar, portão automático, alarme, automóvel (sistema de injeção, ignição, computador de bordo, rádio) e muitos outros.

Considerando o exposto acima, é necessário que o Tecnólogo em Mecatrônica tenha conhecimento das tecnologias citadas neste artigo.

2. O MOTOR DE PASSO

Existem três tipos de motor de passo:

- DC Imã Permanente;

- Relutância Variável;

- Híbridos.

Segundo CARVALHO (2011), as principais características dos motores de passo são:

- O passo angular determina a precisão de deslocamento do motor. Podemos entender como passo angular, como sendo o ângulo de deslocamento provocado por um único pulso aplicado ao motor.

- A velocidade de deslocamento é proporcional à frequência de chaveamento das bobinas do motor.

- Normalmente o motor se mantém inerte quando mantemos energizada uma ou duas bobinas.

O motor que utilizaremos em nosso trabalho é o DC Imã Permanente Unipolar.

Para que ele gire, é necessário alimentar cada uma de suas bobinas seqüencialmente, de acordo com a metodologia desejada.

2.1. Metodologia de Acionamento do Motor de Passo

2.1.1. Acionamento Simples

Consiste em energizar uma bobina por vez seqüencialmente. Quando acionarmos a última bobina, voltamos à primeira. A grande vantagem deste sistema é a simplicidade do algoritmo de controle.

		Bobina			
		1	2	3	4
Sequência	A	1	0	0	0
	B	0	1	0	0
	C	0	0	1	0
	D	0	0	0	1

Tabela1 – Seqüência de acionamento simples

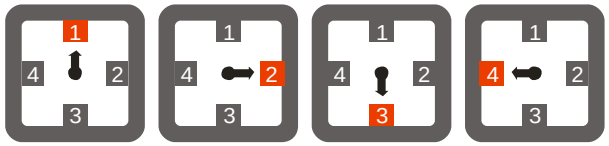

Figura 1 – Acionamento simples

2.1.2. Acionamento Duplo

Acionamos duas bobinas simultaneamente seqüencialmente. Isto proporciona um aumento de torque de 30 a 40%. Quando acionarmos a última bobina, voltamos á primeira.

		Bobina			
		1	2	3	4
Sequência	A	1	1	0	0
	B	0	1	1	0
	C	0	0	1	1
	D	1	0	0	1

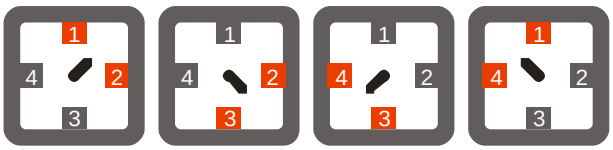

Figura 2 - Acionamento duplo

Tabela2 – Seqüência de acionamento duplo

2.1.3. Acionamento Meio Passo

No acionamento meio passo, como o próprio nome diz, o passo do motor passa a ser a metade, conforme podemos observar abaixo.

		Bobina			
		1	2	3	4
Sequência	A	1	0	0	0
	B	1	1	0	0
	C	0	1	0	0
	D	0	1	1	0
	E	0	0	1	0
	F	0	0	1	1
	G	0	0	0	1
	H	1	0	0	1

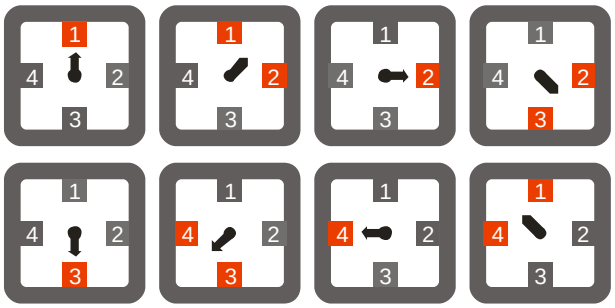

Figura 3 - Acionamento meio passo

Tabela3 – Seqüência de acionamento meio passo

3.MICROCONTROLADOR

3.1.Hardware

O microcontrolador utilizado é MC9S08SH8 da freescale®. Ele é um dispositivo SISC (Complex Instruction Set Computer) de 8 bits, com 8Kbytes de Memória Flash e 256 Kbytes de RAM. Possui 16 pinos e pode ser encontrado em diversos encapsulamentos. Para este trabalho, vamos utilizar o PDIP (Dual in Package – Plastic), por ser de mais fácil manipulação para elaboração do protótipo.

Abaixo segue a descrição de seus pinos:

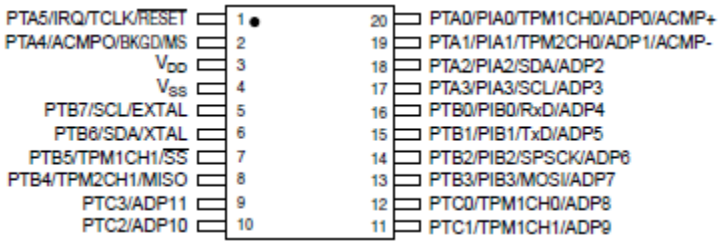

Figura 4 – Pinos MC9S08SH8 – PDIP[2]

Suas principais características são[2]:

- 12 canais de conversor analógico / digital de 8 bits;

- 1 timer geral (MTIM) de 8 bits;

- 2 timers específicos (PWM) de 16 bits com dois canais cada;

- Possui oscilador interno;

- Possui interrupção externa.

A gravação do microcontrolador é feita através de uma interface nativa BDM (Background Debug Mode) e gravador modelo USB Multilink.

O diagrama esquemático completo do circuito está abaixo.

O transformador possui primário 127/220V e dois secundários. O secundário 1 é 16V / 150mA, e secundário 2 é 6V / 500mA. O motor deve ser ligado no J2, sendo os comuns nos terminais 1 e 2 e cada enrolamento respectivamente nos enrolamentos 3, 4, 5 e 6.

O motor de passo a ser utilizado deve ser de 6V e no máximo 400mA.

[1]Dados retirados do Data Sheet: Freescale. MC9S08SH8 Data Sheet HCS08 Microcontrolers, 3a. Revisão. 2008.

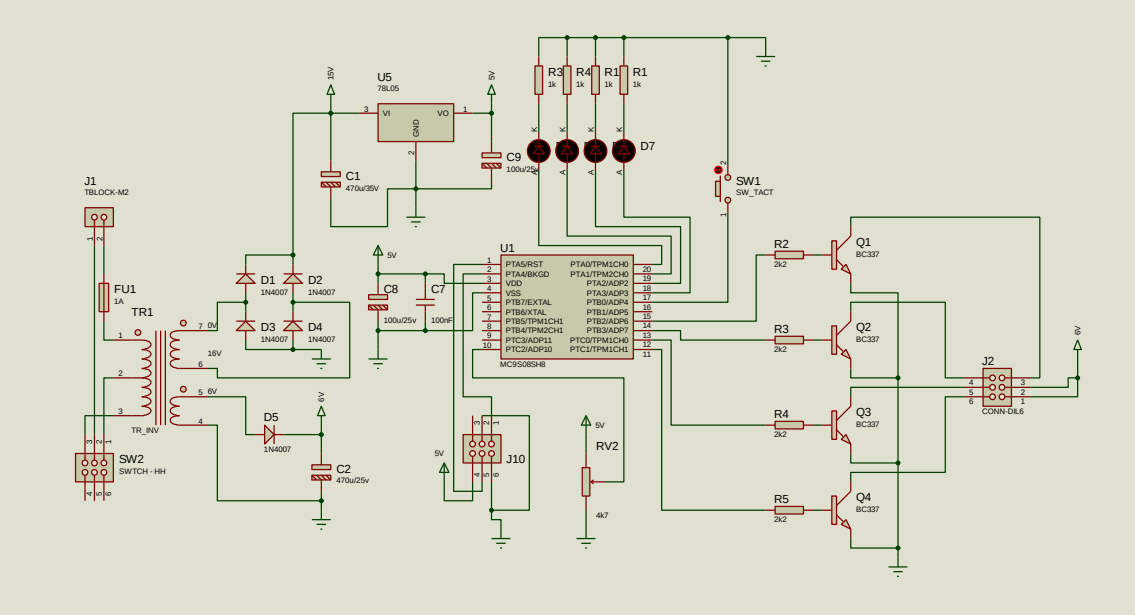

Figura 5 – Diagrama esquemático completo

3.2.Software

Para a elaboração do software, foi utilizado o CodeWarrior IDE 6.3 da Metrowerks. A inicialização dos periféricos foi feita pelo Device Initialization (módulo do CodeWarrior).

O código fonte está abaixo. A chave SW1 comuta o sentido de giro do motor. O potenciômetro RV2 controla a velocidade de giro do motor.

Cada arquivo segue abaixo com seu respectivo nome:

3.2.1. Rotina Principal

```
#include <hidef.h>
#include "derivative.h"
#include "MCUinit.h"
#ifdef __cplusplus
 extern "C"
#endif
void MCU_init(void); /* declaração da função de inicialização */
void main(void) {
  MCU_init(); // Chama rotina de inicialização
  for(;;)
   {
     while (Gira)     //Somente passa na subrotina abaixo a cada 1 ms
       {
        Passo1 = Tempo_Passo;       //Tempo do primeiro passo
        Passo2 = Passo1 + Passo1;   //Tempo do segundo  passo
        Passo3 = Passo2 + Passo1;   //Tempo do Terceiro passo
        Passo4 = Passo2 + Passo2;   //Tempo do Quarto   passo
        if (Chave_Sent)          //Sentido Horário
          {
           if (Timer<Passo1)     //Liga Bobina 1
             {
              Bobina4 = False;
              Bobina1 = True;
             }
           if ((Timer>=Passo1) && (Timer<Passo2))  //Liga Bobina 2
```

```
            {
              Bobina1 = False;
              Bobina2 = True;
            }
          if ((Timer>=Passo2) && (Timer<Passo3))  //Liga Bobina 3
            {
              Bobina2 = False;
              Bobina3 = True;
            }
          if ((Timer>=Passo3) && (Timer<Passo4))  //Liga Bobina 4
            {
              Bobina3 = False;
              Bobina4 = True;
            }
          if (Timer>=Passo4)
            {
              Timer = 0;
            }
        }
       else                  //Sentido Anti-Horário
        {
          if (Timer<Passo1)    //Liga Bobina 4
            {
              Bobina1 = False;
              Bobina4 = True;
            }
          if ((Timer>=Passo1) && (Timer<Passo2))  //Liga Bobina 3
            {
              Bobina4 = False;
              Bobina3 = True;
            }
          if ((Timer>=Passo2) && (Timer<Passo3))  //Liga Bobina 2
            {
              Bobina3 = False;
              Bobina2 = True;
            }
          if ((Timer>=Passo3) && (Timer<Passo4))  //Liga Bobina 1
            {
              Bobina2 = False;
              Bobina1 = True;
            }
          if (Timer>=Passo4)
            {
              Timer = 0;
            }
        }
       Gira = False;        //Desabilita Flag para rodar software
      }
  }
}
```

3.2.2.Rotina de Inicialização

```
/*
** ###################################################################
**
**     Project   : Motor_de_Passo_
**     Processor : MC9S08SH8CPJ
**     Version   : Component 01.007, Driver 01.06, CPU db: 3.00.062
**     Datasheet : MC9S08SH8 Rev. 3 6/2008
**     Date/Time : 06/06/2013, 21:07
**     Abstract  :
**         This module contains device initialization code
**         for selected on-chip peripherals.
**     Contents  :
**         Function "MCU_init" initializes selected peripherals
**
**     Copyright : 1997 - 2009 Freescale Semiconductor, Inc. All Rights Reserved.
**
**     http      : www.freescale.com
**     mail      : support@freescale.com
```

```
** ###################################################################
*/

#include <MC9S08SH8.h>                 /* I/O map for MC9S08SH8CPJ */
#include "MCUinit.h"

/*
** ===================================================================
**     Method      :  MCU_init (component MC9S08SH8_20)
**
**     Description :
**         Device initialization code for selected peripherals.
** ===================================================================
*/
void MCU_init(void)
{
 /* SOPT1: COPT=0,STOPE=0,IICPS=0,BKGDPE=1,RSTPE=0 */
 SOPT1 = 0x02;
 /* SPMSC1: LVWF=0,LVWACK=0,LVWIE=0,LVDRE=1,LVDSE=1,LVDE=1,BGBE=0 */
 SPMSC1 = 0x1C;
 /* SPMSC2: LVDV=0,LVWV=0,PPDF=0,PPDACK=0,PPDC=0 */
 SPMSC2 = 0x00;
 /*  System clock initialization */
 if (*(unsigned char*far)0xFFAF != 0xFF) { /* Test if the device trim value is stored on the specified address */
   ICSTRM = *(unsigned char*far)0xFFAF; /* Initialize ICSTRM register from a non volatile memory */
   ICSSC = *(unsigned char*far)0xFFAE; /* Initialize ICSSC register from a non volatile memory */
 }
 /* ICSC1: CLKS=0,RDIV=0,IREFS=1,IRCLKEN=0,IREFSTEN=0 */
 ICSC1 = 0x04;                       /* Initialization of the ICS control register 1 */
 /* ICSC2: BDIV=2,RANGE=0,HGO=0,LP=0,EREFS=0,ERCLKEN=0,EREFSTEN=0 */
 ICSC2 = 0x80;                       /* Initialization of the ICS control register 2 */
 while(!ICSSC_IREFST) {              /* Wait until the source of reference clock is internal clock */
 }
 /* GNGC: GNGPS7=0,GNGPS6=0,GNGPS5=0,GNGPS4=0,GNGPS3=0,GNGPS2=0,GNGPS1=0,GNGEN=0 */
 GNGC = 0x00;
 /* Common initialization of the CPU registers */
 /* PTASE: PTASE4=0,PTASE3=0,PTASE2=0,PTASE1=0,PTASE0=0 */
 PTASE &= (unsigned char)~0x1F;
 /* PTBSE: PTBSE7=0,PTBSE6=0,PTBSE5=0,PTBSE4=0,PTBSE3=0,PTBSE2=0,PTBSE1=0,PTBSE0=0 */
 PTBSE = 0x00;
 /* PTCSE: PTCSE3=0,PTCSE2=0,PTCSE1=0,PTCSE0=0 */
 PTCSE &= (unsigned char)~0x0F;
 /* PTADS: PTADS4=0,PTADS3=1,PTADS2=1,PTADS1=1,PTADS0=1 */
 PTADS = 0x0F;
 /* PTBDS: PTBDS7=0,PTBDS6=0,PTBDS5=0,PTBDS4=0,PTBDS3=1,PTBDS2=1,PTBDS1=1,PTBDS0=0 */
 PTBDS = 0x0E;
 /* PTCDS: PTCDS3=0,PTCDS2=0,PTCDS1=0,PTCDS0=1 */
 PTCDS = 0x01;
 /* ### Init_ADC init code */
 /* APCTL2: ADPC11=0,ADPC10=0,ADPC9=0,ADPC8=0 */
 APCTL2 = 0x00;
 /* ADCCFG: ADLPC=0,ADIV1=1,ADIV0=0,ADLSMP=0,MODE1=0,MODE0=0,ADICLK1=0,ADICLK0=0 */
 ADCCFG = 0x40;
 /* ADCCV: ADCV9=0,ADCV8=0,ADCV7=0,ADCV6=0,ADCV5=0,ADCV4=0,ADCV3=0,ADCV2=0,ADCV1=0,ADCV0=0 */
 ADCCV = 0x00U;
 /* ADCSC2: ADACT=0,ADTRG=0,ACFE=0,ACFGT=0 */
 ADCSC2 = 0x00;
 /* ADCSC1: COCO=0,AIEN=1,ADCO=0,ADCH4=1,ADCH3=1,ADCH2=1,ADCH1=1,ADCH0=1 */
 ADCSC1 = 0x5F;
 /* ### Init_GPIO init code */
 /* PTAD: PTAD3=0,PTAD2=0,PTAD1=0,PTAD0=0 */
 PTAD &= (unsigned char)~0x0F;
 /* PTADD: PTADD3=1,PTADD2=1,PTADD1=1,PTADD0=1 */
 PTADD |= (unsigned char)0x0F;
 /* ### Init_GPIO init code */
 /* PTBD: PTBD3=0,PTBD2=0,PTBD1=0 */
 PTBD &= (unsigned char)~0x0E;
 /* PTBPE: PTBPE4=1,PTBPE0=1 */
 PTBPE |= (unsigned char)0x11;
 /* PTBDD: PTBDD4=0,PTBDD3=1,PTBDD2=1,PTBDD1=1,PTBDD0=0 */
 PTBDD = (PTBDD & (unsigned char)~0x11) | (unsigned char)0x0E;
 /* ### Init_GPIO init code */
 /* PTCD: PTCD0=0 */
```

```
 PTCD &= (unsigned char)~0x01;
 /* PTCDD: PTCDD0=1 */
 PTCDD |= (unsigned char)0x01;
 /* ### Init_MTIM init code */
 /* MTIMMOD: MOD7=1,MOD6=0,MOD5=0,MOD4=1,MOD3=1,MOD2=1,MOD1=0,MOD0=0 */
 MTIMMOD = 0x9C;
 /* MTIMCLK: CLKS1=0,CLKS0=0,PS3=1,PS2=0,PS1=0,PS0=0 */
 MTIMCLK = 0x08;
 (void)(MTIMSC == 0);               /* Overflow int. flag clearing (first part) */
 /* MTIMSC: TOF=0,TOIE=1,TRST=1,TSTP=0 */
 MTIMSC = 0x60;                     /* Int. flag clearing (2nd part) and timer control register setting */
 /* ### */
 asm CLI;                           /* Enable interrupts */
} /*MCU_init*/
/*
** ===================================================================
**  Interrupção : isrVmtim
**
**  Descrição:
**       Rotina de Interrupção de usuário - Timer interno MTIM
**     Parametro: Nenhum
**     Retorno Nenhum
** ===================================================================
*/
__interrupt void isrVmtim(void)
{
 MTIMSC_TOF = 0;
 Timer_Clk++;            //incrementa variável a cada 1ms
 Timer++;                //base de tempo para os passos
 if (Timer_Clk == 1000)
   {
    Timer_Clk = 0;      //zera a variável a cada 1 s.
    Timer_1seg++;
   }
 Gira = True;       //Sinaliza para passar na rotina principal
}
// fim da isrVmtim */

/*
** ===================================================================
**  Interrupção : isrVadc
**
**  Descrição:
**       Rotina de Interrupção do AD – Conversor Analógico Digital
**     Parametro: Nenhum
**     Retorno Nenhum
** ===================================================================
*/
__interrupt void isrVadc(void)
{
        AD_Passo = ADCRL;
        Tempo_Passo = AD_Passo * 3;
        if (Tempo_Passo <= 50)
          {
           Tempo_Passo = 50;
          }
}
//Fim da isrVadc */

// Inicialização dos registradores da flash
/* NVPROT: FPS7=1,FPS6=1,FPS5=1,FPS4=1,FPS3=1,FPS2=1,FPS1=1,FPDIS=1 */
const unsigned char NVPROT_INIT @0x0000FFBD = 0xFF;
/* NVOPT: KEYEN=0,FNORED=1,SEC01=1,SEC00=0 */
const unsigned char NVOPT_INIT @0x0000FFBF = 0x7E;
extern near void _Startup(void);

// Tabela de Vetores de Interrupção
#ifndef UNASSIGNED_ISR
#define UNASSIGNED_ISR ((void(*near const)(void)) 0xFFFF) /* unassigned interrupt service routine */
#endif
void (* near const _vect[])(void) @0xFFC0 = { /* Interrupt vector table */
       UNASSIGNED_ISR,          /* Int.no. 31 VReserved31 (at FFC0)        Unassigned */
       UNASSIGNED_ISR,          /* Int.no. 30 Vacmp (at FFC2)              Unassigned */
```

```
        UNASSIGNED_ISR,             /* Int.no. 29 VReserved29 (at FFC4)          Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 28 VReserved28 (at FFC6)          Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 27 VReserved27 (at FFC8)          Unassigned */
        isrVmtim,                   /* Int.no. 26 Vmtim (at FFCA)                Used */
        UNASSIGNED_ISR,             /* Int.no. 25 Vrtc (at FFCC)                 Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 24 Viic (at FFCE)                 Unassigned */
        isrVadc,                    /* Int.no. 23 Vadc (at FFD0)                 Used */
        UNASSIGNED_ISR,             /* Int.no. 22 VReserved22 (at FFD2)          Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 21 Vportb (at FFD4)               Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 20 Vporta (at FFD6)               Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 19 VReserved19 (at FFD8)          Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 18 Vscitx (at FFDA)               Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 17 Vscirx (at FFDC)               Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 16 Vscierr (at FFDE)              Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 15 Vspi (at FFE0)                 Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 14 Vtpm2ovf (at FFE2)             Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 13 Vtpm2ch1 (at FFE4)             Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 12 Vtpm2ch0 (at FFE6)             Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 11 Vtpm1ovf (at FFE8)             Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 10 VReserved10 (at FFEA)          Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  9 VReserved9 (at FFEC)           Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  8 VReserved8 (at FFEE)           Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  7 VReserved7 (at FFF0)           Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  6 Vtpm1ch1 (at FFF2)             Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  5 Vtpm1ch0 (at FFF4)             Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  4 VReserved4 (at FFF6)           Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  3 Vlvd (at FFF8)                 Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  2 Virq (at FFFA)                 Unassigned */
        UNASSIGNED_ISR,             /* Int.no.  1 Vswi (at FFFC)                 Unassigned */
        _Startup                    /* Int.no.  0 Vreset (at FFFE)               Reset vector */
};
/* END MCUinit */
```

3.2.3.Arquivo de cabeçalho

```
/*
** ###################################################################
**
**     Project   : Motor_de_Passo_
**     Processor : MC9S08SH8CPJ
**     Version   : Component 01.007, Driver 01.06, CPU db: 3.00.062
**     Datasheet : MC9S08SH8 Rev. 3 6/2008
**     Date/Time : 06/06/2013, 21:07
**     Abstract  :
**         This module contains device initialization code
**         for selected on-chip peripherals.
**     Contents  :
**         Function "MCU_init" initializes selected peripherals
**
**     Copyright : 1997 - 2009 Freescale Semiconductor, Inc. All Rights Reserved.
**
**     http      : www.freescale.com
**     mail      : support@freescale.com
** ###################################################################
*/
#ifndef __Motor_de_Passo__H
#define __Motor_de_Passo__H 1

//Declaração das variáveis
extern unsigned int  Automatico;          // Automático para acionamento teste
extern unsigned int  Timer_Clk;           // Base de tempo geral - 1s
extern unsigned int  Timer;               // Base de tempo para o  motor
extern unsigned char AD_Passo;            // Leitura do AD
extern unsigned int  Tempo_Passo;         // Leitura do AD
extern unsigned int  Timer_1seg;          // Variavel de incremento a cada 1s.
extern unsigned int  Passo1;              //Tempo do primeiro passo
extern unsigned int  Passo2;              //Tempo do segundo passo
extern unsigned int  Passo3;              //Tempo do terceiro passo
extern unsigned int  Passo4;              //Tempo do quarto passo


//Definição dos Ports
```

```
//Port A
#define   Led1            PTAD_PTAD0  //Acionamento do Led1
#define   Led2            PTAD_PTAD1  //Acionamento da Led2
#define   Led3            PTAD_PTAD2  //Acionamento da Led3
#define   Led4            PTAD_PTAD3  //Acionamento da Led4

//Port B
#define   Chave           PTBD_PTBD0  //Chave de Partida
#define   Bobina1         PTBD_PTBD1  //Acionamento da Bobina1
#define   Bobina2         PTBD_PTBD2  //Acionamento da Bobina2
#define   Bobina3         PTBD_PTBD3  //Acionamento da Bobina3
#define   Chave_Sent      PTBD_PTBD4  //Chave de Sentido de Giro

//Port C
#define   Bobina4         PTCD_PTCD0  //Acionamento da Bobina4

//Bits de Flag Diversos
//Flags
#define   True          1
#define   False         0
#define   Gira          _Flags.Bits.M0

/*** Estruturas de Flags */
typedef union {
  byte Byte;
  struct {
    byte M0         :1;
    byte M1         :1;
    byte M2         :1;
    byte M3         :1;
    byte M4         :1;
    byte M5         :1;
    byte M6         :1;
    byte M7         :1;
    byte M8         :1;
    byte M9         :1;
    byte M10        :1;
    byte M11        :1;
    byte M12        :1;
    byte M13        :1;
    byte M14        :1;
    byte M15        :1;
  } Bits;
} Flags;
extern volatile Flags _Flags @0x00000120;
#define Flags                   _Flags.Byte
#define Flags_M0                _Flags.Bits.M0
#define Flags_M1                _Flags.Bits.M1
#define Flags_M2                _Flags.Bits.M2
#define Flags_M3                _Flags.Bits.M3
#define Flags_M4                _Flags.Bits.M4
#define Flags_M5                _Flags.Bits.M5
#define Flags_M6                _Flags.Bits.M6
#define Flags_M7                _Flags.Bits.M7
#define Flags_M8                _Flags.Bits.M8
#define Flags_M9                _Flags.Bits.M9
#define Flags_M10               _Flags.Bits.M10
#define Flags_M11               _Flags.Bits.M11
#define Flags_M12               _Flags.Bits.M12
#define Flags_M13               _Flags.Bits.M13
#define Flags_M14               _Flags.Bits.M14
#define Flags_M15               _Flags.Bits.M15

#ifdef __cplusplus
extern "C" {
#endif
extern void MCU_init(void);
#ifdef __cplusplus
}
#endif

//Protótipo das funções
__interrupt void isrVmtim(void);
```

```
__interrupt void isrVadc(void);
/* END Motor_de_Passo_ */
#endif
```

4. CONSIDERAÇÕES FINAIS

O protótipo foi construído em uma placa matriz de contato. Foi utilizado um motor que atende a especificação acima, e o mesmo funcionou conforme previsto.

Aumentando a potência do enrolamento de 6V (do transformador) e trocando os transistores Q1,Q2, Q3 e Q4 por modelos de maior corrente e ganho, podemos facilmente acionar motores de maior potência.

5. BIBLIOGRAFIA

CARVALHO, Geraldo. Máquinas Elétricas Teoria e Ensaios, São Paulo, 4ª. Edição. Erika, 2011.

Freescale. MC9S08SH8 Data Sheet HCS08 Microcontrolers, 3a. Revisão. 2008.